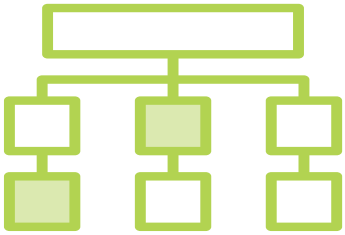

PADRÃO DE PROJETO FLUX PARA SISTEMAS DE INFORMAÇÃO NA ÁREA DA SAÚDE: UM ESTUDO DE CASO SOBRE O SISTEMA RETRATOS DA ATENÇÃO PRIMÁRIA À SAÚDE

Aldiney José Doreto

Mestre em Enfermagem pelo Programa de Pós-Graduação em Enfermagem da Universidade Federal do Paraná - UFPR. Pós-Graduação em Educação Profissional na Área de Saúde: Enfermagem (FIOCRUZ). Graduação em Enfermagem pela Universidade Estadual de Maringá. Pesquisador do Núcleo Avançado de Inovação Tecnológica (NAVI/IFRN). E-mail: aldineydoreto@gmail.com

Diêgo Ferreira de Lima

Bacharel em Ciências e Tecnologia e graduando em Engenharia de Computação pela Universidade Federal do Rio Grande do Norte (UFRN). Técnico em Informática pelo Instituto Federal do Rio Grande do Norte (IFRN). Pesquisador do Núcleo Avançado de Inovação Tecnológica (NAVI/IFRN).
E-mail: diego.lima@lais.huol.ufrn.br

Beatriz Soares de Souza

Bacharel em Ciências e Tecnologia e graduanda em Engenharia de Computação pela Universidade Federal do Rio Grande do Norte (UFRN). Técnica em Informática pelo Instituto Federal do Rio Grande do Norte (IFRN). Pesquisadora do Núcleo Avançado de Inovação Tecnológica (NAVI/IFRN).
E-mail: beatriz.souza@lais.huol.ufrn.br

Ítalo Epifânio de Lima e Silva

Graduando em Tecnologia da Informação pela Universidade Federal do Rio Grande do Norte (UFRN). Técnico em Informática pelo Instituto Federal do Rio Grande do Norte (IFRN). Membro do Programa de Educação Tutorial de Ciências da Computação (PET). Núcleo Avançado de Inovação Tecnológica (NAVI/IFRN). Pesquisador do Laboratório de Inovação Tecnológica em Saúde (LAIS).
E-mail: italo.epifanio@lais.huol.ufrn.br.

Isaque Kaio de Araujo Rodrigues

Graduando em Tecnologia em Análise e Desenvolvimento de Sistemas pelo Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte (IFRN). Técnico em Informática para Internet pelo Instituto Metrópole Digital (IMD). Pesquisador do Núcleo Avançado de Inovação Tecnológica (NAVI/IFRN).
E-mail: isaque.rodrigues@navi.ifrn.edu.br

RESUMO

No desenvolvimento *web* moderno existe uma forte tendência de dividir os elementos de uma interface em uma estrutura de componentes. Por mais que esta tendência esteja fundamentada em bons princípios, há algumas armadilhas que comprometem o ciclo de desenvolvimento de novos projetos e que não são sempre evitadas. Por essa razão, um time de desenvolvimento que não adota um padrão de projeto adequado pode se deparar com impactos negativos no tempo empregado para cada ciclo de desenvolvimento. Conforme a aplicação cresce, torna-se cada vez mais difícil encontrar erros, prover manutenção ao código, e até mesmo desenvolver novas funcionalidades. Nesse contexto, o presente artigo descreve a importância do conhecimento de padrões de projeto e a utilização do padrão Flux, apresentando o estudo de caso de um sistema chamado Retratos da Atenção Primária em Saúde, cuja missão é tornar os dados do Programa Nacional de Melhoria do Acesso e da Qualidade da Atenção Básica (PMAQ) facilmente acessíveis.

Palavras-chave: Padrão de projeto. Flux. Vue. Vuex. Web componentes. PMAQ-AB.

ABSTRACT

Modern web development brings the tendency to divide interface elements in a component structure. Even though this tendency is based on good principles, some pitfalls can compromise the development of new projects and should be avoided. For this reason, a development team that does not adopt an appropriate design pattern may face negative impacts over the time of its development cycles. As the application grows, it gets harder to find bugs, maintain the code and develop new functionalities. Thus, this paper describes the importance of knowing design patterns and using the Flux pattern, presented in the case study of the Retratos da Atenção Primária em Saúde system. Its mission is to provide easy access to the "Programa Nacional de Melhoria do Acesso e da Qualidade da Atenção Básica (PMAQ)" data.

Keywords: Design pattern. Flux. Vue. Vuex. Web componentes. PMAQ-AB.

INTRODUÇÃO

As constantes demandas por aplicações *web* com interfaces ricas para os usuários aumentaram a complexidade de desenvolvimento de *softwares*, tendo em vista o crescimento do código, causando a necessidade de mais desenvolvedores *front-end* para lidar com a aplicação. Para mitigar a dificuldade de desenvolver essas interfaces, surgiram padrões e convenções a fim de aumentar a produtividade e reutilização de código (CARVALHO, 2016).

Tais convenções surgiram no âmbito do desenvolvimento de sistemas orientados a objetos, recuperando abordagens que funcionaram no passado (GAMMA *et al.*, 1993). A reutilização de boas soluções tinha como consequência a descoberta de certos padrões de comunicação de sistema que resolviam problemas específicos e tornavam os projetos mais flexíveis (GAMMA *et al.*, 2000). Essa identificação de padrões não se limita a projetos orientados a objetos, sendo possível encontrar padrões, por exemplo, no desenvolvimento de interfaces, que geralmente possui um paradigma orientado a eventos. Tais padrões foram adotados por projetistas e, posteriormente, incorporados a *frameworks,* com o objetivo de tornar o código reutilizável.

Os *frameworks* modernos de desenvolvimento *front-end* utilizam os *web* componentes, para reaproveitar o código que encapsula uma estrutura HTML, associando estilo e *javascript* de forma

generalizada o suficiente que permita reutilização em várias partes do código (MOZILLA, 2020). Esses *web* componentes são organizados em árvore e conforme a interface fica mais rica ou detalhada, a árvore de componentes aumenta. Com uma árvore grande, o gerenciamento do estado – as informações representadas na interface – pode se tornar muito confuso devido aos dados do estado ficarem espalhados através de vários componentes. Dessa forma, conforme a aplicação cresce, torna-se cada vez mais difícil encontrar erros, prover manutenção ao código, e até mesmo desenvolver novas funcionalidades.

Diante deste contexto e considerando os padrões e boas práticas de desenvolvimentos atuais, o presente trabalho trata-se de um estudo de caso sobre o sistema Retratos da Atenção Primária à Saúde, desenvolvido pelo Núcleo Avançado de Inovação Tecnológica (NAVI/IFRN) a fim de analisar a utilização do padrão Flux utilizando o *framework* Vue.

O portal desenvolvido provê informações sobre o acesso e a qualidade das equipes participantes do 3º ciclo do Programa de Melhoria do Acesso e da Qualidade da Atenção Básica (PMAQ-AB).

Devido ao aumento da complexidade do sistema Retratos, percebeu-se que a falta de um padrão de projeto poderia comprometer a organização do código, implicando no atraso do desenvolvimento do sistema, dificuldade de manutenção e falta de escalabilidade. Em aplicações como o PMAQ-AB, manter o correto estado de cada ponto do sistema torna-se um problema para o desenvolvedor (SCHMITZ, 2016), e a solução aplicada para a dificuldade do gerenciamento de estados e eventos no sistema baseou-se, principalmente, no uso do padrão de arquitetura Flux. Mais especificamente, dentro dessa perspectiva, foi utilizada a biblioteca de gerenciamento de estados Vuex.

MATERIAIS E MÉTODOS

O presente trabalho consiste em uma pesquisa exploratória, de natureza qualitativa, sobre padrões de projetos, em específico o Flux, aplicado ao estudo de caso do sistema Retratos da Atenção Primária à Saúde. Esse estudo foi feito após os desenvolvedores notarem a necessidade de melhor organização de código. Após revisão bibliográfica, foi constatado que a organização padrão de código utilizado por interfaces ricas era o Flux, que foi implementado então no sistema desenvolvido, refatorando completamente o código. A tecnologia escolhida para desenvolver o *software* foi o *framework* Vue, devido à expertise dos desenvolvedores da ferramenta e a adaptação ao padrão Flux foi feita utilizando a biblioteca Vuex.

PADRÕES DE PROJETO

Padrões de projeto são soluções típicas para problemas recorrentes em projetos de *software* (JONES, 1995). Esses padrões descrevem características ou comportamentos de alto nível que devem ser seguidos no seu projeto para que o problema seja resolvido (GAMMA *et al.*, 1993). Um padrão de projeto não é um código específico, que se pode copiar e colar em determinado programa ou importar através de uma biblioteca. Trata-se de um conceito geral para enfrentar um problema particular, como definido por Jones. Desse modo, os padrões de projeto não estão limitados a uma linguagem ou framework, o que possibilita aplicar as indicações de um padrão a diferentes soluções, adequado à realidade de programas específicos.

Importante destacar que essa ideia não deve ser confundida com a de algoritmos. Em ambos se descrevem soluções típicas para problemas conhecidos. Contudo, enquanto um algoritmo sempre define claramente um passo a passo a ser seguido,

como uma receita, o padrão de projeto é uma descrição de mais alto nível de uma solução, como um diagrama de componentes. Por essa razão, o mesmo padrão aplicado a programas diferentes pode resultar em códigos diferentes.

As vantagens de conhecer e utilizar padrões de projetos são inúmeras. A princípio, nota-se que a efetividade das soluções propostas por cada padrão é comprovada pelo teste do tempo: quando uma solução específica começa a se repetir em vários projetos ao longo dos anos, eventualmente alguém decide nomeá-la e documentar o seu funcionamento em detalhe (SHALLOWAY; TROTT, 2004 apud GONÇALVES *et al.*, 2005). Essa é a essência por trás do surgimento de cada padrão. Esses padrões obedecem a alguns princípios de engenharia de *software* que facilitam o desenvolvimento de uma base de código com módulos coesivos e acoplamento mínimo. Dessa maneira, o sistema como um todo torna-se mais fácil de entender e prover manutenção. Graças a isso, a equipe de desenvolvimento pode evitar muitas armadilhas recorrentes e manter um ritmo de produtividade acelerado.

Os padrões de projetos surgiram para solucionar problemas recorrentes no desenvolvimento de *software*, na maioria das vezes, atrelados ao paradigma de programação orientado a objeto (GAMMA *et al.*, 1993). As soluções são amplamente utilizadas na programação *back end,* embora atualmente a demanda por sistemas com interfaces ricas e performáticas tem propiciado o surgimento de padrões de projetos focados na programação *front end*, sendo um deles o Flux.

FLUX

O problema com uma estrutura Model-View-Controller (MVC) é a comunicação bidirecional, que provou ser muito difícil de depurar e entender quando uma mudança em uma entidade causa efeito em cascata em toda a base do código (CECHINEL, 2017). As dificuldades para mapear o estado dos componentes da aplicação quando a complexidade, o tamanho do código e o fluxo de dados aumentam muito, o que pode gerar implicações na manutenção de código, inconsistência de dados e problemas na escalabilidade da aplicação.

Baseados na problemática de lidar com o fluxo de dados da aplicação, os engenheiros do Facebook buscaram desenvolver um sistema restrito onde esse tipo de erro era previsível. Criou-se um código imperativo, em que cada ação do usuário tinha de passar por uma série de condições que determinavam uma mutação do estado. Cada ação do usuário teria de ser previsível e ser passada por um sistema expedidor, que informa essas ações a um controlador geral, responsável por realizar mutações no estado da aplicação e informar à página essa mudança de estado (GOMES, 2016).

A solução encontrada permitia a centralização dos estados da aplicação em único lugar, gerando um fluxo de dados unidirecional, dando origem ao padrão arquitetural Flux (FACEBOOK, 2020). Dessa forma, esse conceito surgiu após os engenheiros do Facebook notarem um ciclo constante de *bugs* advindos das mudanças de estado da aplicação no navegador. Esse padrão, que mapeia as ações dos usuários de forma previsível e permite o fluxo dos dados em apenas uma direção (GOMES, 2016), é uma das arquiteturas mais utilizadas atualmente, visto que resolve os desafios oriundos do aumento de complexidade das aplicações.

Apesar do padrão flux ter sido desenvolvido para uma tecnologia específica, logo outros *frameworks* e linguagens passaram a adotá-lo, visto que os padrões não se limitam à tecnologia. Hoje, o Flux é uma referência no desenvolvimento de interfaces, sendo utilizada em geral na linguagem *javascript* e em vários *frameworks* como o React e o Vue.

VUEX

A implementação da arquitetura Flux no Vue é feita através de uma biblioteca *javascript* chamada Vuex. Essa biblioteca é um padrão de gerenciamento de estado, que centraliza todos os componentes da aplicação, com regras que garantem que o estado só é modificado de forma previsível. O Vuex apresenta ganho de desempenho, manutenibilidade de código e a possibilidade de escalonamento do sistema (VUEX, 2020).

O Vuex serve como uma *store* centralizada para todos os componentes da aplicação, com regras que garantem que o estado só pode ser alterado de forma previsível. No entanto, devido ao uso de uma única árvore de estado, todo o estado de nossa aplicação está contido dentro de um grande objeto. Para também sanar esse problema e garantir uma boa organização do projeto, o Vuex permite ainda modularizar a *store*, para categorizar quais estados pertencem a uma determinada entidade da aplicação, e cada módulo pode conter seu próprio estado, mutações, ações, *getters* e até módulos aninhados (VUEX, 2020).

RETRATOS DA ATENÇÃO PRIMÁRIA À SAÚDE

O Programa Nacional de Melhoria do Acesso e da Qualidade da Atenção Básica (PMAQ) foi utilizado na Atenção Básica (AB), para avaliar e acompanhar o trabalho das equipes de saúde e teve por objetivo incentivar os gestores e equipes de saúde a melhorarem a qualidade de serviços, de modo a propiciar a elevação do repasse de recursos públicos. A síntese de dados do PMAQ é decorrente das avaliações externas realizadas por instituições de ensino e pesquisa, sendo utilizado por gestores do Ministério da Saúde para tomar decisões estratégicas. (BRASIL, 2017).

Como os dados do PMAQ eram utilizados pelos gestores para estudos da sífilis, levou-se em consideração a publicação desses através de um sistema chamado "Retratos da Atenção Primária em Saúde", financiado pela TED 114/2018. O sistema desenvolvido apresenta relatórios dos dados do 3º ciclo do PMAQ, referentes a 42 mil equipes de saúde presentes em 5.324 municípios (NAVI, 2020).

DISCUSSÕES

No desenvolvimento *web* moderno, encontramos uma forte tendência em componentizar as interfaces. A utilização desse conceito decorre do fato de que muitos elementos visuais em uma interface são repetidos em diferentes contextos, como botões e outros elementos que interagem diretamente com o usuário. Por isso, isolar os detalhes de um determinado elemento gráfico em um componente permite que um mesmo código possa ser reutilizado para diferentes finalidades. Essa característica é um sinal de um bom projeto de *software*, visto que a reutilização de código é uma das formas de reduzir custos de desenvolvimento (CECHINEL, 2017).

No desenvolvimento de uma aplicação *web*, é natural que componentes simples (como botões) sejam utilizados como base para construir novos componentes com leiautes mais detalhados (como formulários). Em decorrência disso, forma-se uma árvore de componentes na qual cada componente pode ser composto por outros componentes, como pode ser visto na Figura 1.

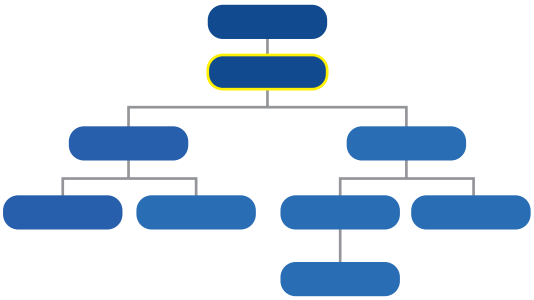

Figura 1 – Exemplo de árvore.
Fonte: autoria própria.

Nesse contexto, dizemos que o estado da aplicação é composto por todas as informações que são mostradas pelos componentes que compõem essa árvore. Em uma aplicação comum, essas informações são guardadas em variáveis dentro de cada componente, e a interação do usuário pode fazer com que um componente altere o estado de outros elementos da tela.

É possível observar que, conforme a interface fica mais rica ou detalhada, a árvore de componentes aumenta. Com uma árvore grande, o gerenciamento do estado pode se tornar muito confuso: os dados do estado se encontram espalhados através de vários componentes, e a interação entre estes também se torna intrincada. Além disso, o compartilhamento de estado entre componentes também pode apresentar desafios. Na Figura 2, pode-se observar uma situação em que um componente é detentor de uma informação que também precisa ser mostrada em dois ramos diferentes da árvore de componentes.

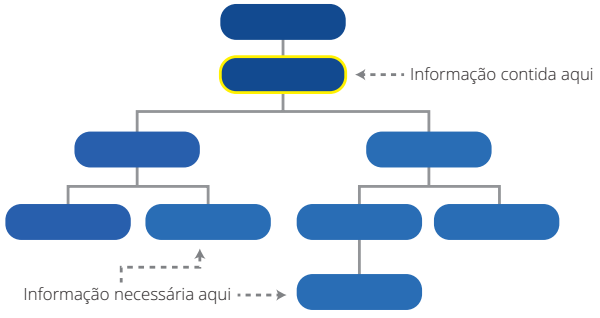

Figura 2 – Árvore de componentes com componente detentor de informação a ser mostrada em outros componentes.
Fonte: autoria própria.

Nessa situação, é necessário "repassar" o estado através da árvore entre os componentes que precisam dele, conforme ilustra a Figura 3.

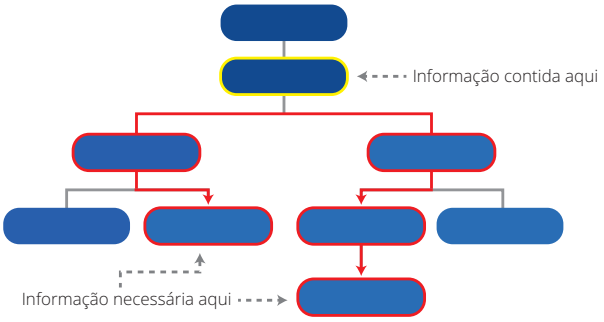

Figura 3 – Árvore de componentes repassando informações entre os componentes.
Fonte: autoria própria.

O problema com essa estratégia é que o estado precisa ser repassado até mesmo para componentes que não utilizam aquela informação. Inclusive, uma consequência direta disso é que a árvore de componentes se torna engessada, dificultando o posicionamento livre de componentes ao longo da árvore. Ou seja, a reutilização dos componentes fica comprometida, e isso vai contra todo o propósito de se utilizar componentes.

Pelos problemas demonstrados anteriormente, um time de desenvolvimento que não adota um padrão de projeto adequado pode ter sérios impactos no tempo de seus ciclos de desenvolvimento. Conforme a aplicação cresce, torna-se cada vez mais difícil encontrar erros, prover manutenção ao código, e até mesmo desenvolver novas funcionalidades.

RESULTADOS

Com o crescimento da complexidade do Retratos, percebeu-se que não seria possível lidar com os dados da aplicação apenas criando variáveis e eventos globais, pois a organização e manutenibilidade do código ficariam comprometidas. Em aplicações como essa, manter o correto estado de cada

ponto do sistema torna-se um problema para o desenvolvedor.

A solução aplicada para o problema do gerenciamento de estados e eventos no sistema Retratos baseou-se, principalmente, no uso do padrão de arquitetura Flux. Como o sistema foi desenvolvido através do *framework* Vue, foi utilizada a biblioteca de gerenciamento de estados Vuex para adaptar o código ao novo padrão.

A refatoração do sistema foi concluída em cerca de 4 semanas. Como o *javascript* é uma linguagem orientada a eventos, várias adaptações dos componentes desenvolvidos anteriormente tiveram que acontecer. A arquitetura Flux mantém um só local dos dados, porém antes de adotar esse padrão de projeto vários componentes comunicavam-se através de eventos globais, então a ideia inicial da refatoração foi eliminar esses eventos.

Antes de adotar o padrão, diversos desenvolvedores criaram eventos globais que realizavam a comunicação entre os componentes. A manutenção do código requisitava que os programadores entendessem qual componente estava emitindo cada evento, o porquê de sua emissão e qual módulo o escutava, o que acaba atrasando a correção de erros e desenvolvimento de novas funcionalidades, visto que a cada ciclo de desenvolvimento era necessário entender cada parte da aplicação. Após a adaptação para o Flux e a remoção dos eventos globais, vários desacoplamentos de componentes foram sendo realizados e de início já se notava mais fluidez para o desenvolvimento de novas funcionalidades, posto que era possível para os programadores observarem quais dados estavam disponíveis para que pudesse trabalhar naquele determinado momento, sem a necessidade de investigar qual componente está compartilhando o dado.

CONSIDERAÇÕES FINAIS

Os padrões de projetos surgem para solucionar problemas recorrentes, geralmente eram atrelados ao paradigma de programação orientado a objeto e, por consequência, estavam relacionados intimamente com o desenvolvimento *back end,* porém recentemente com o aumento de demandas por interfaces ricas e performáticas a complexidade no desenvolvimento *front end* elevou-se gerando o ambiente propício para a criação de padrões e convenções que auxiliam o processo de desenvolvimento de software.

Atualmente os padrões de projetos voltados para o desenvolvimento *front end* tem alcançado popularidade entre empresas e desenvolvedores por facilitar o processo de implementação do *software* e solucionar problemas recorrentes. A arquitetura Flux surgiu com o propósito de gerenciar o estado dos componentes e melhorando a comunicação. Os seus benefícios são a facilidade na manutenção do código, ganho de desempenho e a possibilidade de escalonamento do sistema. Os seus princípios não estão atrelados a uma tecnologia específica permitindo que *frameworks* e bibliotecas a utilizem.

O sistema pôde ser completamente refatorado, introduzindo a arquitetura Flux através da biblioteca Vuex, o que resolveu o problema de gerenciamento de estados. Observou-se que o padrão de projeto adotado traz diversos benefícios como a fácil identificação de que componentes estão modificando os dados em determinado estado da aplicação, o que facilita a depuração, diminuindo o tempo de manutenção e desenvolvimento de novas funcionalidades.

REFERÊNCIAS

BRASIL. Tribunal de Contas da União. **Acórdão n° 2019/2017**. Plenário. Relator: Ministro Bruno Dantas. Sessão de 13/09/2017. Disponível em: https://pesquisa.apps.tcu.gov.br/#/redireciona/acordao-completo/%22ACORDAO-COMPLETO-2261941%22. Acesso em: 19 jan. 2020.

CECHINEL, Alexandre. **Avaliação do framework Angular e das bibliotecas React e Knockout para o desenvolvimento do Frontend de aplicações Web**. 2017. 77 f. Trabalho de Conclusão de Curso (Graduação em Sistemas da Informação) – Universidade Federal de Santa Catarina, Florianópolis, 2017. Disponível em: https://repositorio.ufsc.br/bitstream/handle/123456789/182199/TCC%20PROJETOS%202%20-%20ALEXANDRE%20CECHINEL.pdf?sequence=1&isAllowed=y. Acesso em: 31 jan. 2020.

GAMMA, Erich *et al*. **Design patterns**: Abstraction and reuse of object-oriented design. *In*: EUROPEAN CONFERENCE ON OBJECT-ORIENTED PROGRAMMING. Berlin: Heidelberg, 1993. p. 406-431. Disponível em: http://cseweb.ucsd.edu/~wgg/CSE210/ecoop93-patterns.pdf. Acesso em: 29 jan. 2020.

GAMMA, Erich *et al*. **Padrões de Projeto**: soluções reutilizáveis de software orientado a objetos. Porto Alegre: Bookman, 2000.

MOZILLA. **Web Components**. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/Web_Components. Acesso em: 26 jan. 2020.

SCHMITZ, Daniel; GEORGII, Daniel Pedrinha. **Vue.js na prática**. São Paulo: Casa do código, 2016.

GOMES, Ramon Diogo Gondim Miaja. **Desenvolvimento de uma rede social utilizando Erlang e a arquitetura Flux**. 2016. 62 f. Trabalho de Conclusão de Curso (Graduação em Engenharia de Controle e Automação) – Universidade Federal de Santa Catarina, Florianópolis, 2016. Disponível em: https://repositorio.ufsc.br/bitstream/handle/123456789/171619/PFC_2016-1%20Ramon%20Diogo%20Gondim%20Miaja%20Gomes.pdf?sequence=1&isAllowed=y. Acesso em: 19 jan. 2020.

FACEBOOK. **In-Depth Overview.** Disponível em: https://facebook.github.io/flux/docs/in-depth-overview. Acesso em: 24 jan. 2020.

GONÇALVES, Rodrigo Franco *et al*. Uma proposta de processo de produção de aplicações Web. **Production**, v. 15, n. 3, p. 376-389, dez. 2005. Disponível em: http://www.scielo.br/pdf/prod/v15n3/v15n3a07.pdf. Acesso em: 26 jan. 2020.

CARVALHO, T. (2016). Orientação a Objetos: Aprenda seus conceitos e suas aplicabilidades de forma efetiva. Casa do Código.

JONES, C. Patterns of large software systems: failure and success. **Computer**, v. 28, n. 3, p. 86-87, mar. 1995.

NÚCLEO AVANÇADO DE INOVAÇÃO TECNOLÓGICA – NAVI. **Retratos da Atenção Primária à Saúde**. Disponível em: https://www.navi.ifrn.edu.br/. Acesso em: 31 jan. 2020.

VUEX. **What is Vuex?**. Disponível em: https://vuex.vuejs.org/. Acesso em: 28 jan. 2020.