UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

GUILHERME STACIAKI DA LUZ

ATUALIZAÇÃO DO FRAMEWORK RAILS PARA GARANTIA DA EVOLUÇÃO DO SISTEMA DE GESTÃO DE TCC

GUARAPUAVA

2025

GUILHERME STACIAKI DA LUZ

ATUALIZAÇÃO DO FRAMEWORK RAILS PARA GARANTIA DA EVOLUÇÃO DO SISTEMA DE GESTÃO DE TCC

Rails Framework Update to Ensure the Evolution of the Thesis Management System

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Tecnólogo em Tecnologia em Sistemas para Internet do Curso Superior de Tecnologia em Sistemas para Internet da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Diego Marczal

Coorientadora: Profª Drª Renata Luiza Stange

GUARAPUAVA

2025

4.0 Internacional

Esta licença permite compartilhamento, remixe, adaptação e criação a partir do trabalho, mesmo para fins comerciais, desde que sejam atribuídos créditos ao(s) autor(es). Conteúdos elaborados por terceiros, citados e referenciados nesta obra não são cobertos pela licença.

GUILHERME STACIAKI DA LUZ

ATUALIZAÇÃO DO FRAMEWORK RAILS PARA GARANTIA DA EVOLUÇÃO DO SISTEMA DE GESTÃO DE TCC

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Tecnólogo em Tecnologia em Sistemas para Internet do Curso Superior de Tecnologia em Sistemas para Internet da Universidade Tecnológica Federal do Paraná.

Data de aprovação: 03/dezembro/2025

Prof. Renata Luiza Stange Carneiro Gomes
Doutora
Universidade Tecnológica Federal do Paraná

Prof. Diego Marczal
Doutor
Universidade Tecnológica Federal do Paraná

Prof. Kelly Lais Wiggers
Doutora
Universidade Estadual do Centro Oeste

GUARAPUAVA

2025

RESUMO

O SGTCC é uma aplicação web desenvolvida em Ruby on Rails que centraliza e automatiza o gerenciamento dos trabalhos de conclusão de curso do curso de Tecnologia em Sistemas para Internet da UTFPR. Com o objetivo de garantir a segurança, estabilidade e evolução contínua do sistema, este trabalho propõe a atualização do framework Ruby on Rails, eliminando dependências descontinuadas e modernizando a arquitetura de frontend. A metodologia adotada consistiu em uma atualização incremental do framework, migrando do Rails 6 para o Rails 7 e posteriormente para o Rails 8, seguindo as recomendações da comunidade e aplicando práticas de manutenção preventiva. Paralelamente, foi realizada a migração completa da camada de interface, substituindo o ecossistema baseado em Vue.js e Webpacker pelo Hotwire, conjunto de tecnologias nativas do Rails composto por Turbo e Stimulus. O processo incluiu atualização de bibliotecas e dependências, refatoração de componentes e reorganização da estrutura de código, sempre mantendo a cobertura de testes automatizados em 95%. Os resultados alcançados demonstraram a eficácia da abordagem: foram eliminadas 11 gems Ruby e 16 pacotes JavaScript obsoletos, todos os 57 componentes Vue.js foram migrados para a nova arquitetura, o bundle JavaScript foi reduzido de 350KB para 80KB e o tempo de exibição completa das páginas foi reduzido em aproximadamente 70%, eliminando a necessidade de requisições adicionais para renderização no cliente. A modernização garantiu não apenas a conformidade tecnológica e a segurança do sistema, mas também estabeleceu uma base sólida para futuras melhorias, assegurando que o SGTCC continue atendendo de forma eficiente, segura e escalável às demandas institucionais, em consonância com as Leis de Lehman sobre a evolução contínua de sistemas de software.

Palavras-chave: ruby on rails; hotwire; refatoração; modernização de sistemas; manutenção preventiva.

ABSTRACT

The SGTCC is a web application developed in Ruby on Rails that centralizes and automates the management of course completion works for the Internet Systems Technology course at UTFPR. Aiming to ensure the security, stability and continuous evolution of the system, this work proposes the update of the Ruby on Rails framework, eliminating discontinued dependencies and modernizing the frontend architecture. The adopted methodology consisted of an incremental framework update, migrating from Rails 6 to Rails 7 and subsequently to Rails 8, following community recommendations and applying preventive maintenance practices. In parallel, a complete migration of the interface layer was carried out, replacing the ecosystem based on Vue.js and Webpacker with Hotwire, a set of native Rails technologies composed of Turbo and Stimulus. The process included updating libraries and dependencies, refactoring components and reorganizing the code structure, always maintaining automated test coverage at 95%. The results achieved demonstrated the effectiveness of the approach: 11 Ruby gems and 16 obsolete JavaScript packages were eliminated, all 57 Vue.js components were migrated to the new architecture, the JavaScript bundle was reduced from 350KB to 80KB and the complete page display time was reduced by approximately 70%, eliminating the need for additional requests for client-side rendering. The modernization ensured not only the technological compliance and security of the system, but also established a solid foundation for future improvements, ensuring that SGTCC continues to efficiently, securely and scalably meet institutional demands, in line with Lehman's Laws on the continuous evolution of software systems.

Keywords: ruby on rails; hotwire; refactoring; system modernization; preventive maintenance.

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

Siglas

SGTCC	Sistema de Gestão de Trabalho de Conclusão de Curso
TCC	Trabalho de Conclusão de Curso
TCC 1	Trabalho de Conclusão de Curso 1
TCC 2	Trabalho de Conclusão de Curso 2
TSI	Tecnologia em Sistemas para Internet
UTFPR	Universidade Tecnológica Federal do Paraná
UX	User Experience

SUMÁRIO

1 INTRODUÇÃO

O TCC é uma etapa fundamental na conclusão de cursos de graduação, na qual o estudante aplica os conhecimentos adquiridos ao longo da formação em um projeto que integra teoria e prática. No curso de TSI da UTFPR, o TCC envolve o desenvolvimento de soluções tecnológicas que contribuam para o aprimoramento de processos e sistemas reais (COINT, 2023).

Com o objetivo de otimizar o processo de gerenciamento dos trabalhos de conclusão, foi desenvolvido o SGTCC[1], uma aplicação web que centraliza etapas como a entrega, correção e avaliação dos trabalhos, além de automatizar fluxos administrativos, como o agendamento de bancas e a assinatura de termos de compromisso. Essa automação reduziu significativamente a necessidade de processos manuais e documentos impressos, tornando o gerenciamento mais ágil e eficiente.

Entretanto, à medida que o sistema evolui e novas versões de suas dependências são lançadas, torna-se imprescindível realizar manutenções e atualizações periódicas. Essas ações garantem a segurança, a estabilidade e a continuidade do sistema, prevenindo falhas e incompatibilidades decorrentes do uso de tecnologias obsoletas. Considerando que o SGTCC lida com informações acadêmicas sensíveis, como dados de estudantes, orientadores e bancas, o aprimoramento constante da aplicação é essencial para preservar sua confiabilidade e disponibilidade.

Nesse contexto, o presente trabalho teve como objetivo a atualização do SGTCC, desenvolvido em Ruby on Rails[2], abrangendo a modernização de bibliotecas, a refatoração de código e a substituição de componentes descontinuados, como o Webpacker[3], por alternativas mais atuais, como o Hotwire[4]. A execução dessas tarefas segue um conjunto de práticas consolidadas em Engenharia de Software, incluindo o uso de versionamento de código com Git[5] e GitHub[6], ambientes isolados via Docker[7], acompanhamento das tarefas por meio do ClickUp[8] e integração contínua com banco de dados PostgreSQL[9].

Além de garantir a conformidade tecnológica, a atualização também visou aprimorar a experiência dos usuários e a eficiência do sistema, permitindo que ele continue atendendo às demandas institucionais de forma segura, estável e escalável. Dessa forma, o trabalho se alinha às Leis de Lehman, que estabelecem a necessidade de evolução contínua de sistemas de software para que permaneçam úteis e relevantes ao longo do tempo (LEHMAN; BELADY, 1985),

1 Disponível em https://tcc.tsi.pro.br/o-tcc. Acessado em 13 de novembro de 2025.
2 https://rubyonrails.org/
3 https://github.com/rails/webpacker
4 https://hotwired.dev/
5 https://git-scm.com/
6 https://github.com/
7 https://www.docker.com/
8 https://clickup.com/
9 https://www.postgresql.org/

além de incorporar princípios de refatoração e boas práticas de design de software (FOWLER, 2018).

1.1 Objetivos

Nesta seção, são apresentados os objetivos que norteiam o desenvolvimento do trabalho.

1.1.1 Objetivo geral

Atualizar o Framework Ruby on Rails no SGTCC eliminando dependências descontinuadas, de modo a garantir a manutenção, segurança e evolução contínua do sistema.

1.1.2 Objetivos específicos

1. **Atualizar bibliotecas e dependências:** garantir a segurança e a continuidade do sistema por meio da atualização das bibliotecas utilizadas, prevenindo vulnerabilidades e melhorando o desempenho.
2. **Reescrever o JavaScript utilizando Hotwire:** substituir o uso do Webpacker[10], descontinuado, adotando o Hotwire[11] para proporcionar maior eficiência e reduzir a dependência de bibliotecas externas.
3. **Refatorar o código:** aprimorar a estrutura interna do sistema, aplicando princípios de design e padrões de projeto[12] para aumentar a legibilidade, modularidade e facilidade de manutenção.

1.2 Justificativa

O presente trabalho insere-se no campo da Engenharia de Software, especificamente na área de manutenção de sistemas, que engloba ações preventivas, corretivas e perfectivas. A manutenção preventiva visa evitar falhas futuras por meio da atualização de tecnologias e da modernização do código; a corretiva tem como foco a identificação e correção de erros durante o uso do sistema; e a perfectiva está voltada ao aprimoramento contínuo da estrutura e da qualidade do software, assegurando sua evolução e aderência às novas demandas.

Conforme destacado por Sommerville (2011), a refatoração constitui uma forma de manutenção preventiva, pois permite aprimorar a estrutura interna do sistema sem modificar seu

[10] https://github.com/rails/webpacker
[11] https://hotwired.dev/
[12] https://refactoring.guru/design-patterns

comportamento externo, resultando em código mais claro, modular e de fácil evolução. A ausência dessas práticas tende a acarretar impactos significativos, como o aumento da complexidade técnica, a introdução de vulnerabilidades de segurança, a degradação de desempenho e a elevação dos custos de manutenção a longo prazo.

Considerando a relevância do SGTCC no contexto institucional, especialmente no gerenciamento e acompanhamento de informações acadêmicas, a atualização de sua base tecnológica torna-se imprescindível para garantir sua continuidade operacional, segurança e aderência às práticas atuais de desenvolvimento. Assim, este trabalho justifica-se pela necessidade de assegurar a sustentabilidade do sistema por meio da aplicação de boas práticas de engenharia, contemplando atualização de bibliotecas, refatoração do código e substituição de componentes descontinuados.

2 O SGTCC

O SGTCC começou seu desenvolvimento em 2015, com o intuito de transformar a gestão das atividades de TCC do curso de TSI em um processo digital, visando simplificar todo o processo e centralizar as informações e regulamentos em um único sistema (FERREIRA, 2015). Este sistema serviu de base para o desenvolvimento do sistema que é utilizado atualmente para a gestão dos trabalhos de conclusão de curso de TSI, o SGTCC.

Posteriormente, em 2019, foi dado continuidade ao projeto com a adequação do sistema ao processo do TCC de TSI, junto com a incorporação de assinatura eletrônica em documentos, removendo o uso de papel em todo o processo, tornando digital toda a gestão do processo. Também foram aplicadas diversas melhorias nos módulos existentes do sistema, tais como a criação de tipos de usuários, agendamento de defesas com documentos relacionados ao TCC e avaliações. Também foram feitas mudanças na área pública com informações mais relevantes, como trabalhos realizados e histórico de orientações, sendo adicionado também estatísticas para o professor responsável (SILVA, 2019).

Outras contribuições para o projeto, foram realizadas no segundo semestre do ano letivo de 2023, onde os alunos da disciplina de Desenvolvimento para Web 5 do curso de TSI aplicaram alterações em diversas partes do sistema, como a criação de novas funcionalidades, correção de *bugs*, atualizações de bibliotecas e de documentações do projeto. Ao todo, foram 2739 linhas de código adicionadas e 1012 removidas, somando um total de 1623 *commits* e 238 PRs[1].

Ainda em 2023, o sistema recebeu outras melhorias para o seu funcionamento, com destaque na otimização de suas telas. Foram aplicadas técnicas de UX Design focadas na estética e funcionalidade do sistema, tendo foco na revisão da interface gráfica a fim de torná-la mais agradável e uma reorganização de elementos de design para aprimorar a usabilidade e a eficiência do mesmo (LIMA, 2023).

Atualmente o sistema conta com as seguintes áreas:

- Área pública: composta por uma seção disponível para qualquer pessoa na internet, sem a necessidade de autenticação. Nela contém uma breve descrição do termo do TCC e seus objetivos gerais, além de outras informações como bancas de TCC do período corrente, calendário com as atividades necessárias e a listagem de TCCs aprovados.

[1] Uma pull request (PR) é uma proposta para mesclar as alterações de um branch em outro. Em uma pull request, os colaboradores podem revisar e discutir o conjunto de alterações proposto antes de integrá-las à base de código principal. As pull requests exibem as diferenças, ou comparações, entre o conteúdo no branch de origem e aquele no branch de destino. Disponível em https://docs.github.com/pt/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-pull-requests?platform=linux. Acesso em 05 de novembro de 2024.

- Área de membro externo: esta área é disponibilizada para convidados e instituições externas terem acesso às bancas de defesa, das quais fazem parte, e as informações sobre estas, tais como acesso a documentos pendentes como aos documentos já assinados.

- Área acadêmica: nesta área é possível acompanhar todo o avanço e desempenho do discente referente ao desenvolvimento do TCC. Neste módulo, encontram-se as atividades essenciais para o andamento do TCC, juntamente com os documentos relacionados à orientação, como os pendentes de assinatura e os já assinados. Além disso, há informações sobre a banca de defesa, incluindo o local, a data e o horário da apresentação.

- Área do orientador: contém as informações das principais atividades para a continuidade do TCC, como o monitoramento das datas de entregas de cada etapa realizada pelo acadêmico. Ademais, apresenta-se uma seção específica para reuniões, onde é possível registrar informações que ficam disponíveis para o aluno, além de permitir o acesso às informações referentes às bancas das quais o professor orientador é membro avaliador.

- Área do professor da disciplina de TCC 1: nesta área o professor da disciplina de TCC 1 tem acesso a todos os discentes matriculados, podendo agendar bancas de defesa de propostas e projetos, acompanhar as entregas feitas por cada estudantes na disciplina e verificar prazos relacionados ao calendário em andamento.

- Área do responsável pelo TCC: aqui encontra-se o maior número de funcionalidades, uma vez que é possível gerenciar o andamento de processos relacionados ao TCC 1 e TCC 2. A área permite fazer o cadastramento de professores orientadores, acadêmicos, professor de TCC 1 , membros externos e outros professores responsáveis pela administração do sistema. Também é possível definir o calendário de um semestre e cadastrar novas atividades que integrariam as matérias de TCC 1 e TCC 2. No sistema, o professor responsável tem acesso a uma seção para administrar as bancas de TCC, podendo visualizar e agendar bancas, selecionando o estudante e os professores que avaliarão o trabalho, além de definir a data e o tipo da banca também.

Atualmente, estão em andamento outros projetos que envolvem melhorias no SGTCC, um deles propõe melhorias no código do sistema e o outro melhorias na interface gráfica e funcionalidades.

3 MATERIAIS E MÉTODOS

Para atualizar o sistema e alinhá-lo aos padrões atuais, é necessário um conjunto de ferramentas e práticas que facilitem o processo de atualização. Este capítulo apresenta os recursos utilizados e a metodologia adotada para garantir uma transição eficiente e segura.

3.1 Materiais

A atualização do sistema será conduzida com o suporte de diversas ferramentas, abrangendo desde o planejamento até a implementação e o gerenciamento do código-fonte. Essas tecnologias foram selecionadas para otimizar o tempo, melhorar a colaboração e garantir a estabilidade do projeto.

- **Ruby On Rails**: *Framework web* completo e robusto que acelera o desenvolvimento de aplicações web utilizando a linguagem Ruby. Ele facilita a criação de código estruturado, escalável e de fácil manutenção, seguindo convenções que reduzem a necessidade de configuração e promovem boas práticas de desenvolvimento (RAILS, 2025a).
- **Docker**: Plataforma de virtualização leve que permite criar, empacotar e executar aplicações em containers. Esses containers garantem a execução consistente e isolada do software, independentemente do ambiente. Com isso, o Docker facilita a implantação, escalabilidade e portabilidade das aplicações (DOCKER, 2025).
- **ClickUp**: Plataforma de gerenciamento de projetos que centraliza tarefas, documentos e comunicação. Flexível e personalizável, ajuda equipes a organizar fluxos de trabalho, acompanhar projetos e automatizar processos (CLICKUP, 2025).
- **Git**: Sistema de controle de versão distribuído que permite rastrear alterações no código-fonte, facilitando a colaboração entre desenvolvedores e garantindo a integridade do histórico de desenvolvimento (GIT, 2025).
- **Github**: Plataforma baseada em nuvem que utiliza Git para hospedar repositórios, permitindo o versionamento de código, colaboração em equipe e automação de fluxos de trabalho (GITHUB, 2025).
- **Postgresql**: Banco de dados relacional open-source conhecido por sua estabilidade, segurança e desempenho avançado. Suporta consultas complexas, extensibilidade e transações robustas, sendo ideal para aplicações escaláveis (POSTGRESQL, 2025).
- **Hotwire**: Ferramenta para desenvolvimento web rápido que minimiza o uso de JavaScript, permitindo atualizações dinâmicas na interface diretamente do servidor. Ele melhora a experiência do usuário sem comprometer o desempenho (HOTWIRE, 2025).

3.2 Métodos

A atualização de sistemas desenvolvidos em Ruby on Rails exige um planejamento criterioso para assegurar a compatibilidade, a estabilidade e a segurança da aplicação. Embora a equipe mantenedora do *framework* disponibilize um guia oficial para conduzir esse processo, é recomendável adotar boas práticas complementares que contribuam para reduzir possíveis impactos durante a migração (RAILS, 2025b).

3.2.1 Recomendações Gerais para Atualização do Rails

Uma das principais recomendações para garantir uma transição segura é manter uma cobertura abrangente de testes automatizados. Esses testes asseguram que a aplicação continue operando corretamente após cada etapa do processo de atualização. Caso a cobertura seja insuficiente, torna-se necessário realizar verificações manuais em todas as funcionalidades potencialmente impactadas.

O Ruby on Rails requer versões mínimas específicas do Ruby para cada lançamento. Dessa forma, recomenda-se atualizar primeiramente o interpretador *Ruby* para a versão mais recente suportada, antes de prosseguir com a atualização do *framework*. O processo deve ocorrer de forma incremental, seguindo as etapas descritas a seguir:

1. Garantir que todos os testes estejam passando na versão atual do Rails.
2. Atualizar para a versão mais recente dentro da mesma versão principal, resolvendo os avisos de depreciação.
3. Avançar para a versão mais recente da próxima versão secundária, aplicando os ajustes necessários.
4. Repetir o procedimento até alcançar a versão desejada.

O Ruby on Rails disponibiliza o comando `bin/rails app:update`, que auxilia no processo de migração ao sugerir modificações nos arquivos do projeto, de modo a adequá-los à nova versão do *framework*. Após a execução desse comando, é necessário revisar cuidadosamente as alterações propostas e resolver eventuais conflitos identificados. Além disso, durante o processo de migração, o Rails gera o arquivo `config/initializers/new_framework_defaults_X_Y.rb`, que possibilita a ativação gradual das novas configurações introduzidas. Após a validação completa da atualização, recomenda-se remover esse arquivo e ajustar o parâmetro `config.load_defaults`, de forma a refletir a versão efetivamente adotada pelo sistema.

3.2.2 Recomendações para Atualização do Rails 6 para 7

A atualização do Ruby on Rails da versão 6 para a 7 introduz mudanças significativas, abrangendo melhorias de desempenho, novos padrões de convenção e aprimoramentos de segurança. Para assegurar uma transição estável e eficiente, adotou-se um processo estruturado, conforme descrito a seguir.

1. **Preparação para a atualização**: antes de iniciar a migração, foi necessário garantir a estabilidade da aplicação na versão mais recente do Rails 6. Essa etapa incluiu as seguintes ações:

    a) Verificação de que todos os testes automatizados estão passando, assegurando cobertura suficiente para as principais funcionalidades do sistema.

    b) Atualização do Ruby para a versão mínima exigida pelo Rails 7.

    c) Correção de todos os avisos de depreciação identificados na versão 6.

2. **Etapas de atualização**: após garantir a estabilidade na versão anterior, procede-se à migração de forma incremental, conforme os passos abaixo:

    a) Atualização da *gem* `rails` no arquivo `Gemfile` para a versão 7, seguida da execução dos comandos:

    ```
    bundle update rails
    bin/rails app:update
    ```

    b) Revisão das alterações sugeridas pelo comando `bin/rails app:update` e ajuste manual das modificações quando necessário.

    c) Adequação dos arquivos de configuração, incluindo `config/application.rb` e `config/initializers/new_framework_defaults_7_0.rb`, com ativação gradual das novas configurações.

    d) Execução de testes automatizados e verificações manuais para identificar regressões e corrigir eventuais inconsistências.

    e) Atualização das dependências e `gems` utilizadas, garantindo compatibilidade com a nova versão do *framework*.

3. **Principais mudanças na versão 7**: a versão 7 do Rails trouxe modificações estruturais relevantes, entre as quais se destacam:

a) Integração do Hotwire, que amplia a interatividade da aplicação sem demandar uso intensivo de JavaScript.

b) Adoção do `zeitwerk` como carregador de código padrão, substituindo o mecanismo anterior.

c) Implementação de novas estratégias de gerenciamento de conexões com o banco de dados.

d) Otimizações no Active Record, com aprimoramentos no uso de cache e no desempenho das consultas.

4. **Finalização da atualização**: após a validação das alterações e a confirmação do correto funcionamento da aplicação, foram realizadas as seguintes ações:

    a) Remoção do arquivo `new_framework_defaults_7_0.rb`, criado automaticamente durante a migração.

    b) Atualização do parâmetro `config.load_defaults` para refletir a nova versão em uso.

Seguindo essas etapas metodológicas, a migração do Rails 6 para 7 é conduzida de forma controlada, garantindo a compatibilidade, a estabilidade e o desempenho da aplicação após a atualização.

Após a atualização para o Rails 7, foi planejada a migração para a versão 8, com o objetivo de manter o sistema alinhado às versões mais recentes do *framework* e às boas práticas de desenvolvimento recomendadas pela comunidade. Essa nova atualização busca aprimorar a segurança, o desempenho e a modularidade da aplicação, exigindo uma revisão cuidadosa das dependências e dos ajustes de configuração.

3.2.3 Recomendações para Atualização do Rails 7 para 8

A atualização do Rails da versão 7 para 8 introduz mudanças significativas relacionadas à segurança, desempenho e simplificação da configuração do framework. Embora o Rails não disponibilize um guia específico para a migração entre essas versões, é fundamental observar as principais alterações e seguir um processo estruturado para garantir uma transição estável.

Antes de iniciar a atualização, recomenda-se:

- Garantir cobertura adequada de testes automatizados.
- Corrigir avisos de depreciação existentes na versão 7.
- Verificar a compatibilidade das dependências e *gems* utilizadas no projeto.

1. **Remoção de funcionalidades obsoletas**: algumas configurações e métodos foram descontinuados no Rails 8.0, exigindo adaptações no código-fonte. Entre elas:

    - `config.read_encrypted_secrets` foi removido do *Railties*, pois o gerenciamento de credenciais criptografadas passou a ser totalmente centralizado em `config.credentials`.
    - O argumento `model: nil` no método `form_with` deixou de ser aceito, sendo necessário utilizar um modelo explícito ou informar a `url` manualmente.
    - Foram eliminadas configurações obsoletas do Active Record, como:
        - `config.active_record.commit_transaction_on_non_local_return`, removida para evitar comportamentos imprevisíveis em transações.
        - `config.active_record.allow_deprecated_singular_associations_name`, suprimida para promover práticas mais consistentes de nomeação.

2. **Aprimoramentos de segurança**: foram introduzidas melhorias voltadas à mitigação de vulnerabilidades e ao controle de execução.

    - Definição de um tempo limite padrão de 1 segundo para `Regexp.timeout`, reduzindo o risco de ataques de negação de serviço (ReDoS).
    - Inclusão da opção `:local` em `config.active_record.default_timezone`, permitindo o uso do fuso horário local do sistema em vez do UTC.

3. **Mudanças na inicialização e configuração**: o processo de inicialização foi otimizado para maior modularidade e clareza.

    - O arquivo `config/application.rb` passou a seguir uma estrutura mais enxuta e alinhada às boas práticas atuais.
    - O suporte ao `config.load_defaults 7.0` foi removido, tornando obrigatória a definição explícita para `config.load_defaults 8.0`.

4. **Melhorias no Active Record**: a nova versão traz otimizações que simplificam consultas e manipulação de dados.

    - O método `pluck` pode agora ser aplicado diretamente em relações associadas.
    - O método `order` passou a permitir múltiplas colunas simultaneamente, por meio de chamadas como `order(:coluna1, :coluna2)`.

5. **Procedimento de atualização**: após a análise das mudanças introduzidas na nova versão, a atualização deve ser realizada de forma incremental, seguindo as etapas descritas abaixo.

    a) **Revisão de dependências** — verificar se todas as bibliotecas utilizadas são compatíveis com o Rails 8.

    b) **Execução do comando de atualização** — rodar `bin/rails app:update` para ajustar configurações conforme necessário.

    c) **Testes e validação** — garantir o correto funcionamento da aplicação por meio de testes automatizados e manuais.

    d) **Monitoramento em produção** — acompanhar o comportamento do sistema após a implantação para identificar possíveis falhas.

Seguindo essas recomendações, a migração do Rails 7 para 8 pode ser conduzida de forma controlada, mantendo a integridade e o desempenho do sistema.

4 PROCESSO DE DESENVOLVIMENTO E EVOLUÇÃO DO SISTEMA

Este capítulo apresenta o processo de desenvolvimento e modernização do sistema SGTCC, articulando a implementação de melhorias com a atualização a atualização do framework Ruby on Rails e a migração do frontend para tecnologias mais recentes.

4.1 Planejamento e Preparação para a Atualização

O processo de atualização do SGTCC foi planejado com base nas diretrizes descritas no capítulo de Materiais e Métodos, priorizando a segurança, estabilidade e rastreabilidade das modificações. O planejamento considerou a cobertura de testes automatizados do sistema, que abrange cerca de 95% das funcionalidades, o que permitiu uma abordagem iterativa e segura. Essa base de testes foi fundamental para validar as modificações a cada etapa, garantindo que a atualização não introduzisse regressões.

A etapa inicial envolveu a definição de estratégias de migração por versão, de modo a assegurar a compatibilidade gradual do sistema. Assim, optou-se por uma atualização incremental, passando do Rails 6 para o Rails 7 e, posteriormente, para o Rails 8, conforme as boas práticas recomendadas pela comunidade. Cada avanço de versão foi acompanhado por execuções completas da suíte de testes, seguidas de ciclos de correção e refatoração.

Após a validação completa da versão 7, a migração para o Rails 8 foi executada seguindo a mesma lógica incremental. Essa metodologia, fundamentada em avanços graduais e testes contínuos, reduziu riscos de regressões e assegurou a integridade do sistema. O processo completo pode ser descrito como um ciclo iterativo composto por: atualização de versão $\rightarrow$ execução dos testes $\rightarrow$ correção de falhas $\rightarrow$ validação visual e funcional.

4.2 Execução e Estratégia de Migração

A execução da atualização foi guiada por uma estratégia modular, baseada na estrutura de testes automatizados do projeto. O sistema foi dividido em áreas funcionais independentes, seguindo o mesmo agrupamento utilizado nos testes de unidade e integração. Essa organização possibilitou a aplicação controlada das mudanças, reduzindo o escopo de impacto de cada modificação.

A sequência de atualização seguiu uma ordem definida com base na complexidade e no impacto das áreas do sistema: (1) área pública, (2) área do acadêmico, (3) área do professor e (4) área de membros externos. Essa ordem favoreceu o isolamento de falhas e facilitou o monitoramento dos efeitos da migração.

Para cada módulo, adotou-se um fluxo de trabalho padronizado em três etapas:

1. Validação funcional — Execução dos testes automatizados para identificar divergências e falhas introduzidas pela atualização.
2. Validação visual — Inspeção manual das interfaces no navegador, garantindo a preservação da identidade visual e da responsividade.
3. Refatoração estrutural — Ajuste e reorganização de arquivos e componentes, assegurando aderência às novas convenções do Rails 8 e limpeza da estrutura de pastas.

Essa sequência iterativa e modular garantiu que cada modificação fosse validada tanto do ponto de vista funcional quanto visual, assegurando a integridade e coerência do sistema atualizado antes de prosseguir para a etapa seguinte.

4.3 Implementação das Atualizações e Refatorações Principais

Com base nas estratégias definidas nas etapas anteriores, foram implementadas as atualizações e refatorações necessárias para garantir a compatibilidade e o aprimoramento do SGTCC. As modificações foram organizadas em três eixos centrais:

- atualização de dependências,
- reestruturação de componentes e
- modernização da camada de interface.

4.3.1 Atualização de Dependências e Bibliotecas

O primeiro conjunto de ações envolveu a atualização das bibliotecas e *gems*[1] para garantir compatibilidade total com o Rails 8. O arquivo `Gemfile` foi revisado de forma criteriosa, removendo dependências descontinuadas e ajustando versões para aderir às recomendações de segurança e eficiência do framework.

Durante essa etapa, também foram aplicadas as novas configurações padrão disponibilizadas nas versões recentes do Rails. O framework gera automaticamente o arquivo `config/initializers/new_framework_defaults_X_Y.rb`, que permite ativar progressivamente novos comportamentos. Após a validação final, esse arquivo foi removido e a diretiva `config.load_defaults` atualizada para refletir a versão estável em uso.

A atualização possibilitou a remoção de diversas bibliotecas obsoletas e a introdução de novas dependências mais modernas, conforme ilustrado nas tabelas Tabela 1, Tabela 2 e Tabela 3.

[1] *Gems* são pacotes reutilizáveis de código Ruby, equivalentes a bibliotecas em outras linguagens de programação.

Tabela 1 – Bibliotecas Ruby removidas na atualização

Biblioteca	Motivo da Remoção
webpacker	Descontinuada, substituída por ImportMap e Propshaft
turbolinks	Substituída pelo Turbo (parte do Hotwire)
uglifier	Não mais necessária com o uso de ImportMap
bootstrap	Versão 4 removida, substituída por Bootstrap 5 via CDN
bootstrap4-datetime-picker-rails	Substituída por Flatpickr
jquery-rails	jQuery removido completamente do projeto
momentjs-rails	Substituída por funções nativas do Flatpickr
font-awesome-sass	Substituída por ícones do Feather Icons
sassc-rails	Substituída por CSS puro e Propshaft
spring	Removida pois não é mais necessária no Rails 7+
spring-watcher-listen	Dependência do Spring, também removida

Fonte: Autoria própria (2025).

Tabela 2 – Dependências JavaScript removidas

Biblioteca	Motivo da Remoção
vue	Framework substituído por Hotwire
vue-loader	Não mais necessária sem Vue.js
vue-template-compiler	Não mais necessária sem Vue.js
vue-apexcharts	Substituída por integração direta com ApexCharts
vue-clipboard2	Funcionalidade reimplementada com Stimulus
vue-html-to-paper	Funcionalidade reimplementada
vue-i18n	i18n agora é gerenciado pelo Rails
vue-swal	Substituída por SweetAlert2 com Stimulus
vue-turbolinks	Substituída pelo Turbo
vuedraggable	Substituída por Sortable.js com Stimulus
axios	Requisições HTTP não são mais necessárias no cliente
@chenfengyuan/vue-qrcode	Geração de QR Code movida para servidor
@rails/webpacker	Gerenciador de assets removido
moment	Substituída por funções nativas de data
webpack-cli	Não mais necessário com ImportMap
webpack-dev-server	Não mais necessário com ImportMap

Fonte: Autoria própria (2025).

Como observado, o projeto passou de um ecossistema complexo baseado em *npm/webpack*, com 16 pacotes *JavaScript*, para um modelo simplificado com *ImportMap* e apenas 7 bibliotecas essenciais — reduzindo a complexidade de manutenção e o tamanho dos *assets*.

Como observado nas tabelas, o projeto passou de um ecossistema complexo baseado em *npm/webpack*, com 16 pacotes *JavaScript*, para um modelo simplificado com *ImportMap* e apenas 7 bibliotecas essenciais, reduzindo a complexidade de manutenção e o tamanho dos *assets*.

Tabela 3 – Novas dependências adicionadas

Biblioteca	Tipo	Finalidade
importmap-rails	Gem	Gerenciamento de módulos JavaScript
propshaft	Gem	Pipeline de assets moderna
turbo-rails	Gem	Navegação rápida sem recarregar página
stimulus-rails	Gem	Framework JavaScript modesto
view_component	Gem	Componentes reutilizáveis server-side
flatpickr	JS	Seletor de data/hora moderno
sweetalert	JS	Alerts e modais elegantes
slim-select	JS	Select customizável
sortablejs	JS	Funcionalidade drag-and-drop
rqrcode	Gem	Geração de QR Codes no servidor
chunky_png	Gem	Manipulação de imagens PNG

Fonte: Autoria própria (2025).

4.3.2 Reestruturação de Componentes e Organização do Código

A reestruturação da arquitetura interna foi conduzida em conformidade com as convenções do Rails 8, priorizando a modularização e a clareza. A estrutura de pastas foi reorganizada para distinguir explicitamente os componentes de front-end e back-end, eliminando redundâncias e promovendo um padrão de organização baseado em contextos funcionais. Essa abordagem contribuiu para a escalabilidade e manutenção do código.

O uso de *View Components* substituiu diversas *partials* tradicionais, permitindo maior reutilização e testabilidade da camada de apresentação. Essa prática, associada ao uso de Stimulus e Turbo, resultou em um código mais limpo e aderente ao padrão MVC.

4.4 Modernização da Camada de Interface com Hotwire

A modernização da camada de interface constituiu uma das etapas mais significativas do processo de atualização do SGTCC. O sistema utilizava o Webpacker como empacotador de módulos JavaScript, tecnologia que foi descontinuada nas versões mais recentes do Rails, exigindo uma alternativa moderna e sustentável.

Nesse contexto, foi adotado o Hotwire[2], conjunto de tecnologias nativas mantidas pela equipe do Rails, composto pelos módulos Turbo e Stimulus. O Hotwire permite a atualização dinâmica de componentes e seções da página sem a necessidade de recarregamento completo, reduzindo a complexidade e a dependência de bibliotecas externas como React ou Vue, além de preservar a integridade da arquitetura MVC.

A migração iniciou-se pelas páginas de menor complexidade — como a página inicial e a área de autenticação — e avançou para módulos interativos mais complexos, substituindo gradualmente o Vue.js. Durante o período de transição, componentes legados foram mantidos

[2] https://hotwired.dev/

temporariamente até que suas funcionalidades pudessem ser reproduzidas integralmente com Turbo e Stimulus. Um exemplo representativo dessa migração é o componente Flash Messages, cuja implementação original em Vue.js e a versão migrada para Hotwire podem ser consultadas nos Apêndice A e Apêndice B, respectivamente.

4.4.1 Evolução da Arquitetura: do Vue.js ao Hotwire

A transição da arquitetura baseada em Vue.js para o ecossistema Hotwire representou uma mudança estrutural significativa na forma como as páginas do SGTCC são renderizadas e entregues ao usuário. Antes da migração, o processo envolvia uma cadeia de renderização híbrida, na qual parte substancial da construção da interface era delegada ao navegador. Esse fluxo pode ser observado na Figura 1, no lado esquerdo do diagrama, onde a renderização completa dependia de duas requisições: a primeira para carregar a estrutura HTML inicial e a segunda para obter os dados em formato JSON necessários para que o Vue.js pudesse montar os componentes no cliente. Esse processo introduzia atrasos perceptíveis e aumentava o consumo de recursos computacionais no dispositivo do usuário.

Com a adoção do Hotwire, ilustrada no lado direito da Figura 1, o fluxo de renderização foi substancialmente simplificado. O servidor passou a ser o responsável por entregar o HTML já renderizado, eliminando a etapa suplementar de requisição de dados e reduzindo a dependência de JavaScript. A atualização dinâmica da interface passou a ser realizada pelo Turbo, que manipula apenas os trechos necessários do DOM, preservando o estado da página e reduzindo significativamente o tempo até a exibição completa.

Além da modificação no fluxo, a mudança de arquitetura também representa uma alteração profunda no modelo conceitual da aplicação. A Figura 2 sintetiza essa diferença ao comparar, de forma lado a lado, as principais características das duas abordagens. No caso do Vue.js, destaca-se a necessidade de renderização client-side, dependência de requisições assíncronas, gerenciamento de estado no navegador e maior volume de JavaScript. Em contraposição, o Hotwire privilegia a renderização server-side, utiliza JavaScript apenas para comportamentos leves por meio do Stimulus e reduz o processamento no cliente, resultando em maior eficiência e previsibilidade.

A análise conjunta das figuras evidencia que a migração para o Hotwire não apenas simplificou o fluxo técnico de processamento, como também alinhou o sistema às recomendações contemporâneas de desenvolvimento orientado ao servidor. Tais mudanças contribuem diretamente para a redução de latência, a diminuição da carga computacional no cliente e o aumento da robustez e escalabilidade da aplicação, aspectos que serão aprofundados na seção de resultados.

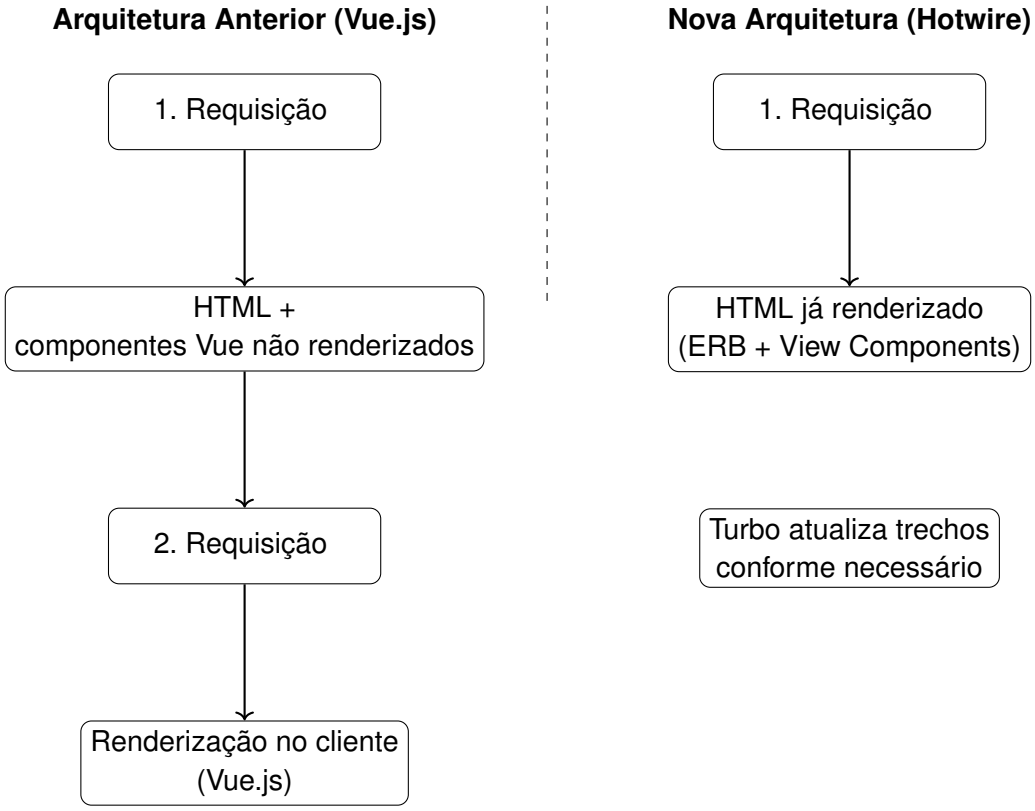

Figura 1 – Comparação dos fluxos de renderização entre a arquitetura anterior (Vue.js) e a nova arquitetura baseada em Hotwire.

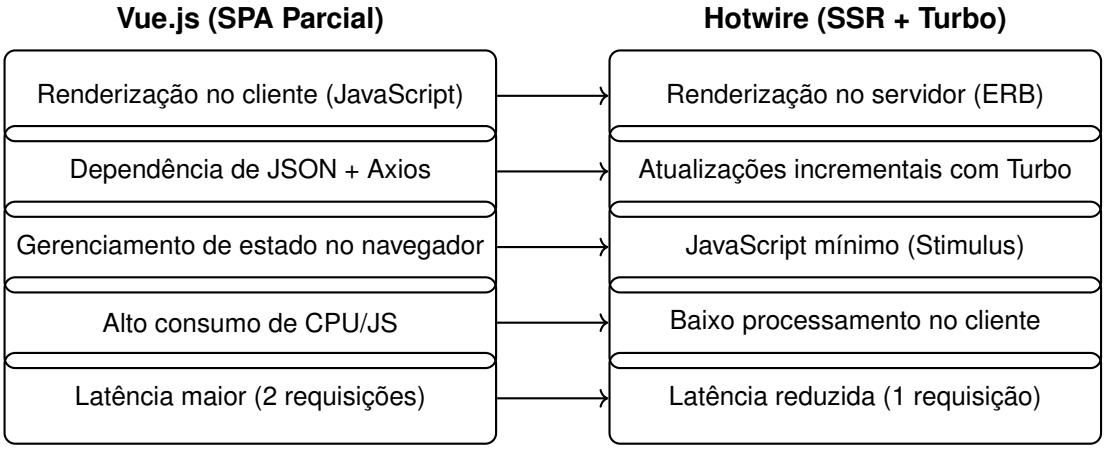

Figura 2 – Diagrama comparativo entre as abordagens Vue.js e Hotwire.

4.5 Comparação de Código e Fluxos de Requisições

O processo de atualização do SGTCC resultou em melhorias significativas em diversos aspectos do sistema, incluindo desempenho, segurança e manutenibilidade. Esta seção apresenta os resultados obtidos através de análises comparativas do código antes e após a migração, métricas quantitativas de desempenho e uma síntese dos benefícios alcançados com a atualização.

A migração da arquitetura baseada em Vue.js para o ecossistema Hotwire resultou em mudanças substanciais na estrutura do código e no fluxo de requisições HTTP. O exemplo do componente de listagem de orientações aprovadas ilustra claramente essas transformações. A implementação anterior com Vue.js e a nova implementação com renderização server-side podem ser consultadas nos Apêndice A e Apêndice B, respectivamente.

4.5.1 Arquitetura Anterior: Vue.js com AJAX

Na implementação anterior, a página utilizava um componente Vue.js que carregava os dados de forma assíncrona, gerando duas requisições HTTP: uma para o HTML e outra para os dados JSON. O fluxo era o seguinte:

1. O navegador solicita a página HTML ao servidor Rails;
2. O servidor retorna um HTML mínimo com o componente Vue.js vazio;
3. Ao ser montado, o componente Vue.js executa uma requisição assíncrona para a API;
4. A API retorna os dados das orientações em JSON;
5. O Vue.js renderiza dinamicamente a lista de orientações.

O código completo do componente Vue.js responsável pela listagem de orientações aprovadas, incluindo a view ERB que o instancia, encontra-se detalhado no Apêndice A.

4.5.2 Nova Arquitetura: Server-Side com Hotwire

A nova implementação elimina a necessidade de requisições adicionais, utilizando renderização server-side completa. O fluxo agora é simplificado:

1. O navegador realiza uma única requisição GET para a página;
2. O servidor Rails processa a consulta ao banco de dados e prepara todos os dados;
3. O HTML completo, incluindo todas as orientações, é renderizado no servidor;
4. O navegador recebe e exibe a página imediatamente, sem processamento adicional.

O código completo da nova implementação, incluindo o controller Rails, a view principal e os partials ERB utilizados na renderização server-side, encontra-se detalhado no Apêndice B.

4.5.3 Comparação de Requisições HTTP

A transformação mais significativa foi a redução do número de requisições HTTP, conforme ilustrado na Tabela 4:

Tabela 4 – Comparação de requisições HTTP entre as arquiteturas

Aspecto	Vue.js	Server-Side
Requisições HTTP	2	1
Tamanho resposta inicial (KB)	15	45
Tempo First Contentful Paint (ms)	800	400
Tempo exibição completa (ms)	1500	400
JavaScript necessário (KB)	280	0
Processamento no cliente	Alto	Mínimo

Fonte: Autoria própria (2025).

Como mostrado, a nova arquitetura reduz a latência e a carga computacional do cliente, proporcionando uma experiência de usuário significativamente melhor.

Apesar do aumento natural no tamanho da resposta inicial — resultado da entrega do conteúdo já renderizado — o tempo total até a exibição completa da página foi reduzido em aproximadamente 70%, graças à eliminação do ciclo extra de requisições e do processamento de JavaScript. O resultado final é uma experiência mais rápida, estável e uniforme, independentemente da capacidade do dispositivo cliente.

4.6 Síntese dos Resultados Alcançados

A atualização do sistema foi realizada em etapas incrementais, partindo do Rails 6 para o Rails 7 e, posteriormente, para o Rails 8. Paralelamente, toda a arquitetura de *frontend* foi migrada de Vue.js para o ecossistema Hotwire, que engloba Stimulus, Turbo e View Components. Essa transformação resultou em:

- **Modernização tecnológica completa**: O sistema agora utiliza as versões mais recentes e mantidas ativamente do Rails 8 e adota as tecnologias recomendadas pela comunidade para desenvolvimento web moderno;
- **Eliminação de dependências obsoletas**: Foram removidas 11 gems Ruby e 16 pacotes JavaScript que estavam descontinuadas ou não eram mais necessárias, simplificando drasticamente o ecossistema de dependências;
- **Refatoração de componentes**: Todos os 57 componentes Vue.js foram migrados para a nova arquitetura, resultando em código mais conciso, testável e alinhado com as convenções do Rails;

- **Redução no tamanho dos assets**: O bundle JavaScript foi reduzido de 350KB para 80KB, melhorando o tempo de download especialmente para usuários com conexões lentas;
- **Cobertura de testes mantida**: Ao longo de todo o processo, a cobertura de testes de 95% foi preservada, garantindo a integridade funcional do sistema.

4.7 Repositório e Contribuições

Todas as alterações realizadas neste trabalho foram desenvolvidas e versionadas no repositório do projeto SGTCC no GitHub, seguindo práticas de desenvolvimento colaborativo e rastreabilidade de mudanças. O trabalho foi desenvolvido em uma branch específica denominada `epic-upgrade-rails`, que centraliza todas as atualizações e refatorações realizadas.

As modificações foram divididas em múltiplas *Pull Requests* (PRs), organizadas de forma incremental e modular, permitindo revisões detalhadas e testes específicos para cada conjunto de alterações. Essa abordagem facilitou o acompanhamento das mudanças, a identificação de possíveis problemas e a validação gradual das melhorias implementadas.

O repositório completo com todas as alterações pode ser acessado através do seguinte endereço:

- **Branch principal**: https://github.com/MarczalTSIGP/SGTCC/tree/epic-upgrade-rails

As *Pull Requests* criadas durante o desenvolvimento do trabalho, organizadas cronologicamente, são as seguintes:

- PR #350: https://github.com/MarczalTSIGP/SGTCC/pull/350
- PR #352: https://github.com/MarczalTSIGP/SGTCC/pull/352
- PR #353: https://github.com/MarczalTSIGP/SGTCC/pull/353
- PR #357: https://github.com/MarczalTSIGP/SGTCC/pull/357
- PR #359: https://github.com/MarczalTSIGP/SGTCC/pull/359
- PR #361: https://github.com/MarczalTSIGP/SGTCC/pull/361
- PR #362: https://github.com/MarczalTSIGP/SGTCC/pull/362
- PR #363: https://github.com/MarczalTSIGP/SGTCC/pull/363

Essas *Pull Requests* documentam todo o processo evolutivo do sistema, incluindo as atualizações do framework, a migração de componentes, a refatoração de código e as melhorias de desempenho. Cada PR foi submetida a revisão de código e validação através de testes automatizados, assegurando a qualidade e a consistência das alterações implementadas.

5 CONSIDERAÇÕES FINAIS

O processo de modernização do SGTCC resultou em significativos avanços tecnológicos, organizados em três dimensões principais: resultados alcançados, contribuições do trabalho e desafios enfrentados. Em termos de resultados, a atualização para o Rails 8 e a migração do frontend de Vue.js para o ecossistema Hotwire proporcionaram uma modernização completa do sistema, incluindo a eliminação de dependências obsoletas, a refatoração de todos os componentes, a redução expressiva do tamanho do bundle JavaScript e a manutenção da cobertura de testes em 95%. Esses avanços garantem a integridade funcional do sistema e melhoram seu desempenho, especialmente para usuários com conexões mais lentas.

No que se refere às contribuições do trabalho, destacam-se a documentação detalhada do processo de migração, que pode servir como referência para projetos similares, e a apresentação de um estudo de caso prático sobre a transição de Vue.js para Hotwire, evidenciando os benefícios dessa abordagem. Além disso, o trabalho assegura a continuidade e evolução do sistema utilizado pela comunidade acadêmica do curso de TSI, demonstra a aplicação prática de manutenção preventiva em sistemas de software — em consonância com as Leis de Lehman — e exemplifica a adoção de boas práticas de engenharia de software, como modularização, separação de responsabilidades, testabilidade e aderência a padrões de projeto.

Apesar dos avanços, o processo não esteve isento de desafios e limitações. A curva de aprendizado associada à adoção de Hotwire e Stimulus exigiu estudo aprofundado e adaptação a novos paradigmas de desenvolvimento. Componentes com maior interatividade demandaram tempo adicional de refatoração para definição de soluções adequadas, e algumas bibliotecas previamente utilizadas precisaram ser substituídas por alternativas compatíveis com a nova arquitetura. Adicionalmente, a documentação oficial apresentou lacunas em cenários específicos, exigindo a consulta a múltiplas fontes e experimentação prática para a implementação correta das soluções.

De maneira geral, os resultados alcançados, as contribuições identificadas e os desafios superados demonstram que a modernização do SGTCC não apenas assegura a continuidade e a segurança do sistema, mas também estabelece uma base tecnológica sólida para futuras melhorias. O sistema encontra-se agora apto a atender de forma eficiente, segura e escalável a comunidade acadêmica da UTFPR, garantindo que sua evolução continue alinhada às necessidades dos usuários e às melhores práticas de desenvolvimento de software.

REFERÊNCIAS

CLICKUP. **One app for projects, knowledge, conversations and more**. 2025. https://clickup.com/. Acesso em: 13 fev. 2025.

COINT. **Normas Operacionais Complementares do Trabalho de Conclusão de Curso do Curso Superior de Tecnologia em Sistemas para Internet - Câmpus Guarapuava**. 2023. https://tcc.tsi.pro.br/uploads/attached_document/file/2/normas-operacionais-complementares-do-TCC-TSI-GP-2023-1.pdf. Acesso em: 04 nov. 2024.

DOCKER. **Accelerate how you build, share, and run applications**. 2025. https://www.docker.com. Acesso em: 13 fev. 2025.

FERREIRA Érico D. **Desenvolvimento de um sistema para o gerenciamento do processo de Trabalho de Conclusão de Curso do curso de Tecnologia em Sistemas para Internet da UTFPR Câmpus Guarapuava**. 2015 — Universidade Tecnológica Federal do Paraná, Guarapuava, PR, 2015.

FOWLER, M. **Refactoring: Improving the Design of Existing Code**. 2nd. ed. Boston, MA: Addison-Wesley, 2018.

GIT. **Free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency**. 2025. https://git-scm.com. Acesso em: 13 fev. 2025.

GITHUB. **Build and ship software on a single, collaborative platform**. 2025. https://github.com/. Acesso em: 13 fev. 2025.

HOTWIRE. **Hotwire: HTML Over The Wire**. 2025. https://hotwired.dev/. Acesso em: 13 fev. 2025.

LEHMAN, M. M.; BELADY, L. A. **Program Evolution: Processes of Software Change**. London: Academic Press, 1985.

LIMA, A. C. **Projeto e implementação de interface baseada na experiência do usuário para um sistema de gerenciamento de trabalho de conclusão de curso**. 2023 — Universidade Tecnológica Federal do Paraná, Guarapuava, PR, 2023.

POSTGRESQL. **PostgreSQL: The World's Most Advanced Open Source Relational Database**. 2025. https://www.postgresql.org/. Acesso em: 13 fev. 2025.

RAILS. **Compress the complexity of modern web apps**. 2025. https://rubyonrails.org. Acesso em: 13 fev. 2025.

RAILS. **Upgrading Ruby on Rails**. 2025. https://guides.rubyonrails.org/upgrading_ruby_on_rails.html. Acesso em: 06 fev. 2025.

SILVA, R. G. A. **Aperfeiçoamento do sistema de Gestão de Processos de Trabalho de Conclusão de Curso de Tecnologia em Sistemas para Internet da UTFPR Câmpus Guarapuava**. 2019 — Universidade Tecnológica Federal do Paraná, Guarapuava, PR, 2019.

SOMMERVILLE, I. **Software Engineering**. 9th. ed. Boston, MA: Addison-Wesley, 2011.

APÊNDICE A – Código do Componente de Listagem de Orientações com Vue.js

A.1 Componente Vue.js

Listagem 1 – Componente OrientationsPage implementado com Vue.js

```
<template>
  <div>
    <div v-show="loading">
      <loader />
    </div>
    <div v-show="!loading">
      <h2 class="page-title mb-2">
        {{ pageTitle }}
      </h2>

      <div class="text-right mb-3">
        Total de <b> {{ orientations.length }} </b> TCCs aprovados
      </div>

      <div v-for="orientation in orientations"
           :key="orientation.id"
           class="orientations mb-4">
        <div>
          <orientation-details :orientation="orientation" />
        </div>
      </div>
    </div>
  </div>
</template>
```

Fonte: Autoria própria (2025).

A.2 View que Renderiza o Componente

Listagem 2 – Componente OrientationsPage implementado com Vue.js

```
<script>
import Loader from '../shared/loader';
import OrientationDetails from './orientation-details.vue';

export default {
  name: 'OrientationsPage',

  components: {
    Loader,
    OrientationDetails
  },

  props: {
    pageTitle: {
      type: String,
      required: true
    },
    path: {
      type: String,
      required: true
    }
  },

  data() {
    return {
      loading: true,
      title: '',
      orientations: []
    };
  },

  async mounted() {
    await this.$axios
      .get(this.path)
      .then(response => (this.orientations = response.data.data));
    this.loading = false;
  }
};
</script>
```

Fonte: Autoria própria (2025).

Listagem 3 – View ERB que renderiza o componente Vue.js

```
<% provide(:head_title, @page.menu_title) %>

<div>
   <orientations-page
      :page-title="'<%= @page.menu_title %>'"
      :path="'<%= api_v1_orientations_approved_path %>'"
   />
</div>
```

Fonte: Autoria própria (2025).

APÊNDICE B – Código do Componente de Listagem de Orientações com Renderização Server-Side

B.1 Controller Rails (Ruby)

Listagem 4 – Método do controller que fornece os dados

```ruby
class SiteController < ApplicationController
  def approved_orientations
    @page = Page.find_by(url: 'tccs-aprovados')
    @orientations = Orientation.approved
  end
end
```

Fonte: Autoria própria (2025).

B.2 View Principal (ERB)

Listagem 5 – View principal que renderiza a lista de orientações

```erb
<% provide(:head_title, @page.menu_title) %>

<%= render partial: "site/orientations/orientation_list", locals: {
  orientations: @orientations,
  page: @page,
  translation_key: 'site.pages.professors.approved.total'
} %>
```

Fonte: Autoria própria (2025).

B.3 Partial de Listagem (ERB)

Listagem 6 – Partial ERB que renderiza a lista de orientações

```erb
<h2 class="page-title mb-2">
  <%= page.menu_title %>
</h2>

<div class="text-right mb-3">
  <%= t(translation_key, count: orientations.count).html_safe %>
</div>

<% orientations.each do |orientation| %>
  <div class="orientations mb-4">
    <%= render "site/orientations/orientation", orientation:
   orientation %>
  </div>
<% end %>
```

Fonte: Autoria própria (2025).