# Um estudo de caso sobre a atualização de uma aplicação Web baseada em Vue.js

Carlos Fabrício da Silva Pontes

CENTRO DE INFORMÁTICA
UNIVERSIDADE FEDERAL DA PARAÍBA

João Pessoa, 2023

Carlos Fabrício da Silva Pontes

# Um estudo de caso sobre a atualização de uma aplicação Web baseada em Vue.js

Relatório técnico apresentado ao curso Ciência da Computação do Centro de Informática, da Universidade Federal da Paraíba, como requisito para a obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Raoni Kulesza

Novembro de 2023

**Catalogação na publicação**
**Seção de Catalogação e Classificação**

P814e Pontes, Carlos Fabricio da Silva.
Um estudo de caso sobre a atualização de uma aplicação Web baseada em Vue.js / Carlos Fabricio da Silva Pontes. - João Pessoa, 2023.
67 f. : il.

Orientação: Raoni Kulesza.
TCC (Graduação) - UFPB/CI.

1. Aplicações Web. 2. Frameworks. 3. Processo de atualização. 4. Vue.js. I. Kulesza, Raoni. II. Título.

UFPB/CI CDU 004.738.5

Elaborado por Michelle de Kássia Fonseca Barbosa - CRB-738

# RESUMO

Desde o surgimento das primeiras aplicações Web, os desenvolvedores têm constantemente buscado maneiras de aprimorar o desenvolvimento desses softwares, resultando em um avanço contínuo das ferramentas disponíveis para essa finalidade. No entanto, as aplicações mais antigas enfrentam o desafio de se manterem atualizadas para aproveitar as novas oportunidades desta evolução. Para organizações, em particular, atualizar sistemas pode ser uma tarefa complexa. Nesse cenário, este trabalho tem como objetivo principal realizar um estudo de caso sobre a atualização para uma aplicação construída com o framework Vue.js. Tal aplicação consiste numa plataforma Web de uma empresa na área de digitalização de processos de gerenciamento de recursos humanos: Workverse. Como atividades iniciais foi realizado um levantamento do estado da técnica na área de desenvolvimento Web e uma análise arquitetural para entender a estrutura da aplicação atual. Posteriormente, foram definidas as etapas e realizada a evolução, garantindo que a aplicação permanecesse atualizada e pudesse se beneficiar de novas funcionalidades. Como principais resultados, foi possível realizar uma compreensão mais profunda das aplicações Web em Vue.js e um estudo de caso que pode contribuir para a definição de um processo que atenda atualizações de outras aplicações Web e/ou outras tecnologias.

**Palavras-chave:** Aplicações Web, Frameworks, Processo de atualização, Vue.js.

# ABSTRACT

Since the emergence of the first web applications, developers have consistently sought ways to enhance the development of these software, resulting in a continuous advancement of tools available for this purpose. However, older applications face the challenge of staying up-to-date to take advantage of the new opportunities brought by this evolution. For organizations in particular, updating systems can be a complex task. In this scenario, the main objective of this work is to conduct a case study on the update process for an application built with the Vue.js framework. This application constitutes a web platform for a company specializing in the digitization of human resources management processes: Workverse. Initial activities included a survey of the state of the art in web development and an architectural analysis to understand the structure of the current application. Subsequently, steps were defined and the evolution was carried out, ensuring that the application remained up-to-date and could benefit from new features. The main results include a deeper understanding of web applications in Vue.js and a case study that may contribute to defining a process that addresses updates for other web applications and/or technologies.



**Key-words:** Frameworks, Updating Process, Web Applications, Vue.js.

# LISTA DE FIGURAS


1 Relação entre Framework, biblioteca e aplicação . . . . . 18
2 Pequeno componente utilizando Options API . . . . . 22
3 Pequeno componente utilizando Composition API . . . . . 22
4 Exemplo de estrutura de código utilizando Options e Composition API . . 23
5 Componente Vue.js construído sem Quasar . . . . . 25
6 Componente Vue.js construído sem Quasar . . . . . 26
7 Visão geral do modelo C4 . . . . . 27
8 Diagrama de navegação da plataforma Workverse . . . . . 32
9 Diagrama de contexto do Sistema Workverse . . . . . 33
10 Diagrama de container do Sistema Workverse com foco na plataforma . . . 34
11 Diagrama de componentes da container Plataforma Workverse . . . . . 37
12 Tabela utilizada para mapear os casos de uso para teste . . . . . 38
13 Fluxo de iteração entre as páginas para atualização . . . . . 39
14 Modal de upload de Mailing antes da atualização do Quasar . . . . . 40
15 Modal de upload de Mailing após da atualização do Quasar . . . . . 40
16 Captura de tela mostrando os erros emitidos pela Migration Build . . . . . 41
17 Captura de tela mostrando a aplicação executando com a Migration Build 42
18 Plataforma sendo executada com Vue.js 3 sem a Migration Build . . . . . 43
19 Página de gerência de usuários com erros de exibição . . . . . 44
20 Página de acompanhamento com erros de exibição e estilização . . . . . 44
21 Mapa de navegação com as páginas antes da atualização do Vue . . . . . 46
22 Mapa de navegação com as páginas após atualização do Vue . . . . . 46
23 Tabela preenchida com as avaliações dos testes das funcionalidades . . . . 47
24 Tabela com a duração de cada etapa do processo . . . . . 47
25 Código que deve ser adicionado ao arquivo quasar.conf para fazer o carregamento em modo de compatibilidade . . . . . 59
26 Correções do vuelidade no QInput . . . . . 61
27 Modificação a ser feita no componente filho . . . . . 63

28 Modificação a ser feita no Mixin . . . . . 63
29 Alterações a serem feitas no componente QTable . . . . . 63
30 Trecho de código com erro na utilização de "v-for"e "v-if" . . . . . 64
31 Correção na utilização do "v-for", substituindo o "v-if"por um objeto filtrado 65
32 Configuração das cores da aplicação . . . . . 66
33 Ajustes realizados para corrigir os ícones . . . . . 67

# LISTA DE ABREVIATURAS

AJAX - Asynchronous JavaScript and XML

API - Application Programming Interface

CDN - Content Delivery Network

CSS - Cascading Style Sheets

EOL - End of Life

HTML – Hypertext Markup Language

IDE - Integrated Development Environment

LTS - Long Term Support

MVC - Model-View-Controller

MVP - Model-View-Presenter

MVVM - Model-View-ViewModel

SPA - Single Page Application

SSO - Single Sign-On

UI - User Interface

XML - Extensible Markup Language

# Sumário

**1 INTRODUÇÃO** **12**

1.1 Tema . . . 12

1.2 Problema . . . 13

1.2.1 Objetivo geral . . . 14

1.2.2 Objetivos específicos . . . 14

1.3 Estrutura do relatório técnico . . . 15

**2 CONCEITOS GERAIS** **16**

2.1 Aplicações Web . . . 16

2.2 Single Page Application . . . 17

2.3 *Frameworks* e bibliotecas *front-end* . . . 17

2.4 Padrões MV* . . . 18

2.4.1 MVC . . . 18

2.4.2 MVP . . . 19

2.4.3 MVVM . . . 19

2.5 Vue.js . . . 20

2.5.1 Histórico de versões do Vue.js . . . 20

2.5.2 Migration Build . . . 21

2.5.3 Options API e Composition API . . . 21

2.6 Quasar . . . 24

2.7 Modelo C4 . . . 26

2.7.1 Diagrama de contexto . . . 27

2.7.2 Diagrama de container . . . 27

2.7.3 Diagrama de componente . . . 28

2.7.4 Diagrama de código . . . 28

2.8 Conclusão do capítulo . . . 28

**3 METODOLOGIA** **29**

3.1 Estudo sobre o desenvolvimento de aplicações Web e o uso de frameworks . 29

3.2 Análise da plataforma e planejamento do processo de atualização . . . . . 29
3.3 Implementação do processo . . . . . 29
3.4 Avaliação da Atualização . . . . . 30

**4 APRESENTAÇÃO E ANÁLISE DOS RESULTADOS 31**

4.1 Visão geral da Plataforma de Gerenciamento Workverse . . . . . 31
4.1.1 Esquema navegacional . . . . . 31
4.2 Arquitetura do sistema da Plataforma de Gerência Workverse . . . . . 32
4.2.1 Visão de mais alto nível - C1 . . . . . 33
4.2.2 Visão de alto nível C2 . . . . . 33
4.2.3 Visão de baixo nível - C3 (Plataforma de gerenciamento) . . . . . 34
4.3 Proposta de atualização para Vue.js 3.0 . . . . . 37
4.3.1 Atualização do Quasar v0.7 para v1 . . . . . 39
4.3.2 Atualização para Vue.js 3 utilizando a Migration Build . . . . . 40
4.3.3 Removendo a Migration Build e realizando os testes finais . . . . . 42
4.3.4 Migrar de Options API para Composition API . . . . . 45
4.4 Avaliação . . . . . 45
4.4.1 Refatorações mapeadas . . . . . 48
4.4.2 Principais dificuldades enfretadas . . . . . 49
4.4.3 Boas práticas para outros processos de atualização . . . . . 49

**5 CONCLUSÕES E TRABALHOS FUTUROS 51**

**REFERÊNCIAS 52**

# 1 INTRODUÇÃO

Desde o advento do primeiro site do mundo em 1991, a World Wide Web tem continuamente evoluído, tornando-se uma das tecnologias mais ubíquas em todo o mundo. Ela é executada em uma ampla variedade de dispositivos e plataformas, permitindo que as aplicações Web alcancem níveis de robustez e funcionalidade comparáveis às aplicações desktop (SCOTT, 2016).

Com o passar dos anos, os desenvolvedores têm buscado aprimorar suas aplicações de forma constante, impulsionando o surgimento de novas arquiteturas e frameworks. Essas inovações possibilitam o desenvolvimento de sistemas cada vez mais complexos, incorporando funcionalidades que, em um passado não tão distante, seriam consideradas impensáveis (SCOOT, 2016). Além disso, a utilização de aplicações Web se expandiu para uma ampla variedade de empresas, desde *startups* até gigantes multinacionais. Cada uma delas adapta sistemas sob medida para atender às suas necessidades específicas, seja para a gestão de processos ágeis ou para o gerenciamento de vastos bancos de dados.

Esta evolução constante no campo das aplicações Web reflete não apenas o poder da tecnologia, mas também a capacidade de adaptação das empresas e desenvolvedores para enfrentar os desafios e oportunidades da era digital. Neste contexto em constante mudança, surge a necessidade de explorar as atualizações tecnológicas a fim de continuar fornecendo soluções inovadoras e competitivas.

## 1.1 Tema

As tecnologias utilizadas no desenvolvimento de aplicações Web passam por atualizações contínuas, com versões anteriores se tornando obsoletas e novas versões trazendo aprimoramentos significativos em funcionalidade, suporte, manutenção de código e outros benefícios tanto para desenvolvedores quanto para os usuários finais. Em um ambiente altamente competitivo, as empresas estão constantemente buscando oferecer as soluções mais avançadas e eficientes.

No entanto, a implementação dessas atualizações não é uma tarefa trivial, especialmente quando se trata de aplicações de médio e grande porte. Segundo a Forbes(2018), atualizar sistemas de software pode ter alguns desafios, como:

- Aceitação do usuário: Usuários de sistemas legados geralmente resistem a mudanças, especialmente aqueles que realizam diversas atividades e já dominam atalhos e processos de trabalho na aplicação.
- Problemas no fluxo de trabalho: A atualização de sistemas legados impacta os fluxos de trabalho estabelecidos. Interrupções nos processos podem resultar em apreensão,

resistência e dificuldades durante a transição;

- Dependências desconhecidas: A falta de conhecimento sobre as dependências, sejam internas ou externas, representa um desafio na atualização. Isso pode resultar em erros durante o processo, dificultando a realinhamento adequado;

- Reprogramação quando necessária: Durante a atualização, pode ser necessário realizar a migração de dados e até mesmo reprogramar do zero, o que implica em custos de tempo e dinheiro;

É essencial que as empresas adotem estratégias de atualização bem planejadas, destacando pontos cruciais e conduzindo avaliações abrangentes. O objetivo é garantir que nenhum aspecto do sistema seja perdido ou prejudicado durante o processo de migração (Forbes, 2018).

## 1.2 Problema

A utilização de sistemas que utilizam versões obsoletas de tecnologias pode acarretar uma série de malefícios que afetam tanto a eficiência operacional das empresas quanto a experiência dos usuários finais. Estes problemas incluem (MCCARTY, 2022):

- Riscos de segurança: Versões desatualizadas de *frameworks* frequentemente deixam as aplicações vulneráveis à ameaças de segurança, uma vez que não recebem mais correções neste quesito. Isso coloca em risco a integridade dos dados, a privacidade dos usuários e a reputação da empresa.

- Perda de competitividade: Em um mercado em constante evolução, a obsolescência tecnológica pode resultar na perda de competitividade. Empresas que não conseguem manter os sistemas atualizados correm o risco de deixar passarem *insights* valiosos, e informações para dar suporte à tomada de decisão fornecidos pelos softwares mais recentes.

- Custos elevados de manutenção: A manutenção de sistemas legados pode se tornar dispendiosa. À medida que os *frameworks* antigos se tornam cada vez mais raros, encontrar profissionais com experiência para manter esses sistemas pode ser um desafio, aumentando os custos de equipe e manutenção.

- Incompatibilidade com novas tecnologias: *Frameworks* obsoletos geralmente não oferecem suporte para as mais recentes tecnologias e recursos. Isso limita a capacidade das empresas de adotar novas inovações tecnológicas que poderiam melhorar a eficiência e a qualidade de seus produtos e serviços.

- Desempenho insatisfatório: Sistemas que utilização versões antigas de *frameworks* podem ter desempenho inferior, resultando em lentidão e experiências de usuário insatisfatórias. Isso pode afetar negativamente a satisfação dos clientes e prejudicar a reputação da empresa.
- Dificuldades na manutenção de código: À medida que um código-fonte envelhece, a manutenção se torna mais complexa e sujeita a erros. Isso dificulta a introdução de novos recursos e a correção de bugs, retardando o desenvolvimento e aprimoramento dos sistemas.

Portanto, é fundamental que as empresas estejam atentas aos riscos associados ao uso de *frameworks* obsoletos e desenvolvam estratégias sólidas para a atualização de seus sistemas. A migração para versões mais recentes e a adoção de melhores práticas de desenvolvimento são passos essenciais para evitar esses malefícios e garantir a competitividade no cenário tecnológico em constante evolução.

### 1.2.1 Objetivo geral

O objetivo primordial deste trabalho foi realizar um estudo de caso sobre a atualização de uma aplicação baseada em Vue.js, a plataforma de gerenciamento da empresa Workverse, e conduzir este processo, realizando a atualização deste *framework* da versão 2.0 para a versão 3.0. A atualização planejada teve como propósito modernizar a plataforma, assegurando que ela esteja em conformidade com as últimas versões das tecnologias utilizadas. Além disso, visou aprimorar as funcionalidades a fim de atender de forma mais eficaz as necessidades da empresa e de seus usuários finais, e proporcionar uma melhor manutenção de código, reduzindo os custos associados à gestão de sistemas legados.

### 1.2.2 Objetivos específicos

Em relação aos objetivos específicos, este trabalho realizou:

1. Um estudo do estado da técnica:
   (a) Investigação da história, conceitos, fundamentos e padrões de projeto relacionados às aplicações web.
   (b) Avaliação das principais soluções de mercado, identificando tendências e melhores práticas para aprimorar a plataforma.
2. Análise da plataforma Workverse:
   (a) Análise abrangente da plataforma de gerenciamento da empresa Workverse.

(b) Planejamento da atualização do framework principal (migração de Vue.js 2.0 para Vue.js 3.0) com base nessa avaliação realizada anteriormente, considerando os requisitos específicos da empresa.

3. Execução do plano de atualização:

   (a) Implementação do plano de atualização, seguindo etapas detalhadas e documentando cuidadosamente cada fase do processo.

   (b) Realização de testes para garantir a funcionalidade adequada da plataforma atualizada e dos componentes relacionados.

4. Proposta de melhorias na arquitetura do sistema:

   (a) Sugestão de melhorias na arquitetura do sistema e possíveis refatorações de código, com base nas observações e resultados obtidos durante o processo de atualização.

## 1.3 Estrutura do relatório técnico

O Capítulo 2 aborda a fundamentação teórica relacionada à aplicações web, incluindo conceitos-chave, *frameworks*, bibliotecas, padrões e outros elementos fundamentais na arquitetura do sistema em foco. Logo após, tem-se o Capítulo 3, onde está descrito em detalhes o processo de estudo, as etapas e metodologias utilizadas para cada fase até atingir os objetivos. Em seguida, no Capítulo 4, é descrito como ocorreu cada fase do estudo, incluindo as análises, a implementação do processo de atualização e a avaliação dos resultados. Por fim, no Capítulo 5, é apresentada a conclusão do estudo, resumindo os resultados alcançados.

# 2 CONCEITOS GERAIS

Nos primeiros anos da World Wide Web (por volta de 1990 à 1995), os "Websites" eram compostos de um conjunto de arquivos de hipertexto conectados que apresentavam informações usando textos e gráficos de forma limitada. Porém, com o passar do tempo, a linguagem de marcação de texto HTML (do inglês, *Hypertext Markup Language*) começou a se desenvolver e foram criadas ferramentas como XML (do inglês, *Extensible Markup Language*) que favoreceu a evolução destas aplicações e permitiu que os engenheiros de software tivessem a capacidade de realizar manipulações dos dados e conteúdos tanto do lado do servidor quanto do cliente, e com isso, surgiu o conceito de aplicações Web (PRESSMAN, 2008).

Estas aplicações também podem ser definidas como um software pelo fato de que são coleções executáveis de instruções e dados que fornecem informações e funcionalidades para o usuário final, e com essa definição, pode-se concluir que é possível desenvolver aplicações Web utilizando vários dos conceitos de sistemas convencionais baseados em computadores (PRESSMAN, 2008).

## 2.1 Aplicações Web

Desde o surgimento das primeiras aplicações Web, os desenvolvedores têm buscado uma forma de desenvolver aplicativos Web com a aparência e a sensação de aplicativos nativos de desktop. Para isso, diversas soluções para uma experiência mais semelhante à nativa, como IFrames, Java Applets , Adobe Flash e Microsoft Silverlight, foram testadas com graus variados de sucesso. Embora sejam tecnologias diferentes, todas têm pelo menos um objetivo em comum: trazer o poder de um aplicativo de desktop para o ambiente leve e multiplataforma de um navegador da Web. Neste contexto, surge a ideia de desenvolver aplicações de página única SPA (do inglês, *Single Page Application*) com objetivo de alcançar uma experiência semelhante à nativa, usando apenas JavaScript, HTML e CSS (do inglês, *Cascading Style Sheets*) (SCOTT, 2016).

Segundo Scott (2016), as origens deste conceito se deu início quando surgiu um controle chamado ActiveX no navegador Microsoft Internet Explorer, usado para enviar e receber dados de forma assíncrona. Porém, a grande mudança se iniciou quando a funcionalidade desse controle foi oficialmente adotada pelos principais fornecedores de navegadores como a API XMLHttpRequest (XHR). Os desenvolvedores então começaram a combinar essa API com JavaScript, HTML e CSS, que resultou numa técnica conhecida como AJAX, ou JavaScript e XML Assíncrono. As requisições de dados não intrusivas do AJAX, combinadas com o poder do JavaScript para atualizar dinamicamente o Modelo de Objetos do Documento - DOM (do inglês, *Document Object Model*) e o uso do CSS para

alterar o estilo da página em tempo real, levaram o AJAX ao centro do desenvolvimento Web moderno.

Com o sucesso deste movimento, o conceito de SPA levou o desenvolvimento Web a um nível completamente novo, expandindo as técnicas de manipulação em nível de página do AJAX para toda a aplicação. Além disso, os padrões e práticas comumente utilizados na criação de uma SPA podem levar a eficiências globais no design de aplicativos, manutenção de código e tempo de desenvolvimento (SCOTT, 2016).

## 2.2 Single Page Application

No modelo arquitetural de SPA, a aplicação inteira roda como uma única página Web. Desta forma, a camada de aplicação é retirada do lado do servidor e controlada apenas do lado do cliente (*browser*). Esta forma de estruturar a aplicação provém alguns benefícios tanto para os desenvolvedores quanto para os usuários, como destaca Scott:

- Renderização como uma aplicação desktop porém rodando no *browser*: A SPA tem a habilidade de renderizar partes selecionadas da tela e o usuário consegue perceber as atualizações instantaneamente.
- Desacoplamento da camada de apresentação: O código responsável para UI (do inglês, *User Interface*) e como ela se comporta é definido do lado do cliente e não apenas do servidor.
- Transações rápidas para o servidor: As transações para o servidor são mais leves e rápidas, porque, depois do carregamento inicial, apenas dados são enviados e recebidos do servidor.

## 2.3 *Frameworks* e bibliotecas *front-end*

*Framework* pode ser definido como um conjunto de classes que incorpora um design abstrato para soluções de uma família de problemas relacionados e suporta reutilização em uma granularidade maior do que as classes. (Johnson, 1998).

Segundo Duarte(2015), *frameworks* impõem um conjunto de padrões de desenvolvimento, ou seja, ao utilizá-lo, o desenvolvedor fica obrigado a ter um código estruturado, fácil de entender e flexível para eventuais alterações que possam ocorrer no futuro.

Os principais benefícios da utilização de frameworks são (Fayad, 1999):

- Melhora da modularização encapsulando os detalhes voláteis de implementação por trás de interfaces estáveis;

- Reusabilidade, com a definição de componentes genéricos que podem ser reutilizados para criar novas aplicações;
- Melhora na extensibilidade, provendo hooks que permitem que as aplicações estendam interfaces estáveis;
- Inversão de controle, na qual o *framework* define quais métodos chamar em resposta a ações externas, controla o fluxo global de execução dos programas em vez de ser o desenvolvedor;

Em comparação aos *frameworks*, segundo Duarte (2015), uma biblioteca é um repositório de funcionalidades a que o desenvolvedor tem acesso para desenvolver a aplicação, porém, é o desenvolvedor que controla o fluxo da aplicação. Além disso, o autor também menciona que um *framework* pode conter várias bibliotecas, e chama o código da aplicação. Por sua vez, uma aplicação pode usar bibliotecas, mesmo que estas não pertençam a nenhum *framework*, para atingir um objetivo. A figura 1 ilustra a relação entre aplicação, bibliotecas e *frameworks*.

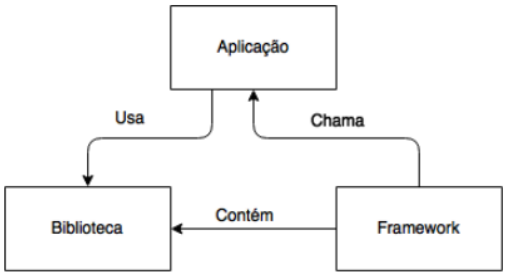


**Figura 1: Relação entre Framework, biblioteca e aplicação**

## 2.4 Padrões MV*

Os padrões arquiteturais MV* são uma família de padrões de design de software amplamente utilizados em desenvolvimento Web que dão suporte à separação de responsabilidades na base de código de um projeto. As duas primeiras letras da sigla MV* representam Model e View, respectivamente, e o asterisco, por sua vez, representa um terceiro componente que pode variar. No caso do modelo arquitetural MVC, por exemplo, o asterisco está para Controller. Além deste, também existem outros padrões amplamente utilizados, como o Model-View-Presenter (MVP) e Model-View-ViewModel (MVVM).

### 2.4.1 MVC

O padrão MVC surgiu na década de 70 como um dos padrões de design mais influentes para aplicações com interfaces gráficas complexas. Ele fornece uma maneira

de dividir a funcionalidade envolvida na manutenção e apresentação dos dados de uma aplicação. Sua sigla representa os 3 principais componentes que precisam ser definidos: Model, View e Controller.

O Model representa os dados da aplicação e as regras do negócio que governam a manipulação destes dados. Ele é responsável por manter o estado persistente da aplicação e fornece ao controlador a capacidade de acessar as funcionalidades encapsuladas pelo próprio modelo (MACORATTI, 2002a). A View é responsável por renderizar o conteúdo de uma parte particular do modelo e encaminhar para o controlador as ações do usuário, além de acessar os dados do modelo via controlador e definir como esses dados devem ser apresentados (MACORATTI, 2002a).

Por fim, o Controller define o comportamento da aplicação, interpretando as ações do usuário e mapeando para chamadas do Model. As ações realizadas incluem ativar processos de negócio ou alterar o estado do Model, e, com base nessas ações e alterações, o Controller seleciona uma View para ser exibida como parte da resposta à solicitação do usuário. Há normalmente um Controller para cada conjunto de funcionalidades relacionadas (MACORATTI, 2002a).

### 2.4.2 MVP

A ideia principal do MVP, do inglês Model-View-Presenter é que toda a lógica que normalmente iria ligar a interface do usuário com os dados seja movida para uma classe separada. No padrão MVP, a interface do usuário não possui processamento algum. Ele adiciona um componente chamado Presenter, que tem como responsabilidade atualizar a View quando o Model é alterado e de sincronizar o Model em relação à View.

Em comparação com o MVC, as principais diferenças é que no MVC, o Controller é o responsável por determinar qual View será exibida em resposta a qualquer ação incluindo quando a aplicação é carregada, e no MVP as ações são roteadas através da View para o Presenter. Além disso, no MVC, a View se comunica (no sentido de poder enviar mensagens) com o Model, e no MVP, a View pode conhecer o Model, mas não envia mensagens para ele. (MACORATTI, 2002a).

### 2.4.3 MVVM

A arquitetura MVVM - Model-View-ViewModel - apresenta a proposta de diminuir o código excessivo de apresentação da View, propondo um novo componente, o ViewModel, que se encarrega pela lógica de apresentação.

Esse novo componente age como um intermediário entre a View e o Model, e é responsável por lidar com a lógica da View. Ele interage com o Model invocando métodos

de suas classes, e, em seguida, fornece seus dados de uma forma que a View possa usar mais facilmente. O ViewModel também fornece implementações de comandos que um usuário da aplicação inicia na View, como quando um usuário clica em um botão na interface, essa ação pode acionar um comando no ViewModel. Esta camada também pode ser responsável por definir mudanças lógicas de estado que afetam algum aspecto da exibição na View, como uma indicação de que alguma operação está pendente.

## 2.5 Vue.js

Vue.js é um framework progressivo que utiliza Javascript para a construção de interfaces de usuário (YOU, 2023). Foi criado por Evan You em 2014 e atualmente se encontra na versão v3.2.45. Ao contrário de outros frameworks monolíticos, Vue.js foi projetado desde sua concepção para ser adotado incrementalmente. Embora não esteja estreitamente associado ao padrão MVVM ( do inglês, Model-View-Viewmodel), os princípios de design do Vue.js foram parcialmente inspirados por ele (You, 2014).

A biblioteca principal é focada exclusivamente na camada visual (*view layer*), sendo fácil adotar e integrar com outras bibliotecas ou projetos existentes. Por outro lado, Vue.js também é perfeitamente capaz de dar poder a sofisticadas *Single-Page Applications* quando usado em conjunto com ferramentas modernas e bibliotecas de apoio.

Dentre as principais características do Vue.js, pode-se destacar que (WOHLGETHAN, 2018):

- Possui uma estrutura baseada em componentes, o que facilita a modularização, reutilização e manutenção do código.
- Baixa curva de aprendizado.
- Facilidade de integração com outras bibliotecas e tecnologias.
- Flexibilidade na estrutura do código.
- Boas práticas e padrões conhecidos pela comunidade.

### 2.5.1 Histórico de versões do Vue.js

Atualmente, as duas versões de Vue.js mais utilizadas, são o Vue.js 2 e o Vue.js 3. Vue.js 2.7 é a versão atual e final do Vue.js 2.x. O Vue.js 2.7 recebeu 18 meses de suporte de longo prazo - LTS (do inglês, *Long Term Support*) a partir de sua data de lançamento em 1º de julho de 2022. Durante esse período, o Vue.js 2 receberá correções de segurança e de *bugs* necessárias, mas não receberá novos recursos. Segundo a documentação oficial

do Vue.js 2, esta versão chegará ao fim da vida útil (EOL) em 31 de dezembro de 2023. Após essa data, o Vue.js 2 continuará disponível em todos os canais de distribuição existentes (CDNs e gerenciadores de pacotes), mas não receberá mais atualizações, incluindo correções de segurança e compatibilidade.

Vue.js 3 é a versão principal atual e mais recente do Vue.js. De acordo com a documentação, ele fornece melhor desempenho, melhor suporte a TypeScript e contém novos recursos que não estão presentes no Vue.js 2, como Teleport, Suspense e vários elementos roots por template. O Vue.js 3 contém alterações importantes que o tornam incompatível com a versão 2, portanto, ao considerar a atualização de uma aplicação Web construída com Vue.js 2 para a versão 3, é crucial estar ciente de que essa migração exigirá esforços adicionais para ser realizada com eficácia, dada a natureza das alterações entre as versões.

### 2.5.2 Migration Build

Segundo o guia oficial de migração do Vue.js 3, a Migration Build é um pacote que fornece uma *build* do Vue.js 3 compatível com a estrutura do Vue.js 2. Essa ferramenta possibilita a execução da aplicação sem a necessidade de implementar todas as alterações exigidas pela versão 3.0 do Vue.js. Além de proporcionar essa compatibilidade, o pacote também gera mensagens de erro e avisos, apontando áreas específicas do código que não são mais compatíveis com o Vue.js 3. Essas mensagens fornecem orientações sobre como corrigir essas partes, servindo como um valioso auxílio para os desenvolvedores durante o processo de migração.

### 2.5.3 Options API e Composition API

Dentre as principais diferenças entre as versões do Vue.js, a que apresenta mais alterações é a opção de utilizar uma nova forma de estruturar a aplicação conhecida como Composition API. Até a versão 2x, só era possível a utilização da Options API.

A Options API tem como foco o conceito de "instância do componente", o que normalmente se alinha melhor com um modelo mental baseado em classes para usuários que vêm de linguagens de programação orientadas a objetos (OOP). Além disso, ela é mais amigável para iniciantes, abstraindo os detalhes da reatividade e impondo a organização do código por meio de grupos de opções.

A Composition API, por sua vez, centraliza-se na declaração de variáveis de estado reativas diretamente no escopo de uma função e na composição de estados a partir de múltiplas funções para lidar com a complexidade. Ela é mais flexível e requer um entendimento de como a reatividade funciona no Vue.js para ser usada de forma efetiva.

Em troca, sua flexibilidade possibilita padrões mais poderosos para organizar e reutilizar lógicas.

As figuras 2 e 3 mostram um exemplo de código que evindecia as diferenças entre essas APIs. Este código consiste em um componente que exibe uma mensagem armazenada na variável "message" e possui um botão para alterar a mensagem, onde ao ser clicado, chama a função "changeMessage", alterando o texto da variável. Esta variável é reativa, ou seja, quando seu conteúdo é alterado, o componente é renderizado novamente para exibir a mudança. Na figura 2, o componente é implementado utilizando Options API, sendo necessário definir esta variável dentro do objeto "data", e a função dentro do objeto "methods". Já na figura 3 que representa a implementação deste componente utilizando Composition API, a implementação da variável e da função é alterada, não precisando que sejam definidas dentro de objetos estruturados, como na Options API, oferecendo mais liberdade ao desenvolvedor.

```
<template>
    <div>
      <p>{{ message }}</p>
      <button @click="changeMessage">Alterar Mensagem</button>
    </div>
  </template>

  <script>
  export default {
    data() {
      return {
        message: 'Olá, Vue 2!',
      };
    },
    methods: {
      changeMessage() {
        this.message = 'Mensagem Alterada!';
      },
    },
  };
  </script>
```

**Figura 2: Pequeno componente utilizando Options API**

```
<template>
  <div>
    <p>{{ message }}</p>
    <button @click="changeMessage">Alterar Mensagem</button>
  </div>
</template>

<script setup>
import { ref } from 'vue';

const message = ref('Olá, Vue 3!');

const changeMessage = () => {
  message.value = 'Mensagem Alterada!';
};
</script>
```

**Figura 3: Pequeno componente utilizando Composition API**

Além deste exemplo, a figura 4 destaca a organização de código possibilitada pela Composition API. O intuito é ilustrar as diferenças estruturais entre o código que utiliza a Options API e aquele que emprega a Composition API. Cada cor na figura representa

uma regra de negócio da aplicação, ou seja, a estrutura que representa uma funcionalidade específica do sistema.

Options API Composition API

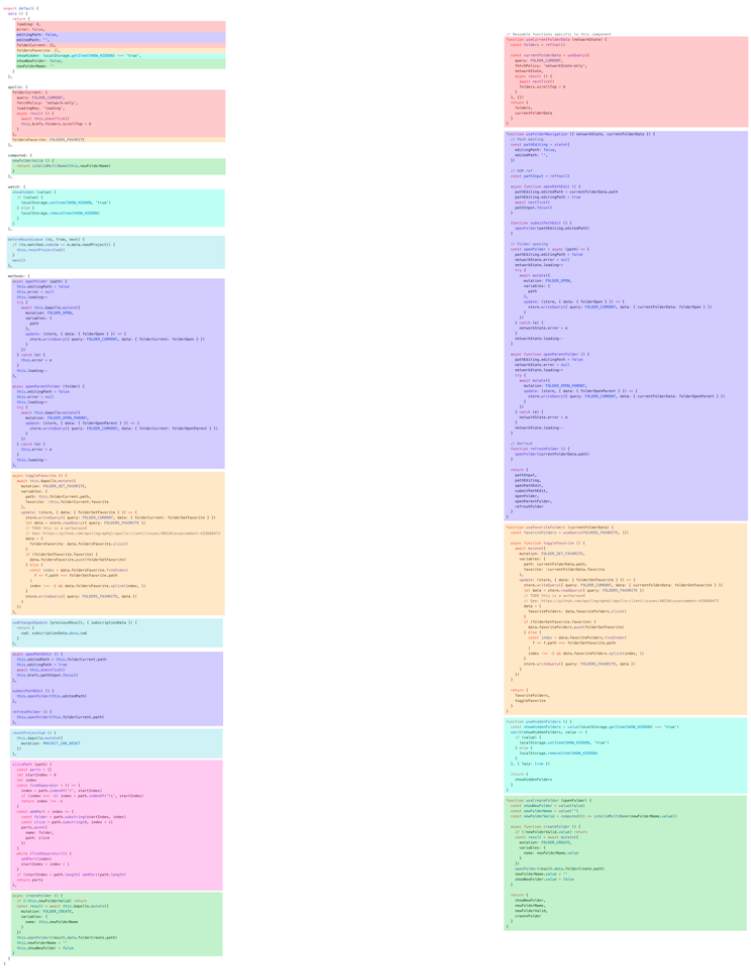

**Figura 4: Exemplo de estrutura de código utilizando Options e Composition API**
Fonte: https://vueschool.io/articles/vuejs-tutorials/options-api-vs-composition-api/

Embora aplicações ou componentes possam ser construídos tanto com a Composition API quanto com a Options API, a documentação oficial do Vue.js faz as seguintes recomendações:

- Escolha a Options API se planeja utilizar o Vue.js em cenários de baixa complexidade, como aplicações ou componentes que possuam código mais simples e não demandem atualizações constantes.

- Escolha a Composition API junto com Single-File Components se planeja construir aplicações completas e robustas.

## 2.6 Quasar

Quasar é um *framework open-source* baseado em Vue.js que permite criar de forma mais rápida aplicações de várias plataformas, como Web e *mobile*. Dentre as principais vantagens, pode-se destacar (QUASAR):

- Boas práticas de desenvolvimento Web integradas por padrão;
- Fácil aprendizado e construção rápida de novas aplicações;
- Suporte a um grande número de plataformas como: Google Chrome, Firefox, Edge, Safari, Opera, iOS, Android, MacOS, Linux, Windows;
- Documentação bem detalhada e comunidade em crescimento;
- Uma ampla biblioteca de componentes

Sobre os componentes disponíveis, as principais características que dão um destaque para este *framework* é que eles foram desenvolvidos tendo como foco a responsividade, performance, interfaces de usuário modernas e alta customização, facilitando o desenvolvimento de novas páginas em aplicações Web.

As figuras 5 e 6 abaixo ilustram a diferença entre uma pequena tabela em Vue.js construída com e sem Quasar, respectivamente. Como pode-se notar, a tabela construída com Quasar tem o código do template bem reduzido, já que o componente QTable encapsula a lógica de composição da tabela, além da estilização, que se faz necessária caso não se utilize a biblioteca de componentes

```vue
<template>
  <div>
    <table class="table">
      <thead>
        <tr>
          <th>ID</th>
          <th>Nome</th>
          <th>Email</th>
        </tr>
      </thead>
      <tbody>
        <tr v-for="item in data" :key="item.id">
          <td>{{ item.id }}</td>
          <td>{{ item.nome }}</td>
          <td>{{ item.email }}</td>
        </tr>
      </tbody>
    </table>
  </div>
</template>

<script>
export default {
  data() {
    return {
      data: [
        { id: 1, nome: "João", email: "joao@example.com" },
        { id: 2, nome: "Maria", email: "maria@example.com" },
        { id: 3, nome: "José", email: "jose@example.com" }
      ]
    };
  }
};
</script>

<style>
.table {
  width: 100%;
  border-collapse: collapse;
}

.table th, .table td {
  border: 1px solid #ddd;
  padding: 8px;
  text-align: left;
}

.table th {
  background-color: #f2f2f2;
}
</style>
```

**Figura 5: Componente Vue.js construído sem Quasar**

```vue
<template>
  <div>
    <q-table
      :rows="data"
      :columns="columns"
      row-key="id"
    ></q-table>
  </div>
</template>

<script>
export default {
  data() {
    return {
      data: [
        { id: 1, nome: "João", email: "joao@example.com" },
        { id: 2, nome: "Maria", email: "maria@example.com" },
        { id: 3, nome: "José", email: "jose@example.com" }
      ],
      columns: [
        {
          name: "id",
          required: true,
          label: "ID",
          align: "left",
          field: "id"
        },
        {
          name: "nome",
          required: true,
          label: "Nome",
          align: "left",
          field: "nome"
        },
        {
          name: "email",
          required: true,
          label: "Email",
          align: "left",
          field: "email"
        }
      ]
    };
  }
};
</script>
```

**Figura 6: Componente Vue.js construído sem Quasar**

## 2.7 Modelo C4

O modelo C4 consiste em um conjunto de diagramas hierárquicos que pode ser utilizado para descrever a arquitetura de um software em diferentes níveis de detalhamento, cada um útil para públicos diferentes. Este modelo considera as estruturas estáticas de um sistema de software em termos de *containers* (aplicativos, armazenamentos de dados, *microservices*, etc.), componentes e código, além de considerar as pessoas que usam este sistema (BROWN, 2018). O termo C4 significa contexto, *containers*, componentes e código, então é possível representar a arquitetura do software nestes 4 níveis de diagrama. A figura 7 apresenta uma visão geral do modelo.

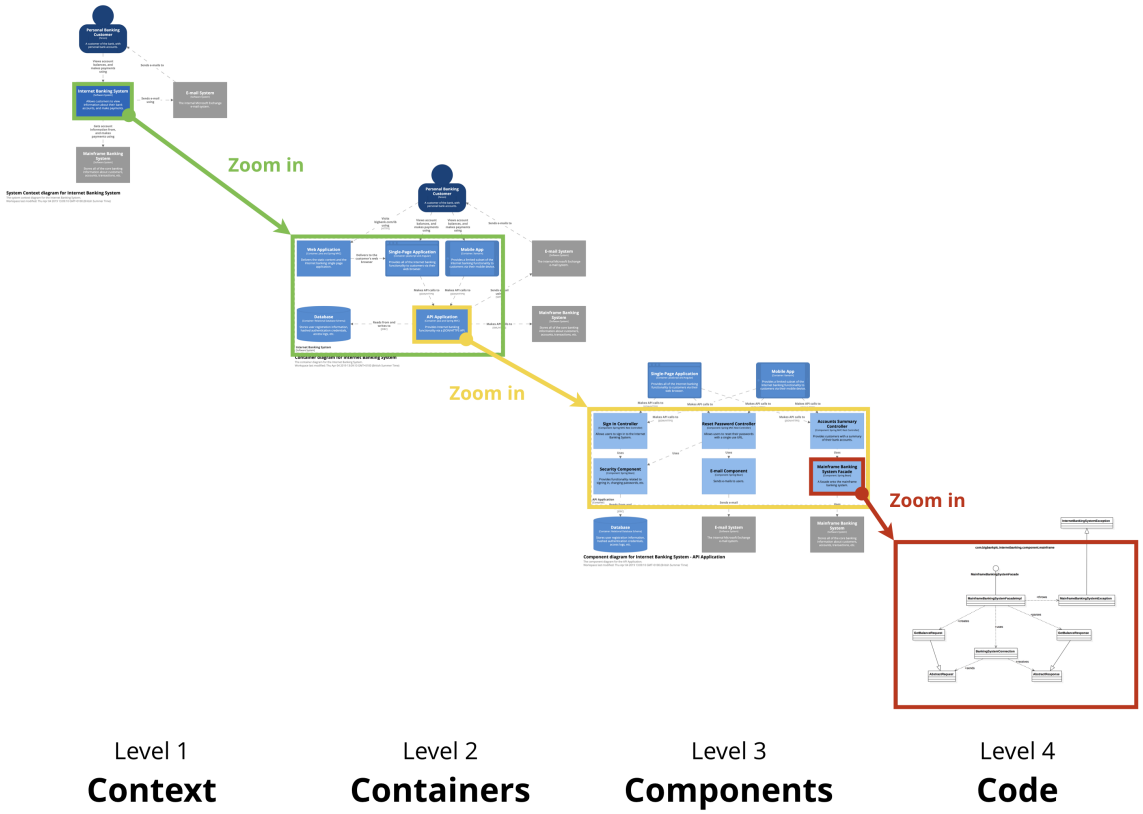


**Figura 7: Visão geral do modelo C4**
Fonte: Brown(2018)

### 2.7.1 Diagrama de contexto

O diagrama de contexto do sistema é o ponto de início da documentação da aplicação, e também possui o maior nível de abstração. Neste nível, é desenhado o sistema que vai ser documentado ao centro, rodeado pelos usuários e outros sistemas que interagem com a aplicação. O foco desta versão devem ser as pessoas e sistemas envolvidos com a aplicação.

### 2.7.2 Diagrama de container

O diagrama de container consiste em uma representação de alto nível da arquitetura do software e como as responsabilidades são distribuídas por ela. Cada container é uma unidade do sistema que é executada separadamente, e que juntas formam o sistema como o todo, por exemplo *server-side Web application, single-page application, desktop app, mobile app, database schema, file system*, entre outros. Ele também mostra as principais opções de tecnologia e como os contêineres se comunicam entre si. É um diagrama simples e focado em tecnologia que é útil para desenvolvedores de software e equipe de

suporte/operações.

### 2.7.3 Diagrama de componente

O nível 3 do modelo c4 consiste em um diagrama de componentes, onde um container individual é ampliado para mostrar os componentes dentro dele. Esses componentes devem mapear para abstrações reais (por exemplo, um agrupamento de código) em uma base de código, onde cada um tem suas responsabilidades e detalhes de implementação e/ou tecnologia.

### 2.7.4 Diagrama de código

Este nível de diagrama permite mostrar como um componente da aplicação é implementado, mostrando as classes utilizadas e detalhes da implementação. Este é um nível opcionado, e normalmente exportado sob demanda pelas IDEs. Além disso, devido ao nível de complexidade e constante ocorrência de atualizações, este diagrama só costuma ser utilizado para alguns componentes complexos.

## 2.8 Conclusão do capítulo

Este capítulo teve como objetivo construir um alicerce para a metodologia e o processo de atualização que são apresentados nos capítulos subsequentes. Foi discutida uma visão geral sobre a evolução das aplicações Web, seus padrões arquiteturais, e em seguida, uma base conceitual sobre os principais frameworks que serão abordados neste processo, o Vue.js e o Quasar, além de alguns detalhes necessários para prosseguir com o processo, como uma explicação breve sobre a Migration Build, e a diferença entre Options API e Composition API. Além disso, também foi especificado o modelo de documentação utilizado, o C4.

# 3 METODOLOGIA

Este trabalho tem como objetivo principal estudar, planejar e aplicar o processo de atualização de um framework Web - o Vue.js - em uma aplicação frontend, que representa a plataforma central de gerenciamento da empresa Workverse. Para atingir esse objetivo, o processo de atualização foi dividido nas seguintes etapas: estudo sobre o desenvolvimento de aplicações Web e o uso de frameworks, análise da plataforma e planejamento do processo de atualização, implementação do processo e avaliação da atualização.

## 3.1 Estudo sobre o desenvolvimento de aplicações Web e o uso de frameworks

Nesta etapa inicial, realizou-se uma análise abrangente do cenário de desenvolvimento de aplicações Web e da importância dos *frameworks* nesse contexto. Foram explorados conceitos, padrões de projeto e as melhores práticas de desenvolvimento Web. Além disso, foram conduzidos estudos aprofundados sobre os frameworks Vue.js e Quasar, os principais componentes da plataforma Workverse. Isso permitiu compreender as mudanças introduzidas nas novas versões desses frameworks e como aplicá-las de forma eficaz na aplicação da Workverse.

## 3.2 Análise da plataforma e planejamento do processo de atualização

Após a fase de estudo, realizou-se uma análise completa da aplicação Workverse, identificando os seus principais componentes e funcionalidades. Essa análise serviu como base para o planejamento do processo de atualização. Para obter uma visão abrangente da plataforma, empregou-se o modelo C4, permitindo uma melhor compreensão da arquitetura e dos relacionamentos dos componentes. Posteriormente, foi realizada uma reunião estratégica com o líder do time de desenvolvimento frontend da empresa para definir as etapas do processo de atualização. Em colaboração com esse líder, os marcos, o cronograma e os prazos para cada fase foram estabelecidos, garantindo um controle eficaz do projeto.

## 3.3 Implementação do processo

Com o processo de atualização cuidadosamente planejado, iniciou-se a implementação, seguindo rigorosamente o fluxo definido na etapa anterior. A cada fase, foram realizadas reuniões com o líder de desenvolvimento para avaliar o progresso e os resultados. Essas avaliações englobaram a verificação da funcionalidade das principais características, a consistência na interface do usuário e a disponibilização de novas funcionalidades. Adicional-

mente, em cada fase, foram registradas possíveis falhas de estrutura, que posteriormente foram agrupadas como sugestões para refatoração.

## 3.4 Avaliação da Atualização

Por fim, realizou-se uma avaliação da aplicação, iniciando com uma comparação das principais páginas da plataforma antes e depois do processo, verificando também se a aplicação continuava a funcionar normalmente com a nova versão do framework Vue.js e se não houve mudanças significativas na aparência dessas telas. Além disso, foram avaliados o cumprimento dos prazos planejados e a taxa de sucesso nos casos de teste.

# 4 APRESENTAÇÃO E ANÁLISE DOS RESULTADOS

Este capítulo tem como propósito fornecer uma visão abrangente da arquitetura do sistema Workverse, com foco especial na plataforma de gerenciamento, avaliando o modelo navegacional e abordando os níveis C1, C2 e C3 do modelo C4, e definir o caminho em que será feita a atualização desta plataforma.

Além disso, este capítulo discutirá os passos e decisões tomadas durante o processo de atualização do *framework* Vue.js e outras bibliotecas e *frameworks* essenciais para suas versões mais recentes. O objetivo é apresentar as considerações e estratégias que orientaram essa atualização, e em seguida, mostrar os resultados alcançados.

## 4.1 Visão geral da Plataforma de Gerenciamento Workverse

A Plataforma de Gerenciamento Workverse é uma aplicação que reúne diversas funcionalidades para os clientes da empresa Workverse. Esses clientes possuem *videobots*, que são softwares interativos feitos para facilitar os procedimentos da área de Recursos Humanos, e buscam realizar todas as operações de gerenciamento, como envio para usuários, monitoramento de dashboards, acompanhamento de dados do *videobot* e cadastro de conteúdos, entre outras funcionalidades. Além disso, usuários com permissões mais elevadas têm a capacidade de gerenciar usuários com níveis de permissões mais baixos.

Construída com o Vue.js como framework principal, a plataforma estava inicialmente na versão 2.0 e precisava ser atualizada para a versão 3.0. Outros elementos fundamentais incluem o uso do framework Quasar, bem como bibliotecas como i18n, Vuelidate e Vuex, responsáveis por internacionalização, validação de formulários e controle de estado global da aplicação, respectivamente.

Os principais usuários dessa plataforma são as empresas-clientes e os colaboradores da Workverse. Os usuários comuns dos videobots não possuem acesso, pois a plataforma é projetada principalmente para o gerenciamento em nível corporativo, proporcionando acesso aos dados enviados pelos colaboradores.

### 4.1.1 Esquema navegacional

Com base no diagrama de navegação apresentado na figura 8, é possível mapear o fluxo de interação na plataforma, identificando as principais páginas que são acessadas pelos usuários e os caminhos de interação. A navegação inicia com a página de Login, onde os usuários fornecem suas credenciais para autenticação. Após a autenticação bem-sucedida, a página Vídeos é carregada, permitindo o acesso aos videobots de uma campanha.

Após o login, é possível por meio de um menu de navegação, acessar outras partes do sistema, como Users, responsável pelo gerenciamento de usuários, e Collaborators, responsável pelo gerenciamento de colaboradores. Além disso, botões para acessar páginas adicionais, como Config, Categories, Mailings, Realtime, DevOptions e Dashboard, são disponibilizados na tela de Vídeos. Outras páginas, como UsersToExclude, SessionsToExclude, CollaboratorsStatistics e MailingSchedule, são consideradas componentes integrados às páginas principais do sistema em uma perspectiva arquitetural. Nos próximos tópicos, será explorada detalhadamente a funcionalidade de cada uma dessas páginas.

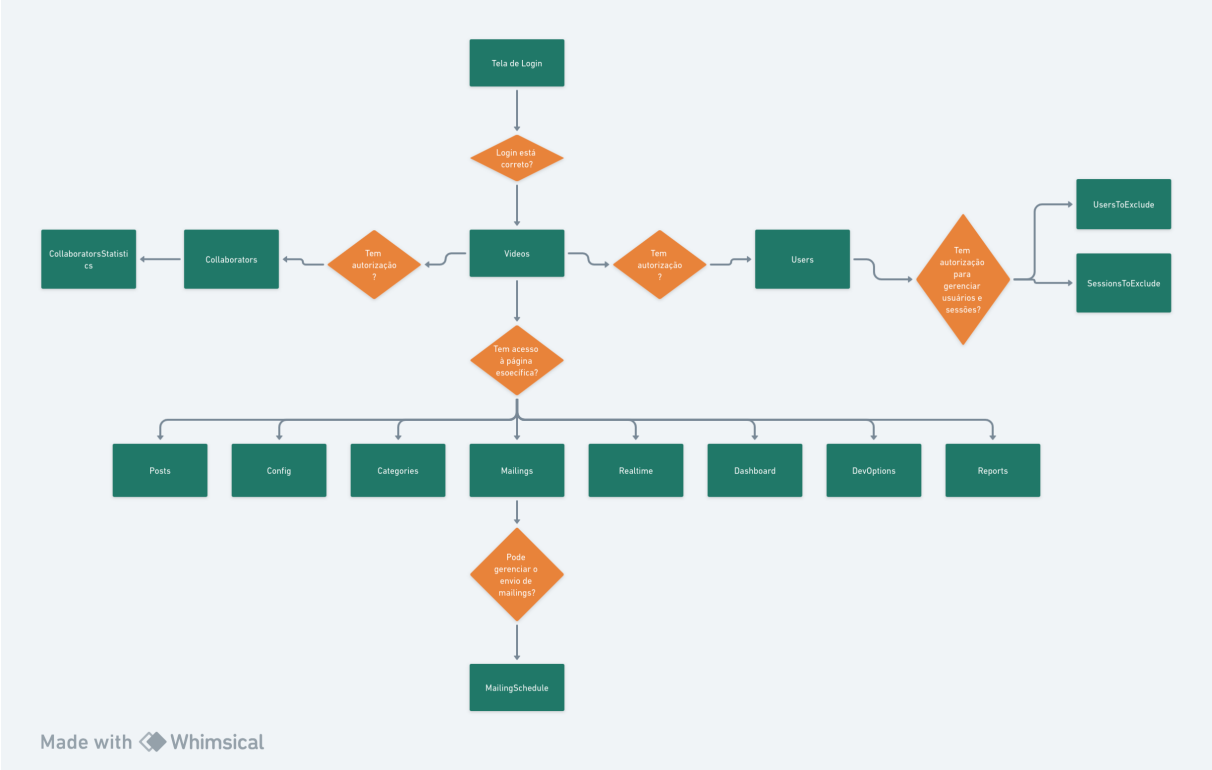


**Figura 8: Diagrama de navegação da plataforma Workverse**

É possível ver a figura 8 em mais detalhes no link a seguir: https://drive.google.com/file/d/12pWIGKsTrMmDGGikhjLnDGIJfkFBWGj_/view?usp=sharing.

## 4.2 Arquitetura do sistema da Plataforma de Gerência Workverse

Após ser feita uma análise da navegação da Plataforma, foi iniciada uma análise arquitetural no Sistema Workverse, com foco especial na plataforma de gerenciamento. Esta será a aplicação-alvo das atualizações deste estudo, pois encontra-se com uma versão desatualizada do Vue.js e desempenha um papel crucial no funcionamento do sistema.

### 4.2.1 Visão de mais alto nível - C1

Como é possível visualizar no diagrama de nível C1 na figura 9, o sistema Workverse é acessado por dois tipos de usuários: usuários comuns que interagem com videobots e gerentes que exercem o controle sobre os usuários e acessam relatórios, geram novas contas e administram videobots. Essas duas categorias de usuários têm interações distintas com o sistema, o que é evidenciado na arquitetura do sistema.

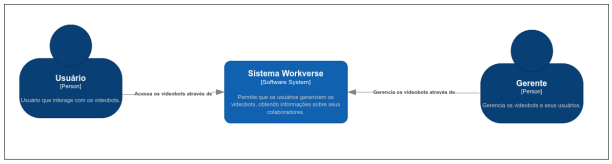


**Figura 9: Diagrama de contexto do Sistema Workverse**

### 4.2.2 Visão de alto nível C2

A próxima camada da arquitetura proporciona uma visão mais aprofundada dos containers que constituem o sistema Workverse. Este diagrama, como evidenciado na figura 10, concentra-se exclusivamente nos containers e atores que interagem com a Plataforma de Gerência Workverse, que é o foco central deste estudo. Essa plataforma representa a aplicação principal para a gestão de videobots e usuários, bem como a visualização de relatórios no formato de um painel de controle.

Os usuários com a função de gerente têm autorização para administrar essa plataforma, e é através deles que ocorre o gerenciamento das operações. Além disso, a plataforma realiza solicitações de leitura e gravação para dois componentes-chave: a API Workverse, que atua como o container principal onde as regras de negócio do sistema são implementadas, e o Tracking Report, responsável pela geração de relatórios relacionados às interações dos videobots. Esses relatórios são posteriormente exibidos em telas específicas dentro da plataforma.

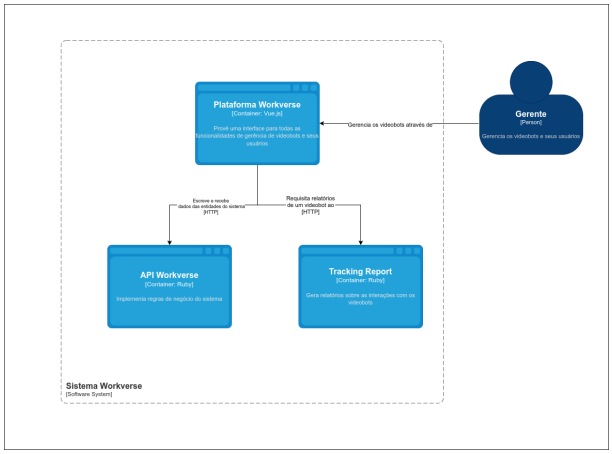


**Figura 10: Diagrama de container do Sistema Workverse com foco na plataforma**

O diagrama C2 está disponível em tamanho maior no link: <https://drive.google .com/file/d/1RquSZI3qL60dzpDWf1FtxFtfa1j3ycyb/view?usp=sharing >

### 4.2.3 Visão de baixo nível - C3 (Plataforma de gerenciamento)

Neste nível, a plataforma de gerenciamento Workverse é examinada em detalhes, conforme ilustrado no diagrama de componentes apresentado na figura 11. O funcionamento interno da plataforma começa com o componente "App", cuja função primordial é criar um template base para a aplicação e personalizá-lo à medida que o usuário navega pelas diferentes rotas. É nesta fase que o código Vue.js é transformado em HTML, CSS e JavaScript, permitindo a renderização da interface.

Todas as rotas de acesso na interface da plataforma são mapeadas a partir do componente "Router,"um elemento compartilhado por diversos componentes dentro da Single Page Application (SPA). Outros componentes que compartilham essa característica incluem a "Api,"responsável pela comunicação entre os componentes da plataforma e as aplicações *backend*; a "Store,"encarregada de fornecer recursos para o gerenciamento do estado global da aplicação; e o "Components,"que abriga a implementação de diversos componentes e layouts usados por várias páginas. Esses layouts são inseridos através de slots no roteamento da aplicação, proporcionando flexibilidade na estrutura da interface.

Dentro do módulo "Pages", encontramos os componentes estruturais que são renderizados dentro do "App"de acordo com as rotas específicas. Esses componentes representam as diversas páginas da plataforma:

Neste nível, a plataforma de gerenciamento Workverse é examinada em detalhes, conforme ilustrado no diagrama de componentes apresentado na figura 11. O funcionamento interno da plataforma começa com o componente "App", cuja função primordial é

criar um template base para a aplicação e personalizá-lo à medida que o usuário navega pelas diferentes rotas. É nesta fase que o código Vue.js é transformado em HTML, CSS e JavaScript, permitindo a renderização da interface. Todas as rotas de acesso na interface da plataforma são mapeadas a partir do componente "Router,"um elemento compartilhado por diversos componentes dentro da Single Page Application (SPA). Outros componentes que compartilham essa característica incluem a "Api,"responsável pela comunicação entre os componentes da plataforma e as aplicações backend; a "Store,"encarregada de fornecer recursos para o gerenciamento do estado global da aplicação; e o "Components,"que abriga a implementação de diversos componentes e layouts usados por várias páginas. Esses layouts são inseridos através de slots no roteamento da aplicação, proporcionando flexibilidade na estrutura da interface.

Dentro do módulo "Pages", encontramos os componentes estruturais que são renderizados dentro do "App"de acordo com as rotas específicas. Esses componentes representam as diversas páginas da plataforma:

- Users: Responsável por fornecer ferramentas para a administração de usuários da aplicação, incluindo o controle de acessos, a geração de novas senhas e outras funcionalidades de gerenciamento. O acesso a esta página é restrito a usuários autorizados.
- UsersToExclude: Gerencia a exclusão do acesso de determinados usuários a um vídeo específico.
- SessionsToExclude: Gerencia a exclusão de sessões relacionadas a um vídeo específico.
- Login: Fornece uma interface de autenticação de usuários na plataforma e suporta a utilização de Single Sign-On (SSO).
- TwoFactors: Oferece ferramentas para autenticação de dois fatores.
- Collaborators: Permite o gerenciamento de colaboradores associados a campanhas na plataforma.
- CollaboratorsStatistics: Disponibiliza uma interface para visualizar estatísticas relacionadas aos colaboradores.

Dentro do módulo "Videos,"que abriga a maioria das páginas da aplicação relacionadas ao gerenciamento de videobots, encontram-se as seguintes páginas:

- Vídeos: Proporciona uma interface para visualizar videobots associados a uma conta específica e inclui botões para acessar outras páginas relacionadas ao gerenciamento desses videobots.

- Realtime: Apresenta uma tabela para visualização dos dados de acompanhamento dos videobots, incluindo eventos acionados e formulários preenchidos. Oferece funcionalidades adicionais, como filtragem e download de dados.

- Categories: Gerencia os conteúdos dos videobots, incluindo uploads de vídeos, textos, imagens e PDFs destinados à exibição nos videobots.

- Posts: Anteriormente responsável pelo gerenciamento de conteúdo, essa função foi substituída pela página "Categories."

- Mailings: Gerencia os mailings de videobots, que são planilhas contendo os dados necessários para a criação de videobots para usuários específicos. A partir desses mailings, é possível gerar vídeos, ativá-los, desativá-los e enviar e-mails e mensagens de texto.

- MailingSchedule: Contém ferramentas para agendamento de mailings, atualmente utilizadas por um cliente e vídeo específicos.

- Dashboard: Oferece uma interface para visualizar dados sob a forma de gráficos analíticos, proporcionando uma visão geral dos dados do sistema.

- Reports: Originalmente responsável por fornecer gráficos analíticos, essa função foi substituída pela página "Dashboard."

- DevOptions: Fornece ferramentas para desenvolvedores, incluindo o upload de ativos para videobots.

- Config: Oferece um conjunto de componentes para configurar videobots, incluindo definições como nome, título, eventos e dados de engajamento, entre outros.

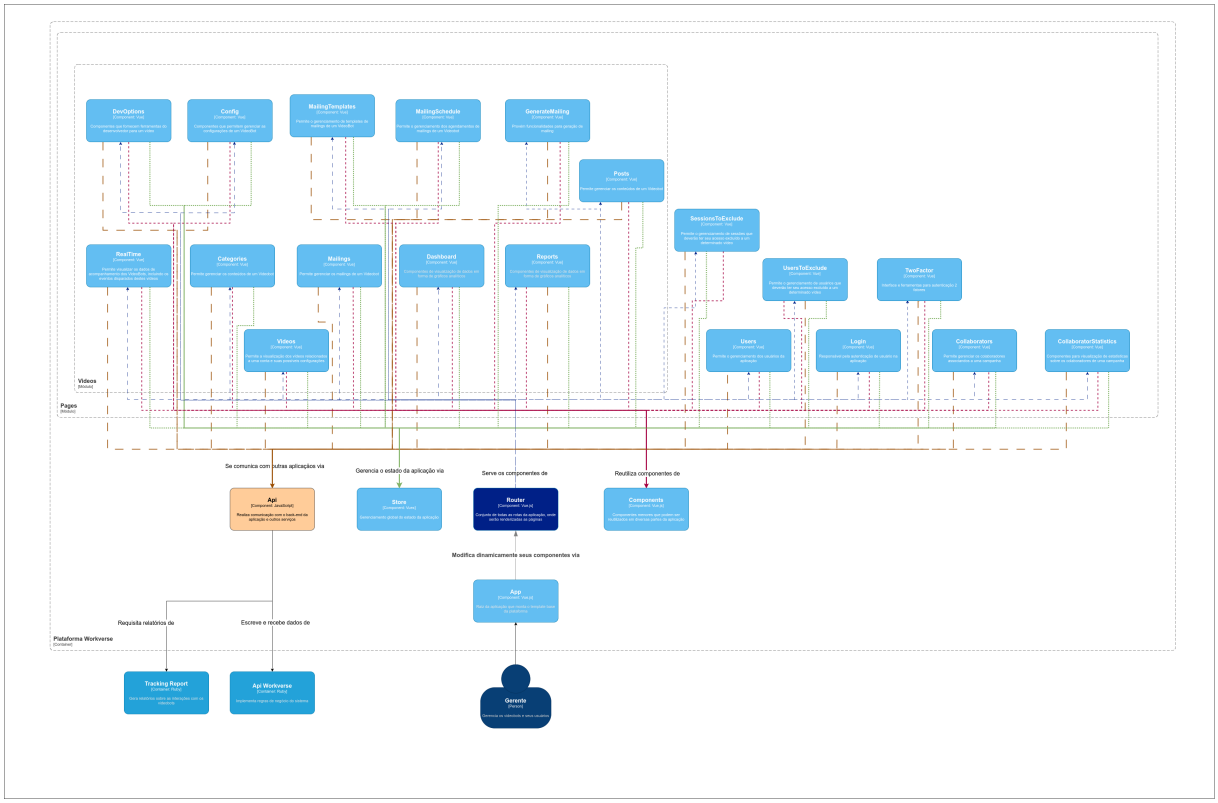


**Figura 11: Diagrama de componentes da container Plataforma Workverse**

O diagrama C3 está disponível em tamanho maior no link: <https://drive.google.com /file/d/1Fu7hZ4sUGBCzNsrI9UkClMHQSsR8lBpk/view?usp=sharing >

## 4.3 Proposta de atualização para Vue.js 3.0

Segundo a documentação do Vue.js, a versão 2.3 utilizada atualmente pela plataforma Workverse terá seu suporte encerrado em 31 de dezembro de 2023, com base nisso, fez-se necessário uma estratégia de atualização para a nova versão.

Tendo esse objetivo, foi analisado, junto com o líder do time de desenvolvimento das aplicações front-end da empresa, qual seria o caminho ideal para esse processo. Após algumas discussões, com auxílio da revisão bibliográfica e análise arquitetural da plataforma, foi definido o fluxo de atualização, que consiste nas seguintes etapas:

1. Atualização intermediária Quasar v0.7 para v1 - Para atualizar para a nova versão do Vue.js, é necessário que o Quasar esteja na versão 2, porém este framework não dá suporte a atualização da v0.7 diretamente para v2, então é necessário fazer este passo intermediário, atualizando primeiro para a v1;

2. Atualizar versão do Vue.js e Quasar v2 utilizando a Migration Build - Após realizar a primeira atualização no Quasar, já é possível atualizar primeiramente a versão do

Vue.js para v3, e, logo em seguida, a versão do Quasar para v2, chegando assim nas versões mais recentes, e então começar as adaptações no código;

3. Remover a Migration Build e realizar novos testes - Este passo ocorre após os ajustes recomendados pela Migration Build, removendo ela e tendo a aplicação funcionando já com o Vue.js 3 sem precisar utilizar a biblioteca de compatibilidade.

4. Migrar de Options API para Composition API - Este é um passo opcional, porém bastante recomendado para aplicações robustas, principalmente porque a nova versão da API (Composition API) possui uma melhor performance e organização de código, tornando mais fácil a manutenção, compreensão e atualização das funcionalidades;

5. Ajustes finos e melhoria de estilização - Com todas as atualizações, uma boa parte da estilização acabou sendo perdida, principalmente com a troca de alguns componentes, então esta etapa será necessária para que os usuários não percebam mudanças ao que já estavam acostumados.

Além disso, teve que ser planejado como seria validada cada etapa, e chegou-se à conclusão de que deveria ser testada página a página verificando as principais funcionalidades, então para isso, foi criada uma tabela, como mostra a figura 12 onde seria possível acompanhar as validações em cada etapa. As últimas 3 etapas não foram incluídas nestes testes porque são opcionais ou não realizam alterações especificamente nos casos de uso ou funcionalidades, então não seria necessário realizar todos os testes novamente.

| Caso de uso | Página | Importância | Quasar v1 | Migration Build | Vue 3 |
|---|---|---|---|---|---|
| Realizar login padrão | Login | ALTA | | | |
| Realizar login two factor | Login | MÉDIA | | | |
| Realizar login SSO | Login | MÉDIA | | | |
| Visualizar vídeos do cliente | Videos | ALTA | | | |
| Exibir opções disponíveis para um video | Videos | ALTA | | | |
| Permitir configurar a brand channel | Config | ALTA | | | |
| Permitir configurar o envio de mailings | Config | ALTA | | | |
| Permitir configurar os eventos para acompanhamento | Config | MÉDIA | | | |
| Permitir configurar os dados de conversão | Config | MÉDIA | | | |
| Permitir configurar os dados de engajamento | Config | MÉDIA | | | |
| Permitir configurar os dados sensíveis | Config | ALTA | | | |
| Exibir os dados de acompanhamento de um vídeo | Realtime | ALTA | | | |
| Alterar o espaço de tempo dos dados exibidos | Realtime | MÉDIA | | | |
| Permitir acessar as telas de perfil para alguns clientes | Realtime | ALTA | | | |
| Gerenciar os conteúdos de um vídeo | Categories | ALTA | | | |
| Fazer uploads de conteúdo de vídeo, imagens, etc | Categories | ALTA | | | |
| Gerenciar os mailings de um videobot | Mailings | ALTA | | | |
| Permitir o download e upload de templates | Mailings | MÉDIA | | | |
| Realizar envio de email e sms | Mailings | ALTA | | | |
| Realizar o agendamento de disparo de mailings | MailingSchedule | MÉDIA | | | |
| Exibir dashboard interativo sobre um videobot | Dashboard | BAIXA | | | |
| Permitir o upload de assets para serem utilizados no videobot | DevOptions | BAIXA | | | |
| Permitir a visualização do último deploy e interatividades | DevOptions | BAIXA | | | |
| Gerenciar os usuários de um cliente (empresa) | Users | ALTA | | | |
| Gerenciar permissões destes usuários | Users | ALTA | | | |
| Gerenciar os colaboradores de uma empresa | Collaborators | MÉDIA | | | |

**Figura 12: Tabela utilizada para mapear os casos de uso para teste**

Com o fluxo definido e os testes planejados, foi possível dar início ao processo de atualização.

### 4.3.1 Atualização do Quasar v0.7 para v1

O primeiro passo para esta atualização terá como foco o framework Quasar, pois a plataforma utiliza a versão v0.7 e o Vue.js 3 suporta apenas a versão 2 em diante, porém não é possível atualizar de forma direta, e com isso, se faz necessário uma atualização prévia para a versão v1.

Esta atualização exige duas etapas: ajustes nos arquivos de configurações do Quasar e a verificação, em todas as páginas, dos componentes que foram atualizados ou deixaram de existir na nova versão. A segunda etapa foi feita de acordo com o fluxo de iteração apresendado na figura 13:

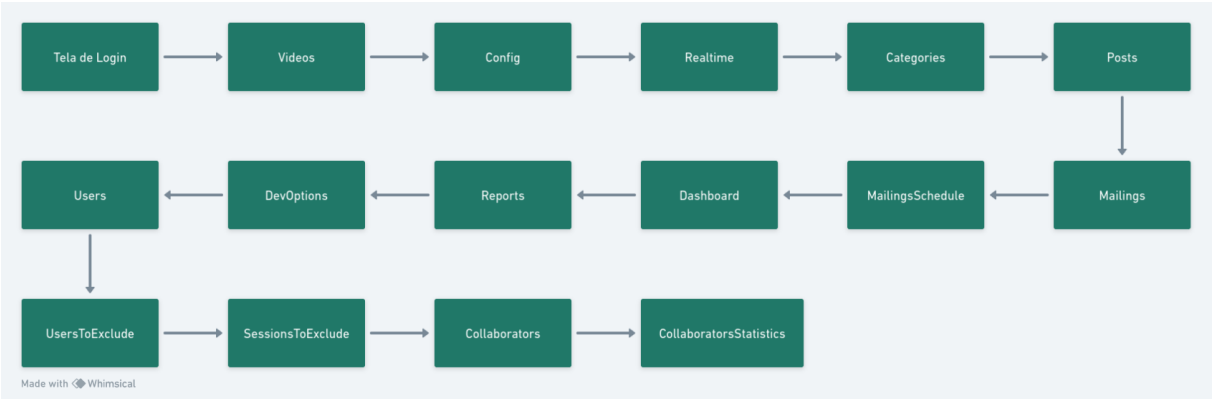


**Figura 13: Fluxo de iteração entre as páginas para atualização**

A ordem sequencial dessa atualização é determinada pelo esquema de navegação da aplicação, que inicia com a tela de Login como a primeira a ser exibida, tornando-a a escolha inicial. Em seguida, os componentes compartilhados por várias páginas, como layouts, são avaliados. E então, as páginas com uma ampla gama de funcionalidades são abordadas sequencialmente.

As principais modificações são documentadas no apêndice A, consistindo principalmente em substituições de componentes, adição ou remoção de atributos e ajustes de estilo. O resultado obtido após a conclusão de todas as etapas é uma versão com componentes atualizados e todos os casos de uso funcionando conforme o esperado. A única diferença notável reside na estilização, já que foi decidido que, devido a futuras atualizações, as correções no estilo visual seriam realizadas no final do processo, quando todas as bibliotecas e frameworks estivessem na versão mais recente para evitar ter que refazer tudo após cada fase.

As figuras 14 e 15 abaixo mostram a diferença entre um modal de upload de arquivo de mailing antes e depois da atualização do Quasar, respectivamente.


**Figura 14: Modal de upload de Mailing antes da atualização do Quasar**

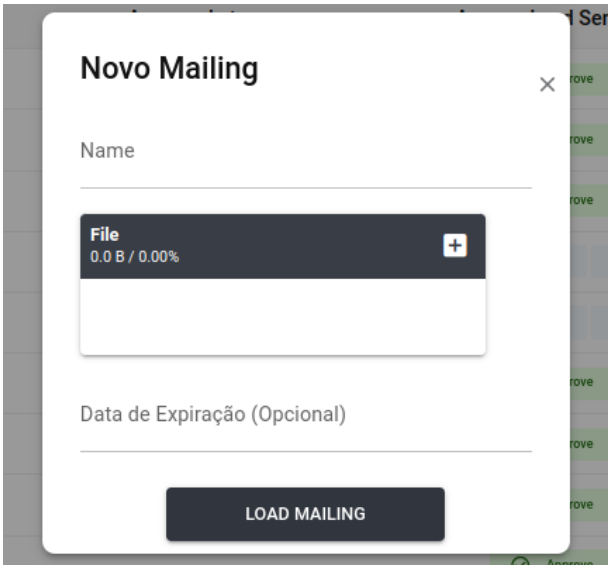


**Figura 15: Modal de upload de Mailing após da atualização do Quasar**

### 4.3.2 Atualização para Vue.js 3 utilizando a Migration Build

A estratégia de atualização sugerida pela documentação do Vue.js consiste em instalar na aplicação uma biblioteca chamada Migration Build, responsável por fazer a compatibilidade entre as versões 2 e 3 deste *framework*, sugerindo ao desenvolvedor os passos necessários para a nova versão. O processo começa com a correção dos erros identificados pela biblioteca, bem como dos alertas que apontam possíveis problemas de compatibilidade. Uma vez concluídas essas correções, a *build* de compatibilidade pode ser

removida, e a aplicação pode ser atualizada para utilizar exclusivamente a nova versão do Vue.js.

Seguindo este guia, foi iniciado a atualização, realizando a instalação da Migration Build, porém devido ao fato da aplicação também utilizar o framework Quasar, foi necessário realizar simultaneamente a sua atualização, pois sem ela, a aplicação não iria ser compilada. Então, também nesta etapa foi realizada a atualização do Quasar da versão 1 para versão 2, ajustando os arquivos de configurações, principalmente o quasar.config onde são definidas as regras de build da plataforma. Realizando essas alterações, a Migration Build começou a funcionar, exibindo as mudanças necessárias que deverão ser implementadas, como mostra a figura 16.

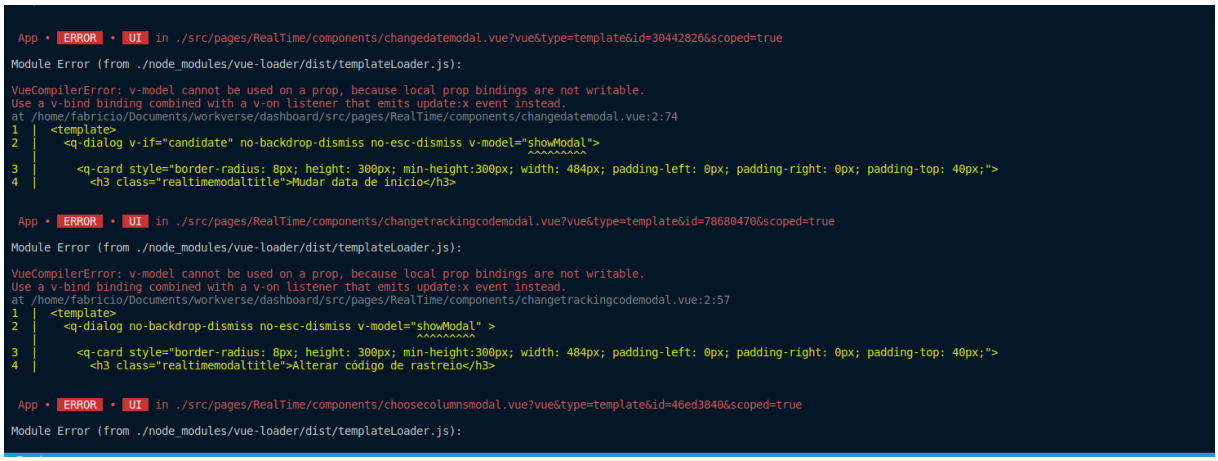


**Figura 16: Captura de tela mostrando os erros emitidos pela Migration Build**

Com a ajuda da biblioteca de migração, foi possível verificar os principais conflitos entre as versões, principalmente os erros críticos que não permitiam subir a aplicação. Após corrigir os erros, já foi possível utilizar a plataforma com a nova versão do Vue.js, porém ainda havia erros de conflitos de outras bibliotecas, mas muitas funcionalidades já estavam funcionando com a nova versão.

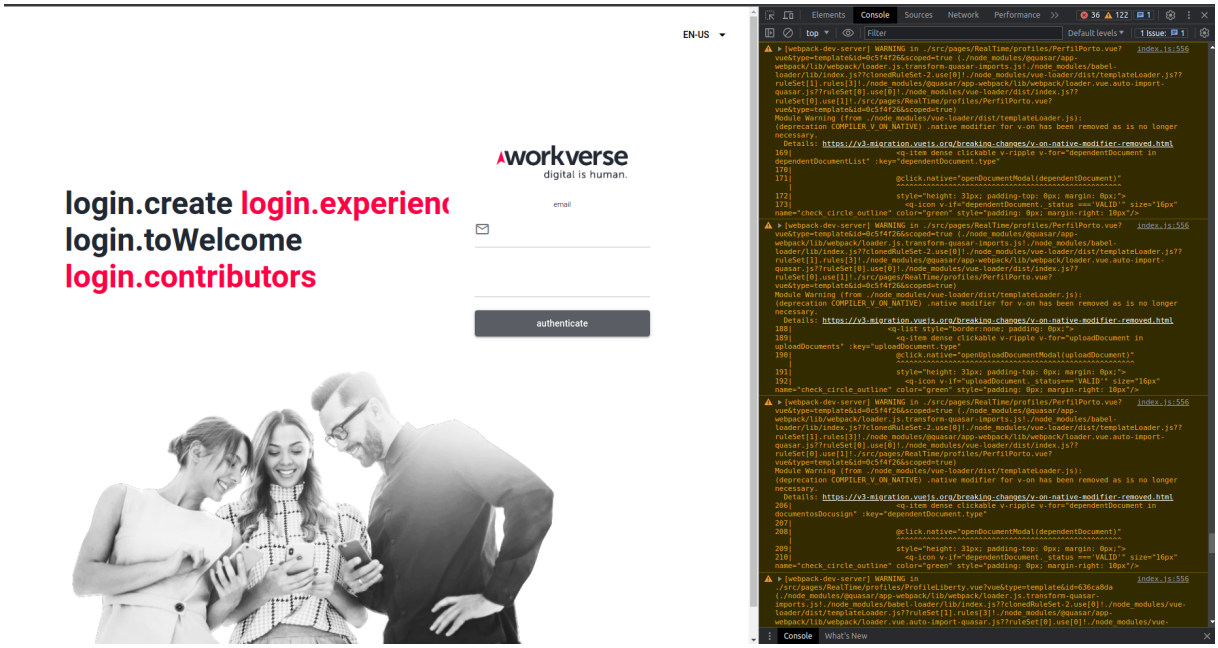


**Figura 17: Captura de tela mostrando a aplicação executando com a Migration Build**

A figura 17 mostra a primeira execução da plataforma utilizando a versão 3 do Vue.js, e, como é possível observar, há vários *warnings* no console emitidos pela biblioteca de compatibilidade, informando as coisas que não funcionam mais ou que devem ser atualizadas para oferecer a funcionalidade desejada. Além disso, nesta imagem, nota-se um erro nos textos, e isso acontece porque mesmo que esteja fazendo a compatibilidade de versões, a plataforma utiliza outras bibliotecas, como a i18n, responsável pela internacionalização dos textos, e a versão utilizada não é mais compatível com a nova versão do Vue.js, portanto, também deve ser atualizada. Além desta, também há outras bibliotecas a serem corrigidas, como a Vuelidate, responsável pela validação de formulários na plataforma.

### 4.3.3 Removendo a Migration Build e realizando os testes finais

Após realizar as correções que foram sugeridas nas mensagens de *warnings*, e também atualizar as bibliotecas que estavam com conflitos de versão, a biblioteca de compatibilidade foi desinstalada, para agora, executar a plataforma apenas utilizando o Vue.js 3, e começar a próxima etapa que seria o mapeamento, página a página, corrigindo os conflitos restantes e testando os casos de uso. Após a desinstalação da biblioteca, a plataforma continuou executando, e após alguns ajustes na tela de login, foi possível entrar e acessar a página principal, como mostra a figura 18.

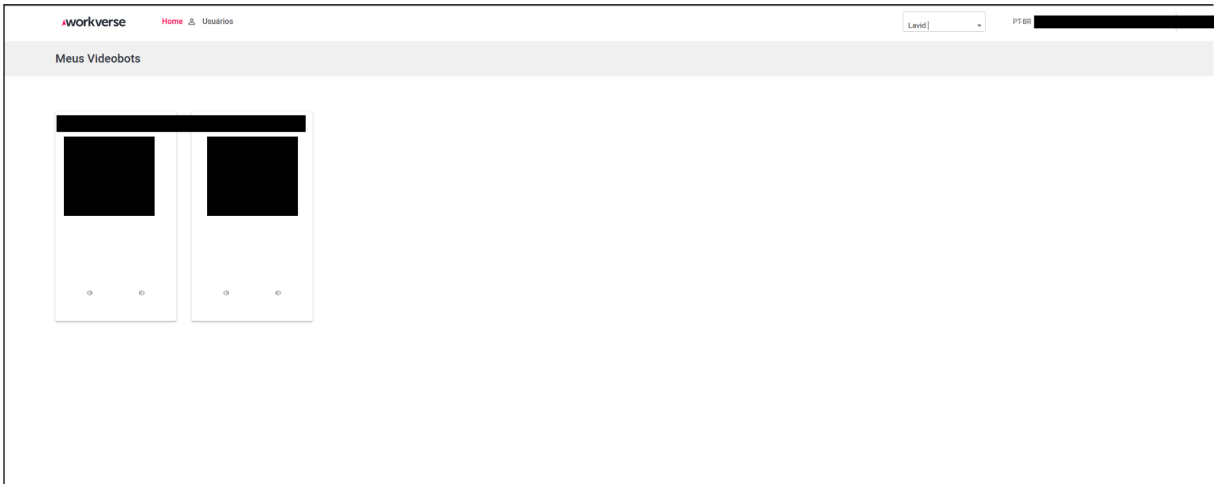


**Figura 18: Plataforma sendo executada com Vue.js 3 sem a Migration Build**

Com acesso ao menu principal, ficou mais fácil fazer o mapeamento das páginas, acessando diretamente pelos botões da interface. Apesar da Migration Build ter auxiliado nas correções principais entre as versões, ela apenas exibe os erros em tempo de compilação, então ainda sim, é necessário realizar as correções restantes da nova versão, pois mesmo que a aplicação esteja rodando, muitas funcionalidades não estão funcionando como antes. Um exemplo que pode ser citado nesta fase, é que, nesta tela principal onde é possível trocar a campanha, se faz uso do *event bus* do Vue.js, que consiste em um mecanismo de eventos, onde um componente como o layout aciona um evento na página de vídeos, e esta funcionalidade mudou na nova versão, então foi necessário realizar os ajustes sugeridos pela documentação do Vue.js, onde foi necessário instalar uma nova biblioteca responsável por emitir eventos, a Mitt. Além disso, a maior parte dos erros que foram verificados ocorreram devido às outras dependências utilizadas, como o Quasar e Vuelidate, também sendo necessário realizar os ajustes de migração.

As ilustrações abaixo mostram um exemplo de duas páginas que já estavam funcionando em Vue.js 3, porém ainda não apresentaram o comportamento nem a estilização totalmente como esperado.

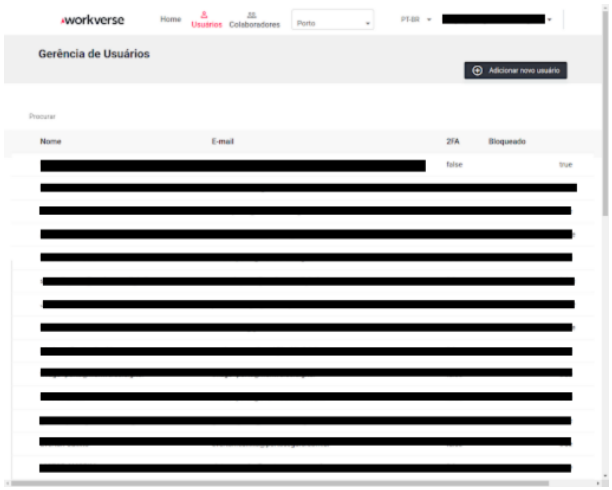


**Figura 19: Página de gerência de usuários com erros de exibição**

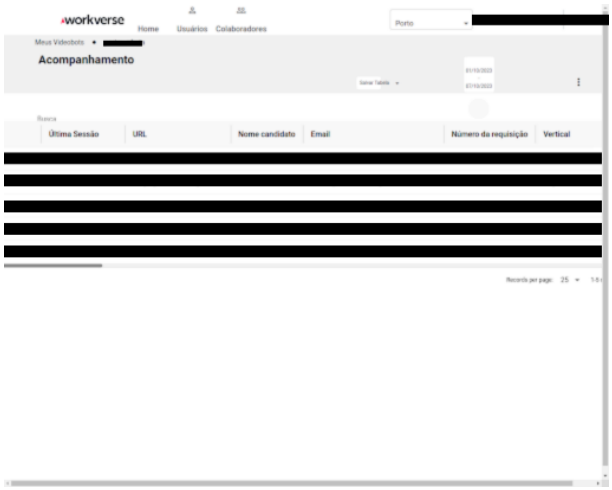


**Figura 20: Página de acompanhamento com erros de exibição e estilização**

A partir de então, foram sendo realizadas as correções seguindo a ordem de páginas descritas na seção de atualização do Quasar. As mudanças em cada página serão descritas no apêndice B de migração do Vue.js. Dentre as principais mudanças, podem ser destacadas:

- Correções na importação e utilização dos componentes do Vuelidate;
- Alterações na utilização de slots em componentes do Quasar, como QTable;
- Ajustes na utilização do v-if e v-for em mesmos componentes, fazendo necessário uso de filtros computados;
- Ajustes nos componentes atualizados do Quasar;

Além disso, também foi necessário realizar ajustes em algumas configurações globais, como a definição das cores de estilização utilizadas em toda a aplicação, e o carregamento dos ícones. Estes passos também estão descritos no apêndice de migração.

### 4.3.4 Migrar de Options API para Composition API

Como mencionado anteriormente, essa etapa foi inicialmente considerada opcional. Com isso, após algumas reuniões de tomada de decisão com membros da empresa, foi decidido que a abordagem mais eficaz seria manter os componentes existentes usando a mesma API. Em vez de refatorar todos os componentes existentes, a estratégia escolhida foi escrever novos componentes, à medida que fossem necessários, usando a API mais recente.

Essa decisão visa garantir a contínua evolução da aplicação. À medida que novas demandas, páginas e casos de uso surjam, esses novos componentes serão desenvolvidos e implementados com a versão mais recente, a Composition API, garantindo a flexibilidade e adaptabilidade da aplicação a novos requisitos e funcionalidades.

## 4.4 Avaliação

A primeira avaliação realizada foi baseada no esquema navegacional, pois foi necessário verificar que toda a aplicação estava com sua integridade mantida após a atualização, e de preferência, quase idêntica antes do processo ocorrer. As duas imagens abaixo mostram a comparação das capturas de tela das principais páginas antes do processo de atualização, e depois da última etapa, que foi quando a aplicação estava totalmente funcional em Vue.js 3.0. Como observado, a atualização foi bem sucedida, pois funcionou de forma semelhante, com suas funcionalidades mantidas e com os frameworks e bibliotecas atualizados.

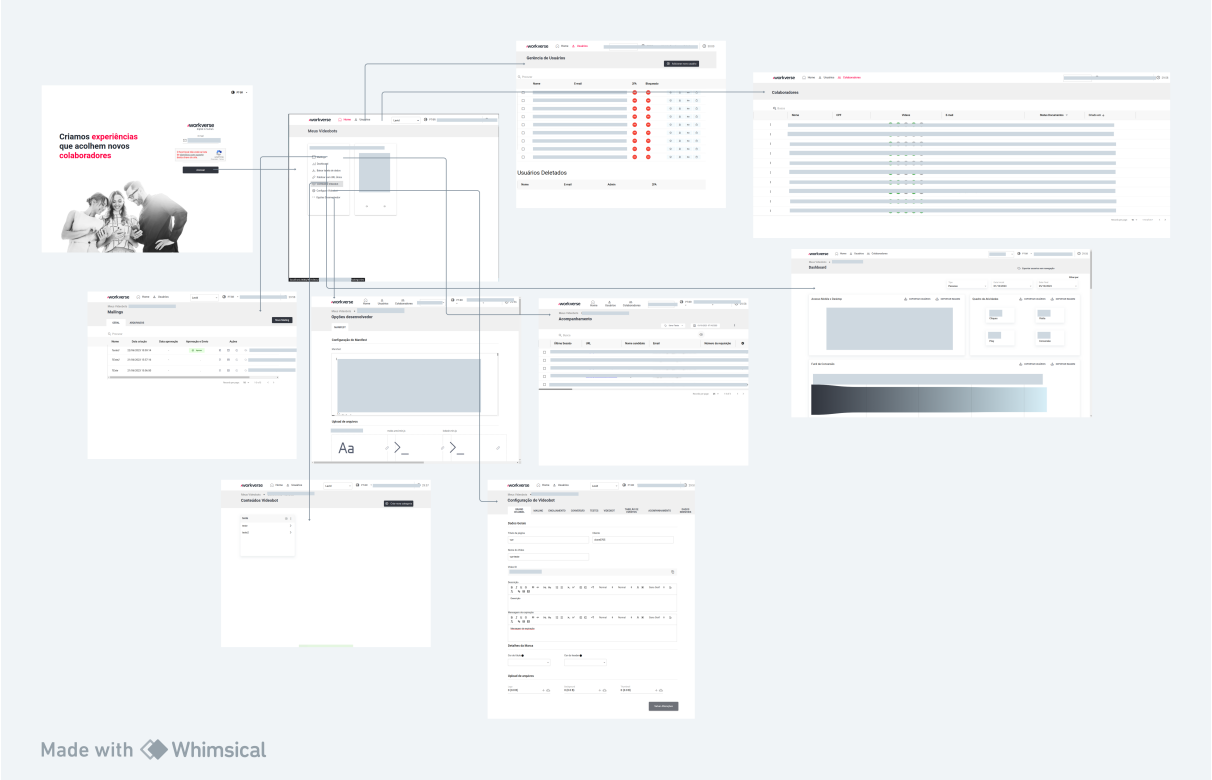


**Figura 21: Mapa de navegação com as páginas antes da atualização do Vue**

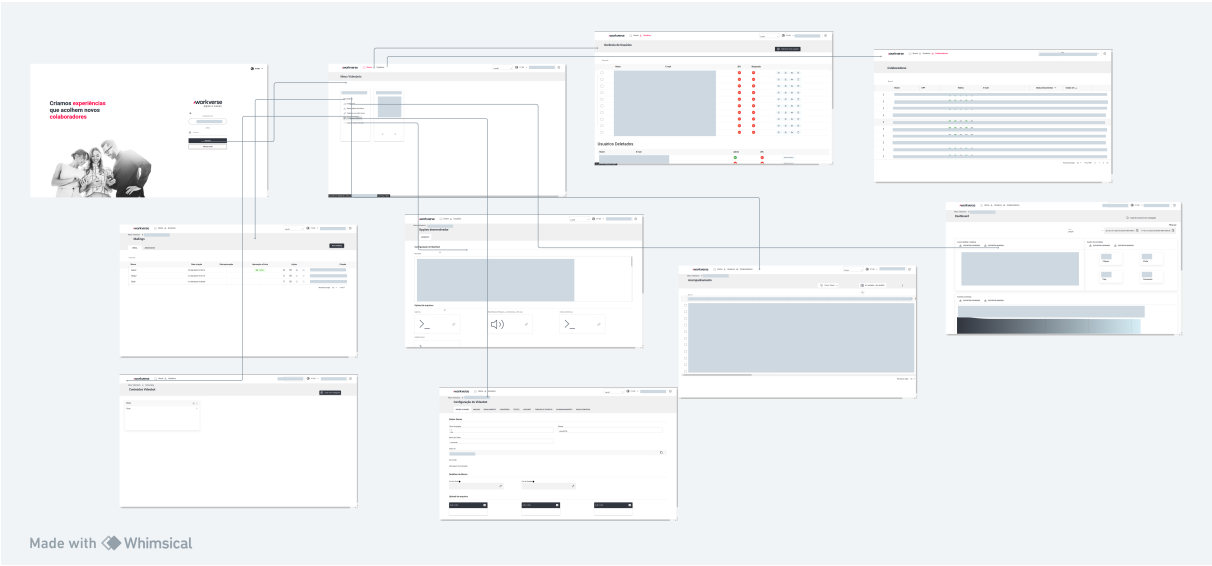


**Figura 22: Mapa de navegação com as páginas após atualização do Vue**

Para ver em mais detalhes, é possível acessar o mapa de navegação antes da atualização por este link: <https://drive.google.com/file/d/1Eb2YVjuximhT9WW3El4cI6cbXuXVhnSd/view?usp=sharing>. O mapa de navegação após a atualização se encontra neste outro link: <https://drive.google.com/file/d/1cF9WlLVDSm2zL0cDFxtb44Ip8049QgRs/view?usp=sharing>.

Outro ponto avaliado foi em relação aos testes referentes às principais funcionalidades da plataforma. A figura 20 mostra a tabela que foi utilizada para o acompanhamento e avaliação de cada etapa do processo, e como mostram os resultados, praticamente todos os casos de uso foram mantidos, com o funcionamento desejado, com exceção da funcionalidade da página Dashboard, que precisou da atualização de alguns componentes referentes aos gráficos mostrados, mas a maior parte da página continuou funcionando, por isso o preenchimento com "CORREÇÕES" na planilha exibida na Figura 23.

| Caso de uso | Página | Importância | Quasar v1 | Migration Build | Vue 3 |
|---|---|---|---|---|---|
| Realizar login padrão | Login | ALTA | OK | OK | OK |
| Realizar login two factor | Login | MÉDIA | OK | OK | OK |
| Realizar login SSO | Login | MÉDIA | OK | OK | OK |
| Visualizar vídeos do cliente | Videos | ALTA | OK | OK | OK |
| Exibir opções disponíveis para um video | Videos | ALTA | OK | OK | OK |
| Permitir configurar a brand channel | Config | ALTA | OK | OK | OK |
| Permitir configurar o envio de mailings | Config | ALTA | OK | OK | OK |
| Permitir configurar os eventos para acompanhamento | Config | MÉDIA | OK | OK | OK |
| Permitir configurar os dados de conversão | Config | MÉDIA | OK | OK | OK |
| Permitir configurar os dados de engajamento | Config | MÉDIA | OK | OK | OK |
| Permitir configurar os dados sensíveis | Config | ALTA | OK | OK | OK |
| Exibir os dados de acompanhamento de um vídeo | Realtime | ALTA | OK | OK | OK |
| Alterar o espaço de tempo dos dados exibidos | Realtime | MÉDIA | OK | OK | OK |
| Permitir acessar as telas de perfil para alguns clientes | Realtime | ALTA | OK | OK | OK |
| Gerenciar os conteúdos de um vídeo | Categories | ALTA | OK | OK | OK |
| Fazer uploads de conteúdo de vídeo, imagens, etc | Categories | ALTA | OK | OK | OK |
| Gerenciar os mailings de um videobot | Mailings | ALTA | OK | OK | OK |
| Permitir o download e upload de templates | Mailings | MÉDIA | OK | OK | OK |
| Realizar envio de email e sms | Mailings | ALTA | OK | OK | OK |
| Realizar o agendamento de disparo de mailings | MailingSchedule | MÉDIA | OK | OK | OK |
| Exibir dashboard interativo sobre um videobot | Dashboard | BAIXA | OK | OK | CORREÇÕES |
| Permitir o upload de assets para serem utilizados no videobot | DevOptions | BAIXA | OK | OK | OK |
| Permitir a visualização do último deploy e interatividades | DevOptions | BAIXA | OK | OK | OK |
| Gerenciar os usuários de um cliente (empresa) | Users | ALTA | OK | OK | OK |
| Gerenciar permissões destes usuários | Users | ALTA | OK | OK | OK |
| Gerenciar os colaboradores de uma empresa | Collaborators | MÉDIA | OK | OK | OK |

**Figura 23: Tabela preenchida com as avaliações dos testes das funcionalidades**

Por fim, também foi avaliado a questão dos prazos para que os objetivos fossem atingidos. A figura 24 mostra os dados referentes aos prazos esperados e ao tempo de duração de cada etapa do processo de migração. Foi possível perceber que as estimativas chegaram perto do tempo real, porém ainda sim, algumas fases necessitaram de um pouco mais de esforço para serem concluídas.

| Data de inicio | Etapa do processo | Resultado alcançado | Data de finalização | Tempo esperado | Tempo total |
|---|---|---|---|---|---|
| 01/06/2023 | Estudo sobre o desenvolvimento de aplicações Web e o uso de frameworks | Uma base teórica e técnica para dar inicio ao processo de atualização | 21/06/2023 | 21 dias | 21 dias |
| 22/06/2023 | Análise da plataforma e planejamento do processo de atualização | Análise arquitetural da plataforma e etapas da atualização definidas | 30/06/2023 | 6 dias | 8 dias |
| 03/07/2023 | Atualização intermediária do Quasar v0.7 para v1 | Migração concluída com sucesso para Quasar v1 | 09/08/2023 | 30 dias | 36 dias |
| 23/08/2023 | Atualização da versão do Vue.js e Quasar utilizando a Migration Build | Aplicação configurada e identificação de erros críticos do Vue 3 | 27/08/2023 | 7 dias | 4 dias |
| 28/08/2023 | Ajustes com auxílio da Migration Build | Aplicação funcional com Vue 3, mas com erros em tempo de execução | 09/09/2023 | 14 dias | 13 dias |
| 16/09/2023 | Remoção da MIgration Build e testes finais | Aplicação totalmente operacional com Vue 3 | 27/10/2023 | 40 dias | 41 dias |

**Figura 24: Tabela com a duração de cada etapa do processo**

### 4.4.1 Refatorações mapeadas

Nesta seção, são apresentadas as refatorações mapeadas para aprimorar a estrutura e a manutenção do código do projeto.

- Remoção de páginas não utilizadas: Foi identificado que certas páginas, tais como "Posts"e "Reports,"não estão sendo utilizadas por nenhum cliente. Portanto, é recomendado remover essas páginas, a fim de reduzir a complexidade e o tamanho do projeto;
- Organização da componentização e dos diretórios: A estrutura de componentização e organização de diretórios pode ser otimizada para melhorar a manutenção e a escalabilidade do projeto. Algumas recomendações incluem:
  - Reorganização de componentes relacionados: Os componentes "UsersToExclude"e "SessionsToExclude"devem ser incorporados como subcomponentes dentro do bloco "Users,"enquanto "CollaboratorStatistics"deve ser colocado como um subcomponente dentro do bloco "Collaborators."
  - Desvinculação de componentes desnecessários: No bloco de dashboard, há um componente sendo importado diretamente do bloco "Reports,"que não está mais em uso. Recomenda-se mover esse componente para a pasta "Dashboard"para manter uma estrutura limpa e organizada.
  - Refatoração do componente raiz da página Realtime: O componente raiz da página "Realtime"(RealTime.vue) atualmente possui aproximadamente 1500 linhas de código, o que dificulta a manutenção e a evolução. É aconselhável dividir esse componente em subcomponentes mais gerenciáveis.
- Revisão da pasta "Components": A pasta "components"localizada no diretório raiz (src) deve ser revisada. Atualmente, ela contém apenas 6 componentes que são usados por páginas específicas, cada uma delas já possuindo sua própria pasta "components."Sugere-se que os componentes da pasta raiz sejam movidos para diretórios locais correspondentes às páginas ou, se necessário, manter a pasta raiz apenas para componentes compartilhados por várias páginas.
- Adoção da Composition API: Para garantir uma maior escalabilidade e facilitar a manutenção do código, é altamente recomendado que novos componentes e funcionalidades sejam desenvolvidos utilizando a Composition API, conforme foi comunicado durante a última atualização do projeto. Essa abordagem contribuirá significativamente para a clareza e a qualidade do código.

A implementação dessas refatorações propostas contribuirá para uma estrutura mais eficiente e sustentável do projeto, garantindo sua manutenção e expansão contínuas.

### 4.4.2 Principais dificuldades enfretadas

Dentre as principais dificuldades durante este estudo de atualização da aplicação, é possível dar destaque as seguintes atividades:

- Realizar a avaliação global da aplicação e entender como os componentes e páginas se comunicavam: Uma das principais dificuldades encontradas ao longo deste projeto foi a necessidade de compreender profundamente a arquitetura da plataforma Workverse. Isso envolveu a análise minuciosa de como os diversos componentes e páginas que se relacionavam e trocavam informações. Muitas vezes, essa complexa teia de interações não estava devidamente documentada, o que exigiu um esforço adicional na identificação das dependências e fluxos de dados. Compreender a interconectividade de todos os elementos da aplicação era crucial para garantir que a atualização não quebrasse funcionalidades essenciais e proporcionasse uma transição suave;

- Entender o processo de como as bibliotecas interagiam e os erros que estavam sendo gerados pelos conflitos: Outro desafio significativo ao longo do processo de atualização foi lidar com as incompatibilidades e conflitos entre as bibliotecas e frameworks utilizados. À medida que as versões eram atualizadas, novos erros e problemas surgiam, muitas vezes devido a mudanças sutis na forma como as bibliotecas interagiam. Identificar a causa raiz desses erros, bem como encontrar soluções viáveis, exigiu uma profunda compreensão não apenas das bibliotecas em questão, mas também de como elas se integravam na aplicação. Isso implicou horas de pesquisa, depuração e testes para garantir que a atualização não resultasse em funcionalidades defeituosas;

- Estimar os prazos: A incerteza associada a problemas inesperados, complexidade das tarefas e a necessidade de manter a continuidade do negócio tornaram a definição de prazos precisos uma tarefa delicada. Manter um equilíbrio entre a necessidade de concluir o projeto de forma eficiente e a garantia de que a qualidade não fosse comprometida foi desafiador;

### 4.4.3 Boas práticas para outros processos de atualização

Esta seção destaca uma série de práticas valiosas que podem ser aplicadas durante a atualização de outras aplicações e frameworks, proporcionando uma abordagem mais eficiente e estruturada.

- Divisão em etapas - Uma prática comum, adotada não apenas neste estudo, mas também em outras atualizações de frameworks, é a divisão do processo em duas

etapas distintas: ajustes nos arquivos de configuração seguidos pela iteração por todas as páginas, verificando as mudanças da nova versão.

- Iteração entre os componentes com base na navegação - A sequência da atualização, neste trabalho, foi determinada estrategicamente com base no esquema navegacional. Iniciar pelas telas de acesso prioritário do usuário simplifica os testes, especialmente os de sistema.

- Atualizações de estilo ao final do processo - Uma decisão prática adotada aqui, que pode ser uma diretriz útil para outros processos, é realizar correções de estilo apenas no estágio final. Essa abordagem evita retrabalhos frequentes, concentrando-se inicialmente em ajustes essenciais para o funcionamento e, em seguida, dedicando uma atenção mais detalhada à aparência.

- Documentação de dependências - Um ponto crucial é documentar minuciosamente as dependências da aplicação. Esta prática simplifica os próximos processos de atualização, uma vez que uma das maiores complexidades reside na compreensão das conexões entre as dependências.

# 5 CONCLUSÕES E TRABALHOS FUTUROS

O presente estudo abordou os desafios enfrentados durante o processo de atualização de aplicações web, em um cenário de constante evolução tecnológica. Ao longo deste processo, foram identificados vários obstáculos, incluindo a complexidade da atualização de sistemas de médio e grande porte e a necessidade de garantir compatibilidade com as mais recentes inovações no desenvolvimento desses softwares.

Com o avanço neste estudo, os passos planejados mostraram ser efetivos, pois ao fim de todo o processo, a atualização da plataforma foi bem sucedida, e isso permitou que ela permanecesse atualizada, além de abrir caminho para a adoção de novas funcionalidades e melhorias.

Ademais, com este trabalho, foi possível obter uma compreensão mais aprofundada das aplicações Web atuais e da evolução das tecnologias relacionadas. As descobertas evidenciam a importância de manter os sistemas atualizados para atender às crescentes demandas dos usuários e permanecer competitivo no mercado. Também é importante mencionar que as lições aprendidas durante todo o processo podem servir como um guia para organizações que enfrentam desafios semelhantes e desejam otimizar seus sistemas.

Dentre os principais aprendizados obtidos com este estudo, pode-se destacar:

- Compreensão aprofundada das tecnologias utilizadas: O estudo destacou a importância de uma compreensão aprofundada das tecnologias utilizadas em uma aplicação. Isso inclui não apenas o conhecimento das versões mais recentes de frameworks, como o Vue.js, mas também a compreensão de como essas tecnologias se integram e interagem entre si.

- Planejamento estratégico da atualização: A necessidade de um planejamento estratégico eficaz foi evidente. A atualização de sistemas corporativos não é apenas uma questão técnica, mas também estratégica. A definição de etapas claras, estipulação de prazos e testes de avaliação são essenciais para que tenha sucesso.

- Manutenção da funcionalidade durante atualizações: Garantir a continuidade das funcionalidades críticas durante o processo de atualização foi ressaltado como uma prioridade. A implementação de estratégias que minimizem o risco de erros e garantam que os usuários continuem a ter acesso a recursos essenciais é fundamental para o sucesso do processo.

- Aprendizado contínuo e flexibilidade: Por fim, a necessidade de uma mentalidade de aprendizado contínuo e flexibilidade foi evidente. Em um ambiente tecnológico em constante evolução, a capacidade de se adaptar a novos desafios, aprender com experiências anteriores e ajustar estratégias conforme necessário é crucial.

Como trabalhos futuros sugere-se:

1. Aplicação das sugestões de refatoração na plataforma: Uma extensão natural deste estudo seria a aplicação prática das refatorações mapeadas, realizando o planejamento com o líder de desenvolvimento e implementando os pontos identificados.

2. Manutenção contínua da plataforma com bibliotecas atualizadas: Recomenda-se a continuidade da prática de manter a plataforma Workverse com as versões de bibliotecas e frameworks mais recentes. Isso não apenas assegura a utilização das últimas funcionalidades e melhorias de segurança, mas também estabelece uma cultura de manutenção contínua que pode ser essencial para a longevidade do sistema.

3. Generalização do modelo de atualização para outros frameworks: Este trabalho pode servir como uma base sólida para estudos similares em outros frameworks populares, como Angular ou React. A adaptação das estratégias e lições aprendidas aqui para diferentes contextos tecnológicos pode enriquecer a compreensão geral das práticas de atualização em ambientes corporativos.

# REFERÊNCIAS


[1] YOU, Evan. Documentação oficial do Vue.js. Disponível em: <https://vuejs.org/>. Acesso em: 20 jul. 2023.

[2] MACORATTI, José Carlos. **Padrões de Projeto: Design Patterns**. 15 mai. 2002. Disponível em: <https://www.macoratti.net/vbn_mvc.htm>. Acesso em: 05 set. 2023.

[3] MACORATTI, José Carlos. **Padrões de Projeto: Design Patterns - MVP**. 15 mai. 2002. Disponível em: <https://www.macoratti.net/10/07/vbn_mvp.htm>. Acesso em: 05 set. 2023.

[4] LEUSCHKE, Alex. **Majesty of Vue**. Packt Publishing, 2015.

[5] WOHLGETHAN, E. **Supporting Web Development Decisions by Comparing Three Major JavaScript Frameworks: Angular, React and Vue.js**. Bachelor's Thesis — IT department, Haw Hamburg, 2018.

[6] YOU, Evan. First Week of Launching an OSS Project. Evan You's Blog, 2014, fevereiro 11. Disponível em: <https://blog.evanyou.me/2014/02/11/first-week-of-launching-an-oss-project/>. Acesso em: 10 ago. 2023.

[7] PRESSMAN, Roger; LOWE, David. **Web Engineering: A practioner's approach**. McGraw-Hill, Inc., 2008.

[8] SCOTT, Emmit. **SPA Design and Architecture: Understanding single-page Web applications**. Shelter Island: Manning Publications Co., 2016.

[9] DUARTE, N. F. B. **Frameworks e Bibliotecas Javascript**. Tese (Doutorado) — Instituto Superior De Engenharia Do Porto, 2015.

[10] R. JOHNSON; B. FOOTE. **Designing Reusable Classes**. **Journal of Object-Oriented Programming**, vol. 1, pp. 22–35, June/July 1988.

[11] QUASAR. Documentação oficial do Quasar. Disponível em: <https://quasar.dev/introduction-to-quasar>. Acesso em: 29 ago. 2023.

[12] MICROSOFT. **The MVVM Pattern**. US: Microsoft, 2012. Disponível em: <https://docs.microsoft.com/en-us/previous-versions/msp-n-p/hh848246(v=pandp.10)>. Acesso em: 05 set. 2023.

[13] MCCARTY, D. What Are the Biggest Problems with Legacy Software? 2020. <https://www.gavant.com/library/what-are-the-biggest-problems-with-legacy-software>. Acesso em: 04 ago. 2023.

[14] FORBES TECH COUNCIL. **10 Challenges To Think About When Upgrading From Legacy Systems**. Forbes, 1 de junho de 2018. Disponível em: <https://www.forbes.com/sites/forbestechcouncil/2018/06/01/10-challenges-to-think-about-when-upgrading-from-legacy-systems/?sh=69e083511c0d>. Acesso em: 13 nov. 2023.

# APÊNDICE A - ATUALIZAÇÃO QUASAR V1

Alguns passos de configuração devem ser realizados para iniciar a migração de versão. Os passos são os seguintes:

- Remover a pasta do *quasar-cli*;
- Remover pastas *.quasar, node modules* e *package-lock.json* ou *yarn.lock file*;
- Instalar: *quasar* e *@quasar/extras* como dependência;
- Instalar: *@quasar/app* como dependência de desenvolvimento;
- Reinstalar todos os pacotes *npm*;
- Remover os arquivos *.babelrc* e criar um novo arquivo chamado *babel.config.js*;
- Atualizar o arquivo *babel.config.js* com o código: *module.exports = presets: [ '@quasar/babel-preset-app' ]*

No arquivo *quasar.conf.js*:

- Renomear a pasta "*src/plugins*"para "*src/boot*";
- Renomear a seção *plugins* para *boot*;
- Renomear o valor *fontawesome* para "fontawesome-v6", "mdi"para "mdi-v6"e "ionicons"para "ionicons-v4"dentro da chave *extras*, se são utilizados;
- Dentro do caminho "framework ¿ iconSet"fazer as mesmas renomeações ("fontawesome"para "fontawesome-v6", "mdi"para "mdi-v6"e "ionicons"para "ionicons-v4");
- Renomear "framework ¿ i18n"para "lang";
- Remover todas as referências a "ctx.theme";
- Criar um novo arquivo chamado "quasar.variables.styl"(ou ".sass", ".scss") na pasta " /src/css", caso ainda não exista, e adicionar as cores primárias e secundárias.;

A partir dessas alterações, é necessário ajustar cada página, fazendo as atualizações dos componentes. Algumas páginas não foram mencionadas por não terem ajustes significativos com a atualização.

**Layout: App Layout**

- QLayoutHeader e QLayoutFooter renomeados para QHeader e QFooter, respectivamente;
- QLayoutHeader e QLayoutFooter teve a remoção do atributo *color*;
- QSelect: Ajuste na filtragem dos dados, agora deve se passar uma função no "v-on:filter"e adição de um *slot* para exibir o item selecionado;
- QList: Adição de *padding* necessária;

**Layout: TableModal**

- QInput: Adicionar "type='password'"nos campos de senha;

**Página: Login**

- QField: Atualizado completamente. Não é mais utilizado para encapsular QInput ou QSelect, pois estes componentes já possuem as funcionlidades dentro deles;
- QInput: Atributo before passa a ser "v-slot:prepend";
- QInput: Adicionar "type='password'"nos campos de senha;

**Página: Vídeos**

- QModal: Substituido pelo QDialog;
- Adicionar QCard dentro do QDialog para aplicar a estilização que antes era feita pelo "content-css";
- QDatetime removido, deve ser utilizado QDate e QTime;

Refatorações possíveis:

- Remover ConversionConfigModal, não está mais sendo usado, "conversionConfig"está sendo usando na página de configurações;
- Remover ConfigEngagementModal, configuração de engagemento foi adicionanda na página de configurações;
- CreateVideobotForm e StatusVideobotModal não estão sendo usados no momento;

**Página: Users**

- QSearch: removido, deve ser utilizado QInput com a propriedade *debounce*;

- Substituição da classe ".q-display-1"para ".text-h4";
- QModal: Substituido pelo QDialog;
- Adicionar QCard dentro do QDialog para aplicar a estilização que antes era feita pelo "content-css";
- QField retirado;
- QInput: Substituição de "stack-label='label'"para "stack-label label='label'";
- Refatorações possíveis:
  - ModifyVideosModal não está sendo usado;

**Página: Categories**

- Q-popover: Q-menu;
- QItemMain: QItemLabel;
- Q-Modal: Q-DIalog;
- QUploader:
  - Atríbuto *extensions* é substituido por *accept*;
  - Atributo "url-factory"não existe mais, apenas *factory*, e na função retorna uma chave *url*;
  - Definir *header* para o "content-type";

Refatoração: Páginas AddContent e EditContent removidas por não estarem sendo utilizadas

**Página: Dashboard**

- QCardTitle: removido, utilizar QCardSection de QCard;
- QCardMain: removido, utilizar QCardSect;
- QCardSeparator: removido, utilizar QSeparator;
- QDatetime: removido, utilizar QDate and QTime;

# APÊNDICE B - ATUALIZAÇÃO PARA VUE.JS 3.0

## 1- Instalando a Migration Build

O primeiro passo na atualização do Vue.js foi instalar a nova versão, juntamente com a biblioteca de compatibilidade Migration Build, então a primeira tentativa foi instalar essas bibliotecas e tentar rodar a aplicação. As versões instaladas foram:

- "vue: 3.3.4"
- "@vue/compat: 3.3.4"

Entretanto, essa tentativa foi falha, pois a plataforma fazia uso de outras bibliotecas como Quasar, Vuelidate e i18n. Para chegar perto de subir a aplicação, a primeira ideia foi comentar os trechos destas outras bibliotecas que estavam dando erro , porém também não deu certo, pois elas estavam sendo usadas por vários componentes, então foi decidido atualizar também as outras bibliotecas da plataforma.

- "quasar: 2.12.5";
- "vue-router: 4.2.4";
- "vuex: 4.1.0";
- "@vuelidate/core: 2.0.3";
- "@vuelidade/validators: 2.0.4";

Após estas atualizações, foi necessário fazer os ajustes nos arquivos de configurações que não será entrado em detalhes pelo estudo ter foco apenas nas atualizações do Vue.js. Porém, as alterações foram seguidas de acordo com a documentação oficial de cada biblioteca.

Quando realizados os principais ajustes, principalmente no arquivo de configuração do Quasar, o quasar.conf, foi possível subir a aplicação e iniciar a próxima etapa. A principal mudança neste arquivo foi a adição do seguinte trecho de código dentro do bloco *build* mostrado na figura 25.


**Figura 25: Código que deve ser adicionado ao arquivo quasar.conf para fazer o carregamento em modo de compatibilidade**

Esse código é responsável por carregar o Vue.js utilizando como opção de compilação a compatibilidade com a versão 2 do *framework*. Após isso, foi possível perceber a aplicação mostrando os erros e ajustes críticos a serem feitos.

### 2- Obtendo ajuda da Migration Build

Como descrito anteriormente, após algumas atualizações de bibliotecas e ajustes nos principais arquivos de configuração, foi possível compilar a aplicação e obter os erros indicados da Migration Build.

Primeiramente, essa biblioteca indicou os erros críticos, que impedem a aplicação de compilar totalmente. Entre eles, pode-se destacar:

- Ajuste no atributo v-model
  - Alterações na nomenclatura:
    - "value"− > "modelValue";
    - "input"− > "update:modelValue";
- Remoção do "v-bind.sync", deve-se utilizar apenas o "v-model";
- Pode ser criado novos modificadores "v-model"customizados;
- É possível ter múltiplos "v-model";

- Alteração na sintaxe dos *named* e *scoped slots*;
- Remoção de filtros, agora deve-se usar métodos ou propriedades *computed*;
- Remoção de algumas funcionalidades nativas de emissão de eventos, fazendo-se necessário utilizar uma nova biblioteca, como a *mitt*;
- Alteração na precedência entre "v-for"e "v-if";
  - Antes o "v-for"tinha precedência sobre o "v-if";
  - Agora no Vue.js 3, o "v-if"sempre tem maior precedência;

**3- Remoção da Migration Build e avanço por páginas**

Após corrigir os erros críticos e avisos apontados pela biblioteca de compatibilidade, foi possível removê-la e conseguir rodar a aplicação em Vue.js 3, porém ainda sim, há alguns erros de tempo de execução que só podem ser notados durante a navegação em cada página, além de funcionalidades que não estão realizando as ações como antes, então se fez necessário um mapeamento por cada página, corrigindo os erros de cada uma.

**Login**

Na tela de login, foi necessário ajustar a utilização do Vuelidate, pois ainda estava fazendo a validação dos campos na sintaxe da versão anterior, então foi feito os ajustes necessários, e também removido temporariamente o Captcha, pois a biblioteca estava depreciada, então vai ser necessário fazer uso de uma nova versão.

O principal ajuste no Vuelidate foram as importações dos utilitários por outro pacote, no caso:

- Antes: "import  required, email  from "vuelidate/lib/validators";"
- Depois: "import  required, email  from "@vuelidate/validators"; import useVuelidate from "@vuelidate/core";"

Além de ter que iniciar uma instância local do useVuelidate, lembrando que na nova versão do Vue, cada criação de instância local se refere a mesma configuração da instância global. A linha de código a ser adicionada é "v$ = useVuelidate()"

Após isso, em cada *input*, deve ser feita a substituição da utilização dessa instância, como mostra a figura 26 abaixo:

```
// ANTES
<q-input
       v-model="email"
       :bottom-slots="true"
       :error="$v.email.$error"
       :error-message='$t("errorMessages.unvalidEmail")'
       type="email"
       @blur="$v.email.$touch"
       @click="emailError = false"
>
// DEPOIS
<q-input
       v-model="email"
       :bottom-slots="true"
       :error="v$.email.$error"
       :error-message='$t("errorMessages.unvalidEmail")'
       type="email"
       @blur="v$.email.$touch"
       @click="emailError = false"
>
```

**Figura 26: Correções do vuelidade no QInput**

### Videos

Tendo corrigido a tela de *login*, foi possível acessar a tela principal da plataforma que é a de vídeos, e nesta tela, não teve muitos problemas, então foi possível obter as funcionalidades desejadas.

### Config

Por meio da tela de vídeos, foi possível acessar a tela de configurações, que necessitou de alguns ajustes de validação, assim como na tela de *login*, como o uso do *useVuelidate* dentro do arquivo principal da página, depois foram sendo testados os casos de uso, e a principal diferença foi principalmente em estilização, mas isso se dá devido a atualização do Quasar

### Realtime

Esta página não apresentou erros críticos, porém não estava funcionando devido a uma mudança no *framework* Quasar, que foi a alteração do atributo "data" para "rows", e sem essa alteração, os dados não eram carregados na tabela principal de acompanhamento, então após corrigir isso, já foi possível verificar o principal caso de uso que é ter acesso a tela de acompanhamento, porém ainda faltava ajustar e reposicionar elementos que ficaram diferentes devido as atualizações do Quasar

### Categories

Neste conjunto de páginas, a principal mudança necessária para ter as funcionalidades corretamente em Vue.js 3 foi a correção no sistema de comunicação e eventos entre os componentes pai e filhos, que é o caso do menu de adicionar ou editar conteúdo (componentes pais) e os formulários individuais (componentes filhos), como preenchimento de texto ou *upload* de imagens.

As duas imagens abaixo mostram as alterações que devem ser feitas no componente filho e no *mixin* responsável por fazer o *submit*, respectivamente.

```
//antes
this.$on('close-modal2', () => {
this.$emit('close-modal')
})

//depois
this.$root.emitter.on('close-modal2', () => {
this.$emit('close-modal')
})
```

**Figura 27: Modificação a ser feita no componente filho**

```
//antes
this.$emit("close-modal2");
//depois
this.$root.emitter.emit("close-modal2");
```

**Figura 28: Modificação a ser feita no Mixin**

Além disso, também foi necessário fazer os ajustes mencionados anteriormente nos campos de formulários que utilizavam Vuelidate, passando a criar uma instância no bloco de dados do componente: ”v$ useVuelidate()”.

### Mailings

Nesta tela, teve alguns ajustes também relacionados ao Vuelidate como nas páginas anteriores, porém os ajustes mais críticos eram em relação ao Quasar, com mudanças no funcionamento do componente QTable. As principais mudanças no componente QTable são mostradas na figura 29

```
//ANTES
<q-tr slot="body" slot-scope="props" :props="props">
//DEPOIS
<template v-slot:body="props">
<q-tr :props="props">
```

**Figura 29: Alterações a serem feitas no componente QTable**

### DevOptions

Este componente não teve muitos problemas com a atualização, apenas alguns ajustes referentes ao Quasar, como a mudança na definição do *slot* a ser usado no componente QExpansionItem. No código, deve ser alterado de «template slot=”header»”para «template v-slot:header¿”

### Users

Neste conjunto de telas, foi necessário alterar a forma que estavam sendo definidas as colunas da tabela QTable, utilizada tanto para listar os usuários ativos e desativados. Essa alteração foi a mesma realizada no componente QTable da tabela Mailings.

Dessa forma, encapsulando o ”q-tr”dentro de um *template*, e utilizando ele para definir o *slot* utilizado. Com essa alteração, a listagem voltou a funcionar normalmente.

Além disso, teve as alterações relacionadas ao Vuelidate, como nas páginas anteriores, neste caso, nos momentos em que alguns inputs eram validados.

Depois destas alterações, foi necessária uma mudança no componente Permissions-Modal, responsável por definir as permissões de um usuário específico, neste caso, havia um problema que consiste em utilizar um ”v-if”juntamente com o ”v-for”, como mostra o código abaixo:

```
v-for="(category, categoryName) in userPermissions"
:key="categoryName"
group="accordions"
:label="categories[categoryName]"
:header-class="'cursor-pointer'"
style="border-bottom: solid 1px rgba(0,0,0,0.12);"
v-if="permissionsGranted[categoryName]"
>
```

**Figura 30: Trecho de código com erro na utilização de ”v-for”e ”v-if”**

A documentação recomenda que seja utilizado no ”v-for”já um objeto filtrado, ficando da seguinte forma:

```
// Template //
v-for="(category, categoryName) in filteredPermissions"
:key="categoryName"
group="accordions"
:label="categories[categoryName]"
:header-class="'cursor-pointer'"
style="border-bottom: solid 1px rgba(0,0,0,0.12);"
>

// Script //
filteredPermissions() {
        let auxPermissions = { ...this.userPermissions }
        for (const key in auxPermissions) {
                if (!this.permissionsGranted[key]) {
                        delete auxPermissions[key]
                }
        }
        return auxPermissions;
},
```

**Figura 31: Correção na utilização do "v-for", substituindo o "v-if"por um objeto filtrado**

### Collaborators

Esta página precisou apenas do ajuste nas tabelas QTable como ocorreu na página de usuários, e então ela voltou a funcionar corretamente. Por ser uma página mais simples, o esforço de atualização foi bem menor.

Durante a atualização, foi percebido que o componente CollaboratorsStatistics não está sendo usado no momento, então foi adicionado à sua remoção à lista de refatorações.

### Outras correções

Após os ajustes, algumas coisas precisaram ser modificadas, como as cores de estilização globais e a definição dos ícones.

### Cores de estilização globais

A atualização do Quasar fez com que mudasse a forma como as cores são definidas por meio de variáveis globais, já que antes era feita definido em arquivo "stylus", porém agora, foi necessário criar um arquivo "javascript", e aplicar as definições, criando as variáveis globais, da seguinte forma:

```
import { boot } from 'quasar/wrappers'
import { setCssVar } from 'quasar'


export default boot(() => {
setCssVar('accent', '#9C27B0');
setCssVar('primary', '#31353E');
setCssVar('primarydark', '#2883AD');
setCssVar('primarylight', '#F3FAFD');
setCssVar('secondary', '#26A69A');
setCssVar('tertiary', '#555');
setCssVar('neutral', '#E0E1E2');
setCssVar('positive', '#47BF2C');
setCssVar('negative', '#DD4B4B');
setCssVar('info', '#31353E');
setCssVar('warning', '#F2C037');
setCssVar('gray1', '#F8F8F8');
setCssVar('gray2', '#F0F0F0');
setCssVar('gray3', '#C5C5C5');
setCssVar('gray4', '#797E80');
setCssVar('gray5', '#55585A');
})
```

**Figura 32: Configuração das cores da aplicação**

Esse arquivo deve ser adicionado ao diretório *boot*, definindo as variáveis globais utilizando "setCssVar", passando a *label* da variável, e o valor, que neste caso é a cor que vai ser utilizada. Após isso, deve ser feita uma alteração no arquivo "quasar.conf", adicionando a referência dentro da chave "boot", da seguinte forma:

"boot: ["i18n", "vuelidate", server: false, path: 'brand-colors' ]"

Dessa forma, ao subir a aplicação, é feito o carregamento do i18n, do Vuelidate e do novo arquivo que define as variáveis globais de estilização.

## Ícones

Com as atualizações do Vue e Quasar, os ícones customizados pararam de funcionar, e isso foi devido a mudança no carregamento desses ícones na hora de subir a aplicação.

A correção consistiu em mover os arquivos "icons.woff", "icons.woff2" e "icons.css" para a pasta "css" dentro do diretório "src", e após isso, adicionar no arquivo "quasar.conf" a referência ao arquivo "css"dentro da chave "css", da seguinte forma:

```
module.exports = function(ctx) {
return {
// app plugins (/src/plugins)
boot: ["i18n", "vuelidate", { server: false, path: 'brand-colors' }],
css: ["app.styl", "icons.css"],
```

**Figura 33: Ajustes realizados para corrigir os ícones**