INTERFACE TECNOLÓGICA

# MIGRAÇÃO DE PARTES DE UMA APLICAÇÃO DESKTOP PARA O FORMATO DE API REST: estudo de caso

## *MIGRATION OF PARTS OF A DESKTOP APPLICATION TO THE REST API FORMAT: case study*

Victor Hugo Manduca Rizo – victor.manduca@hotmail.com

Felipe do Espírito Santo – felipe.santo@fatectq.edu.br

Faculdade de Tecnologia de Taquaritinga (Fatec) – Taquaritinga – São Paulo – Brasil

**DOI: 10.31510/infa.v17i1.773**

## RESUMO

Anos atrás, era comum desenvolvermos sistemas baseados na arquitetura de dois tiers para sistemas *Desktop*, porém muito se evoluiu através do tempo, de avanços tecnológicos a processuais, além da demanda do mercado por aplicações que fossem mais performáticas, com melhor usabilidade, maior quantidade de profissionais para atender a demanda e com custo reduzido para desenvolver e manter a mesma. Percebeu-se que, com o avanço da tecnologia, que os sistemas *Desktop* não forneciam uma abrangência necessária para atender grandes quantidades de usuários, afetando a disseminação dos sistemas desenvolvidos. Porém, com a chegada da *Web*, houve uma massiva quantidade de adeptos a esse novo formato de desenvolvimento, uma vez que qualquer pessoa poderia se conectar na rede e acessar a aplicação de qualquer lugar, o que contribuiu também para disseminar o conhecimento sobre essas tecnologias. O paradigma qualitativo orientou essa investigação e a metodologia de pesquisa utilizada para a coleta de dados deste trabalho foi o estudo de caso da migração parcial de um sistema Desktop para a Web de uma empresa de automação comercial na região de Ribeirão Preto, São Paulo. Este artigo visa abordar as vantagens de uma migração, mesmo que parcial, do sistema *Desktop* para sistema *Web*, utilizando as arquiteturas MVC e *Rest*, apresentando suas características e seu impacto nos produtos desenvolvidos pelas organizações.

**Palavras-chave:** Migração de sistemas. REST. MVC.

## ABSTRACT

Years ago, it was common to develop systems based on two-tier architecture for Desktop systems, but much has evolved over time, from technological to procedural advances, in addition to the market demand for applications that have better performance, usability, and more professionals to meet the market demand and with reduced cost to develop and maintain it. It was noticed that, with the advancement of technology, that Desktop systems did not provide the necessary coverage to serve large numbers of users, affecting the dissemination of developed systems. However, with the arrival of the Web, there was a massive number of adherents to this new development format, since anyone could connect to the network and access the application from anywhere, which also contributed to disseminate knowledge about

these technologies. The qualitative paradigm guided this investigation and the research methodology used to collect data from this work was the case study of the partial migration of a Desktop system to the Web of a commercial automation company in the region of Ribeirão Preto, São Paulo. This article aims to present the advantages of a migration, even if partial, from the Desktop system to the Web system, using the MVC and Rest architectures, presenting their characteristics and their impact on the products developed by organizations.

**Keywords:** System migration. REST. MVC.

## 1 INTRODUÇÃO

A evolução da *Web* nas últimas décadas proporcionou o surgimento de uma nova opção para a construção de sistemas, o desenvolvimento de aplicações totalmente *online,* permitindo acesso aos seus dados independentemente de local e hora.

De acordo com a pesquisa realizada pelo *Stack Overflow* em 2019 (*Developer Survey Results*, 2019), a linguagem de programação Javascript, acompanhado das linguagens de marcação e estilização: HTML e CSS são as mais utilizadas e populares. Estatisticamente mais de 50% das respostas são de desenvolvedores *Web* (*fullstack, backend* e *frontend*) e cerca de 21% são *Desktop*, sendo assim, vê-se claramente, que este último modelo de desenvolvimento está entrando em decadência, dado que, no estudo da União Internacional de Telecomunicações (UIT), 3,9 bilhões de pessoas estão conectadas à internet.

Baseado nos dados apresentados acima, com a quantidade de desenvolvedores que o estudo mostrou para a plataforma *Web*, o custo de desenvolvimento para a mesma se torna reduzido, pois para se desenvolver um MVP é mais fácil e fiel a realidade por todos os sistemas operacionais terem um navegador e poderem executar a aplicação e não estarem presos à capacidade e localização da máquina do cliente, além de, como mais pessoas conhecem as tecnologias utilizadas, a manutenção do código gerado e dos sistemas desenvolvidos se tornam mais eficientes.

*Minimum Viable Product* ou MVP é o que Moogk (2012) define como um modelo de experimento para confirmar ou refutar a hipótese de valor e a capacidade de crescimento de um produto, que deve ter apenas as funcionalidades essenciais para seu funcionamento, a fim de mensurar sua tração no mercado.

Portanto, com base em tais informações, será realizada uma breve introdução, neste artigo, sobre os temas: sistemas *Desktop* e *Web,* a definição de cada um, além das arquiteturas

que são mais utilizadas para cada um dos sistemas; apresentação de um caso real sobre migração de um sistema *Desktop* para *Web* de uma empresa de automação comercial na região de Ribeirão Preto, São Paulo, que será tratada como Empresa X; por fim, discutir os resultados dessa migração e relacionar com a pesquisa aqui realizada.

## 2 SISTEMAS DESKTOP

> *Desktop* é uma palavra da língua inglesa comumente utilizada para referir-se aos sistemas de software(programas) que funcionam em um computador independentemente de estar conectado em rede a um servidor ou a outros computadores. Um sistema *desktop* é uma opção para explorar os recursos do computador, quando não há servidores ou conexão à Internet. (ANDRADE,2015, p.22)

Com tal definição, pode-se chegar na arquitetura mais comum para esta plataforma que, segundo Silveira et al. (2011), nos sistemas mais antigos, como *Desktop* e *Mainframes,* era utilizada a arquitetura de *tiers*, uma divisão focada em dividir fisicamente as partes do sistema, como *Cloud Computing*, servidor *Web*, banco de dados, cliente remoto, servidor de aplicação, entre outros, logo pode-se concluir que *tiers* diferentes são módulos do sistema que são executados e se comunicam de maneira remota através da rede.

Nos sistemas *Desktop* é ainda comum a utilização de dois *tiers,* geralmente empregadas em aplicações Delphi e Visual Basic. No *tier* de interface é empregado, comumente, muito código a fim de abrandar os esforços do servidor, além disso emprega-se o uso das *stored procedures* que são "[...] rotinas definidas no banco de dados, identificadas por um nome pelo qual podem ser invocadas." (CARVALHO, 2015, p.94), visando assegurar e isolar a lógica, e garantir o mínimo de perda de performance ao adicionar mais um *tier*. Contudo, é gerado problemas de dependência do banco de dados, integrações custosas e baixo desacoplamento de responsabilidades.

No estudo de caso será observada a versão *Desktop* de um sistema que estava instalado nas máquinas do cliente e que realizava todo seu processamento localmente, nesses mesmos computadores onde estava instalado a aplicação, conforme os autores citados previamente, pode haver tipos de sistemas *Desktops*, que podem utilizar diferentes arquiteturas e trabalhar de formas variadas com relação às suas camadas e onde os processos são realizados.

## 3 SISTEMAS WEB

Segundo Berenguel et al. (2008), a *World Wide Web* (*Web*) é essencialmente baseada no tráfego de documentos de hipermídia interligados uns com os outros por *hiperlinks*, que segundo a W3C (1995), é a relação entre duas partes, das quais o link parte da origem e chega ao destino. Tal propriedade é reconhecida também como encadeamento. Logo, partindo de um documento, consegue-se ascender a muitos outros na *Web*, vulgarmente conhecido como "navegar na internet". Outra particularidade é o endereçamento, que originando-se de um identificador conhecido como URI (*Uniform Resource Identifier*), pode-se encontrar diretamente qualquer documento na *Web*.

Para o presente trabalho ficará definido como sistema *Web*, todo o sistema que tenha recursos acessíveis por meio de requisições HTTP (*HyperText Transfer Protocol*) ou HTTPS (*HyperText Transfer Protocol Secure*), mesmo que não tenha seu cliente todo desenvolvido em tecnologias da *Web*, para a compreensão desse processo e da migração que será apresentada ao decorrer do artigo, serão apresentados os padrões de arquitetura mais comuns, tais como o *Model-View-Controller* (MVC) e *Rest* (*Representational State Transfer*), com suas devidas restrições.

O HTTP foi desenvolvido para prestar suporte à transferência de tais documentos de hipermídia. Este é um protocolo de camada de aplicação, que é a camada em que há a interação com o usuário, segundo Kumar, Dalal e Dixit (2014), do qual oferece uma interface uniforme para a transmissão de documentos. Qualquer cliente e servidor que implementa o HTTP pode se comunicar e trafegar documentos.

### 3.1 MODEL-VIEW-CONTROLLER (MVC)

Sommerville (2013) descreve o modelo MVC como, a separação da apresentação e a interação dos dados do sistema. A estruturação do sistema é configurada em três componentes lógicos que se comunicam que são: o *Model* (Modelo), que realiza o gerenciamento do sistema de dados e suas operações; a *View* (Visão), na qual é responsável por definir e administrar como os dados irão ser expostos ao usuário; e o *Controller* (Controlador), no qual administra a interação do usuário com o sistema e as repassa para a *View* e o *Model*.

Quanto ao uso deste modelo, o autor aponta que ele é usado quando há diversos modos

de ver e interagir com os dados. Além de quando os futuros requisitos de visualização e interação com os mesmos são desconhecidos. Como vantagem, é ainda afirmado que: “Permite que os dados sejam alterados de forma independente de sua representação, e vice-versa. Apoia a apresentação dos mesmos dados de maneiras diferentes, com as alterações feitas em uma representação aparecendo em todas elas.” SOMMERVILLE (2013, p.109). Já como desvantagem, o autor descreve que quando o modelo de dados, que Lisboa Filho e Iochpe (1999) definem como agrupamento de conceitos que podem ser utilizados para relatar um grupo de dados e as operações para manuseá-los, e as interações são simples, há a possibilidade de aumentar a quantidade de código e sua complexidade.

## 3.2 REST

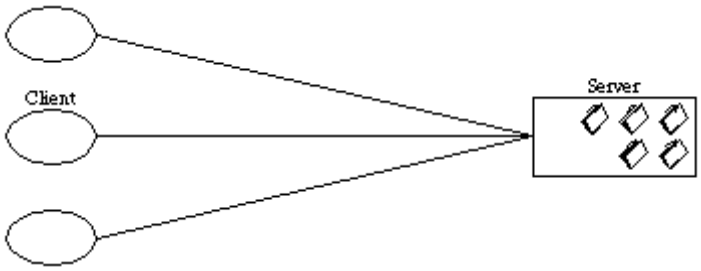
Ferreira e Knop (2016) apresentam que o modelo de arquitetura *Rest* (*Representational State Transfer*), criado em 2000 por Roy Fielding, é uma coleção de princípios e restrições, que se entende como um conjunto de boas práticas. Tais restrições visavam determinar o modo como os padrões deveriam ser moldados, por exemplo o HTTP e URI. Ainda neste capítulo, serão abordadas algumas das restrições, que são: cliente-servidor, sem estado, cache, interface uniforme, sistema em camadas, código sob demanda.

A restrição de cliente-servidor tem como premissa dividir as responsabilidades das partes de um sistema. Uma forma de aplicá-la seria dividir a camada de interface do usuário do back-end e seu desacoplamento do banco de dados. Desta forma, há mais espaço para escalabilidade, que é a habilidade de um sistema acomodar um aumento do número de elementos ou objetos, para um processo de crescimento de volume de trabalho, e/ou para ser suscetível à ampliações (BONDI, 2000), pois o servidor poderá ser chamado por qualquer aplicação cliente, independentemente da plataforma ou da linguagem em que foi escrito o *Software* (FERREIRA; KNOP, 2016).

A restrição *sem estado* é derivada de cliente-servidor, porém nenhum estado de sessão é tolerado no componente de servidor. Uma requisição realizada pelo cliente ao servidor conterá todas as informações para entender a solicitação, mas não em se aproveitar de possíveis contextos armazenados no servidor (Figura 1). O estado da sessão é guardado do lado do cliente. Utilizando esta restrição, as APIs começam a possuir visibilidade, confiabilidade e escalabilidade. Com a utilização de balanceadores de carga será possível adicionar inúmeros

servidores para a aplicação e o cliente não os identificará, uma vez que estão transparentes (FIELDING, 2000 apud FERREIRA; KNOP, 2016).

Figura 1 - Restrição *sem estado*




Fonte: Fielding (2000)

Segundo Ferreira e Knop (2016) para uma melhor performance, no modelo *Rest*, deve haver o uso de *cache*, que segundo Smith (1982) é a memória usada para guardar momentaneamente porções de conteúdo da memória principal que está em uso, pois o cenário ideal é aquele em que a requisição não necessite se comunicar com o servidor uma vez que já há uma cópia no navegador do usuário, deste modo, é possível eliminar a latência da rede.

Fielding (2000) mostra que ao se aplicar o princípio de generalização da engenharia de *Software* - que Hürsch e Lopes (1995) definem como um novo paradigma com o objetivo modularizar a aplicação, deixando cada componente separado do restante do sistema - para a interface dos componentes, a arquitetura geral do sistema será simplificada e a visibilidade das interações melhorada. Além disso, há o encorajamento de melhoria dos módulos de forma independente quando as implementações são desacopladas dos serviços que os proveem. Sendo assim, a restrição de interface uniforme tem o objetivo de tornar independente a comunicação entre servidor e cliente. Para se atingir uma interface uniforme, definem-se algumas restrições arquiteturais: identificação de recursos, manipulação de recursos por meio de representações, mensagens auto descritivas, por fim, hipermídia como um motor de estado da aplicação.

A restrição de sistema em camadas permite que uma arquitetura seja composta de camadas hierárquicas por meio de limitações no comportamento dos componentes de modo que cada componente apenas tenha conhecimento de sua própria camada. Ao fazer isso, é reduzida a complexidade do sistema, além de promover encapsulamento para cada serviço novo ou legado e aumentar a escalabilidade do sistema (FIELDING, 2000).

Por fim, Fielding (2000) aborda a restrição de código sob demanda, na qual a arquitetura *Rest* permite que uma funcionalidade do cliente seja estendida por meio do download e

execução do código no formato de plugin ou script. Desta forma, há uma redução de funcionalidades que necessitam serem pré-instaladas, possibilitando que novas funcionalidades possam ser baixadas pelo cliente, o que aumenta a escalabilidade do sistema.

## 4 MIGRAÇÃO DE SISTEMAS

Um estudo feito pela empresa Esteio Engenharia e Aerolevantamentos S.A. em 2008, no qual aborda as principais características e diferenças nos Sistemas de Informações Geográficas (SIG) *Desktop* e *Web*, mostra algumas características de sistemas *Desktop*, sendo: sua abrangência está limitada à estação operacional do SIG, a arquitetura do *Software* está envolvida no padrão *Desktop*, as atualizações e implementações de dados são exclusivas aos setores de trabalho, os bancos de dados são instalados nas estações de trabalho ou nos servidores, sua velocidade é cerceada às configurações do hardware que está em operação e sua segurança está restringida à autorização do uso da estação (FURQUIM; FURQUIM, 2008).

Para o ambiente *Web*, o estudo da Esteio Engenharia e Aerolevantamentos S.A., aponta que com o sistema *Web* há uma maior cobertura quanto a quantidade de usuários, pois aqueles que possuírem acesso à rede poderão usufruir do sistema, além de atualizações e implementações que podem ser realizadas remotamente, arquitetura *Rest* (cliente-servidor), sua velocidade agora depende da rede e não das configurações do maquinário que está sendo operado, por fim, a segurança da aplicação se torna mais eficiente dados os protocolos de autenticação.

Segundo um estudo realizado por Lunelli, Mendonça e Delazari (2012), no qual foi realizado a migração de uma aplicação de visualização cartográfica em ambiente *Desktop* para *Web*, foi observado uma melhora na usabilidade, interatividade e aceitabilidade do sistema, além de ter permitido que novas funcionalidades possam ser adicionadas.

## 5 METODOLOGIA

O paradigma qualitativo orientou essa investigação e a metodologia de pesquisa utilizada para a coleta de dados deste trabalho foi o estudo de caso da migração parcial de um sistema Desktop para a Web da Empresa X, para melhor compreender os conceitos aplicados nessa migração também foi realizado um levantamento bibliográfico para a discussão do

referencial teórico apresentado nos primeiros capítulos do trabalho.

Para Freitas e Jabbour (2011, p. 9) “quando a finalidade é explicar ou descrever um evento ou uma situação, a abordagem adotada deve ser a qualitativa”. A escolha desta metodologia se mostrou oportuna e foi feita devido ao objetivo do trabalho centrar-se na interpretação do fenômeno do objeto de estudo, ou seja, no seu processo e significado (GODOY, 1995).

## 6 RESULTADOS E DISCUSSÃO

Partindo dos conceitos e estudos abordados neste documento, foi possível tratar de um *case* real, citado no capítulo de introdução, no qual foi realizada a migração parcial de um sistema gerenciador de postos de gasolina de *Desktop* para *Web*, em que atuava nos preços das bombas de gasolina, alterando também a arquitetura do sistema.

Antes da migração, o sistema *Desktop*, desenvolvido em Delphi, utilizava uma arquitetura com dois *tiers*, dos quais um *tier* era a base de dados e o outro o sistema em si, com toda a regra de negócio e seus componentes visuais. Após um ano de uso, os usuários relataram que o sistema começou a apresentar lentidão (não foram realizados testes pela empresa para medir a velocidade de execução do sistema, a constatação da lentidão foi realizada através da observação da utilização do sistema), e foi constatado que isso devia-se ao fato de que a máquina em que o sistema estava para cuidar de todo o posto possuía uma configuração de baixo desempenho, a máquina tinha mais de 4 anos, possuía apenas 8GB de memória RAM e trabalhava com o *SQL Server* 2019 instalado localmente, por esse motivo não suportava as atividades em que estava sendo submetida.

Então, optou-se por rever toda a arquitetura e a tecnologia dado que isso também poderia ajudar na performance da aplicação. Por fim, foi escolhido a arquitetura *Rest*, optando por utilizar as restrições de cliente-servidor, interface uniforme, sistema em camadas e código sob demanda, pois foi estruturado que o sistema *Desktop* passaria a ser apenas um cliente de vários micro serviços que a tecnologia Node.JS (uma aplicação interpretada em *runtime* desenvolvida sobre o motor Javascript) iria prover, tornado o sistema Delphi em cliente e o sistema em Node.JS em servidor, consumindo o banco de dados. Entretanto, muitos serviços e funcionalidades do sistema *Desktop* ainda irão ser mantidos.

Após a migração do serviço e sua devida implantação no posto de gasolina, o cliente

relatou um aumento significativo da performance no serviço que fora migrado (a Empresa X não realizou testes para quantificar a melhoria em tempo exato), impedindo também que houvesse lentidão no sistema em Delphi que também cuida do estoque, caixa e emissão de Notas Fiscais. Além disso, por causa da arquitetura *Rest*, a aplicação se tornou mais segura e possibilitou que mais funcionalidades fossem implementadas, tornando-se escalável.

Cabe reforçar aqui mais uma vez que os dados exatos da melhoria de performance não foram liberados pela Empresa X, não foi realizado um acompanhamento exato do tempo de resposta por exemplo, a melhoria foi constatada com base na afirmação dos usuários, na redução dos chamados de suporte e manutenção e no acompanhamento realizado durante a migração.

Poppendieck e Poppendieck (2011) citando o modelo Kano afirmam que "aumenta-se a satisfação do cliente na proporção direta do aumento de desempenho", por isso mostrou-se eficaz essa abordagem da migração para potencializar o desempenho da aplicação.

## 7 CONCLUSÃO

Observando os resultados da reformulação do sistema desktop para adoção de tecnologias *Web* em sua estrutura pode-se concluir que o processo foi benéfico para o cliente, o principal interessado na aplicação, o aumento de desempenho na percepção dele proporcionou a redução de chamados de suporte.

Conforme citado pela pesquisa do Stackoverflow na introdução deste artigo, observa-se que a adoção de uma tecnologia mais nova e moderna proporciona uma ampliação nas ofertas de profissionais para desenvolver na *stack* adotada, algo que vai complicando-se ao passar do tempo quando ela já não é atualizada e as empresas ficam apenas com sistemas legados, os profissionais que entendem daquela tecnologia vão saindo do mercado ou atualizando-se e as empresas se encontram com a mão de obra escassa.

Conclui-se que a migração de tecnologias *Desktop* para tecnologias *Web* é benéfica, mesmo que realizada parcialmente, assim como neste estudo de caso, ainda pode entregar resultados positivos de performance, agradar o cliente gerando uma sobrevida para o *Software* legado e proporcionar uma migração de forma gradual até a adoção completa de tecnologias *Web*, com isso espera-se que pesquisas futuras possam observar mais casos de migrações realizadas e discutir eventuais vantagens e desvantagens que o tempo possa apresentar do processo de migração.

## REFERÊNCIAS

ANDRADE, R. A. Z. de. Sistema desktop para gerenciamento de culturas agrícolas. 2015. 57 f. Trabalho de Conclusão de Curso (Graduação) - Universidade Tecnológica Federal do Paraná, Pato Branco, 2015.

BERENGUEL, A. L. A. et al. Arquitetura AAA em sistemas Web baseados em REST. 2008. Disponível em: <http://www.alice.cnptia.embrapa.br/alice/handle/doc/24592>. Acesso em: 06 fev. 2020.

BONDI, André B.. Characteristics of scalability and their impact on performance. Proceedings Of The Second International Workshop On Software And Performance - Wosp '00, [s.l.], p. 195-203, set. 2000. ACM Press. http://dx.doi.org/10.1145/350391.350432. Disponível em: < https://dl.acm.org/doi/pdf/10.1145/350391.350432 >. Acesso em: 16 maio 2020.

CARVALHO, V. Mysql come com o principal banco de dados open source do mercado. Vila Mariana: Casa do Código, 2015.

DEVELOPER SURVEY RESULTS 2019. Stack Overflow. [S.l.]. 2019. Disponível em: < https://insights.stackoverflow.com/survey/2019 >. Acesso em: 16 dez. 2019.

DEVELOPER SURVEY RESULTS 2016. Stack Overflow. [S.l.]. 2016. Disponível em: < https://insights.stackoverflow.com/survey/2016 >. Acesso em: 04 de fev. 2020.

FERREIRA, W. O.; KNOP, I. O.. Estruturação de Aplicações Distribuídas com a Arquitetura REST. 2016. Disponível em: <https://seer.cesjf.br/index.php/cesi/article/view/812>. Acesso em: 10 fev. 2020.

FIELDING, Roy T. Architectural Styles and the Design of Network-based Software Architectures. 2000. 162 f. Dissertação (Mestrado) - Curso de Information And Computer Science, University Of California, Irvine, 2000. Disponível em: <https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>. Acesso em: 08 fev. 2020.

FREITAS, W. R. S.; JABBOUR, C. J. C. Utilizando estudo de caso(s) como estratégia de pesquisa qualitativa: boas práticas e sugestões. *Estudo & Debate*, Lajeado, v. 18, n. 2, p. 7-22, 2011. Disponível em: https://edisciplinas.usp.br/pluginfile.php/2148238/mod_resource/content/1/Protocolo%20de%20estudo%20de%20caso.pdf

FURQUIM, A.; FURQUIM, M. Principais características e diferenças entre sistemas SIG desktop e SIG web. Esteio, 2008. Disponível em: < https://www.esteio.com.br/downloads/pdf/SIG-Desktop_e_SIG-Web.pdf >. Acesso em: 11 jan. 2020.

GODOY, A. S. Introdução a pesquisa qualitativa e suas possibilidades. Revista de

Administração de Empresas. São Paulo, v. 35, n. 2, p. 57-63, Mar./Abr. 1995

HÜRSCH W. L.; LOPES C. V. Separation of Concerns. Northeastern University, Boston, Estados Unidos da América. Technical report NU-CCS-95-03, 1995.

KUMAR, S.; DALAL, S.; DIXIT, V. THE OSI MODEL: OVERVIEW ON THE SEVEN LAYERS OF COMPUTER NETWORKS. International Journal Of Computer Science And Information Technology Research. [s. I.], p. 461-466. jul. 2014.

LISBOA FILHO, J.; IOCHPE, C. Um Estudo sobre Modelos Conceituais de Dados para Projeto de Bancos de Dados Geográficos. Ip-informática Pública, Belo Horizonte, p.67-90, dez. 1999.

LUNELLI, Pâmela A.; MENDONÇA, André Luiz A. de; DELAZARI, Luciene S. MIGRAÇÃO DE ATLAS ELETRÔNICO EM AMBIENTE DESKTOP PARA AMBIENTE WEB. Iv Simpósio Brasileiro de Ciências Geodésicas e Tecnologias da Geoinformação, Recife, p.1-7, maio 2012.

MOOGK, Dobrila R. Minimum Viable Product and the Importance of Experimentation in Technology Startups: Technology Innovation Management Review. Echnology Innovation Management Review, Ottawa, v. 2, n. 3, p.23-26, Mar. 2012. Disponível em: <https://timreview.ca/article/535>. Acesso em: 04 fev. 2020.

OPENJS FOUNDATION. About Node.js. Disponível em: <https://nodejs.org/en/about/>. Acesso em: 17 fev. 2020.

PEZZOTTI, R.Com 3,9 bilhões de usuários no mundo, o que acontece na web em um minuto. UOL. São Paulo, abr. 2019. Disponível em: < https://economia.uol.com.br/noticias/redacao/2019/04/01/com-39-bilhoes-de-usuarios-no-mundo-o-que-acontece-na-web-em-um-minuto.htm >. Acesso em: 16 dez. 2019.

POPPENDIECK, M.; POPPENDIECK, T. Implementando o Desenvolvimento Lean de Software: Do conceito ao dinheiro. Porto Alegre: Bookman, 2011.

SILVEIRA, P. et al. Introdução à Arquitetura e Design de Software: Uma visão sobre a plataforma Java. Vila Marina: Elsevier, 2011.

SMITH, Alan Jay. Cache Memories. 1982. Disponível em: < https://dl.acm.org/doi/abs/10.1145/356887.356892 >. Acesso em: 16 maio 2020.

SOMMERVILLE, I. Engenharia de Software. 9. ed. São Paulo: Pearson Education do Brasil Ltda., 2013.

W3C. Terms. 1995. Disponível em: <https://www.w3.org/MarkUp/html-spec/html-spec_2.html#SEC2>. Acesso em: 08 fev. 2020.