UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO

BSI Sistemas de Informação UFRPE

Uso de microfrontends em aplicações reais: integração entre aplicações Next.js e SingleSPA com React através de Module Federation e Proxy Reverso

---

Relatório Técnico relativo ao Trabalho de Conclusão Curso do Bacharelado em Sistemas de Informação na modalidade Empresa

---

Aluno
Vinícius Vieira de Santana

Orientadora
Silvana Bocanegra
Departamento de Estatística e Informática

Coorientador
Renan Costa Alencar
CESAR School

19 de fevereiro de 2026

Resumo

A crescente complexidade de aplicações web corporativas tem impulsionado a adoção de arquiteturas baseadas em microfrontends, com o objetivo de promover escalabilidade, reutilização de código e autonomia entre equipes de desenvolvimento. Neste trabalho é apresentado um estudo técnico realizado em um laboratório de inovação de uma grande empresa do setor varejista, cujo objetivo foi investigar e viabilizar a integração entre duas aplicações web distintas: um sistema de Marketplace desenvolvido em Next.js e uma plataforma interna denominada Workspace, composta por múltiplos microfrontends em React orquestrados por Single-SPA. Inicialmente, foram exploradas estratégias de integração baseadas em Module Federation, incluindo sua aplicação em conjunto com o framework Next.js e a tentativa de consumo direto de módulos do Workspace. Contudo, limitações técnicas relacionadas à incompatibilidade entre configurações de empacotamento, ao acoplamento interno dos módulos e à descontinuação do suporte oficial ao Module Federation no Next.js inviabilizaram a adoção dessa abordagem em ambiente de produção. Diante dessas restrições, foi proposta e validada uma solução alternativa baseada no uso de proxy reverso, utilizando o Nginx para realizar o roteamento transparente de requisições entre as aplicações. Essa abordagem viabilizou a integração efetiva entre o Marketplace e o Workspace, permitindo a reutilização das telas já existentes, o compartilhamento do contexto de autenticação e a navegação unificada entre os sistemas, sem a necessidade de refatorações estruturais ou de acoplamento direto entre as bases de código. Como resultado, o trabalho evidencia que o uso de proxy reverso mostrou-se uma solução tecnicamente viável e aderente ao contexto corporativo analisado, atendendo aos requisitos de integração estabelecidos e mantendo a independência arquitetural das aplicações envolvidas.

Palavras-chave: Microfrontends; Module Federation; Single-SPA; Proxy Reverso; Arquitetura de Software.

Sumário

1 Introdução

1.1 Contexto

Nos últimos anos, a arquitetura de microfrontends tem-se consolidado como uma alternativa viável para lidar com a crescente complexidade de aplicações web de grande porte. Inspirados na arquitetura de microsserviços, os microfrontends podem ser definidos como uma abordagem de desenvolvimento de interfaces em que uma aplicação é dividida em múltiplas partes menores e independentes, onde cada uma representa um módulo específico do sistema. Essa abordagem possibilita que diferentes equipes desenvolvam e mantenham partes independentes de um mesmo sistema, utilizando tecnologias e ciclos de entrega distintos. Contudo, a integração dessas partes em um produto coeso ainda representa um desafio, sobretudo quando os sistemas pertencem a repositórios separados e possuem contextos de execução heterogêneos.

Hoje em dia, existem duas abordagens amplamente utilizadas na engenharia de software para implementar microfrontends nas aplicações: são elas o Single SPA [1] e o Module Federation [2]. O Single SPA é um framework JavaScript que agrupa múltiplas aplicações menores em uma só e possui ciclos de vida para orquestrar estes submódulos. Já o Module Federation é uma funcionalidade disponível em bundlers (empacotadores de arquivo otimizados) como o Webpack [3] e Rspack [4] e baseia-se na exportação de bundles entre aplicações (uma aplicação expõe os recursos e outra os consome), trazendo maior flexibilidade e independência entre as aplicações.

No ambiente de um laboratório de tecnologia, criado por uma grande empresa de *home center*, surgiu a necessidade de integrar duas aplicações distintas: o Marketplace, responsável pela gestão de vendedores, produtos e operações comerciais, desenvolvido com o framework Next.js [5]; e o Workspace, uma plataforma de produtividade interna distribuída em vários submódulos, sendo eles feitos em React e orquestrados com o Single SPA. Como requisito, essa integração deveria permitir a reutilização de componentes entre as aplicações, sem comprometer a independência de cada uma nem aumentar a complexidade de manutenção. O desafio, portanto, consistiu em viabilizar a comunicação e o consumo de módulos entre sistemas isolados, garantindo escalabilidade, previsibilidade e baixo acoplamento.

1.2 Justificativa

A integração entre os sistemas Marketplace e Workspace apresenta relevância significativa do ponto de vista organizacional e estratégico, uma vez que impacta diretamente a eficiência operacional, a escalabilidade das soluções digitais e a sustentabilidade da arquitetura de software adotada pela empresa. Em ambientes corporativos de grande porte, especialmente aqueles caracterizados por múltiplos times e domínios de negócio distintos, a capacidade de reutilizar funcionalidades e reduzir duplicações de esforço torna-se um fator determinante para a competitividade e a agilidade no desenvolvimento de novos produtos [6].

No contexto da empresa analisada, o Marketplace representa um sistema crítico para as operações comerciais, concentrando funcionalidades relacionadas à gestão de vendedores e produtos. O Workspace, por sua vez, atua como uma plataforma transversal, responsável por centralizar ferramentas internas utilizadas por diferentes áreas da organização. A inexistência de um mecanismo eficaz de integração entre essas aplicações resultaria na replicação de funcionalidades, no aumento do custo de manutenção e na fragmentação da experiência do usuário, além de dificultar a evolução consistente das soluções ao longo do tempo.

Dessa forma, a adoção de uma estratégia que permita o compartilhamento de módulos entre sistemas isolados, preservando a autonomia tecnológica de cada aplicação, contribui diretamente para a redução do acoplamento entre equipes, a padronização de processos e a previsibilidade das entregas. Tais aspectos são fundamentais para organizações que operam em larga escala e que necessitam conciliar inovação contínua com estabilidade operacional.

A solução apresentada neste trabalho impacta diretamente a governança técnica da organização ao propor um modelo de integração alinhado às restrições organizacionais e às limitações do ambiente corporativo, sem descaracterizar os princípios da arquitetura de microfrontends. A sistematização das decisões técnicas e de seus efeitos no contexto de negócio fornece subsídios tanto para a replicação da abordagem em outros sistemas da empresa quanto para a avaliação crítica de tecnologias emergentes em ambientes reais de produção.

Assim, este trabalho justifica-se não apenas pela relevância técnica da integração entre microfrontends, mas também pelo seu potencial de contribuir para a melhoria dos processos de desenvolvimento, manutenção e evolução de sistemas corporativos complexos, alinhando decisões arquiteturais às necessidades estratégicas da empresa.

1.3 Objetivos

O objetivo geral deste trabalho é analisar e viabilizar a integração entre duas aplicações web corporativas distintas — Marketplace e Workspace — por meio de abordagens baseadas em microfrontends, de forma a permitir a reutilização de funcionalidades existentes, preservando a autonomia arquitetural e reduzindo o acoplamento entre os sistemas.

Como objetivos específicos, este trabalho propõe-se a:

- Analisar as características arquiteturais das aplicações Marketplace e Workspace, considerando suas tecnologias, formas de orquestração e restrições técnicas;
- Investigar abordagens de integração baseadas em microfrontends, com ênfase no uso de Module Federation e proxy reverso;
- Avaliar a viabilidade da integração via Module Federation em diferentes cenários, considerando a ausência e a necessidade de refatoração da aplicação Workspace;
- Mapear possíveis limitações técnicas e arquiteturais relacionadas às abordagens;
- Propor uma estratégia para o compartilhamento do contexto de autenticação entre aplicações integradas, garantindo uma experiência de uso contínua.

2 A empresa e sua atuação

A empresa objeto deste estudo atua há mais de um século no setor varejista brasileiro, com foco no segmento de *home center*. Trata-se de uma organização de grande porte, com operação consolidada tanto no varejo físico quanto no comércio eletrônico, incluindo a atuação por meio de uma plataforma de *marketplace*. Seu portfólio abrange produtos voltados à construção civil, reforma, decoração e utilidades para o lar, atendendo tanto consumidores finais quanto profissionais especializados.

Com o objetivo de sustentar sua estratégia de transformação digital, a organização mantém um laboratório interno de tecnologia e inovação, estruturado como uma unidade dedicada ao desenvolvimento de soluções digitais. Esse laboratório adota práticas modernas de engenharia de software, como arquiteturas baseadas em microsserviços e microfrontends, metodologias ágeis e automação de processos, sendo responsável por criar e evoluir produtos digitais que apoiam os canais de venda e a eficiência operacional da empresa.

Atuei nesse laboratório entre maio de 2024 e maio de 2025, exercendo o cargo de engenheiro de software pleno, com foco principal no desenvolvimento *frontend*. Durante esse período, participei do desenvolvimento de uma plataforma web interna destinada à gestão de vendedores e produtos no ambiente de *marketplace* da empresa. Essa solução é considerada estratégica no processo de digitalização organizacional, pois permite que os vendedores parceiros gerenciem cadastros de produtos, pedidos, entregas e demais informações operacionais.

Contribuí desde a concepção da arquitetura do sistema até sua implementação, utilizando tecnologias modernas como React, Next.js, Single-SPA, Module Federation e configuração de *proxy* reverso via Nginx, no contexto de uma arquitetura baseada em microfrontends. Essa abordagem favoreceu a escalabilidade, a manutenção e a integração da plataforma com outros sistemas internos.

Além das responsabilidades técnicas, também desempenhei funções de mentoria para profissionais juniores, revisão de código, disseminação de boas práticas e participação em rituais ágeis da equipe. A célula de desenvolvimento era composta por aproximadamente seis a oito desenvolvedores, além de *product managers*, *designers* de produto e profissionais de qualidade.

A área de *marketplace* mantinha interlocução constante com os setores comercial, logístico e de atendimento ao cliente, assegurando o alinhamento entre as soluções tecnológicas desenvolvidas e as demandas de negócio.

Atualmente, exerço a função de engenheiro de software iOS em uma instituição de pesquisa e desenvolvimento tecnológico de renome nacional.

3 Desenvolvimento realizado na empresa

Esta seção descreve o desenvolvimento do trabalho realizado no ambiente corporativo da empresa, apresentando as etapas, decisões técnicas e soluções adotadas para viabilizar a integração entre os sistemas Marketplace e Workspace. O capítulo está organizado de forma a contextualizar o problema, fundamentar as tecnologias utilizadas, especificar os requisitos da integração e detalhar o processo de implementação da solução proposta.

3.1 O problema e solução proposta

No contexto de aplicações corporativas de grande porte, é comum a existência de múltiplos sistemas desenvolvidos de forma independente, utilizando diferentes tecnologias, arquiteturas e estratégias de organização. Esse cenário, embora natural, frequentemente resulta em duplicação de funcionalidades e esforço de desenvolvimento, especialmente quando aplicações distintas compartilham domínios de negócio semelhantes.

Neste trabalho, o problema central consiste na necessidade de integrar duas aplicações web distintas com o objetivo de reutilizar telas já existentes. O Marketplace, aplicação responsável por consumir essas funcionalidades, foi projetado utilizando o framework Next.js. Por outro lado, o sistema Workspace, aplicação que contém as telas a serem reutilizadas, é composto por múltiplos módulos desenvolvidos em React e orquestrados por meio do SingleSPA. As diferenças arquiteturais e tecnológicas entre essas aplicações impõem desafios à reutilização eficiente de interfaces e à manutenção da coerência arquitetural.

Como solução, este trabalho propõe a análise comparativa de três abordagens de integração baseadas em microfrontends: a integração via Module Federation sem refatoração do Workspace, na qual o Marketplace conecta-se diretamente ao módulo de Produto e obtém as telas; a integração via Module Federation com refatoração do Workspace, permitindo a integração do Marketplace à raiz do SingleSPA; e a integração via proxy reverso, em que as aplicações permanecem desacopladas em nível de código, sendo o acesso às funcionalidades do Workspace mediado por uma camada intermediária. A avaliação dessas abordagens considera critérios como esforço de implementação, impacto arquitetural e grau de acoplamento entre as aplicações, visando identificar a alternativa mais adequada ao contexto do trabalho.

3.2 Tecnologias utilizadas

3.2.1 Microsserviços

Microsserviços [7] são uma abordagem arquitetural na qual uma aplicação é dividida em pequenos serviços independentes, cada um responsável por uma funcionalidade específica. Esses serviços operam de forma isolada, possuem ciclos de desenvolvimento autônomos e comunicam-se por meio de protocolos leves. Essa estratégia promove escalabilidade, facilita a manutenção e possibilita a implantação contínua de partes individuais do sistema sem interromper o funcionamento geral. Além disso, favorece a autonomia de equipes, que podem desenvolver e versionar cada serviço de maneira independente.

3.2.2 Microfrontends

Microfrontends [8] estendem os princípios dos microsserviços para a camada de interface. Nessa abordagem, a interface do usuário é segmentada em múltiplos módulos independentes, cada um podendo ser desenvolvido com tecnologias diferentes, desde que sigam um contrato de integração comum. A principal motivação é permitir que equipes distintas trabalhem paralelamente em partes da interface, reduzindo o acoplamento e aumentando a escalabilidade organizacional. Assim como nos microsserviços, cada módulo pode ser versionado, implantado e atualizado de forma independente, sem afetar a aplicação como um todo.

3.2.3 React

React [9] é uma biblioteca para construção de interfaces que utiliza componentes reutilizáveis como unidade básica de desenvolvimento. Sua principal característica é o uso de uma representação virtual da interface — conhecida como *DOM Virtual* — que otimiza atualizações ao minimizar alterações na estrutura real exibida ao usuário. React adota um modelo declarativo, no qual o desenvolvedor descreve o estado desejado da interface, e a própria biblioteca se encarrega de sincronizar essas mudanças com a tela. Essa abordagem contribui para uma maior previsibilidade e organização do código, especialmente em aplicações extensas.

3.2.4 Next.js

Next.js [5] é um ambiente de desenvolvimento que estende o React com funcionalidades avançadas voltadas ao desempenho e otimização. Entre suas capacidades estão a geração de páginas de forma estática, a renderização no servidor e a divisão automática de módulos para diminuir o tamanho do pacote enviado ao navegador. Esse conjunto de recursos também melhora a indexação por motores de busca e a velocidade de carregamento, fatores essenciais para aplicações de grande porte. Além disso, o Next.js oferece uma estrutura padronizada que facilita a organização do código.

3.2.5 Webpack

Webpack [3] é um empacotador de módulos que analisa os arquivos de uma aplicação, identifica suas dependências e os transforma em um conjunto otimizado de pacotes. Por meio de carregadores e extensões, é capaz de processar diversos tipos de arquivos, como códigos, estilos e imagens. Ele também oferece recursos como eliminação de código não utilizado e divisão inteligente dos pacotes, contribuindo para melhorar o desempenho e reduzir o tempo de carregamento das páginas. É amplamente utilizado em aplicações modernas devido à sua flexibilidade e vasta comunidade de extensões.

3.2.6 SingleSPA

SingleSPA [1] é um conjunto de ferramentas destinado a integrar múltiplas aplicações de interface dentro de um mesmo ambiente. Ele organiza cada aplicação como um módulo com ciclo de vida próprio, controlando sua inicialização, renderização e descarregamento conforme rotas ou eventos. Essa abordagem permite que diferentes tecnologias convivam no mesmo espaço, facilitando a adoção progressiva de novos padrões ou a migração gradual de sistemas legados. O SingleSPA também auxilia a reduzir o acoplamento e a promover maior autonomia entre equipes de desenvolvimento. O repositório [10] demonstra como uma aplicação orquestrada com o SingleSPA pode unir submódulos feitos com React.

3.2.7 Module Federation

Module Federation [2] é um recurso introduzido no Webpack que permite o compartilhamento de módulos entre aplicações distintas em tempo de execução. Esse mecanismo possibilita que diferentes sistemas consumam componentes ou funcionalidades hospedadas em outros ambientes, sem a necessidade de empacotá-los localmente. Dessa forma, torna-se possível a implementação de arquiteturas de microfrontends de forma mais eficiente, reduzindo duplicações e permitindo atualizações independentes. A federação de módulos também ajuda a mitigar problemas relacionados a dependências conflitantes, permitindo a negociação dinâmica de versões compatíveis. O repositório [11] reúne exemplos práticos de integração entre diferentes tecnologias por meio do uso de Module Federation.

3.2.8 Proxy Reverso

Um proxy reverso [12] é um servidor intermediário que recebe as requisições dos usuários e as encaminha aos serviços internos apropriados, atuando em nome dos servidores. Essa abordagem permite abstrair a infraestrutura interna e oferecer funcionalidades como balanceamento de carga, cache de conteúdo e aplicação de regras de segurança, sendo amplamente utilizada em arquiteturas distribuídas e cenários com múltiplas aplicações e microfrontends. Nesse contexto, o Nginx [13] é uma das ferramentas mais empregadas para a implementação de proxy reverso. Trata-se de um servidor web eficiente, baseado em um modelo de processamento assíncrono, capaz de lidar com grande volume de conexões simultâneas com baixo consumo de recursos. Sua configuração flexível possibilita o redirecionamento de rotas, a distribuição de tráfego entre serviços e a mediação do acesso a aplicações modernas, contribuindo para o desempenho e a estabilidade do sistema.

3.2.9 Comparação entre Module Federation e Proxy Reverso como estratégias de Micro Frontend

A integração entre aplicações independentes no contexto de microfrontends pode ser realizada por meio de diferentes estratégias arquiteturais, entre as quais se destacam o *Module Federation* e o uso de Proxy Reverso. O Module Federation, introduzido no Webpack 5, possibilita o compartilhamento dinâmico de módulos em tempo de execução, promovendo reutilização de código e composição de funcionalidades no lado do cliente [2, 3]. Por outro lado, a abordagem baseada em Proxy Reverso opera em nível de infraestrutura, realizando o roteamento de requisições HTTP para aplicações distintas, sem a necessidade de compartilhamento direto de dependências ou de integração em nível de código [13, 12].

Apesar de ambas as estratégias terem como objetivo viabilizar arquiteturas de Micro Frontends, elas apresentam diferenças significativas quanto ao acoplamento entre aplicações, à complexidade de configuração e ao impacto sobre a arquitetura existente. Enquanto o Module Federation tende a exigir maior alinhamento técnico entre os times e as ferramentas de empacotamento utilizadas, o Proxy Reverso oferece maior isolamento entre os sistemas e maior flexibilidade em ambientes heterogêneos [8, 14]. A Tabela 1 apresenta uma comparação entre essas abordagens, destacando os principais critérios técnicos considerados neste trabalho.

Critério	Module Federation	Proxy Reverso
Modelo de integração	Integração em tempo de execução por meio do compartilhamento direto de módulos JavaScript entre aplicações.	Integração baseada em roteamento de requisições HTTP para aplicações distintas, sem compartilhamento de código.
Acoplamento entre aplicações	Alto, devido ao compartilhamento de dependências e à necessidade de compatibilidade entre ambientes de execução.	Baixo, pois as aplicações permanecem isoladas e se comunicam apenas via requisições HTTP.
Gerenciamento de dependências	Dependências compartilhadas são configuradas explicitamente, exigindo controle rigoroso de versões.	Cada aplicação gerencia suas próprias dependências de forma independente.
Complexidade de configuração	Elevada, especialmente em ambientes com múltiplos frameworks, versões e orquestradores.	Moderada, concentrada principalmente na configuração do servidor Proxy Reverso.
Impacto na arquitetura existente	Pode exigir mudanças estruturais nas aplicações para suportar o compartilhamento de módulos.	Mínimo, pois não requer alterações significativas na arquitetura interna das aplicações.
Compatibilidade com diferentes frameworks	Limitada, dependendo da compatibilidade entre ferramentas de build e versões de bibliotecas.	Alta, pois opera em nível de infraestrutura, independentemente do framework utilizado.
Isolamento entre Micro Frontends	Parcial, uma vez que erros em dependências compartilhadas podem afetar múltiplas aplicações.	Alto, já que falhas em uma aplicação não impactam diretamente as demais.
Adequação a ambientes corporativos complexos	Indicada quando há forte padronização técnica entre os times e controle centralizado das dependências.	Indicada quando há heterogeneidade tecnológica e necessidade de rápida integração entre sistemas.

Tabela 1: Comparação entre Module Federation e Proxy Reverso como estratégias de microfrontend.

3.3 Especificação da Integração

Esta seção reúne os principais elementos que definem a integração entre as aplicações Marketplace e Workspace. São apresentados os casos de uso relacionados à integração dos sistemas, os requisitos não funcionais do Marketplace e os critérios de aceite adotados para validar o correto funcionamento da integração, servindo como base para o desenvolvimento e a avaliação da solução proposta.

3.3.1 Casos de uso

Esta subseção apresenta alguns dos casos de uso contidos no escopo do projeto Marketplace, destacando os casos de uso que são implementados pelo Workspace. A Figura 1 apresenta o diagrama de casos de uso em UML que descreve o relacionamento entre as ações do sistema.

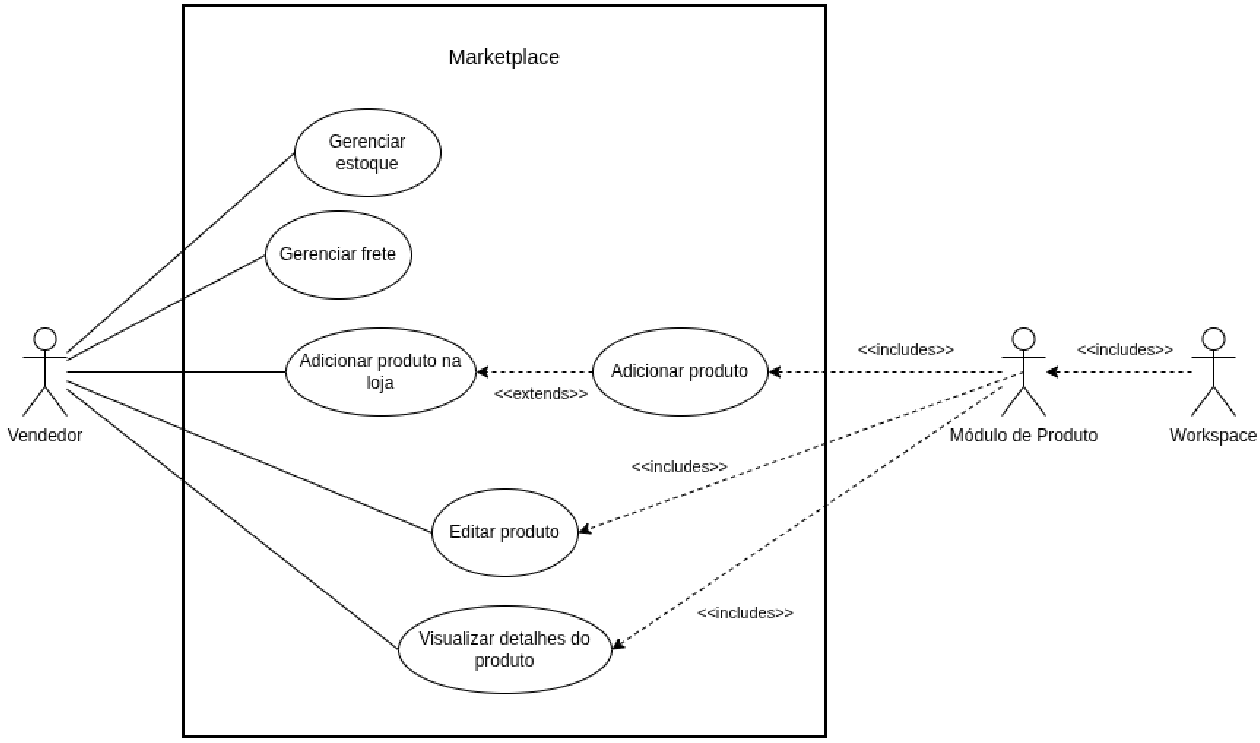


Figura 1: Casos de uso da aplicação Marketplace.

Os casos de uso “Adicionar produto”, “Editar produto” e “Visualizar detalhes do produto” são implementados no Módulo de Produto, que está inserido no Workspace e deve ser acessado a partir dele. A Tabela 2 traz mais detalhes sobre essas ações do sistema.

Nome	Atores	Descrição
Adicionar Produto	Vendedor	Permite que o usuário cadastre um produto não existente no catálogo do Marketplace. Uma vez cadastrado, vendedores das demais lojas poderão adicioná-lo às suas lojas.
Editar Produto	Vendedor	Permite que o usuário atualize informações gerais do produto, como dimensões, peso, entre outras características. As alterações realizadas para um determinado produto serão aplicadas a todas as lojas.
Visualizar Detalhes do Produto	Vendedor	Permite que o usuário visualize os detalhes de um determinado produto.

Tabela 2: Casos de Uso comuns entre Marketplace e Workspace.

3.3.2 Requisitos não funcionais do sistema Marketplace

Para garantir a correta integração entre os módulos envolvidos, bem como assegurar aspectos relacionados à reutilização de código, controle de acesso e consistência das funcionalidades disponibilizadas ao usuário, serão apresentados os requisitos não funcionais definidos para a aplicação Marketplace. A Tabela 3 consolida esses requisitos, descrevendo as diretrizes que orientam a arquitetura da solução proposta.

ID	Descrição
RNF-0001	O sistema Marketplace deve consumir a tela "Adicionar Produto" do módulo de Produto, de modo que não haja replicação de código.
RNF-0002	O sistema Marketplace deve consumir a tela "Detalhes de Produto" do módulo de Produto, de modo que não haja replicação de código.
RNF-0003	O sistema Marketplace deve consumir a tela "Edição de Produto" do módulo de Produto, de modo que não haja replicação de código.
RNF-0004	Ao realizar o deploy no módulo de Produto com alterações em qualquer uma das telas, as modificações devem ser refletidas automaticamente no Marketplace.
RNF-0005	O menu lateral das páginas do módulo de Produto não deve ser exibido no sistema Marketplace.
RNF-0006	A tela "Adicionar produto" deve estar preparada para ocultar ou apresentar campos específicos para o Workspace e para o Marketplace.
RNF-0007	Os usuários do Marketplace não devem ter acesso a outras páginas do Workspace que não sejam aquelas explicitamente mencionadas nos requisitos anteriores.
RNF-0008	O sistema Markeplace deve ser criado de forma agnóstica, de modo que não fique fortemente acoplado ao framework Next.js e seus recursos.

Tabela 3: Requisitos Não Funcionais do Marketplace.

3.3.3 Critérios de aceite da integração entre Marketplace e Workspace

Os critérios de aceite descritos a seguir devem ser atendidos para que a integração seja considerada válida, garantindo que as funcionalidades implementadas no Módulo de Produto, pertencente ao Workspace, possam ser corretamente consumidas pelo sistema Marketplace, sem a necessidade de duplicação de código ou acoplamento excessivo entre as aplicações. Tais critérios foram estabelecidos com base nos requisitos não funcionais e casos de uso.

- O sistema Marketplace deve ser capaz de consumir as telas *Adicionar Produto*, *Editar Produto* e *Visualizar Detalhes do Produto* diretamente do módulo de Produto, sem a replicação de código-fonte ou componentes visuais.
- Alterações realizadas no módulo de Produto, incluindo ajustes funcionais ou visuais em qualquer uma das telas compartilhadas, devem ser refletidas automaticamente no sistema Market-

place após o processo de deploy, sem a necessidade de modificações adicionais no código do Marketplace.

- O menu lateral e demais elementos de navegação específicos do Workspace não devem ser exibidos quando as telas do módulo de Produto forem acessadas a partir do sistema Marketplace, garantindo a coerência visual e a experiência adequada ao contexto da aplicação consumidora.
- Usuários autenticados no sistema Marketplace devem ter acesso exclusivamente às funcionalidades previstas nos casos de uso definidos para a integração, não sendo permitido o acesso a outras páginas ou módulos pertencentes ao Workspace.
- A integração via microfrontend deve permitir que o Marketplace permaneça desacoplado de frameworks específicos utilizados pelo Workspace, de modo que a aplicação consumidora não dependa diretamente de recursos proprietários do framework Next.js para seu funcionamento.
- O mecanismo de integração adotado deve preservar o correto funcionamento dos casos de uso *Adicionar Produto*, *Editar Produto* e *Visualizar Detalhes do Produto*, assegurando que as ações realizadas apresentem comportamentos consistentes tanto no Workspace quanto no Marketplace.

Dessa forma, o atendimento aos critérios de aceite estabelecidos aqui assegura que a integração entre Marketplace e Workspace, por meio da arquitetura de microfrontends, atenda aos objetivos do projeto, proporcionando reutilização de funcionalidades, redução de acoplamento entre sistemas e maior manutenibilidade da solução proposta.

3.4 A integração dos sistemas

Esta seção descreve o processo de desenvolvimento da integração entre as aplicações, abordando as estratégias adotadas, as tecnologias empregadas e os resultados obtidos ao longo da implementação. Para fins de organização, a seção está subdividida em três partes principais: a primeira apresenta a arquitetura de comunicação entre os sitemas; a segunda apresenta as tentativas de integração por meio do Module Federation, contemplando tanto a abordagem sem refatoração do Workspace quanto a estratégia que envolveu sua refatoração para possibilitar a integração do Marketplace diretamente na raiz do Single-SPA; a terceira discute a utilização de proxy reverso como alternativa para o roteamento transparente da página desejada.

3.4.1 Arquitetura

A Figura 2 ilustra o desenho arquitetural proposto para resolver o problema do consumo das telas do Workspace dentro do Marketplace.

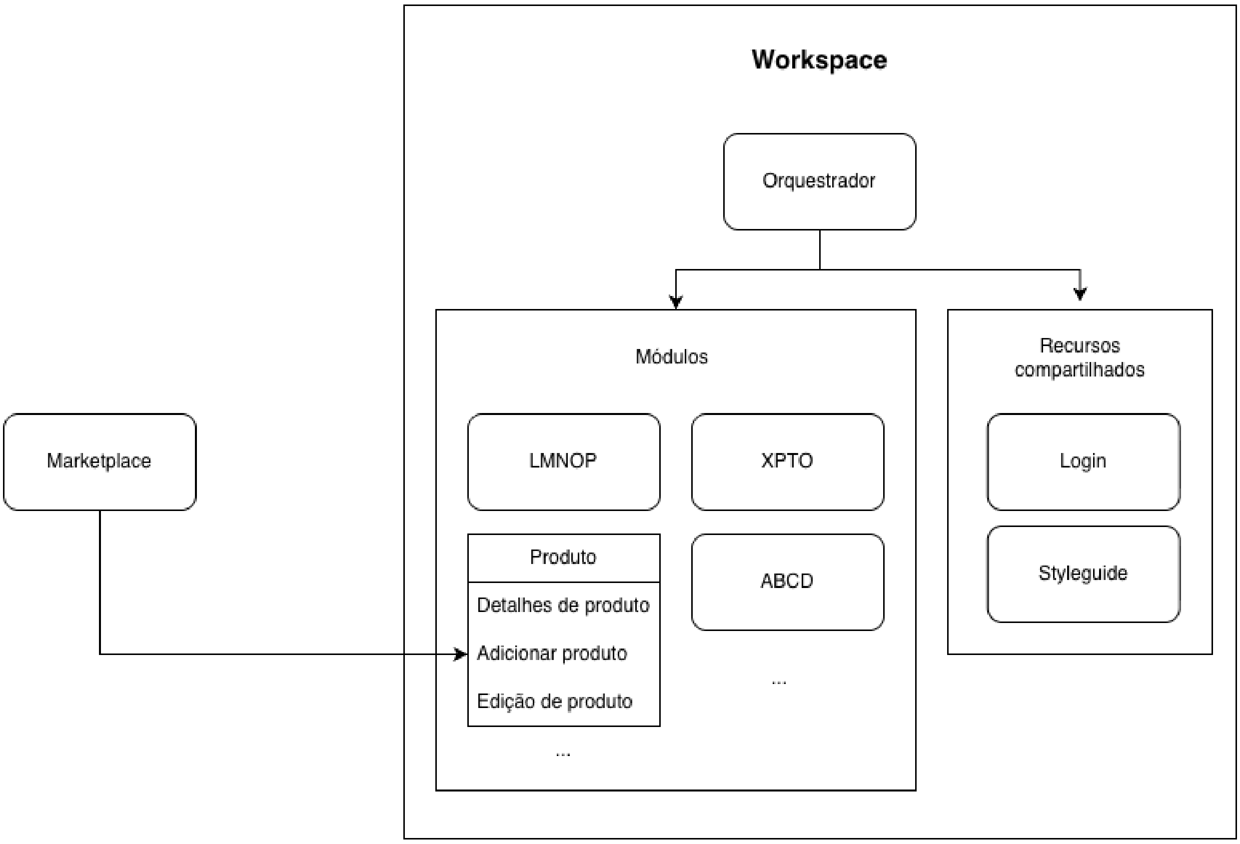


Figura 2: Desenho arquitetural da comunicação entre Marketplace e Workspace.

Para este trabalho, queremos acessar, do Workspace, o módulo de Produto para consumir as telas que já foram implementadas. Para fins práticos, vamos utilizar como exemplo para validar esta integração o consumo da tela “Adicionar produto”, que representa o fluxo de preenchimento das informações necessárias para cadastrar um produto no catálogo da empresa.

As aplicações Marketplace e Workspace encontram-se organizadas em repositórios distintos, não compartilhando código-fonte diretamente, o que reforça o isolamento entre os domínios e favorece a independência evolutiva de cada sistema. Ambas as aplicações adotam a abordagem de monorepositório, porém com estratégias e ferramentas distintas de gerenciamento e orquestração.

A aplicação Marketplace é estruturada como um monorepo que abriga uma aplicação desenvolvida em Next.js utilizando a arquitetura de *Page Router*. Além da aplicação principal, o repositório contempla um conjunto de pacotes reutilizáveis, tais como o *design system* da aplicação, uma biblioteca de gerenciamento de estado com abordagem agnóstica, bem como configurações compartilhadas de ferramentas auxiliares, incluindo Jest para testes unitários, HttpClient para requisições HTTP e ESLint para análise de estática de código. O gerenciamento do monorepo é realizado por meio do *Turborepo*, permitindo otimização do fluxo de desenvolvimento e execução eficiente de tarefas. A aplicação Marketplace encontra-se em desenvolvimento há pouco mais de um ano, período no qual atuei diretamente em sua evolução até maio de 2025. A Tabela 4 apresenta as principais dependências e tecnologias usadas no Marketplace.

Tecnologia	Finalidade
Next.js (v14)	Framework principal para desenvolvimento da aplicação Marketplace, utilizando a arquitetura de Page Router.
React (v18)	Biblioteca base para construção da interface de usuário.
Turborepo	Ferramenta utilizada para gerenciamento do monorepo, otimizando execução de tarefas e compartilhamento de pacotes.
Design System	Conjunto de componentes reutilizáveis responsáveis por padronizar a interface visual da aplicação.
Biblioteca de Gerenciamento de Estado	Solução agnóstica utilizada para controle de estado da aplicação, independente de framework específico.
Jest	Ferramenta utilizada para execução de testes automatizados.
Cliente HTTP	Biblioteca responsável pela comunicação com serviços externos e APIs.
ESLint	Ferramenta de análise estática de código, utilizada para padronização e qualidade do código-fonte.

Tabela 4: Principais dependências e tecnologias utilizadas no Marketplace.

Por sua vez, a aplicação Workspace também é organizada como um monorepo, porém gerenciado pela ferramenta Lerna. O Workspace é composto por aproximadamente dez submódulos independentes, todos desenvolvidos em React, versão 17, em que cada submódulo representa um time operacional distinto da organização. A aplicação é tratada como uma *Single Page Application* (SPA), sendo utilizada a biblioteca SingleSPA para realizar a orquestração dos microfrontends. Ademais, o Workspace disponibiliza dois submódulos compartilhados e consumidos pelos demais: o módulo de autenticação (*login*) e o módulo de *styleguide*, que atua como um *design system*, promovendo padronização visual e reutilização de componentes entre os submódulos. A Tabela 5 apresenta as dependências usadas no Workspace.

Tecnologia	Finalidade
React (v17)	Biblioteca utilizada para o desenvolvimento dos submódulos que compõem o Workspace.
Lerna	Ferramenta responsável pelo gerenciamento do monorepo do Workspace e seus submódulos.
SingleSPA	Biblioteca utilizada para orquestração dos microfrontends, permitindo a composição da aplicação como SPA.
Submódulo de Login	Módulo compartilhado responsável pela autenticação de usuários e controle de acesso.
Styleguide	Submódulo compartilhado que atua como design system, promovendo reutilização de componentes e padronização visual.

Tabela 5: Principais dependências e tecnologias utilizadas no Workspace.

3.4.2 Estudo sobre uso de Module Federation com Next.js

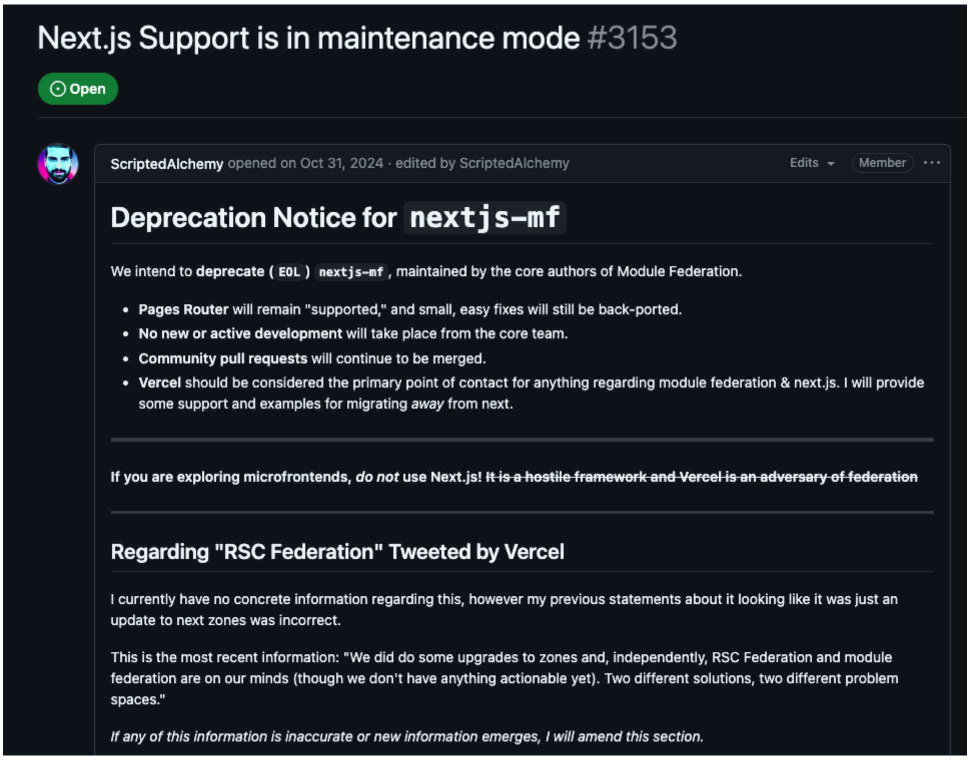
Conforme definido junto a arquitetos de software da empresa, decidimos seguir com a estratégia de integrar o marketplace com o Workspace via Module Federation. Desde o começo, encontramos dificuldades neste processo.

Fizemos uma série de estudos no intuito de testar na prática quais configurações seriam necessárias para viabilizar a comunicação entre duas aplicações. Para isso, criamos duas aplicações para simular esta comunicação, sendo a aplicação host feita em Next.js e a aplicação remote feita com React.js. O Next.js não usa o Module Federation nativo do Webpack, mas sim um plugin chamado @module-federation/nextjs-mf. Isto se dá pelo fato do Next.js ser otimizado para funcionar com SSR (*Server Side Rendering*) enquanto o Webpack é um empacotador que funciona no *client-side*. Esse plugin foi criado para tornar possível o uso da funcionalidade no Next.js.

Inicialmente tentamos exportar um simples componente feito em React da aplicação *remote* (responsável por exportar o conteúdo) e consumí-lo na aplicação *host* (aplicação consumidora). Durante o processo, detectamos um bug no plugin, e entramos em contato com um dos desenvolvedores de @module-federation/nextjs-mf e o problema foi resolvido. Dessa forma conseguimos exportar um componente e provar que seria possível trabalhar com Module Federation no Next.js para consumir componentes de uma aplicação React simples; porém, ainda era necessário validar a integração com o sistema Workspace, uma vez que eles não usavam somente React, mas também o Single SPA para orquestrar diversos módulos internos e convergí-los para uma única aplicação.

Contudo, escolher usar este plugin teria implicações no futuro do Marketplace visto que os desenvolvedores do plugin alertaram que ele somente funciona para arquitetura de *Page Router*, que renderiza as telas no lado do cliente, e que não haveria suporte à arquitetura de *App router*(padrão atual do Framework, renderiza as telas no lado do servidor e as enviam prontas para o cliente). Decidir pelo uso desta ferramenta representava uma limitação técnica para o projeto uma vez que não seria possível utilizar alguns dos recursos mais novos do framework. Porém com este plugin seria reutilizar um fluxo que já estava desenvolvido e validado em outra aplicação, atendendo a expectativa do time de negócios.

Antes de iniciarmos a validação da integração do Next.js com o Single SPA, foi anunciado a descontinuação do plugin @module-federation/nextjs-mf, conforme ilustrado na Figura 3. Após consultas a arquitetos e engenheiros de software de nível sênior, não foram encontradas alternativas que viabilizassem o uso de microfrontends para o Next.js. Foi optado por aguardar um tempo por uma reação da Vercel (empresa por trás do Next.js) e por novas soluções de microfrontend para o Next.js.


Next.js Support is in maintenance mode #3153

Open

ScriptedAlchemy opened on Oct 31, 2024 · edited by ScriptedAlchemy

Edits · Member · ···

Deprecation Notice for `nextjs-mf`

We intend to **deprecate** (`EOL`) `nextjs-mf`, maintained by the core authors of Module Federation.

- **Pages Router** will remain "supported," and small, easy fixes will still be back-ported.
- **No new or active development** will take place from the core team.
- **Community pull requests** will continue to be merged.
- **Vercel** should be considered the primary point of contact for anything regarding module federation & next.js. I will provide some support and examples for migrating *away* from next.

If you are exploring microfrontends, *do not* use Next.js! ~~It is a hostile framework and Vercel is an adversary of federation~~

Regarding "RSC Federation" Tweeted by Vercel

I currently have no concrete information regarding this, however my previous statements about it looking like it was just an update to next zones was incorrect.

This is the most recent information: "We did do some upgrades to zones and, independently, RSC Federation and module federation are on our minds (though we don't have anything actionable yet). Two different solutions, two different problem spaces."

If any of this information is inaccurate or new information emerges, I will amend this section.

Figura 3: Anúncio da descontinuação do @module-federation/nextjs-mf. Fonte [15].

3.4.3 Integração Via Module Federation sem refatoração do Workspace

Após três meses de espera, como não foram apresentadas alternativas de microfrontends para o Next.js, decidimos criar uma aplicação React na versão 18 no intuito de simular o comportamento da aplicação consumidora, sem frameworks ou outras dependências além do Webpack e as demais necessárias para fazê-la funcionar. Além disso, criamos uma aplicação que consistia em um monorepo com SingleSPA, com módulos feitos em React, para simular o comportamento de exportação de um componente.

Configuramos o Webpack de forma correta (conforme demonstrado no Apêndice A) e foi possível exportar o componente Footer da aplicação *remote* e consumí-lo na aplicação *host*. A Figura 4 mostra o componente Footer sendo apresentado na aplicação consumidora. Em seguida, realizamos um estudo de viabilidade da integração de uma aplicação consumidora com o Workspace sem necessidade de refatoração. O Workspace consiste em mais de 10 módulos sendo orquestrados pelo Single SPA. Existia um grande acoplamento entre esses módulos da forma como foi desenvolvido, então uma refatoração traria grandes impactos ao projeto, podendo gerar *side effects* em um projeto estável, além de aumentar a complexidade e o tempo de entrega desta funcionalidade.

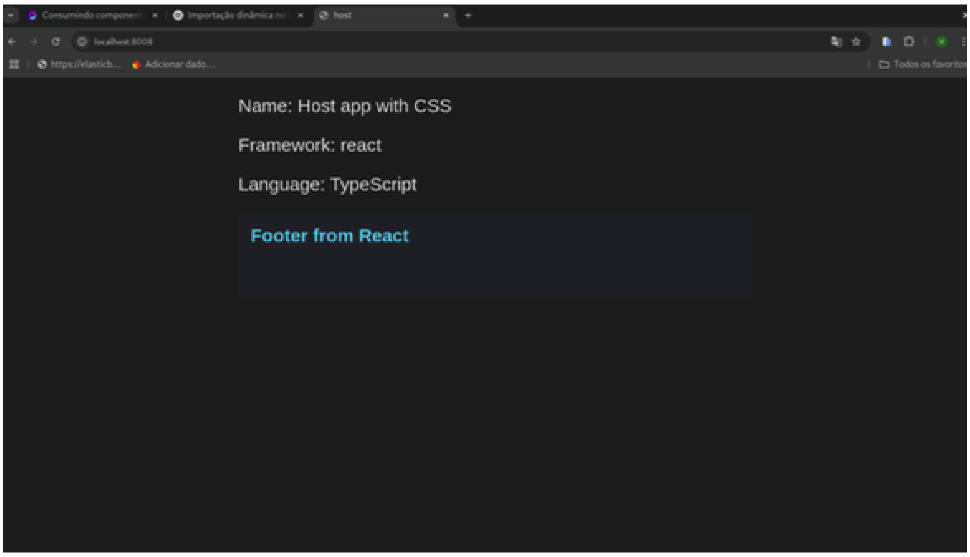


Figura 4: Componente Footer (conteúdo na cor azul com texto ”*Footer from React*”) consumido na aplicação Host.

Uma vez que só queríamos integrar, em um primeiro momento, o Marketplace com o módulo Produto, e ambas possuíam o Webpack como ferramenta de empacotamento, partimos do pressuposto de que seria possível comunicar o Marketplace diretamente com o módulo sem ter interferências do orquestrador do Single SPA. Para testar esta hipótese foi criada uma nova aplicação *host* para testar o consumo. Criamos essa aplicação deixando somente o React 17 (mesma versão do módulo de Produto) e o Webpack como dependências.

Após cinco semanas de tentativas, percebemos que a abordagem não era viável devido à incompatibilidades de atributos definidos na configuração do Webpack. Para o Workspace funcionar o Single SPA adicionava por meio de seu *plugin* um atributo do Webpack chamado *externals* que é uma forma de excluir as dependências declaradas do bundle da aplicação, enquanto o Module Federation tem uma propriedade chamada *shared* que é a responsável por declarar as dependências que seriam compartilhadas entre as aplicações (uma vez que todos os módulos do Workspace eram compilados com uma aplicação única, com um único *bundle* a dependência era declarada uma vez e compartilhada com os demais módulos). A aplicação *host* estava tentando acessar o *React* do módulo de Produto para renderizar a página “Adicionar produto”, mas como esta dependência era excluída do *bundle* graças à propriedade *externals* definida pelo Single SPA, não era possível ter acesso ao *React* e, então, a aplicação quebrava.

3.4.4 Integração Via Module Federation com refatoração do Workspace

Como não foi possível integrar usando a estratégia de Module Federation, sem impactar a estrutura do Workspace como um todo, optamos por refatorar o Workspace para torná-lo compatível com o Module Federation. Para isso, foi preciso alterar todos registros das aplicações de Import Map para Module Federation conforme sugerido pela documentação do Single SPA. Com isso, conseguimos configurar o módulo de Produto para exportar um componente para uma aplicação externa ao monorepo. Na aplicação consumidora, foi renderizado o componente exportado assim como o previsto.

Porém, o Workspace não estava funcionando, pois a estratégia de registrar todas as aplicações por meio do Module Federation fez com que o Single SPA passasse a não reconhecer os seus módulos.

Após dois meses de investigação, percebemos que o módulo de Produto não estava sendo registrado pois ao tentar importar o módulo src/index.ts o SingleSPA analisava todos os aplicativos em cascata e ao chegar nos componentes não encontrava o pacote styleguide, que é um outro módulo do sistema Workspace. Seria preciso alterar todos os demais módulos para que importassem o styleguide via Module Federation. Uma outra opção seria criar um Package Registry para gerenciar os artefatos do styleguide e tratá-lo como uma biblioteca interna da organização em vez de um módulo à parte. Contudo, os *stakeholders* concluíram que a abordagem de integração entre as aplicações através de Module Federation se mostrava bastante sensível, e os estudos para viabilizar a integração por este caminho foram interrompidos.

3.4.5 Estudo do uso de Proxy Reverso como estratégia de microfrontend

Após reuniões com arquitetos de software, lideranças de negócio e outros desenvolvedores de software foi decidido que não era mais viável continuar utilizando Module Federation e optamos por fazer o redirecionamento da navegação para uma página do módulo de Produto através de um Proxy Reverso.

Foram criadas duas aplicações chamadas de Aplicação A (que simula o papel do Marketplace em consumir o recurso) e Aplicação B (que simula o do Workspace em fornecer o recurso). A Aplicação A consiste em uma tela de listagem de produtos, enquanto a Aplicação B contém uma tela chamada "Adicionar Produto" e simula o cenário da integração. Ao clicar no produto, é feito o redirecionamento para a rota /remote/add-product e o usuário obtém acesso à tela "Adicionar Produto". Para controlar o redirecionamento com Nginx, foi criado um arquivo com as configurações necessárias, conforme apresentado no Apêndice B. A Figura 5 ilustra o funcionamento do proxy reverso.

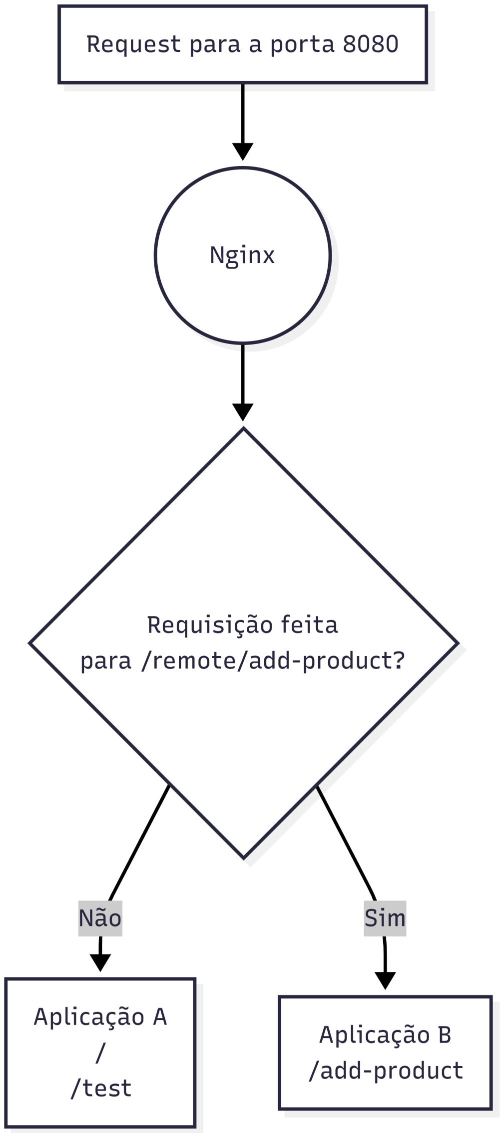


Figura 5: Diagrama da proposta de integração via Proxy Reverso.

Primeiramente, realizamos uma prova de conceito (PoC) para testar o funcionamento da abordagem. Além de realizar a comunicação, esta PoC também tinha como objetivo nos ajudar a entender quais recursos poderiam ser compartilhados entre si e de que forma. Por exemplo, o Workspace armazenava o token de usuário no Local Storage, então para cada requisição feita, o Workspace obtinha este valor do Local Storage para validar se o usuário ainda estava logado. Portanto, precisaríamos garantir que o token gerado no login do usuário do Marketplace poderia ser reutilizado pelo Workspace para manter o comportamento da tela de Adicionar Produto. A Figura 6 mostra a tela de listagem de produtos da Aplicação A.

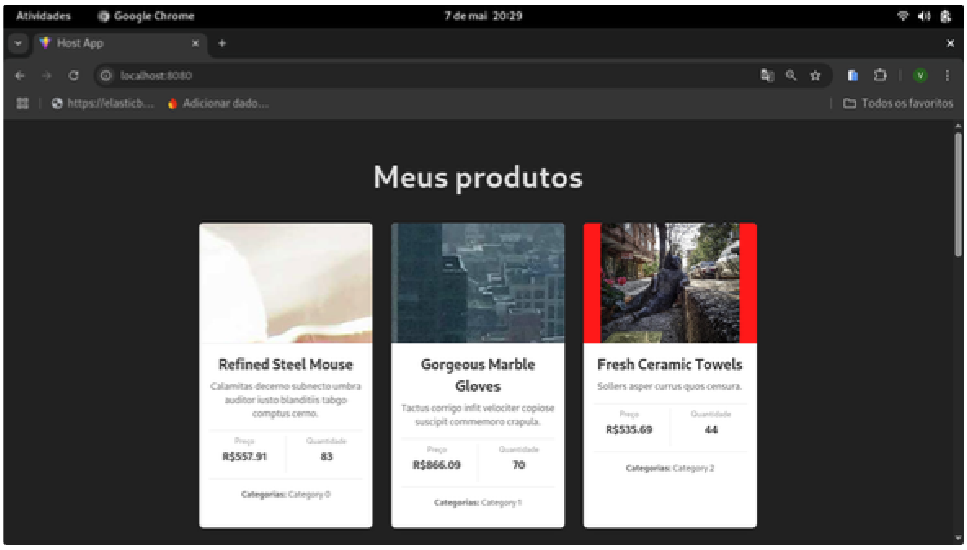


Figura 6: Tela de listagem de produtos da Aplicação A.

Ao clicar em um dos produtos, as informações são salvas no Local Storage do navegador. Na tela “Adicionar Produto”, as informações salvas são apresentadas na tela. O usuário é mantido na mesma aba e não percebe que o redirecionamento é feito entre duas aplicações distintas, uma vez que ele continua sob o domínio da aplicação A. A Figura 7 mostra a tela “Adicionar produto” com os dados carregados após o usuário clicar em um dos produtos.

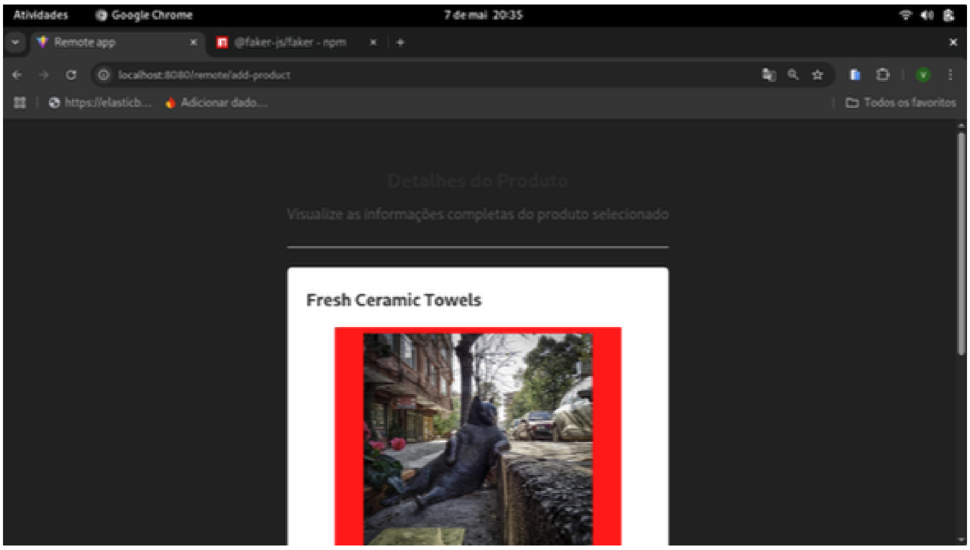


Figura 7: Tela “Adicionar Produto” consumida na Aplicação A.

Além do funcionamento da navegação, validamos que com o Proxy Reverso a aplicação consumidora e a funcionalidade exportada compartilham o mesmo Local Storage por estarem sob a mesma origem.

3.4.6 Importação da tela “Adicionar Produto” através de Proxy Reverso

Após os avanços da prova de conceito de proxy reverso, reutilizamos a Aplicação A (consumidora) e alteramos as configurações de Nginx para interceptar as requisições para o recurso /remote/add-product e fazer o proxy para o recurso de destino. As configurações utilizadas podem ser encontradas no Apêndice C.

Na aplicação A, foi adicionado um botão com o texto “Adicionar produto”. Ao ser clicado, o usuário é redirecionado para a página “Adicionar produto” do Workspace, mas ainda sob o domínio da Aplicação A. A Figura 8 apresenta a tela de listagem de produtos com o novo botão.

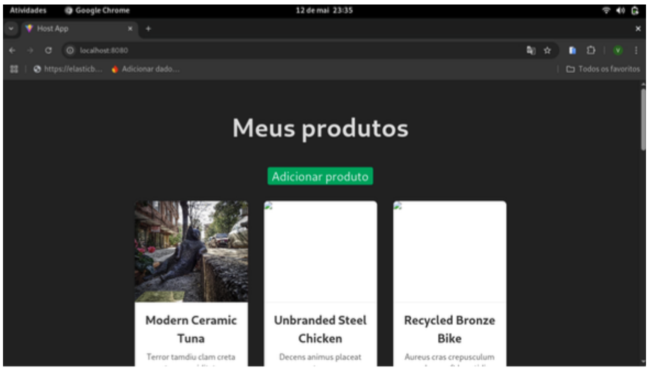


Figura 8: Aplicação A com botão “Adicionar produto”.

Para acessar a página, é necessário que o usuário esteja logado. Essa validação é feita com um token JWT que está armazenado no Local Storage do navegador, sendo possível compartilhar esta informação entre as aplicações, visto que elas estão sob a mesma origem. Para simular essa integração, foi armazenado um token com as permissões de usuário do Marketplace gerado pelo serviço de autenticação da empresa. Este token foi salvo em uma chave esperada pela tela “Adicionar Produto”.

```
localStorage.setItem("<chave\_esperada>", "<seu\_token\_de\_usuário>")
```

O redirecionamento para a página é feito por meio da função:

```
const onRedirectToAddProduct = () => {
    window.location = ${window.origin}/partners/suppliers/catalog/product/add-product`
        as string & Location;
}
```

A Figura 9 apresenta a tela “Adicionar produto” sendo consumida na Aplicação A.

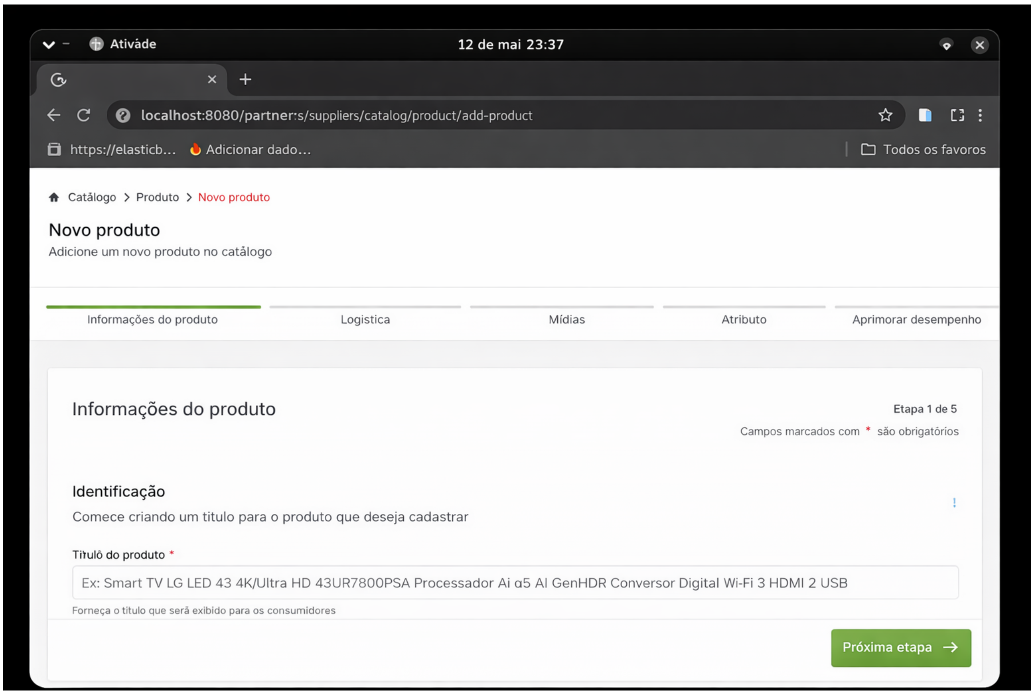


Figura 9: Tela de Adicionar Produto sendo renderizada na Aplicação A.

Conforme o requisito não-funcional RNF-0006 presente na Tabela 3, a tela "Adicionar produto" deve estar preparada para ocultar ou apresentar campos específicos para cada aplicação. Para realizar este controle, foi criado um componente React responsável por ocultar essas informações, com a responsabilidade de apresentar somente o que for necessário para o usuário. Mais detalhes sobre a implementação deste componente podem ser encontrados no Apêndice D.

Conforme o requisito não-funcional RNF-0005, o menu lateral será apresentado somente para o Workspace, evitando que o usuário do Marketplace possa interagir com ele.

Outra observação identificada foi que, embora o token do Marketplace estivesse armazenado no Local Storage e o Workspace tenha conseguido encontrá-lo, o token não tinha permissão para executar ações relacionadas ao módulo de Produto.

No frontend do Workspace, o token era corretamente recuperado; entretanto, ao realizar requisições HTTP à API utilizando esse token, os *status codes* [16] 401 (Unauthorized) ou 403 (Forbidden) eram retornados. Isso ocorria porque o token do usuário do Marketplace não continha as *claims* necessárias associadas ao módulo de Produto. Dessa forma, o usuário não conseguia realizar ações como "Adicionar Produto" ou executar interações na interface, como aplicar filtros e buscar categorias específicas, até que as *claims* correspondentes fossem incluídas em seu token pelo time responsável pelo serviço de autenticação. Durante esta prova de conceito, também foram mapeadas quais seriam as permissões necessárias para o token de um usuário do Marketplace para que ele pudesse ter os acessos necessários para efetuar as ações desta funcionalidade.

Por fim, destaca-se que, para viabilizar a comunicação entre o Marketplace e as telas do Workspace, é necessário habilitar e configurar adequadamente o mecanismo de CORS (Cross-Origin Resource Sharing) tanto no frontend quanto no backend do Workspace.

3.5 Contribuição

Até o término de minha atuação como colaborador da empresa, a solução de integração via Proxy Reverso não havia sido lançada em produção. Contudo, além de ter deixado uma documentação completa citando todos os passos necessários, também realizei treinamentos com todos os envolvidos na integração, time de desenvolvedores do marketplace, do workspace ou dos profissionais de infraestrutura que atuam de forma *cross-projeto*.

Além de fornecer este passo a passo, neste trabalho também foram levantados vários riscos que poderiam impedir a integração das aplicações.

O trabalho gerado recebeu *feedbacks* positivos de todos os envolvidos. Desenvolvedores de ambos os times agradeceram pelo passo a passo deixado, assim como pelas explicações fornecidas. Pessoas do time de Negócio (como Product Owners e pessoas do time de Comercial) confirmaram o entendimento do que foi apresentado e validaram a solução.

4 Dificuldades encontradas

Dada a especificidade do problema abordado, alguns desafios foram enfrentados ao longo do desenvolvimento. Esta seção visa descrever as limitações técnicas encontradas durante esse processo.

4.1 Incompatibilidade nas configurações de Webpack entre o Module Federation e o Single-SPA

Conforme definido inicialmente junto ao arquiteto responsável à época, bem como em conjunto com outros desenvolvedores, a integração entre o Marketplace e o módulo de Produto seria realizada por meio do Module Federation, permitindo a exportação das telas desejadas do módulo e sua apresentação dentro do Marketplace. A Figura 10 demonstra como seriam, visualmente, as páginas do Marketplace com os conteúdos obtidos do Workspace.

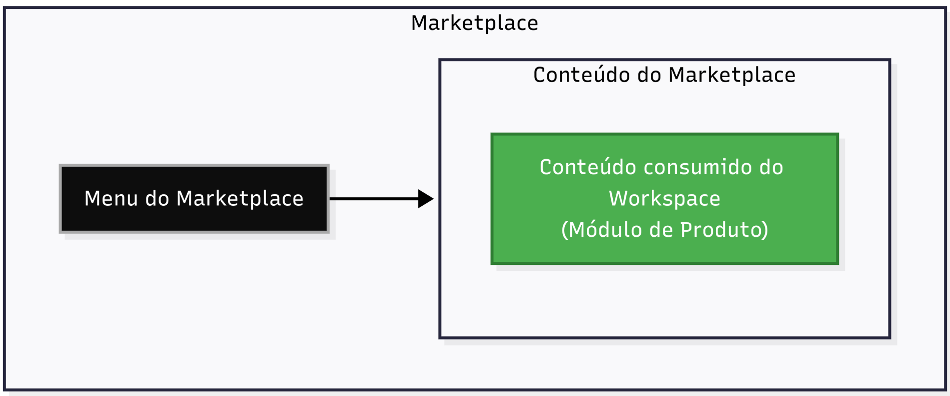


Figura 10: Esboço ilustrativo da página “Adicionar Produto” do Marketplace com o conteúdo exportado do módulo de Produto

Entretanto, após meses de tentativas, não foi possível consumar a integração por esse caminho. Foram identificadas limitações técnicas relacionadas à forma como o Workspace está estruturado. O Workspace consiste em um conjunto de aplicações separadas, desenvolvidas por equipes distintas, que utilizam o Single-SPA como solução de microfrontend para orquestrar essas aplicações e tratá-las como uma aplicação única. Contudo, o Single-SPA requer internamente uma série de configurações específicas do Webpack, dentre as quais se destaca a propriedade *externals*. Essa propriedade é responsável por declarar quais dependências devem ser excluídas do *bundle* final da aplicação. Em contrapartida, o Module Federation depende de uma propriedade denominada *shared*, responsável por declarar quais dependências serão compartilhadas entre as aplicações, como, por exemplo, a dependência React. Em síntese, a incompatibilidade entre as formas como essas duas tecnologias gerenciam e acessam as dependências impossibilitou o consumo das telas do módulo de Produto no Marketplace.

4.2 Plugin @module-federation/nextjs-mf em modo de manutenção

O Marketplace foi desenvolvido utilizando o *framework* Next.js, aproveitando suas vantagens relacionadas à renderização no lado do servidor, o que contribui para um carregamento mais rápido da aplicação e reduz a necessidade de processamento no navegador do usuário.

Para viabilizar o uso do Module Federation, foram pesquisadas soluções que possibilitassem a habilitação dessa funcionalidade no Next.js. Verificou-se que seria necessário utilizar um plugin específico, desenvolvido pelos mantenedores do Module Federation, denominado @module-federation/nextjs-mf. Desde o início, esse plugin demonstrou-se uma solução limitada, pois exige que as aplicações Next.js sejam desenvolvidas utilizando a arquitetura *Page Router*. Atualmente, o Next.js adota como padrão a arquitetura *App Router*, o que faz com que a aplicação deixe de usufruir de alguns recursos mais recentes e futuros do *framework*.

Após dois meses de estudo em torno desse plugin, o principal mantenedor do Module Federation, Zack Jackson, publicou uma série de postagens na rede social X com críticas ao Next.js e à Vercel, informando que o suporte ao Next.js entraria em modo de manutenção. Posteriormente, no dia 31 de outubro de 2024, foi criada uma *issue* no GitHub anunciando que o time principal não daria continuidade ao desenvolvimento do plugin. Até o término das tentativas de integração, o Next.js não havia providenciado uma solução estável para habilitar a comunicação entre aplicações por meio de microfrontends.

4.3 Acoplamento entre os módulos internos do Workspace

Outro problema identificado durante o desenvolvimento da integração foi o forte acoplamento existente entre os módulos internos do Workspace. Na tentativa de exportar as telas do módulo de Produto por meio do Module Federation, constatou-se que esse módulo dependia de outros módulos do Workspace, os quais são orquestrados pelo Single-SPA. Especificamente, o módulo de Produto apresentava dependências diretas dos módulos Styleguide e Login.

O módulo Styleguide concentrava todos os componentes de interface utilizados pelos submódulos do Workspace. Dessa forma, tratava-se, na prática, de uma aplicação independente, executada em uma porta específica. Assim, caso o Module Federation fosse utilizado para consumir as telas do módulo de Produto, seria igualmente necessário obter esses componentes a partir do módulo Styleguide. Entretanto, não foi possível consumir os componentes do Styleguide via Module Federation. Para viabilizar essa abordagem, seria necessário remover o acoplamento existente entre o módulo de Produto e o Styleguide. Essa remoção demandaria a migração de todas as referências aos componentes do Styleguide, o que implicaria um esforço elevado e um tempo considerado inviável. Alternativamente, seria necessário redeclarar provisoriamente os componentes no módulo de Produto ou criar um *Package Registry* específico para o compartilhamento desses componentes.

4.4 Adaptações no módulo de Produto para atender a requisitos de outra aplicação

Para viabilizar a integração por meio do Proxy Reverso, tornou-se necessário implementar um mecanismo de controle de visibilidade dos conteúdos exibidos na página *Adicionar Produto*. Determinadas informações não deveriam ser apresentadas ao usuário do Marketplace, enquanto alguns elementos da interface seriam exclusivos desse ambiente e, portanto, deveriam permanecer ocultos para os usuários do Workspace.

Um exemplo relevante é o menu lateral do Workspace, que permite a navegação entre diversas páginas da aplicação. Como apenas um subconjunto dessas páginas está disponível para os usuários do Marketplace, esse menu não poderia ser exibido nesse contexto.

Com o objetivo de gerenciar essa lógica de forma centralizada e consistente, foi desenvolvido um componente responsável por filtrar e controlar a exibição de elementos da interface com base no domínio de origem da aplicação. Detalhes adicionais sobre essa implementação podem ser consultados no Apêndice D.

5 Impactos da sua formação no seu trabalho

Minha formação acadêmica exerceu papel fundamental no desenvolvimento das competências técnicas, analíticas e arquiteturais que aplico atualmente em minha atuação profissional, especialmente no contexto de sistemas distribuídos e arquiteturas de software modernas. Ao longo do curso, os conhecimentos adquiridos em disciplinas teóricas, projetos práticos e atividades extracurriculares contribuíram diretamente para a minha capacidade de projetar, implementar e evoluir soluções computacionais escaláveis e alinhadas às necessidades organizacionais.

A disciplina Fundamentos de Problemas Computacionais I foi essencial para o fortalecimento do raciocínio lógico, da capacidade de abstração e da formulação algorítmica. Embora eu já possuísse contato prévio com programação, essa disciplina me desafiou a resolver problemas progressivamente mais complexos, estimulando a decomposição de problemas em partes menores e a construção de soluções eficientes por meio de algoritmos. Essa base conceitual é diretamente aplicável ao desenvolvimento de arquiteturas modulares, nas quais a decomposição do sistema em componentes independentes é um princípio central [17]. Na prática profissional, esse aprendizado mostrou-se indispensável para a análise de complexidade, definição de responsabilidades e redução de acoplamento entre módulos de sistemas.

Na disciplina Desenvolvimento de Sistemas de Informação, tive a oportunidade de desenvolver minha primeira aplicação móvel aplicando conceitos fundamentais da engenharia de software. Durante essa experiência, utilizei práticas como versionamento de código com Git, organização do trabalho em sprints e colaboração em equipe, aproximando o ambiente acadêmico da realidade do desenvolvimento profissional. Esses conceitos estão alinhados a modelos iterativos e incrementais amplamente utilizados no desenvolvimento de sistemas complexos [18]. Essa vivência contribuiu para minha compreensão inicial sobre separação de responsabilidades, organização de código e evolução contínua de sistemas — aspectos que mais tarde se mostraram essenciais para o trabalho com arquiteturas baseadas em microfrontends. Além disso, a experiência com o framework Flutter foi determinante para minha inserção inicial no mercado de trabalho por meio do primeiro estágio.

As disciplinas de projeto desempenharam papel central no desenvolvimento da minha capacidade de conceber soluções arquiteturalmente mais complexas e orientadas a problemas reais. Em especial, destaco a disciplina Projeto de Desenvolvimento Tecnológico para o Mundo I, na qual desenvolvemos, em grupo, um aplicativo de acompanhamento para pacientes portadores de Esclerose Múltipla. Nesse projeto, adotamos a metodologia *Human-Centered Design* (HCD), que prioriza a compreensão profunda das necessidades dos usuários ao longo de todo o ciclo de desenvolvimento [19]. O contato direto com pacientes, cuidadores, familiares e profissionais de saúde reforçou a importância de alinhar decisões técnicas e arquiteturais aos requisitos humanos e contextuais do sistema. Essa experiência influenciou diretamente minha atuação profissional posterior, especialmente na concepção de arquiteturas flexíveis e evolutivas, capazes de atender diferentes perfis de usuários e domínios de negócio.

Além das disciplinas curriculares, minha atuação na empresa júnior Seed a Bit teve impacto significativo na minha formação profissional. Essa experiência representou meu primeiro contato direto com projetos reais e clientes, permitindo a aplicação prática de conceitos de engenharia e arquitetura de software. Ao longo desse período, evoluí do cargo de analista para o de diretor de tecnologia, o que contribuiu para o desenvolvimento de habilidades de comunicação, liderança, negociação e tomada de decisão técnica. Essas competências são particularmente relevantes em contextos de arquiteturas distribuídas, como microserviços e microfrontends, nos quais a coordenação entre equipes e a definição clara de responsabilidades são fatores críticos de sucesso [6, 20]. Essa vivência também reforçou minha compreensão sobre governança arquitetural, padronização e autonomia de times, conceitos diretamente relacionados ao tema central deste trabalho.

Por fim, expresso minha gratidão a todos os professores, orientadores, colegas de curso e profissionais que contribuíram para minha trajetória acadêmica. Os ensinamentos, orientações e experiências compartilhadas ao longo da graduação foram fundamentais para minha formação técnica, humana e profissional, tendo impacto direto na construção da base conceitual que sustenta minha atuação atual na área de arquitetura de software.

6 Conclusão

Este trabalho teve como objetivo analisar e viabilizar a integração entre duas aplicações web corporativas — Marketplace e Workspace — por meio de uma arquitetura baseada em microfrontends, considerando restrições técnicas, organizacionais e de governança presentes em um ambiente de produção real. Ao longo do estudo, foram investigadas diferentes abordagens de integração, com ênfase inicial no uso do Module Federation, amplamente reconhecido como uma solução moderna para compartilhamento de módulos entre aplicações independentes.

As tentativas de adoção do Module Federation evidenciaram desafios relevantes, tais como a incompatibilidade entre configurações do Webpack e do Single-SPA, o forte acoplamento entre os módulos internos do Workspace e as limitações impostas pelo uso do framework Next.js, especialmente após a descontinuação do suporte oficial ao plugin @module-federation/nextjs-mf. Esses fatores demonstraram que, embora tecnicamente promissora, a solução não se mostrou viável no contexto específico da organização estudada, sobretudo quando considerados os riscos de refatoração em larga escala e os impactos na estabilidade de sistemas já consolidados.

Diante desse cenário, a adoção de um proxy reverso como mecanismo de integração mostrou-se uma alternativa eficaz e pragmática. A solução permitiu o roteamento transparente entre aplicações, o reaproveitamento de funcionalidades existentes e o compartilhamento de contexto de autenticação, preservando a experiência do usuário final e respeitando as limitações arquiteturais e organizacionais do ambiente corporativo. Além disso, a abordagem adotada manteve o desacoplamento entre os sistemas, favorecendo a manutenibilidade e a evolução futura das aplicações.

Embora a solução não tenha sido implantada em produção até o término de minha atuação na empresa, os artefatos gerados, a documentação elaborada e os treinamentos realizados contribuíram para a disseminação do conhecimento e para a tomada de decisões técnicas mais embasadas pelos times envolvidos. Assim, este trabalho evidencia que a escolha de uma arquitetura de integração deve ir além da adoção de tecnologias emergentes, exigindo uma análise crítica do contexto organizacional, da maturidade técnica das ferramentas e dos impactos a longo prazo sobre a governança e a sustentabilidade dos sistemas.

Como trabalhos futuros, sugere-se a avaliação de novas abordagens de microfrontends compatíveis com frameworks orientados à renderização no servidor, bem como estudos comparativos entre soluções baseadas em proxy reverso e integrações nativas à medida que o ecossistema de ferramentas evolua.

Referências Bibliográficas

[1] S.-S. Community, “Single-spa documentation,” *single-spa.js.org*, 2023.

[2] Z. Jackson, “Module federation in webpack 5,” *Webpack Documentation*, 2020.

[3] W. Contributors, “Webpack documentation,” *Webpack.js.org*, 2023.

[4] Rspack Team, “Rspack documentation,” https://www.rspack.dev/, 2024, acesso em: 15 jan. 2026.

[5] V. Inc., “Next.js documentation,” *Nextjs.org*, 2023.

[6] I. Sommerville, *Engenharia de Software*. São Paulo: Pearson, 2011.

[7] S. J. Fowler, *Microsserviços Prontos para a Produção*. São Paulo: Novatec Editora, 2017.

[8] Z. Luo and Q. Zhu, “Micro-frontends architecture: A systematic review,” *IEEE Access*, vol. 9, 2021.

[9] F. O. Source, “React documentation,” *React.dev*, 2023.

[10] react-microfrontends, “React microfrontends example with single-spa,” GitHub repository, 2024, acesso em: 16 fev. 2026. [Online]. Available: https://github.com/react-microfrontends

[11] module-federation/module-federation-examples, “Module federation examples,” GitHub repository, 2024, acesso em: 16 fev. 2026. [Online]. Available: https://github.com/module-federation/module-federation-examples

[12] R. T. Fielding, J. Gettys, J. C. Mogul, H. F. Nielsen, L. Masinter, P. J. Leach, and T. Berners-Lee, *HTTP: The Definitive Guide*. Sebastopol: O’Reilly Media, 2014.

[13] N. Inc., “Nginx documentation,” *nginx.org*, 2023.

[14] S. Newman, *Building Microservices*, 2nd ed. Sebastopol, CA: O’Reilly Media, 2021.

[15] Module Federation Core Team, “Issue #3153: Problems with shared dependencies in module federation,” https://github.com/module-federation/core/issues/3153, 2024, acesso em: 15 jan. 2026.

[16] MDN Web Docs, “Http response status codes,” Mozilla Developer Network Documentation, 2026, acesso em: 16 fev. 2026. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Status

[17] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. Cambridge, MA: MIT Press, 2009.

[18] R. S. Pressman and B. R. Maxim, *Software Engineering: A Practitioner’s Approach*, 8th ed. New York: McGraw-Hill Education, 2016.

[19] *ISO 9241-210: Ergonomics of human-system interaction — Part 210: Human-centred design for interactive systems*, International Organization for Standardization Std., 2019.

[20] M. Richards and N. Ford, *Fundamentals of Software Architecture*. Sebastopol, CA: O’Reilly Media, 2020.

A Configuração do Module Federation no Host

O presente apêndice descreve a configuração adotada para habilitar o consumo de componentes remotos em uma aplicação do tipo *host*, desenvolvida em React, utilizando Webpack e o mecanismo de *Module Federation*. As configurações apresentadas são responsáveis por definir os módulos remotos consumidos, bem como as dependências compartilhadas entre as aplicações envolvidas.

A.1 Arquivo `mfe.js`

Este arquivo concentra as definições do *Module Federation* utilizadas pela aplicação *host*. Nele, são especificados o identificador da aplicação, os módulos remotos consumidos e as dependências compartilhadas, garantindo a reutilização de bibliotecas comuns e evitando conflitos de versão entre as aplicações integradas.

```
const deps = require("./package.json").dependencies;

const MODULE_FEDERATION_CONFIG = {
  name: "host",
  filename: "remoteEntry.js",
  remotes: {
    navigation: "navigation@http://localhost:3001/remoteEntry.js"
  },
  shared: {
    ...deps,
    react: {
      singleton: true,
      eager: true,
      requiredVersion: deps.react
    },
    "react-dom": {
      singleton: true,
      eager: true,
      requiredVersion: deps["react-dom"]
    },
    "styled-components": {
      singleton: true,
      eager: true,
      requiredVersion: deps["styled-components"]
    }
  }
};

module.exports = { MODULE_FEDERATION_CONFIG };
```

Listing 1: Arquivo `mfe.js`

A.2 Arquivo `webpack.config.js`

Este arquivo integra a configuração do *Module Federation* ao processo de empacotamento do Webpack. Nele são definidos a URL pública da aplicação *host*, o ponto de entrada da aplicação, as configurações do servidor de desenvolvimento, os *loaders* responsáveis pelo processamento dos arquivos e os *plugins* utilizados, com destaque para o `ModuleFederationPlugin` e o `HtmlWebPackPlugin`.

```
const HtmlWebPackPlugin = require("html-webpack-plugin");
const ModuleFederationPlugin = require("webpack/lib/container/ModuleFederationPlugin");
const { MODULE_FEDERATION_CONFIG } = require("./mfe");
const path = require("path");

module.exports = {
  output: {
    publicPath: "http://localhost:8008/",
  },
  entry: "./src/index.tsx",
  resolve: {
    extensions: [".tsx", ".ts", ".jsx", ".js", ".json"],
  },
  devServer: {
    port: 8008,
    historyApiFallback: {
      disableDotRule: true,
      verbose: true,
    },
    static: {
      directory: path.resolve(__dirname, "dist"),
    },
  },
  module: {
    rules: [
      {
        test: /\.m?js/,
        type: "javascript/auto",
        resolve: {
          fullySpecified: false,
        },
      },
      {
        test: /\.(css|s[ac]ss)$/i,
        use: ["style-loader", "css-loader", "postcss-loader"],
      },
      {
        test: /\.(ts|tsx|js|jsx)$/,
        exclude: /node_modules/,
        use: {
          loader: "babel-loader",
        },
```

```
      },
    ],
  },
  plugins: [
    new ModuleFederationPlugin(MODULE_FEDERATION_CONFIG),
    new HtmlWebPackPlugin({
      template: "./src/index.html",
    }),
  ],
};
```

Listing 2: Arquivo `webpack.config.js`

B Configurações de Nginx utilizadas na Prova de Conceito da integração via Proxy Reverso

Abaixo, segue a configuração utilizada para habilitar a comunicação da aplicação consumidora com a aplicação remota.

```
server {
    listen 80;

    # Permitir somente /remote/add-product e seus assets
    location ^~ /remote/add-product {
        proxy_pass http://172.19.0.3:8080/add-product;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    # Permitir também os assets estáticos usados pela página
    location ^~ /remote/assets/ {
        proxy_pass http://172.19.0.3:8080/assets/;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    # Bloquear todo o restante do microfrontend
    location ^~ /remote/ {
        return 403;
    }

    # Rota padrão para o shell ou host app
    location / {
        proxy_pass http://172.19.0.2:8080/;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
    }
}
```

C Configurações de Nginx utilizadas para acessar tela “Adicionar Produto” via Proxy Reverso

Abaixo, segue a configuração utilizada para habilitar a comunicação da aplicação consumidora com a aplicação remota.

```
server {
    listen 80;

    # Permitir somente /remote/add-product e seus assets
    location ^~ /remote/add-product {
        proxy_pass http://172.19.0.3:8080/add-product;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    # Permitir também os assets estáticos usados pela página
    location ^~ /remote/assets/ {
        proxy_pass http://172.19.0.3:8080/assets/;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    # Bloquear todo o restante do microfrontend
    location ^~ /remote/ {
        return 403;
    }

    # Redirecionamento para rota de adicionar produto em Partners
    location ^~ /partners/suppliers/catalog/product/add-product {
        proxy_pass
            http://host.docker.internal:9000/partners/suppliers/catalog/product/add-product;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    # Rota padrão para o shell ou host app
    location / {
        proxy_pass http://172.19.0.2:8080/;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
```

```
        proxy_cache_bypass $http_upgrade;
    }
}
```

D Componente responsável por fazer controle de visualização de conteúdo

Abaixo, segue a implementação do componente responsável por ocultar ou apresentar informações de acordo com o domínio da página.

```
import { MarketplaceSharingHandlerProps } from './types'
import { useMemo } from 'react'
import { MarketplaceUseCaseProps } from './types'

// Mensagem de erro caso a aplicação não encontre a variável de ambiente
const MKT_ORIGIN_ENV_NOT_DEFINED =
  'A variável de ambiente EXPECTED_MKT_ORIGIN não foi definida. Esta variável é
      necessária para validar a origem do Marketplace. Verifique o arquivo .env ou as
      configurações do ambiente.'

export const useMarketplaceValidation = (
  behaviour: 'hide-to-marketplace' | 'show-to-marketplace'
): MarketplaceUseCaseProps => {
  const isMarketplaceOrigin = useMemo(() => {
    const appOrigin = window.origin

    if (!process.env.EXPECTED_MKT_ORIGIN) {
      throw new Error(MKT_ORIGIN_ENV_NOT_DEFINED)
    }

    return appOrigin === process.env.EXPECTED_MKT_ORIGIN
  }, [window.origin])

  const componentMustBeVisible = useMemo(() => {
    return behaviour === 'show-to-marketplace'
      ? isMarketplaceOrigin
      : !isMarketplaceOrigin
  }, [isMarketplaceOrigin, behaviour])

  return { isMarketplaceOrigin, componentMustBeVisible }
}

export const MarketplaceSharingHandler = ({
  behaviour,
  children,
}: MarketplaceSharingHandlerProps) => {
  const { componentMustBeVisible } = useMarketplaceValidation(behaviour)

  return componentMustBeVisible ? <>{children}</> : <></>
}
```