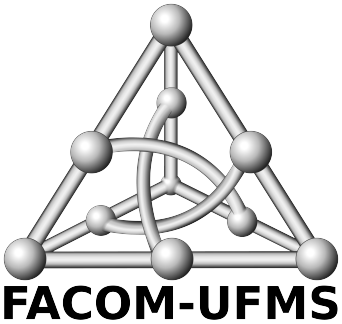


Universidade Federal de Mato Grosso do Sul
Faculdade de Computação
Mestrado Profissional em Computação Aplicada

Integração de Módulos Utilizando Micro Frontends na Plataforma +Precoce

Alan Balen Schio

Campo Grande - MS, 13 de Março de 2023

Alan Balen Schio

Integração de Módulos Utilizando Micro Frontends na Plataforma +Precoce

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação Aplicada da Faculdade de Computação/FACOM pela Universidade Federal do Mato Grosso do Sul/UFMS, como parte dos requisitos para a obtenção do título de Mestre em Computação Aplicada.

Universidade Federal do Mato Grosso do Sul – UFMS

Faculdade de Computação

Mestrado Profissional em Computação Aplicada

Orientador: Prof. Dr. Evandro Mazina Martins

Coorientador: Prof. Dr. Milton Ernesto Romero Romero

Campo Grande - MS

13 de Março de 2023

Alan Balen Schio

Integração de Módulos Utilizando Micro Frontends na Plataforma +Precoce/ Alan Balen Schio. – Campo Grande - MS, 13 de Março de 2023-

?? p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Evandro Mazina Martins

Dissertação (Mestrado) – Universidade Federal do Mato Grosso do Sul – UFMS
Faculdade de Computação
Mestrado Profissional em Computação Aplicada, 13 de Março de 2023.

1. Micro Frontends. 2. Plataforma +Precoce. I. Prof. Dr. Evandro Mazina Martins II. Universidade Federal do Mato Grosso do Sul. III. Faculdade de Computação. IV. Integração de Módulos Utilizando Micro Frontends na Plataforma +Precoce

Alan Balen Schio

Integração de Módulos Utilizando Micro Frontends na Plataforma +Precoce

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação Aplicada da Faculdade de Computação/FACOM pela Universidade Federal do Mato Grosso do Sul/UFMS, como parte dos requisitos para a obtenção do título de Mestre em Computação Aplicada.

Trabalho aprovado. Campo Grande - MS, 0 13 de Março de 2023:

Prof. Dr. Evandro Mazina Martins
Orientador

Me. Fernando Rodrigues Teixeira Dias
Convidado

Dr. Camilo Carromeu
Convidado

Prof. Dr. Gedson Faria
Convidado

Campo Grande - MS
13 de Março de 2023

Para criança que um dia fui, que até em seus melhores sonhos, achava impossivel ser o adulto que me tornei.

Agradecimentos

Agradeço a minha família por me apoiar e incentivar a sempre seguir o caminho dos estudos. Principalmente a minha mãe Selia por sempre me mostrar e fazer enxergar a importância do exemplo, o valor do meu esforço, as conquistas e de seus significados toda vez que pensei em desistir durante esse período. Ao meu sobrinho que mesmo criança muitas vezes trouxe paz para minha cabeça pensante e que inúmeras vezes foi o motivo de seu tio querer ser alguém melhor, mais instruído e mais sábio.

A minha eterna professora e hoje amiga Priscila Silva Martins por sempre me incentivado em continuar estudando, buscando e nunca desistindo. Agradeço mais do que o incentivo, por ser um exemplo.

Agradeço ao meu orientador, Evandro Mazina Martins, por todos os conselhos, pela paciência pelas conversas e ajuda durante esse período, suas palavras sempre foram muito importantes para mim.

Ao professor Milton por várias vezes me lembrar que independente do que estava sendo feito e para que ou quem, este trabalho era mais que tudo uma extensão de mim.

Ao Dr. Camilo e Me. Fernando por terem conseguido entender meus devaneios sobre o tema e projeto quando nem eu mesmo conseguia explicar tão bem a minha ideia.

Ao Me. Fernando por todas as reuniões, e-mails, mensagens, ligações, conversas, explicações e discussões que juntas fizeram com que esse trabalho chegasse ao seu desenvolvimento.

A Mayara Alencar e Tássio Jordão, por terem plantado a semente do Micro Frontends em minha mente, o que fez que eu tivesse tanto apego, vontade e gosto por este trabalho. E por ao longo desses dois anos termos realizado inúmeras conversas, testes e aprendizados a cerca do tema. Nunca se sabe quando algo impacta a vida de alguém, e vocês com certeza impactaram.

A Stephany, por todo amor, carinho e acalento ao longo deste último ano, que apesar de ter sido repleto de angustias, nervosismo, frustrações, se tornou imensamente mais fácil por ter você do lado. Obrigado por me lembrar quase que diariamente o valor do que estava fazendo.

Aos todos amigos e familiares que ao longo desses dois anos não estive tão presente ou bem humorado, obrigado por entenderem e me apoiar.

A todos que fazem da computação e tecnologia uma área de impacto na vida das pessoas.

Por último aquele que se tornou minha maior referência desde os primeiros dias de aula na graduação até hoje: Alan Mathison Turing. Que o mundo inteiro possa um dia conhecer e reconhecer sua grandeza, obrigado Alan.

Resumo

A Plataforma +Precoce vem sendo desenvolvida ao longo dos últimos anos em parceria entre Embrapa e a Universidade Federal do Mato Grosso do Sul, de maneira modularizada, onde cada novo desenvolvimento é realizado fora da base de código principal, evitando conflitos e efeitos colaterais com demais códigos do sistema que se encontra em uso. Ao fim de cada desenvolvimento, estes módulos não se incorporavam ao código final, seja por causa das versões de dependências conflitantes ou por alta demanda operacional de reavaliar os códigos. A falta de entendimento de uma técnica eficiente de unificação de código também estava presente. A técnica de micro serviços é amplamente difundida e utilizada para se fazer o gerenciamento e conexão dos módulos de um sistema sem que necessite estar dentro do mesmo código. Derivada desta, nasceu a abordagem Micro Frontends, que permite no desenvolvimento de plataformas web a integração dos módulos em tempo de execução. Se valendo da técnica de Micro Frontends, da maneira modular com que a plataforma foi desenvolvida e da necessidade de disponibilizar as novas funcionalidades ao usuários, este trabalho provê a integração entre os módulos da plataforma utilizando a técnica Module Federation para realizar a integração entre os módulos com a plataforma principal, e fornecendo maneiras de validar a disponibilidade dos módulos e garantir que seu uso seja possível mesmo em modo offline.

Palavras-chaves: frontend. micro frontends. computação distribuída. webpack. module federation. plataforma +precoce. engenharia de software.

Abstract

The +Precoce Platform has been developed over the last few years in partnership between Embrapa and the Federal University of Mato Grosso do Sul, in a modularized way, where each new development is carried out outside the main code base, avoiding conflicts and side effects with other system codes that are in use. At the end of each development, these modules were not incorporated into the final code, either because of conflicting dependency versions or the high operational demand of reevaluating the codes. The lack of understanding of an efficient code unification technique was also present. The micro services technique is widely spread and used to manage and connect the modules of a system without needing to be within the same code. Derived from this, the Micro Frontends approach was born, which allows the integration of modules at runtime in the development of web platforms. Taking advantage of the Micro Frontends technique, the modular way in which the platform was developed and the need to make new features available to users, this work provides integration between the platform modules using the Module Federation technique to perform the integration between modules with the core platform, and providing ways to validate the availability of modules and ensure their use is possible even in offline mode.

Keywords: frontend. micro frontends. distributed computing. webpack. module federation. +precoce.

Lista de ilustrações

Figura 1 – Interface da P+P. Fonte: (**??**). 6
Figura 2 – Visualização da interface do Módulo Fluxograma. Fonte: (**??**). 6
Figura 3 – Visualização da interface do Módulo Cronograma. Fonte: (**??**). 7
Figura 4 – Visualização da interface do Módulo Otimização Matemática. Fonte: (**??**) 7
Figura 5 – Visualização da interface do Módulo Análise de Sensibilidade. Fonte: (**??**) 8
Figura 6 – Visualização da interface do Gestor. Fonte: Autor 8
Figura 7 – Exemplos de arquitetura monólito e micro serviços. Fonte: Autor . . . 9
Figura 8 – Exemplos de Micro Frontends. Fonte: (**??**) 11
Figura 9 – Micro Frontends com *iframe*. Fonte: (**??**) 13
Figura 10 – Exemplo de integração visual de *host* e *remotes*. Fonte: Autor 14
Figura 11 – Papeis de cada projeto na integração. Fonte: Autor 19
Figura 12 – Configuração *remotes* relativa aos dados na configuração *host*. Fonte: Autor 23
Figura 13 – Janela do Fluxograma sendo exibida na Aplicação Gestor. Fonte: Autor 24
Figura 14 – Opção de Fluxograma nas ações. Fonte: Autor 25
Figura 15 – Visualização do código e da interface do componente ***Optimization.vue***. Fonte: Autor 27
Figura 16 – Novo componente reutilizável de gráfico. Fonte: Autor 29
Figura 17 – Componente SensitivityAnalysis.vue. Fonte: Autor 30
Figura 18 – Lista de opções. Fonte: Autor 30
Figura 19 – Novo componente do gráfico em Janela de Diálogo. Fonte: Autor 31
Figura 20 – Código do novo componente ChartDialog. Fonte: Autor 31
Figura 21 – Código do novo componente reutilizável do gráfico. Fonte: Autor 31
Figura 22 – Utilização de componente dinâmico de formulário. Fonte: Autor 32
Figura 23 – *Card* de simulação. Fonte: Autor 34
Figura 24 – Visualização do componente de opções. Fonte: Autor 35
Figura 25 – *Host* pwa carregando arquivos dos seus *remotes*. Fonte: Autor 37
Figura 26 – *Host* pwa exibindo componente carregado do *remote* pluginGantt. Fonte: Autor 37
Figura 27 – *Host* manager exibindo arquivos de inicialização do *remote* pluginFlow. Fonte: Autor 38
Figura 28 – *Host* manager exibindo componente carregado do *remote* pluginFlow. Fonte: Autor 38
Figura 29 – *Host* sem prejuízo de inicialização mesmo com *remotes offline*. Fonte: Autor 43

Figura 30 – *Remotes* pluginFlow *offline* e *host* manager não disponibilizando sua funcionalidade. Fonte: Autor 44
Figura 31 – *Remotes offline* e *host* pwa exibindo somente funcionalidade do *remote* disponível. Fonte: Autor 45
Figura 32 – *Host* pwa carregando todos os arquivos do *remotes* pluginGantt para uso *offline*. Fonte: Autor 49
Figura 33 – *Host* pwa sendo utilizado sem internet. Fonte: Autor 50
Figura 34 – *Host* sendo utilizado sem internet e exibindo funcionalidade de Cronograma do *remote* pluginGantt. Fonte: Autor 51
Figura 35 – Pastas dos projetos *host* e *remote*. Fonte: Autor 63
Figura 36 – Configuração webpack do projeto *host*. Fonte: Autor 66
Figura 37 – Configuração webpack do projeto *remote*. Fonte: Autor 67
Figura 38 – *Build* do projeto *remote*. Fonte: Autor 69
Figura 39 – Importação circular. Fonte: (**??**) 71

Lista de abreviaturas e siglas

MFE	Micro Frontend
MF	Module Federation
MFP	ModuleFederationPlugin
MFEP	ModuleFederationEnhancedPlugin
P+P	Plataforma + Precoce
PWA	*Progressive Web Apps*
SPA	*Single Page Application*
BD	Banco de dados
BE	*Backend*
FE	*Frontend*
MS	Micro Serviços
HTML	*HyperText Markup Language*
HTTP	*Hypertext Transfer Protocol*
URL	*Uniform Resource Locator*

Sumário

Introdução **1**
Motivação **2**
Objetivos **2**

1 FUNDAMENTAÇÃO TEÓRICA **5**
1.1 Pecuária e Sistemas de Produção **5**
1.2 Plataforma +Precoce **5**
1.2.1 Aplicação Portal 5
1.2.2 Módulo Fluxograma 5
1.2.3 Módulo Cronograma 6
1.2.4 Módulo Otimização Matemática 7
1.2.5 Módulo Análise de Sensibilidade 7
1.2.6 Aplicação Gestor 7
1.3 Micro Serviços **8**
1.4 Micro Frontends **9**
1.4.1 Benefícios 10
1.4.2 Desafios 11
1.4.3 Implementação 12
1.5 Module Federation **13**
1.6 ModuleFederationPlugin **15**

2 TRABALHOS RELACIONADOS **17**

3 INTEGRAÇÃO **19**
3.1 Adequação de Projetos **20**
3.1.1 Utilização do webpack 20
3.1.2 Pré integração 20
3.2 Integração Aplicação Gestor **21**
3.2.1 Remote pluginFlow 21
3.2.2 Host manager 22
3.3 Integração Aplicação Portal **25**
3.3.1 Remote pluginGantt 25
3.3.2 Remote pluginOptimize 26
3.3.3 Remote pluginSensitivity 28
3.3.4 Host pwa 33
3.4 Monitoramento **35**

3.5 **Disponibilização** **36**
3.6 **Resultados da Integração** **37**

4 **CONTROLE DE INDISPONIBILIDADE** **39**
4.1 **Plugin ModuleFederationEnhancedPlugin** **39**
4.2 **Inicialização Assíncrona** **40**
4.3 **Disponibilização Offline** **45**
4.4 **Resultados do Controle de Disponibilidade** **48**
4.5 **Considerações Finais** **48**

5 **MELHORIAS NO MÓDULO FLUXOGRAMA** **53**

Resultados e Conclusão **55**

Trabalhos Futuros **57**

REFERÊNCIAS **59**

APÊNDICES **61**

APÊNDICE A – MODULEFEDERATIONPLUGIN **63**
A.0.1 Configuração 63
A.0.2 Dependências 64
A.0.3 Configuração 64
A.0.4 Funcionamento 67
A.0.5 Compilação 68
A.0.6 Execução 69

APÊNDICE B – ATUALIZAÇÕES E ADAPTAÇÕES DOS PROJETOS **73**
B.1 **plugin-sensitivity** **73**
B.1.1 Dependências 73
B.2 **plugin-optimize** **73**
B.2.1 Dependências 73
B.2.2 Adaptação 74
B.3 **plugin-flow** **74**
B.3.1 Dependências 74
B.3.2 Adaptação 74

APÊNDICE C – CRIAÇÃO DO PROJETO PLUGIN-GANTT . . . **75**

APÊNDICE D – DEPENDÊNCIAS QUE UTILIZAM O WEBPACK 77
D.1 Compoentente CardSimulationMenu 77

APÊNDICE E – HELPER DE CARREGAMENTO DINÂMICO DE MÓDULOS 81

ANEXOS 85

ANEXO A – MANUAL DE INTEGRAÇÃO EM NOVOS MÓDULOS 87

ANEXO B – DECLARAÇÃO DE AUTORIA E RESPONSABILIDADE 89

Introdução

O crescimento da área de tecnologia nos últimos anos se faz por números expressivos, desde o início da pandemia o crescimento chegou a mais de 600% (**??**), isso nos leva a pensar como lidar com esse crescimento também dentro dos times de tecnologia e nas linhas de código. A habilidade de lidar com esse crescimento sem que se perca o desempenho e capacidade é conhecida por escalabilidade (**??**) e se torna cada vez mais necessária para o crescimento de times de tecnologia e a sua divisão continue possibilitando atuação simultânea no desenvolvimento de ferramentas, e que esses times consigam trabalhar na mesma base de código ou projeto sem que a alteração de um time comprometa o que foi realizado pelo outro.

Enquanto de um lado existem as metodologias ágeis oferecidas como ferramenta organizacional para gerenciamento de projetos e pessoas, do outro existem desenvolvedores e linhas de código que buscam também não serem impactados. Garantir uma experiência de desenvolvimento adequada e uma boa produtividade aos desenvolvedores de software caminha lado a lado com os avanços tecnológicos que se tornam disponíveis nas linguagens e ferramentas de desenvolvimento. Enquanto melhorado o desempenho na execução de um código, sua qualidade e confiabilidade, deve-se permitir que escalabilidade também se faça presente, permitindo com que times atuem em um mesmo projeto de maneira fluida e sem barreiras. Um passo importante para garantir essa escalabilidade no time de desenvolvimento é fazer com que o mesmo se torne modular, isso implica em transformar uma grande e única base de código, conhecida por monólito, em pequenas ou médias partes que se unem umas aos outras, ou a uma determinada parte principal que rege os demais, em tempo de execução de código conforme a necessidade. Utilizando arquitetura de Micro Serviços, se torna possível realizar o desenvolvimento modular de um sistema que seja possível obter pontos positivos desse tipo de desenvolvimento sem que se perca produtividade de desenvolvedores ou que se necessite de uma infraestrutura mais robusta. Outra maneira de utilizar-se do desenvolvimento modular é quando se possui o desenvolvimento em etapas, assim, uma nova parte do sistema é desenvolvida sem que isso afete as que já estão em funcionamento e que essa se integra as demais no momento necessário.

A Plataforma +Precoce é uma ferramenta *online* que visa a organização e disponibilização de informações de qualidade relacionadas a economia e ambiente, para auxiliar a tomada de decisão dos produtores rurais visando melhorar o sistemas de cria, recria e engorda de bovinos (**??**). Além de sua contribuição para a pecuária, traz também contribuições tecnológicas pois o seu desenvolvimento que vem sendo feito ao longo dos anos (**????????**) encontra-se realizado de maneira modular, porém seus módulos ainda não estão integrados.

Este trabalho utiliza a abordagem Micro Frontend (MFE), baseada na arquitetura de Micro Serviços, com a ferramenta *Webpack ModuleFederationPlugin* (MFP) para implementar a integração dos módulos da Plataforma +Precoce, garantindo que todos os módulos atuais se integrem a plataforma principal de maneira simples e eficaz, garantindo que o projeto continue tendo sua premissa de funcionamento sem conexão para os agricultores no campo.

Motivação

A plataforma +Precoce(P+P) desenvolvida em parceria da Empresa Brasileira de Pesquisa Agropecuária (Embrapa) com a Faculdade de Computação (FACOM) da Universidade Federal de Mato Grosso do Sul (UFMS) desde 2018 (**??**). A plataforma que fornecerá para produtores e técnicos rurais modelos matemáticos de sistemas de produção pecuária. Atualmente a P+P se divide em duas aplicações web, Portal e Gestor, e mais quatro módulos de funcionalidade, Cronograma, Fluxograma, Análise de Sensibilidade e Otimização. Os módulos encontram-se desenvolvidos, porém não estão integrados, fazendo com que tais funcionalidades não estejam disponíveis ao usuário.

Objetivos

O Objetivo geral deste trabalho é a integração dos módulos desenvolvidos para a P+P utilizando Module Federation. Para atingir este objetivo geral, deve-se atingir os seguintes objetivos específicos:

- Atualizar dependências dos projetos e os adequar ao padrão da empresa;
- Garantir que a indisponibilidade de um módulo não afete as funcionalidades restantes da aplicação;
- Desenvolver uma administração de módulos para uso sem conexão a internet;
- Garantir a continuidade de funcionamento dos módulos após a integração;
- Elaborar um guia de configuração para criação de novos módulos;
- Fornecer um exemplo de uma nova arquitetura distribuída para aplicações *frontend*;
- Proporcionar ao programa de Mestrado Profissional uma nova abordagem utilizando Micro Frontends, para impulsionar a comunidade e promover inovações sobre o desenvolvimento *frontend*;
- Produzir conteúdo acadêmico sobre o tema Micro Frontends e Module Federation.

Além dos objetivos relacionados à integração entre os módulos, também foram realizadas melhorias no módulo Fluxograma.

1 Fundamentação Teórica

1.1 Pecuária e Sistemas de Produção

O Brasil possui o maior rebanho bovino comercial do mundo, além de ser um dos líderes mundiais na produção e exportação de carne (**??**). Para que esses números possam crescer cada vez mais, é imprescindível que se possuam bons sistemas de produção pecuária. Os sistemas de produção são resultados de pesquisas e desenvolvimentos que definem modelos e padrões para obter melhores resultados.

1.2 Plataforma +Precoce

A Embrapa começou o desenvolvimento da Plataforma +Precoce(P+P) para que o conhecimento de décadas de pesquisa sobre sistemas de produção (**??**) sejam melhor utilizadas pelos produtores. A P+P é uma ferramenta que disponibiliza para produtores e técnicos rurais modelos matemáticos de sistemas de produção pecuária que são o resultado das pesquisas realizadas pela Embrapa por meio de simulações de sistemas de produção (**????????**).

As próximas seções apresentam os módulos desenvolvidos para a P+P.

1.2.1 Aplicação Portal

Desenvolvida no trabalho de **??**), a aplicação fornece o cadastro e configuração de uma simulação. Desenvolvido inicialmente utilizando AngularJS o projeto evoluiu dentro da própria Embrapa e hoje utiliza o *framework* Vue.jS para incorporar os conceitos mais atualizados e fazer uso das possibilidades do modelo de desenvolvimento conhecido como Progressive Web Apps, que pode ser visto na Figura **??**, este inclusive é o motivo de que o projeto em desenvolvimento possui o nome de pwa (**??**).

O Portal oferece a funcionalidade de Simulação, se comunica com o servidor e faz o carregamento, a criação e a edição de simulações.

1.2.2 Módulo Fluxograma

O módulo Fluxograma visa a criação de modelos de sistema de produção utilizados pela plataforma utilizando ferramentas gráficas para realizar o desenho de fluxogramas. O módulo utiliza a biblioteca D3.js que cria representações gráficas utilizando vetores escalares e tags HTML, com ele foram criadas representações de grafos orientados onde as

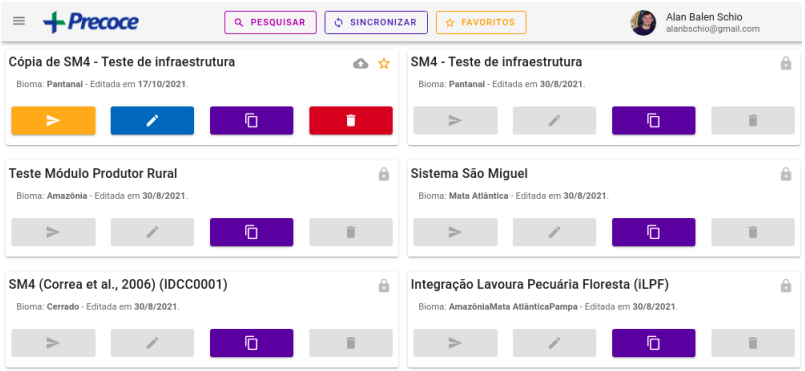


Figura 1 – Interface da P+P. Fonte: (**??**).

arestas representam fluxos de animais, insumos e produtos (**??**). Inicialmente o projeto foi desenvolvido para ser chamado a partir da aplicação P+P dentro do módulo Portal, contudo após definições internas da Embrapa foi decidido que o módulo passa a ser integrado a Aplicação Gestor, explicado na seção **??**. A Figura **??** traz um exemplo de fluxograma que pode ser criado com este módulo.

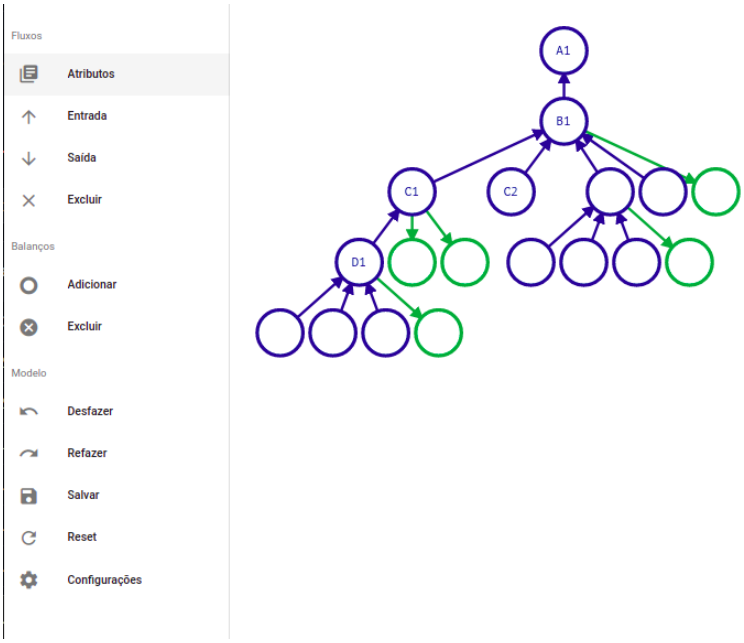


Figura 2 – Visualização da interface do Módulo Fluxograma. Fonte: (**??**).

1.2.3 Módulo Cronograma

Este módulo exibe os resultados do simulador do P+P na forma de cronogramas de implantação de sistema, isto é, o gráfico de Gantt (**??**). A interface do módulo pode ser visto na Figura **??**.

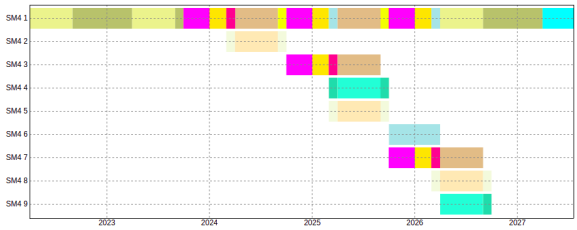


Figura 3 – Visualização da interface do Módulo Cronograma. Fonte: (**??**).

1.2.4 Módulo Otimização Matemática

Este módulo gera para o usuário um arquivo de *script* na linguagem matemática *AMPL* (**??**) para experimentos de otimização matemática do sistema de produção. A interface do módulo pode ser visto na Figura **??**.

Figura 4 – Visualização da interface do Módulo Otimização Matemática. Fonte: (**??**)

1.2.5 Módulo Análise de Sensibilidade

O módulo de Análise de Sensibilidade foi desenvolvido por **??**) e fornece a funcionalidade de identificar o quanto o resultado de um indicador pode ser influenciado por um ou mais parâmetros. O usuário consegue visualizar essa influência a partir de quatro tipos diferentes de gráficos existentes na funcionalidade. A interface do módulo pode ser visto na Figura **??**.

1.2.6 Aplicação Gestor

O Gestor fornece uma plataforma de parametrização e criação de sistemas para os responsáveis e pesquisadores. A interface do Gestor pode ser visto na Figura **??**.

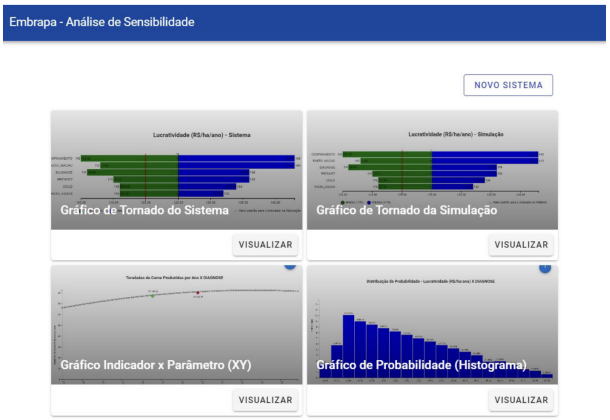


Figura 5 – Visualização da interface do Módulo Análise de Sensibilidade. Fonte: (**??**)

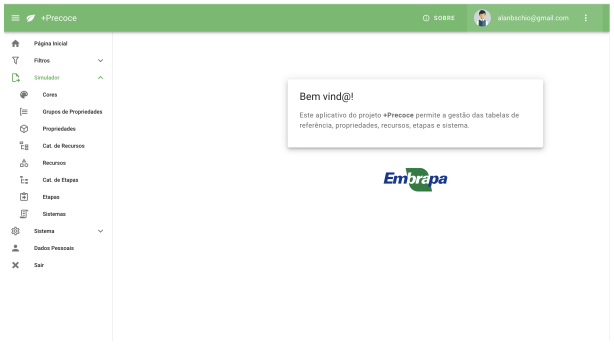


Figura 6 – Visualização da interface do Gestor. Fonte: Autor

1.3 Micro Serviços

Durante muitos anos o padrão de desenvolvimento mundial foi o que se conhecia por monólito, ou arquitetura monolítica, um único e grande projeto onde todos realizavam suas contribuições. Com o passar do tempo novos paradigmas foram aparecendo entre eles o de divisão dos projetos e micro serviços. Com o movimento ágil, a quebra de grande times em pequenos grupos onde cada qual tendo uma única responsabilidade, sentiu-se a necessidade de que os projetos e seus códigos também adotassem essa mesma abordagem.

Assim surgiu a arquitetura de micro serviços trazendo interoperabilidade, flexibilidade e baixo acoplamento que a mesma possibilidade em serviços de *backend*. Dentre seus pontos positivos o que mais se destaca é a independência que a mesma possibilita tanto em quesitos técnicos, como tecnologias, padrões de desenvolvimento, mas também em negócios, como uma maior liberdade e autonomia do time de desenvolvimento. Sendo a arquitetura mais indicada visando escalabilidade, é preferível essa abordagem em vez da monolítica, por esta ser de difícil dimensionamento e ainda conter problemas ao se desenvolver diferentes partes simultaneamente (**??**).

Na arquitetura de micro serviços cada módulo do sistema é desenvolvido de maneira isolada e, a nível de código, em vários projetos diferentes, de maneira distinta a da

arquitetura de monólito onde toda a base de código se encontra em um mesmo projeto, como na Figura **??**.

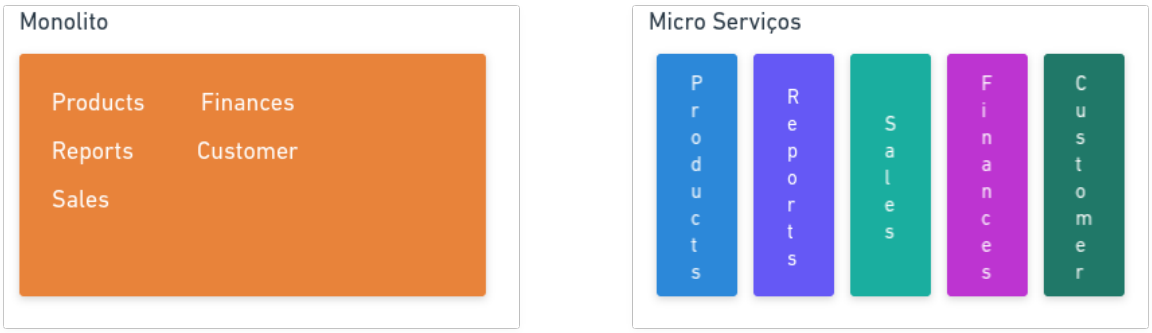


Figura 7 – Exemplos de arquitetura monólito e micro serviços. Fonte: Autor

1.4 Micro Frontends

Evoluindo cada vez mais rápido e constante nos últimos anos, e com o crescente número de usuários nos mais diversos lugares e condições tecnológicas, as áreas de desenvolvimento web se encontraram em uma posição onde não só suas arquitetura e abordagens necessitavam de constante evolução, mas também a maneira como um produto ou ferramenta é entregue ao seu usuário final. Sabemos que as páginas web possuem os mais diferenciados nichos e que com isso tem-se cada vez mais a necessidade de seu carregamento e fornecimento em um tempo menor, garantindo que seu usuário interaja de maneira mais rápida possível.

Técnicas como o *Chunk Splitting*, (**??**) surgiram para que seja possível que o código de um projeto seja dividido em artefatos menores e pontuais sendo carregados no momento exato que são necessários. Ainda assim, a equipe como um todo e a base de código continua sendo dependente e unificada. Sabendo dos benefícios existentes na arquitetura de micro serviços pode-se imaginar que em algum momento seriam desejados para o *frontend*, porém por ser utilizado no lado cliente, na maquina do usuário, existem preocupações quanto a desempenho e performance. Esta abordagem esteve durante bons anos apenas como hipótese e possibilidade. Em novembro de 2016, a (**??**) listou em seu relatório anual de tendências pela primeira vez o termo *Micro Frontends*. Já em seu artigo digital (**??**) explica que percebeu-se que existia um padrão emergindo na maneira que as empresas estavam fazendo *frontend*, mais desacoplado onde seus pequenos *pedaços* pudessem ser desenvolvidos, testados e implementados de maneira independente e ainda assim se apresentado ao usuário como um único produto.

??) definem em seu livro que os dois principais atributos de um MFE são auto-suficiente e independência de implantação. Um MFE deve ser auto-suficiente porque toda a

lógica de negócios que é específica para sua função e comportamento deve ser internalizadas em si, e independência de implantação se dá pela disponibilização do software, neste caso, a implantação de um MFE deve ocorrer de maneira que o mesmo possa ser acessado e, de certo modo até mesmo, utilizado de maneira independente, sem que haja dependência de outro software ou módulo.

Mesmo com o devido entendimento dos benefícios da utilização de Micro Frontends e a adoção de grandes empresas como Ikea, Starbucks, DAZN (**??**), Amazon (**??**), ainda existe uma certa desconfiança acerta do tema, vários autores como (**??**), (**??**), (**??**), (**??**) e (**??**) trazem em seus trabalhos o benefícios e desafios existentes sobre o tema de Micro Frontends e as adversidades existente para essa abordagem.

Algo importante de ser pontuado é que Micro Frontends compartilham as mesmas dores e vitórias da arquitetura de micro serviço. A sua adoção não pode ser embasada apenas em questões tecnológicas, como frisado por (**??**), a estruturação de times, divisões organizacionais, tomada de decisões, e até mesmo desenvolvimentos paralelos ou com escopo estritamente delimitado são fatores a serem levados em consideração ao escolher uma arquitetura, **??**) em seu livro traz um ótimo exemplo de uso e separação de times voltado para a arquitetura de Micro Frontends, onde cada um dos times realiza o desenvolvimento de uma parte do sistema, que se integram em uma única tela, isso pode ser visto na Figura **??**, no canto superior direito a imagem de uma tela de computador trazendo conteúdo desenvolvido por time A, B e C.

1.4.1 Benefícios

Os benefícios existentes são semelhantes ao da arquitetura de micro serviços, de maneira macro temos quatro principais áreas de benefícios:

Independência tecnológica: cada time possui total independência de qual tecnologia utilizar para fazer sua parte, desde que esta cumpra os *contratos* acordados do desenvolvimento e se integre com o projeto num todo.

Isolamento de escopo: Com a delimitação de responsabilidades e escopo de cada projeto, se tornam mais certeiras as análises, definições de modelos e identificação de problemas.

Autonomia de time: Uma vez que o escopo técnico de cada Micro Frontends definido, o time tem mais autonomia para elencar a melhor solução para um problema ou o melhor desenvolvimento de alguma funcionalidade.

Atualizações incrementais e pontuais: Possuindo cada parte do projeto isolado, se tornam mais fáceis as correções pontuais, atualizações de funcionalidades e bem como incremento de novas. Uma vez que seu escopo isolado faz com o que esta sobre responsabilidade do mesmo time não impacte os demais. Pode-se ainda abranger atualizações de

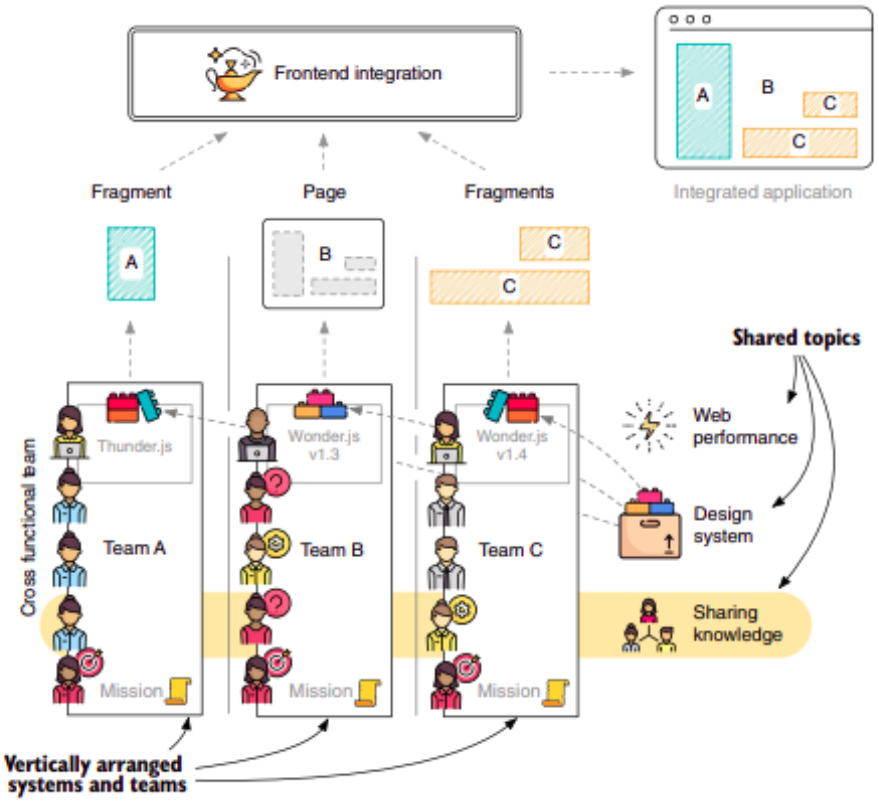


Figura 8 – Exemplos de Micro Frontends. Fonte: (**??**)

dependências ou bibliotecas utilizadas pela aplicação *web*.

1.4.2 Desafios

Mesmo com suas vantagens e com as diversas abordagens utilizadas na tentativa de atingir o melhor desenvolvimento de uma arquitetura de Micro Frontends, existem alguns pontos que são os desafios cruciais para a funcionalidade, tais como:

Consistência na integração: A aplicação num todo deve funcionar de maneira que o usuário não perceba que são varias partes separadas se unificando para criá-lo.

Consistência de interface: Para que uma aplicação se apresente como conciso para usuário, não só sua funcionalidade deve ser de maneira consistente mas sua interface também. Os times devem estar bem alinhados para que toda a aplicação siga uma mesma identidade visual.

Comunicação entre projetos: Mesmo com escopos isolados dento de cada projeto construído, em algumas ocasiões se faz necessária a comunicação entre duas ou mais partes, e para implementação dessa comunicação deve ser levada em consideração que não seja prejudicada a performance e que se faça possível nas diferentes tecnologias que possam ser utilizadas.

Duplicidade de código: Ao se possuir mais de um projeto servindo dentro de um

mesmo contexto geral, deve-se atentar-se a duplicidade de código no que tange o uso de mesmas bibliotecas ou dependências, e o desenvolvimento de trechos de código que possam ser reutilizáveis por mais de um projeto, de modo que esses se tornem compartilháveis.

Administração de diversas versões de dependências: Deve-se possibilitar a interoperabilidade de versões diferentes de bibliotecas em cada um dos projetos. Se um determinado projeto A depender de uma biblioteca X na versão 1.0 e um determinado projeto B depender da mesma biblioteca na versão 2.0, ambas necessitam estar disponíveis para os projetos e de maneira que não interfiram nos demais.

Sobrecarga operacional: Como dito anteriormente, a adoção de Micro Frontends não deve ser pautada em apenas questões técnicas, uma vez que a quebra de uma aplicação monólito em vários outros projetos pode levar a fadiga ao time, pela necessidade de troca de projeto e configuração para seu desenvolvimento. Por isso deve-se atentar ao tamanho do seu time, estrutura organizacional, ou ainda casos como o desenvolvimento da P+P, onde cada participante de um projeto, necessita de desenvolver e elaborar o seu único projeto.

1.4.3 Implementação

Ao longo dos anos diversas abordagens foram utilizadas, a primeira grande empresa que se tem conhecimento de tentar utilizar técnicas e abordagem que se assemelham ao Micro Frontends foi o Spotify (**??**), que utilizando de *iframe*, uma funcionalidade do HTML que permite incorporar páginas de outro *software web* dentro de um determinado espaço, onde cada parte do seu reprodutor de música era um projeto como pode ser visto na Figura **??**.

Figura 9 – Micro Frontends com *iframe*. Fonte: (**??**)

Também é possível adaptar funcionalidades já existentes, como por exemplo, o carregamento de informações e arquivos sob demanda, utilizando Ajax e a funcionalidade

de criação de componentes customizáveis do JavaScript da implementações custosas de integração utilizando JavaScript, os *Web Components*. Incluindo também a utilização de roteamento de servidores Nginx (**????**), onde era criado um gerenciador de tráfego, conhecido como *gateway*, onde cada URL acessa não diretamente o conteúdo do software, o *gateway* que é acessado e esse esconde as URL originais dos softwares, assim o conteúdo de um ou outro software é retornado, mas para o usuário ele sempre esta dentro da mesma URL, mesmo tendo origens diferente por trás do *gateway*. Outra abordagem é o uso do roteamento da internet fornecido pelo protocolo HTTP, onde cada diretório de uma distribuição de *software web* é uma parte da URL. Assim dentro de um devido *software web* por exemplo ***meu-software.com.br*** se dentro do servidor ele possuir o diretório **produtos** está será exibida como o *software* ao acessar ***meu-software.com.br/produtos***.

Algumas empresas ainda tentavam criar suas próprias soluções, como foi o caso da multinacional SAP que em 2020 criou o *framework* Luigi (**????**), e o caso da Canopy criando o SingleSPA (**??**) este é o *framework* mais antigo relacionado a Micro Frontends tendo sua criação datada de 2016, mesmo ano que se começou a ter conhecimento da técnica (**??**), e considerado com o maior alto estágio de maturidade (**??**), este inclusive sendo uma das ferramentas mais utilizadas para MFE. Contudo dois pontos negativos do SingleSPA são: a adaptabilidade de código e o carregamento por demanda (**??**). Em contra partida em meados de 2020 surgiu uma teoria sobre como se realizar a arquitetura de Micro Frontends, o *Module Federation* e a tecnologia baseada na mesma.

1.5 Module Federation

O Module Federation (MF) tem como objetivo resolver um dos maiores problemas do desenvolvimento *frontend*: o compartilhamento de código em sistemas distribuídos (**??**). Esta solução se trata de um modelo mental de organização e compartilhamento de código em tempo real (**??**). Este modelo mental possui dois papéis principais: os *remotes* e o *host*. Cada um desses papéis é responsável por exportar ou importar módulos, respectivamente. Um *remote* identifica os itens que serão exportados: define um nome externo para o arquivo e qual o caminho referente ao arquivo original, pseudo-código de exemplo:

```
1    exposes:{
2      "./ArquivoFinal": "./caminho/ArquivoDaFuncionalidade.js",
3    },
```

Já o *host* identifica um módulo a ser importado e o *remote* utilizado, pseudo-código de exemplo:

```
1    consumes: [
2      {
3          module: "./ArquivoFinal",
4          remote: "http://urlDoRemote.com/....",
```

```
5       }
6     ],
```

Com o MF, diferente da arquitetura de micro serviços onde cada serviço se auto-gerencia, é necessário que exista um módulo principal, afim de que este seja o responsável por gerenciar e exibir os demais. Os *remotes* são cada um dos módulos que irá se unir ao *software* principal para adicionar suas funcionalidades e se tornar, na visão do usuário, uma única plataforma ou aplicação. Os *remotes* exportam somente o que é necessário de ser conhecido externamente. Já o *host*, também conhecido como *root*, é o ponto de entrada e parte principal do seu *software web*, é este que será acessado pelo usuário e que o mesmo terá conhecimento, quem importará, incorporará e usará todos os *remotes*, irá administrá-los, organizá-los e definir onde e em que momento cada um será exibido ou utilizado. Um exemplo de orquestração de *host* e *remotes* é exibido na Figura **??** onde o *host* incorpora a sua exibição dois módulos *remotes* e os exibe como um único *software web*.

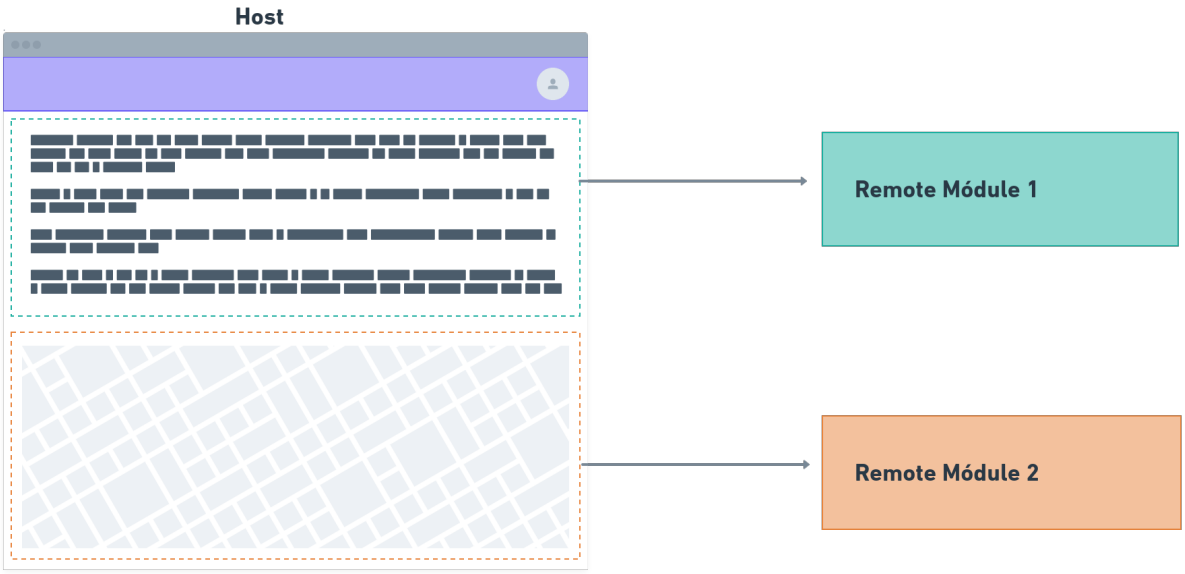


Figura 10 – Exemplo de integração visual de *host* e *remotes*. Fonte: Autor

1.6 ModuleFederationPlugin

Se valendo do Module Federation, junto a ferramenta mais popular e adotada de compilação e distribuição de código da atualidade(**??**), o *webpack*, Zackary Jackson e Marais Rossouw criaram o *plugin* ModuleFederationPlugin com auxílio e guia do criador do próprio *webpack*, Tobias Koppers (**??**), ficando disponível na versão 5 da ferramenta (**??**). Além da importação de código externo como dito no MF, o *plugins* permite ainda que as dependências entre o *host* e seus *remotes* sejam auto gerenciadas, caso o *host* possua a mesma dependência que seu *remote* e na mesma versão, o *remote* utilizará do

conteúdo existente no *host*, caso o *remote* precise de uma versão diferente de alguma dependência o *webpack* garante a utilização da versão necessária. Isso se dá porque o *plugin* atua na camada mais baixa do desenvolvimento, onde os código são divididos, administrados e carregados, dentro do próprio *webpack*, este que se encarrega de encapsular e fechar o escopo de uma ou mais dependências apenas sobre o módulo necessário, para que assim não interfira em suas outras instâncias, e ainda caso necessite de alguma outra dependência inexistente em seu *host*, o *webpack* inteligentemente irá carrega-lá do código original do *remote* e incorpora-lá. Com esta facilidade se consegue melhorar a performance de carregamento de dependências e se evita duplicidade de código entre os módulos (**??**).

Além de fornecer a possibilidade de atuar com a abordagem de compartilhamento dinâmico e unidirecional, onde vários módulos agem como *remote* exportando componente e apenas um age como *host*, seus criadores quiseram ir além, tornaram possível o compartilhamento bidirecional onde um mesmo módulo pode ao mesmo tempo ser *host* e *remote*, ou seja, importar outros módulos e também ser importado.

Ao solucionar os desafios existentes na implementação de Micro Frontends o *plugin* faz com que a solução seja facilitada, uma vez que a responsabilidade da administração e configuração acaba ficando a cargo do próprio *webpack* e não do desenvolvedor pois definições de otimização, quebra de código e compilação já se encontram na sua responsabilidade, com isso toda a configuração onde se identifica se um projeto é *host* ou se é *remote* está fora do escopo do código fonte, apenas sendo necessário identificar na configuração o nome do módulo e o *webpack* fica responsável de identificar se o que esta sendo utilizado é de origem de um *remote* ou de alguma dependência instalada. Assim outro ponto forte é que ao remover a necessidade de aumentar a complexidade de código para a configuração de um Micro Frontends, a implementação de um projeto para sua utilização se torna ainda mais tranquila pois não existem adaptações de grande desconforto para o desenvolvedor, tanto para a criação de um novo projeto utilizando esta tecnologia, quanto a adaptação de um projeto existente.

Detalhes de como funciona e como configurar um projeto para utilizar O ModuleFederationPlugin estão no apêndice **??**.

2 Trabalhos Relacionados

A principal referencia sobre Module Federation durante todo o trabalho será o livro (**??**), escrito por um dos criadores da ferramenta para descrever a tecnologia e tudo relativo a mesma.

Os presentes trabalhos relacionados são responsáveis por cada um dos módulos que são alvo do processo de integração elaborado neste presente trabalho: Em (**??**) é descrita a criação da Plataforma +Precoce, suas decisões arquiteturais, bem como sua motivação e seus benefícios. Este trabalho também é o responsável pelo desenvolvimento da parte principal do projeto que será chamado neste trabalho de Aplicação Portal; no trabalho de (**??**) é realizado a pesquisa e o desenvolvimento da funcionalidade de criação e representação de um sistemas de produção por meio de desenho gráfico, o Fluxograma, que será chamado neste presente trabalho de módulo Fluxograma e *pluginFlow*; (**??**) traz em seu trabalho a funcionalidade de visualização as etapas do processo de produção realizado pela P+P em forma de gráfico de Gantt. Este será chamado por módulo Cronograma e *pluginGantt*; (**??**) fornece uma identificação de como os valores de parâmetros utilizados em um sistema de produção podem influenciar no resultado final de um indicador da mesma por meio da técnica conhecida como análise de sensibilidade. Serão utilizadas as nomeações módulo Análise de Sensibilidade e *pluginSensitivity* para referenciar o mesmo ao longo deste trabalho; (**??**) introduz o uso de uma linguagem de programação matemática para realizar a otimização dos valores dos sistemas de produção da plataforma. Neste trabalho esta funcionalidade será chamada de módulo Otimização e *pluginOptimize*.

3 Integração

O Module Federation foi escolhido pois se encaixa com a necessidade de integração dos projetos por atender aos dois requisitos básicos: **Pouca alteração de códigos atuais dos projetos** e **Baixa adaptabilidade de código**. Esses dois requisitos significam que a mudança de como era realizado o desenvolvimento antes e depois do Module Federation causa baixo impacto em como o desenvolvedor trabalha. Assim, quando um novo desenvolvedor entrar para o time de desenvolvimento da Embrapa, ou um novo projeto for desenvolvido em conjunto com a UFMS, a pessoa por traz do código não precisará aprender uma nova tecnologia ou então gastar um grande tempo fazendo configurações para que os projetos se integrem. O processo de integração foi divido em duas partes: Aplicação Portal e Gestor. A primeira sendo acessada pelos usuários finais, e a segunda utilizada internamente pela Embrapa, onde as aplicações web são os *hosts* e os módulos são os *remotes*, além disso como explicitado no Capítulo **??**, cada módulo recebeu uma nomeação interna que será utilizada em código e nos vínculos entre os módulos, os papeis e a nomeação interna de cada projeto podem ser vistos na Figura **??**.

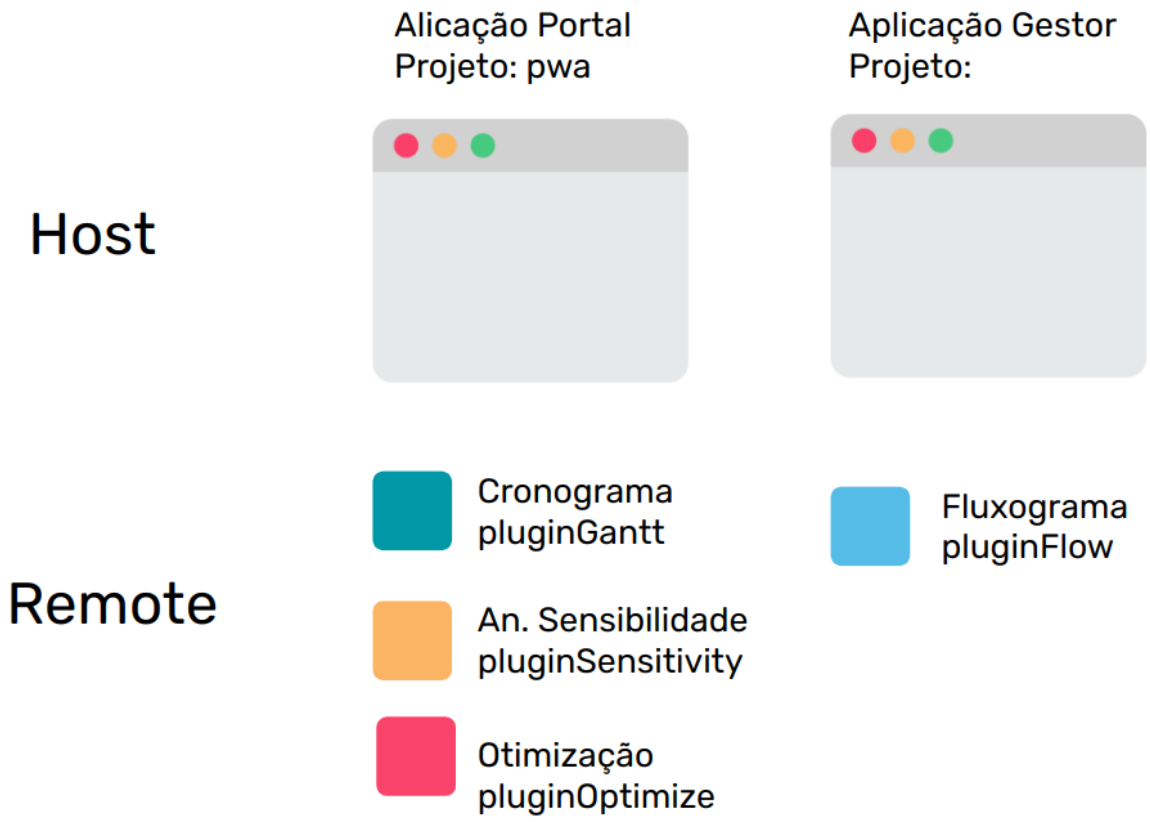


Figura 11 – Papeis de cada projeto na integração. Fonte: Autor

Deste ponto em diante, será usada a nomeação interna para se referir a cada módulo.

Também será utilizada a palavra projeto em vez de módulo, uma vez que será falado sobre o projeto em termos de código.

Para ser realizada a integração dos pré requisitos são necessários: que todos os projetos estejam de acordo com os padrões de projeto da Embrapa; e que todos os projetos utilizem o *webpack* em sua 5ª versão.

3.1 Adequação de Projetos

O padrão de projeto *frontend* da Embrapa[1] define que projetos utilizem o *framework* Vue.js, e que tenham sido criados a partir do projeto modelo da empresa que utiliza bibliotecas de funcionalidades, monitoramento, armazenamento e interface pré definidas. Os projetos dos pluginSensitivity, pluginOptimize e pluginFlow já se encontravam desenvolvidos em uma versão anterior do projeto modelo. Foram então realizadas atualizações e adaptações, informações sobre as atualizações estão no apêndice **??**. O projeto pwa também estava desenvolvido na versão anterior do projeto modelo, porém durante o decorrer deste trabalho a própria Embrapa se encarregou de realizar a atualização e ajustes. O projeto manager foi desenvolvido já utilizando o projeto modelo e não necessitou de alterações. O código desenvolvido por **??**) contava apenas com a interface em HTML necessária para o gráfico de gantt, o mesmo não encontrava-se desenvolvido como projeto, foi então realizada a criação de um novo projeto e foi reorganizado código desenvolvido, bem como desenvolvidos componentes Vue.js para o projeto e sua funcionalidade.

3.1.1 Utilização do webpack

Como dito na seção **??** do capítulo **??**, a implementação do Module Federation só se tornou disponível na 5ª versão, com a disponibilização do plugin ModuleFederationPlugin. Como os projetos foram adequados para utilizar o modelo padrão da Embrapa, todos já utilizavam o *webpack* em sua 4ª versão. Sendo então necessária a atualização para a versão 5.74.0[2], também foram necessárias atualizar a versão do @vue/cli-service, para a versão 5.0.0, pois este faz parte do processo de compilação do *framework* Vue.js que utilizam do *webpack*.

3.1.2 Pré integração

Como os projetos possuem diferentes URL em cada ambiente, testes internos, testes externos e publicação para o usuários finais[3], é necessário que se tenha a administração

[1] Informação fornecida durante as reuniões com a Embrapa e disponível em documentações internas da empresa

[2] A última versão disponível até a data em que estava sendo desenvolvida a integração

[3] https://www.embrapa.io/docs/build/

das URLs de cada *remote* para cada um desses ambientes. Para realizar o manuseio de forma simplificada, foram utilizadas variáveis de ambiente baseadas em arquivo ***.env***[4]. Nos *remotes* a variável dentro do arquivo ***.env*** que se refere a URL é **VUE_APP_URL** e seu valor é definido pela infraestrutura de disponibilização da empresa. Já nos projetos *host* o arquivo ***.env*** possui também variáveis dedicadas a cada *remote* e tem seu nome definido de acordo com nome do *remote*.

3.2 Integração Aplicação Gestor

3.2.1 Remote pluginFlow

Primeiramente foi adicionado o *plugin* MFP ao projeto pluginFlow, alterando o arquivo ***vue.config.js*** da seguinte maneira:

```
1  const { defineConfig } = require("@vue/cli-service");
2  const webpack = require("webpack");
3  const ModuleFederationPlugin = webpack.container.ModuleFederationPlugin;
4  const dependencies = require("./package.json").dependencies;
5  module.exports = defineConfig({
6    publicPath: process.env.VUE_APP_URL,
7    configureWebpack: {
8      plugins: [
9        new ModuleFederationPlugin({
10         name: 'pluginFlow',
11         filename: 'remoteEntry.js',
12         exposes: {
13           './FlowTree': './src/components/shared/FlowTree.vue',
14         },
15         shared: {
16           vue: {
17             eager: true,
18             requiredVersion: dependencies.vue,
19           },
20           vuetify: {
21             eager: true,
22           },
23         },
24       }),
25     ],
26   },
27 });
```

Na configuração foi definida a propriedade *publicPath* que deve ser a URL do projeto, esta propriedade é muito importante pois é o seu valor que informa ao *webpack* que ao gerar os arquivos necessários para um componente, de onde esses arquivos serão requisitados. Foi adicionada a propriedade *exposes* que contém o componente ***FlowTree.vue*** este sendo exportado para ser utilizado, o nomeia como ***./FlowTree*** seguindo os padrões definidos pelo *plugin* MFP. A configuração também define quais as dependências que pluginFlow irá utilizar de maneira compartilhada com seu *host*, o manager, que são o *framework*

[4] https://cli.vuejs.org/guide/mode-and-env.html

vue e a biblioteca de interface ***vuetify***. O pluginFlow é um excelente caso do desafio de administração de versões que existe em Micro Frontends e que foi solucionado pelo MPF, pois o mesmo exige uma versão bem especifica da biblioteca gráfica D3 na sua versão 4.2.2 por conta de algumas funcionalidades, porém a D3 já encontra-se em versões superiores a 7.0.0[5], assim caso seja necessário que o *host* ou algum outro modulo também necessite utilizar dessa biblioteca, a versão a ser escolhida pelo outro projeto não oferece risco às funcionalidades existentes dentro do pluginFlow. Após adicionar a configuração é preciso alterar a inicialização do projeto para utilizar importação dinâmica, para isso, o arquivo ***main.js*** foi renomeado para ***bootstrap.js***, e criado novo arquivo ***main.js*** contendo apenas a seguinte linha:

```
1    import("./bootstrap");
```

Somente essas duas alterações são necessárias em projeto *remote* para que a integração sejá realizada. Quando importado no *host* precisará apenas utilizar

```
1    import FlowTree from "pluginFlow/FlowTree"
```

da mesma maneira que é feita a importação de alguma biblioteca JavaScript. E o *webpack* se encarregará de identificar se pluginFlow é uma dependência instalada ou um remote.

3.2.2 Host manager

Uma vez realizadas as adaptações necessárias ao pluginFlow, foi inciada a adaptação do projeto manager. Primeiramente foi criado no arquivo ***.env*** a variável *VUE_APP_FLOW_REMOTE* para definir a URL do *remote* pluginFlow de acordo com o ambiente. Então foi adicionado o *plugin* MFP a configuração do projeto manager ***vue.config.js***, como no código abaixo:

```
1  const { defineConfig } = require("@vue/cli-service");
2  const webpack = require("webpack");
3  const ModuleFederationPlugin = webpack.container.ModuleFederationPlugin;
4  const dependencies = require("./package.json").dependencies;
5  module.exports = defineConfig({
6    publicPath: process.env.VUE_APP_URL,
7    configureWebpack: {
8      plugins: [
9        new ModuleFederationPlugin({
10         name: "manager",
11         filename: "remoteEntry.js",
12         remotes: {
13           pluginFlow: "pluginFlow@"+process.env.VUE_APP_FLOW_REMOTE+"/remoteEntry.js"
14         },
15         shared: {
16           vue: {
17             eager: true,
18             singleton: true,
19             requiredVersion: dependencies.vue,
20           },
21           vuetify: {
22             eager: true,
```

[5] https://www.npmjs.com/package/d3

```
            singleton: true,
            requiredVersion: dependencies.vuetify,
          },
        },
      }),
    ],
  },
});
```

Diferente da configuração realizada no pluginFlow, para o manager não existe a propriedade *exposes*, em seu lugar esta a propriedade *remotes*, que traz a definição de cada *remote* a ser utilizado, seguinte o modelo necessário para o MFP:

```
nomeDoRemoteParaSerUtilizado: "propriedadeNameNaDefinicaoDoRemote@URL.do.remote/
    remoteEntry.js"
```

Na Figura **??** pode ser visto como cada valor definido na configuração do *remote* pluginFlow é utilizado na configuração do *host* manager.

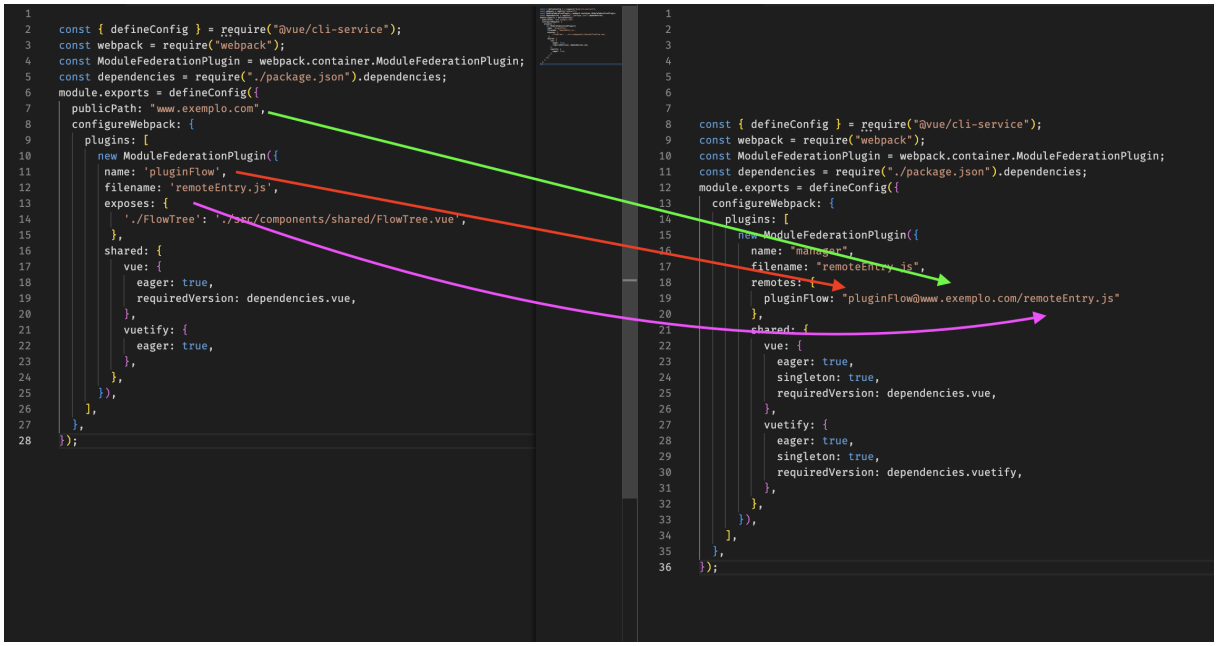


Figura 12 – Configuração *remotes* relativa aos dados na configuração *host*. Fonte: Autor

Da mesma maneira que realizado no pluginFlow, arquivo inicial ***main.js*** foi renomeado para ***bootstrap.js***, e criado novo arquivo ***main.js*** contendo apenas a importação do arquivo ***bootstrap.js***. Assim o projeto pluginFlow já pode ser utilizado dentro do projeto manager. Foi definido pela Embrapa que o pluginFlow será exibido na tela de Sistemas, componente ***CrudSystem.vue***, sendo exibido como Janela de Diálogo quando solicitado, seguindo definições internas do projeto. A importação FlowTree ficou da seguinte maneira no arquivo:

```
import FlowTree from 'pluginFlow/FlowTree'
export default {
  components: {
    MessageWrapper,
```

```
5       Combobox,
6       CrudDescriptionAdd,
7       RichEdit,
8       NameField,
9       FlowTree
10    },
11  }
```

E sua utilização FlowTree ficou da seguinte maneira na seção **<template>** do arquivo:

```
1  <template v-else-if="dialogMode == 'F'">
2    <FlowTree :simulation="json" v-if="dialogMode == 'F' && dialog" @saveChanges="
         callback"/>
3    <v-card-actions>
4      <v-spacer></v-spacer>
5      <v-btn color="blue darken-1" text @click="closeForm">
6        Fechar
7      </v-btn>
8    </v-card-actions>
9  </template>
```

E exibindo o componente de Fluxograma como mostrado na Figura **??**.

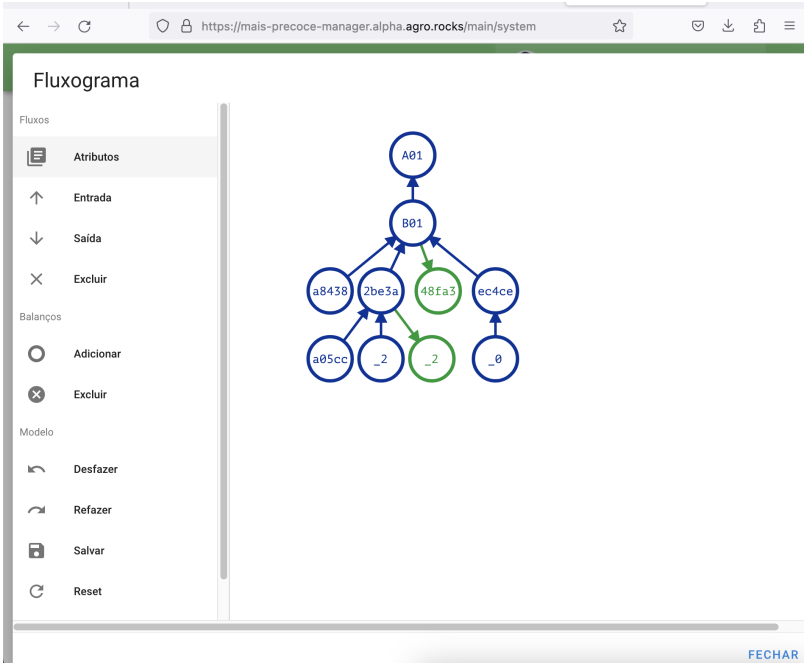


Figura 13 – Janela do Fluxograma sendo exibida na Aplicação Gestor. Fonte: Autor

Como o pluginFlow deve somente ser exibido quando o usuário tiver a intenção de visualizá-lo, foi adicionado ao componente ***CrudSystem.vue***, uma nova opção no menu de ação, o código dessa nova ação se dá por:

```
1  <v-tooltip bottom :open-on-focus="false" :open-on-click="false" >
2    <template v-slot:activator="{ on, attrs }">
3      <v-icon small class="mr-2" @click="flowForm(item)"  v-bind="attrs" v-on="on">
4        mdi-play-circle
5      </v-icon>
6    </template>
7    <span>Abrir o Fluxograma</span>
```

```
8  </v-tooltip>
```

Com isso a ação de exibir o componente do pluginFlow fica disponível ao usuário como pode ser visto na Figura **??**.

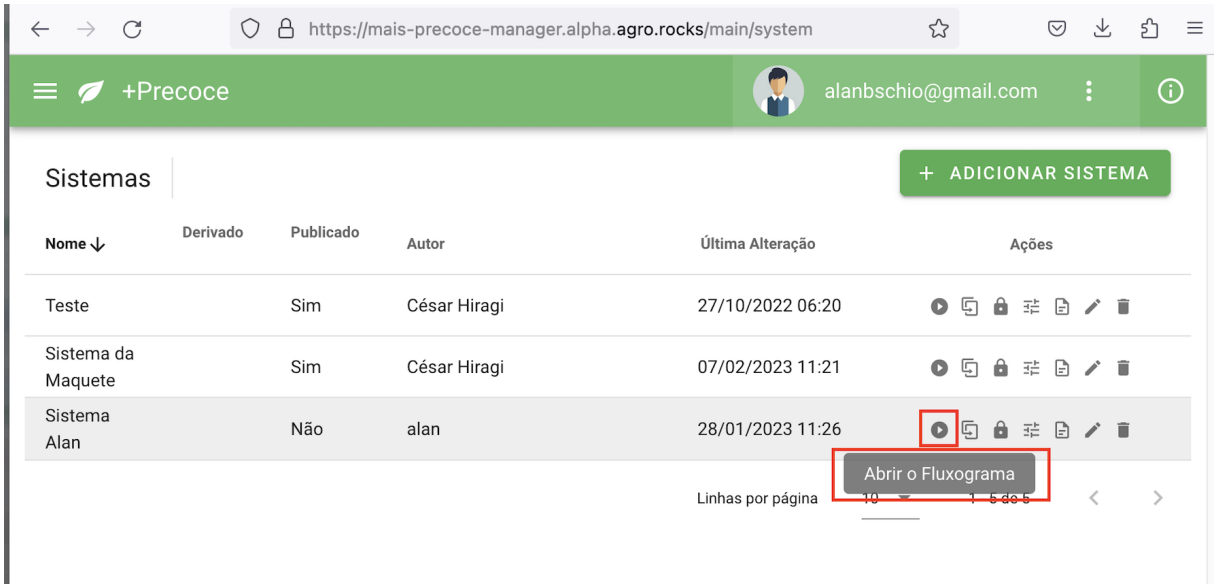


Figura 14 – Opção de Fluxograma nas ações. Fonte: Autor

3.3 Integração Aplicação Portal

3.3.1 Remote pluginGantt

Para o pluginGantt foi criado um novo projeto baseado no projeto modelo Embrapa, e foram feitas reestruturações dos códigos anteriores. Primeiramente foi criado um componente exclusivo para a interface do gráfico de Gantt, seguindo o modelo de componentes Vue.js, encapsulando todo o conteúdo do projeto inicial desenvolvido em HTML e JavaScript. Foi utilizado o diretório **helpers** onde foi separado as funções de geração do gráfico. Após a adaptação do projeto iniciou-se o desenvolvimento da integração, primeiramente sendo adicionado ao arquivo ***vue.config.js*** as configurações como no código abaixo:

```
1  const { defineConfig } = require("@vue/cli-service");
2  const webpack = require("webpack");
3  const ModuleFederationPlugin = webpack.container.ModuleFederationPlugin;
4  const dependencies = require("./package.json").dependencies;
5  module.exports = defineConfig({
6    publicPath: process.env.VUE_APP_URL,
7    configureWebpack: {
8      plugins: [
9        new ModuleFederationPlugin({
10         name: "pluginGantt",
11         filename: "remoteEntry.js",
12         exposes: {
13           './GanttChart': './src/components/shared/GanttChart.vue'
```

```
14          },
15          shared: {
16            vue: {
17              eager: true,
18              singleton: true,
19              requiredVersion: dependencies.vue,
20            },
21            d3: {
22              requiredVersion: dependencies.d3
23            }
24
25          },
26        }),
27      ],
28    },
29  });
```

Na configuração, o pluginGantt define na propriedade *exposes* que o componente ***GanttChart.vue*** está sendo exportado com o nome ***./GanttChart***. A configuração também define quais as dependências que pluginGantt irá utilizar de maneira compartilhada com seu *host*, o pwa. No item *shared* são adicionados o *framework* ***vue***, que o pluginGantt utilizará do seu *host*, e a biblioteca D3, semelhando ao pluginFlow neste projeto também existe a dependência de uma biblioteca em uma determinada versão, a D3 em sua versão 3.0.0, então aqui é informado que este *remote* utiliza exclusivamente a versão 3.0.0, e que além disso, por esta estar definida no *shared*, caso algum outro *remote* também utilize dessa dependência, o segundo a ser inicializado utiliza a importação do primeiro a ser inicializado, reduzindo a quantidade de código a ser carregando no navegador do usuário. Da mesma maneira que realizado nos projetos anteriores, o arquivo ***main.js*** foi renomeado para ***bootstrap.js***, e criado novo arquivo ***main.js*** contendo apenas a importação do arquivo ***bootstrap.js***.

3.3.2 Remote pluginOptimize

No projeto pluginOptimize foram realizadas algumas alterações que precedem a adaptação para a integração, como por exemplo o componente ***ErrorModal.vue*** foi removido pois não estava sendo utilizado. Os componentes ***Indicador.vue, MaximizeIndicator.vue e ParametroUnico.vue*** foram melhorados, sendo assim possível utilizar a facilidade conhecida como **v-model** para se obter o valor do(s) indicador(es) selecionados. No que tange a biblioteca matemática utilizada, ***mathjs***, no arquivo ***src/plugins/ampljs/ampl.js*** a maneira que se utiliza a biblioteca também foi atualizada, deixado de ser importado como *const math = require('mathjs')* e sendo importado agora como *import { simplify } from 'mathjs'*. Por fim, o componente principal cujo qual o usuário realiza ações, o ***FormNewOptimization.vue*** possuía um grande número de funções em código o que pode dificultar sua manutenção e legibilidade, com isso foi implementa uma melhoria de converter algumas funções do mesmo em arquivos utilitários, com isso foram

criados os arquivos **json.helper.js**, **download.helper.js** e **otimization.helper.js** na pasta **helpers**. Essa separação também ajuda na organização e entendimento do código do componente. O componente também foi renomeado para um nome mais adequado: ***Optimization.vue***. Além disso foram aplicadas algumas melhorias:

- adicionado a funcionalidade de desabilitar o formulário enquanto a otimização esta sendo gerada;
- substituídos a descrição dos campos do formulário que estavam utilizando o elemento HTML **h4** para o correto a ser utilizado para esta finalidade que é o **label**;
- substituído elemento HTML **div** por **section**, o correto a ser utilizado para esta finalidade.

Algumas das alterações relacionadas ao ***Optimization.vue*** podem ser vistas na Figura **??**.

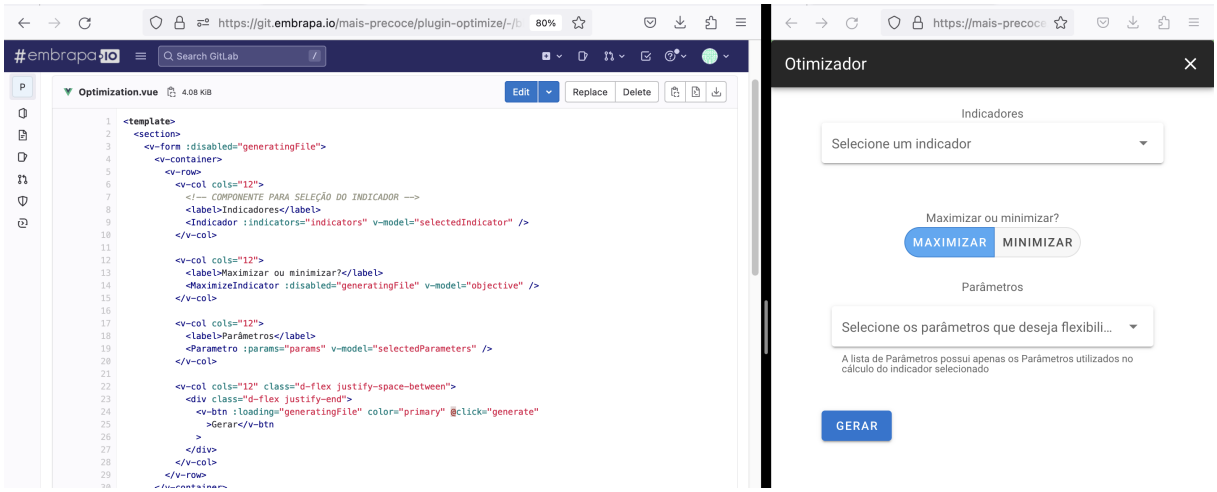


Figura 15 – Visualização do código e da interface do componente ***Optimization.vue***. Fonte: Autor

Após realizada as alterações necessárias nos componentes, foi então possível iniciar configuração para a integração, primeiramente adicionado ao arquivo ***vue.config.js*** as configurações como no código abaixo:

```
1  const { defineConfig } = require("@vue/cli-service");
2  const webpack = require("webpack");
3  const ModuleFederationPlugin = webpack.container.ModuleFederationPlugin;
4  const dependencies = require("./package.json").dependencies;
5  module.exports = defineConfig({
6    publicPath: process.env.VUE_APP_URL,
7    configureWebpack: {
8      plugins: [
9        new ModuleFederationPlugin({
10         name: "pluginOptimize",
11         filename: "remoteEntry.js",
12         exposes: {
```

```
13              './Optimization': './src/components/shared/Optimization.vue'
14            },
15            shared: {
16               vue: {
17                 eager: true,
18                 singleton: true,
19                 requiredVersion: dependencies.vue,
20               },
21               vuetify: {
22                 eager: true,
23                 singleton: true,
24                 requiredVersion: dependencies.vuetify,
25               },
26            },
27          }),
28        ],
29      },
30    });
```

Na configuração do pluginOptimize a propriedade *exposes* possui ***Optimization.vue*** sendo exportado como ***./Optimization***. A configuração também define quais as dependências que pluginOptimize irá utilizar de maneira compartilhada com seu *host*, o manager, que são o *framework* ***vue*** e a biblioteca de interface ***vuetify***. Da mesma maneira que realizado nos projetos anteriores, o arquivo ***main.js*** foi renomeado para ***bootstrap.js***, e criado novo arquivo ***main.js*** contendo apenas a importação do arquivo ***bootstrap.js***.

3.3.3 Remote pluginSensitivity

O último *remote* a ser configurado foi o pluginSensitivity, antes de realizar a integração foram realizadas adaptações e simplificações no projeto. Inicialmente para manter a consistência de nomenclatura dos componentes, o arquivo ***TornadoSimulação.vue*** foi renomeado para ***TornadoSimulation.vue***. Foi identificado que um mesmo código se repetia em todos os gráficos gerados, o informativo de ajuda que contém a imagem explicativa do gráfico. Foi então criado um novo componente, o ***HelpDialog.vue*** que inclui o botão que lida com a exibição do informativo, como mostrado na Figura **??**. Foi identificado que os componentes de seleção de indicador e parâmetro não estavam utilizando a funcionalidade **v-model**[6] para repassar o seu valor ao formulário de origem. Os componentes ***Parametro.vue*** e ***ParametroUnico.vue*** possuíam comportamento semelhante sendo apenas diferenciado por uma propriedade, **multiple**, então o componente foi reescrito de maneira que abrange os dois tipos de uso. Além das melhoria, o componente ***Indicador.vue*** foi renomeado para ***SelectIndicador.vue***, e o ***Parametro.vue*** foi renomeado para ***SelectParametro.vue***.

Durante a análise do código deste projeto foi identificado que apresentava o uso de duas tecnologias, que apesar de bem utilizadas adicionavam um complexidade que não era mais necessária ao projeto, uma vez que ao integrá-lo com o projeto pwa todos os dados

[6] https://v2.vuejs.org/v2/guide/forms.html

```
<v-row>
   <v-col cols="11" class="text-center"> </v-col>
   <v-col cols="1" class="text-center">
     <v-btn
       class="help"
       color="primary"
       fab
       dark
       title="Ajuda"
       @click="showModal = true">?</v-btn>
   </v-col>
   <v-dialog fullscreen v-model="showModal">
     <v-card>
       <v-card-title class="text-h5 grey lighten-2">
         <h1>
           Ajuda do gráfico <small>{{ title }} </small>
         </h1>
         <v-spacer></v-spacer>
         <v-btn color="primary" text @click="close">Fechar </v-btn>
       </v-card-title>
       <slot />
     </v-card>
   </v-dialog>
  </v-row>
</template>
```

Figura 16 – Novo componente reutilizável de gráfico. Fonte: Autor

sobre a simulação virão como parâmetros. Assim eliminando o uso da dependência **Vuex** para armazenamento de dados. Já a dependência **Vue Router** estava sendo utilizada para fazer gerenciamento de telas e rotas de maneira a ter de ser redirecionado, o que não se faz mais necessário uma vez que por definições realizadas pela empresa, os *remotes* do projeto pwa serão exibidos como janelas ou caixas de diálogo de acordo com a simulação selecionada. Foi criado um novo componente ***SensitivityAnalysis.vue***, este é ponto principal da funcionalidade, que pode ser utilizado tanto para uso em *host* quando dentro do próprio projeto. Este componente recebe os dados da simulação como propriedade obrigatória e exibe as quatro opções de análise de sensibilidade que podem ser geradas, como mostrado na Figura **??**. As opções exibidas são geradas por meio de uma lista que contém a imagem do fundo, o título e qual o componente de formulário que será aberto em forma de janela de diálogo, como mostrado na Figura **??**.

O redirecionamento de tela estava sendo feito por meio de 3 tipos de rotas: Escolha de sistema (/sis); Escolha e preenchimento das informações de geração do gráfico(/ ,

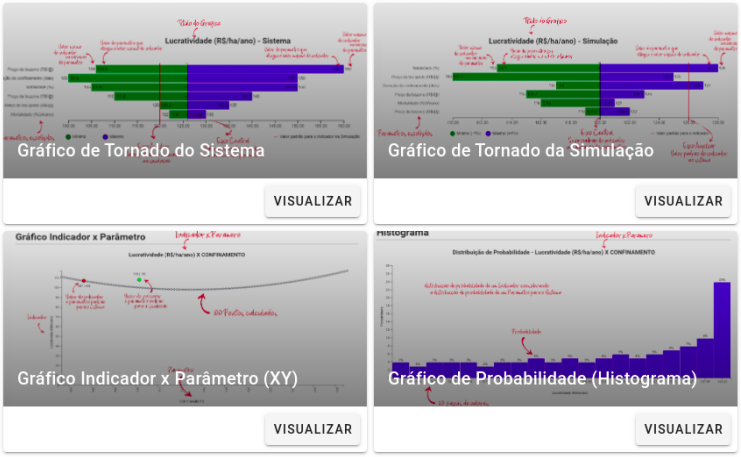


Figura 17 – Componente SensitivityAnalysis.vue. Fonte: Autor

```
card = [
 {
  title: "Gráfico de Tornado do Sistema",
  src: "img/legenda-sistema.jpg",
  component: FormSistema,
 },
 {
  title: "Gráfico de Tornado da Simulação",
  src: "img/legenda-tornado.jpg",
  component: FormSimulacao,
 },
 {
  title: "Gráfico Indicador x Parâmetro (XY)",
  src: "img/legenda-xy.jpg",
  component: FormXY,
 },
 {
  title: "Gráfico de Probabilidade (Histograma)",
  src: "img/legenda-histograma.jpg",
  component: FormHistograma,
 }
]
```

Figura 18 – Lista de opções. Fonte: Autor

/formxy , /formhistograma , /formsimulação , /formsistema); e Exibição do gráfico (/line , /bar ,/tornadosimulação , /tornadosistema). O redirecionamento de escolha de sistema, se faz desnecessário após a integração com o projeto pwa, a simulação de sistema será definida pela escolha do usuário no *host*. A exibição do gráfico é resultado do preenchimento de informações do formulário, o que se entende com o resultado de uma ação, sendo assim transformado em uma janela de diálogo, como pode ser visto na Figura **??**. A funcionalidade de exibição de gráficos possui uma implementação igual para ambos os casos, por isso criou-se um componente genérico chamado **ChartDialog.vue** que utiliza do elemento HTML **<slot/>** para fazer a injeção do gráfico, e que exibe o título de acordo um parâmetro, como visto em seu código na Figura **??**.

Com essa alteração, os componentes ***BarChart.vue, LineChart.vue, Torna-***

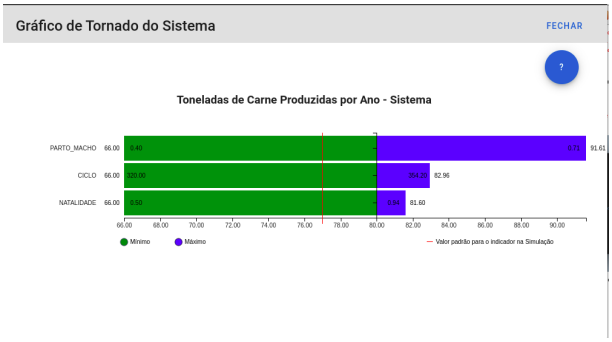


Figura 19 – Novo componente do gráfico em Janela de Diálogo. Fonte: Autor

```
<template>
  <v-dialog fullscreen v-model="showModal"
    ><v-card>
      <v-card-title class="text-h5 grey lighten-2">
        {{ title }}
        <v-spacer></v-spacer>
        <v-btn color="primary" text @click="close">Fechar </v-btn>
      </v-card-title>
      <slot />
    </v-card>
  </v-dialog>
</template>
```

Figura 20 – Código do novo componente ChartDialog. Fonte: Autor

doSimulation.vue e TornadoSystem.vue passam a ser utilizados respectivamente dentro dos arquivos ***FormHistogram.vue, FormXY.vue, FormSimulation.vue, e FormSystem.vue***, sendo encapsulados dentro do componente ***ChartDialog.vue*** como no exemplo da Figura **??**. Com essa alteração as rotas **/line, /bar, /tornadosimulação e /tornadosistema** foram descartadas.

```
<ChartDialog
  title="Histograma"
  @close="onChartModalClose"
  :showWhen="showChartDialog">
  <Bar
    v-if="showChartDialog"
    :title="`Distribuição de Probabilidade - ${indicador} X ${parametros}`"
    :simulacao="simulacao"
    :indicador="indicador"
    :parametros="parametros"
    :form="{ type: 'system', model: 'histograma' }"/>
</ChartDialog>
```

Figura 21 – Código do novo componente reutilizável do gráfico. Fonte: Autor

Ao criar o componente central ***SensitivityAnalysis.vue***, a responsabilidade de exibir os fomulários de geração dos gráficos é trazida para dentro do componente, e assim pode-se remover o uso de redirecionamento para o formulário por meio de janela de

diálogo, que utiliza a o elemento *<component>* do Vue.js para injetar um componente por propriedade, tornando assim dinâmico qual componente deve formulário deve ser renderizado, como mostrado no código da Figura **??**.

```
<v-dialog fullscreen v-model="showDialog"
    ><v-card v-if="showDialog">
      <v-card-title class="text-h5 grey lighten-2">
        {{ selectedCard.title }}
        <v-spacer></v-spacer>
        <v-btn color="primary" text @click="showDialog = false"
          >Fechar
        </v-btn>
      </v-card-title>
      <component
        v-if="showDialog"
        :is="selectedChart"
        :simulacao="simulation"
        :title="selectedCard.title"
      >
      </component>
    </v-card>
  </v-dialog
```

Figura 22 – Utilização de componente dinâmico de formulário. Fonte: Autor

Com a mudança de abordagem de retirada de roteamento, e com a utilização deste projeto sendo integrado dentro do projeto pwa, todas as informações necessárias agora serão passadas como parâmetros, bem como as informações de indicadores e parâmetros agora utilizam a atribuição em **v-model** para aguardar o valor definido dentro do componente onde são utilizados as suas seleções. Assim a dependência que possui o intuito der realizar o gerenciamento de dados em estado global com compartilhamento de um mesmo dado em diversas partes de um sistema, não se faz mais necessária. O primeiro arquivo a ser removido foi o arquivos ***src/store/modules/forms***, detentor da função desenvolvida por **??**) chamada de *searchRelation*, que se trata da função responsável por analisar os dados selecionados no formulário e realizar a chamada da função de cálculo da análise de sensibilidade, e alimentação dos dados para plotagem do gráfico. Toda a lógica que estava encapsulada foi movida para um novo arquivo auxiliar chamado ***src/helpers/grafico/index.js***, está recebe os valores necessários como parâmetros. Os dados de simulação, o tipo de gráfico a ser gerado, o(s) indicador(es) e o(s) parâmetro(s) definidos que antes eram obtidos por meio do gerenciamento global de estado, agora são passados sempre que necessário buscar uma nova *relation*, nome utilizado na ferramenta. Em seguida foi removido o arquivo ***src/store/modules/graph.js*** que tinha como finalidade apenas guardar os dados de cada tipo de gráfico isoladamente e o nome dado a cada gráfico.

Com os ajustes pré-integração finalizados, foi iniciado a adaptação das configurações do pluginSensitivity, primeiramente adaptando o arquivo ***vue.config.js***, semelhante aos *remotes* anteriores:

```
1 const { defineConfig } = require("@vue/cli-service");
2 const webpack = require("webpack");
```

```
3  const ModuleFederationPlugin = webpack.container.ModuleFederationPlugin;
4  const dependencies = require("./package.json").dependencies;
5  module.exports = defineConfig({
6    publicPath: process.env.VUE_APP_URL,
7    configureWebpack: {
8      plugins: [
9        new ModuleFederationPlugin({
10          name: 'pluginSensitivity',
11         filename: 'remoteEntry.js',
12         exposes: {
13           './SensitivityAnalysis': '@/components/SensitivityAnalysis.vue'
14         },
15         shared: {
16           vue: {
17             eager: true,
18             singleton: true,
19             requiredVersion: dependencies.vue,
20           },
21           vuetify: {
22             eager: true,
23             singleton: true,
24             requiredVersion: dependencies.vuetify,
25           },
26           d3: {
27             requiredVersion: dependencies.d3
28           },
29         },
30       }),
31     ],
32   },
33 });
```

Na definição o pluginSensitivity adiciona a propriedade *exposes* que o componente ***SensitivityAnalysis.vue*** esta sendo exportado e o tornando possível de ser utilizado externamente, o nomeia como ***./SensitivityAnalysis*** seguindo os padrões definidos pelo *plugin* MFP. A configuração também define quais as dependências que pluginSensitivity irá utilizar de maneira compartilhada com o *host* que são: *framework* ***vue***; a biblioteca de interfaces ***Vuetify***; e a biblioteca de gráficos ***D3***. Apesar de não utilizar nenhuma funcionalidade específica da versão utilizada para a biblioteca D3, ou que não existe em versões posteriores, optou-se por adiciona-lá ao *shared* de maneira preventiva. Da mesma maneira que realizado nos projetos anteriores, o arquivo ***main.js*** foi renomeado para ***bootstrap.js***, e criado novo arquivo ***main.js*** contendo apenas a importação do arquivo ***bootstrap.js***.

3.3.4 Host pwa

Por último, e mais importante, foi realizada a adaptação do *host* que utilizará o projetos anteriores. O arquivo ***vue.config.js*** foi alterado adicionando o *plugin* MFP e os *remotes* utilizados por este projeto, da seguinte maneira:

```
1  const { defineConfig } = require("@vue/cli-service");
2  const webpack = require("webpack");
```

```
3  const ModuleFederationPlugin = webpack.container.ModuleFederationPlugin;
4  const dependencies = require("./package.json").dependencies;
5  module.exports = defineConfig({
6    publicPath: process.env.VUE_APP_URL,
7    configureWebpack: {
8      plugins: [
9        new ModuleFederationPlugin({
10         name: 'pwa',
11         filename: 'remoteEntry.js',
12         remotes: {
13           pluginGantt: "pluginGantt@"+process.env.VUE_APP_PLUGIN_GANTT_REMOTE+"/
                remoteEntry.js",
14           pluginSensitivity: "pluginSensitivity@"+process.env.
                VUE_APP_PLUGIN_SENSITIVITY_REMOTE+"/remoteEntry.js",
15           pluginOptimize: "pluginOptimize@"+process.env.
                VUE_APP_PLUGIN_OPTIMIZE_REMOTE+"/remoteEntry.js"
16         },
17         shared: {
18           vue: {
19             eager: true,
20             singleton: true,
21             requiredVersion: dependencies.vue,
22           },
23           vuetify: {
24             eager: true,
25             singleton: true,
26             requiredVersion: dependencies.vuetify,
27           },
28         },
29       }),
30     ],
31   },
32 });
```

E semelhante aos demais projetos, o arquivo ***main.js*** foi renomeado para ***bootstrap.js***, e criado novo arquivo ***main.js*** contendo apenas a importação do arquivo ***bootstrap.js***.

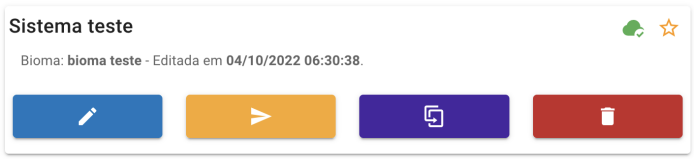
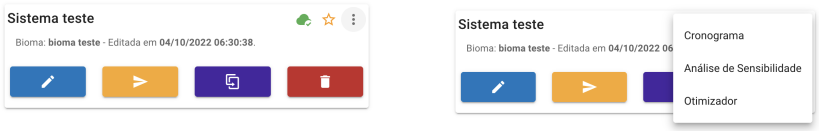


Figura 23 – *Card* de simulação. Fonte: Autor

Na P+P cada simulação de sistema de produção é exibido como um *card* (Figura **??**) onde todas as ações possíveis são resultados de botões presentes no *card*, e se apresenta ao usuário como uma janela de diálogo. Para isso, foi então criado um componente de listagem de opções dentro do *card*, o ***SimulationCardMenu.vue***, o componente completo pode ser visto no apêndice **??**. Em resumo o componente importa os três *remotes* e os coloca numa lista de opções. As opções são exibidas em tela (**??**) e ao ser selecionada uma opção, o código vindo do *remote* é injetado dinamicamente no HTML, código resumido:



(a) Botão de opções no Card. (b) Opções listadas.

Figura 24 – Visualização do componente de opções. Fonte: Autor

```
<template>
<v-list>
 <v-list-item :key="menu.name" v-for="menu in menus" @click="select(menu)">
   <v-list-item-title>{{ menu.name }}</v-list-item-title>
 </v-list-item>
</v-list>
<v-dialog fullscreen v-model="showModuleDialog">
 <component :is="moduleToShow.component" :simulation="simulation"/>
</v-dialog>
</template>
<script>
import Optimization from 'pluginOptimize/Optimization'
import Cronograma from 'pluginGantt/GanttChart'
import Sensibilidade from 'pluginSensitivity/SensitivityAnalysis'
export default {
  data () {
    return {
      menus: [{
        name: 'Cronograma',
        component: Cronograma
      },{
        name: 'Analise de Sensibilidade',
        component: Sensibilidade
      },{
        name: 'Otimizador',
        component: Optimization
      }],
      moduleToShow: null,
    }
  },
  methods: {
    select (menuItem) { this.moduleToShow = menuItem},
  }
}
</script>
```

3.4 Monitoramento

A empresa utiliza uma ferramenta de monitoramento para uma melhor detecção e atuação em *bugs*. Por uma determinação do setor de tecnologia da Embrapa, foi definido que mesmo sendo integrados aos *hosts*, os *remotes* devem fazer o rastreio de seus erros separadamente. Para abranger essa necessidade, nos *remotes* foi criado um arquivo *sentry.js*, que possui sua função ***useSentry***, a mesma é utilizada dentro do componente exportado.

```
import Vue from 'vue'
import * as Sentry from '@sentry/vue'
```

```
3  import { BrowserTracing } from '@sentry/tracing'
4  export const useSentry = (router) => {
5    const props = {
6      tracingOrigins: ['localhost', window.location.hostname, /^\//]
7    }
8    if (router) props.routingInstrumentation = Sentry.vueRouterInstrumentation(router)
9
10   Sentry.init({
11     Vue,
12     dsn: process.env.VUE_APP_SENTRY_DSN,
13     release: process.env.VUE_APP_VERSION.split('-')[0],
14     environment: process.env.VUE_APP_STAGE,
15     integrations: [
16       new BrowserTracing(props)
17     ],
18     tracesSampleRate: 1.0
19   })
20 }
```

Com isso para garantir um melhor otimização de dependências, na propriedade *shared* do arquivo ***vue.config.js*** de todos os projetos, sejam eles *remotes* ou *hosts*, o compartilhamento das dependências da ferramenta utilizada pela empresa, da seguinte maneira:

```
1  '@sentry/vue': {
2      eager: true,
3      singleton: true,
4      requiredVersion: dependencies['@sentry/vue']
5    },
6  '@sentry/tracing': {
7      eager: true,
8      singleton: true,
9      requiredVersion: dependencies['@sentry/tracing']
10   },
```

3.5 Disponibilização

A disponibilização online de *remotes* e *hosts* ficou a cargo do setor de tecnologia da empresa, por meio da plataforma interna Embrapa.io[7]. A plataforma utiliza o sistema de controle de versionamento de código Git para criar versões dos sistemas e projetos. Assim quando finalizada a implementação de uma funcionalidade ou correção, basta seguir o fluxo de liberação da plataforma e o código se torna online em poucos minutos. Com a disponibilização dos projetos sendo feita de maneira singular, ao ser atualizado um *remote* seguindo o fluxo de disponibilização, nada precisa ser alterado no *host* para que a atualização surta efeito: e é automaticamente refletida no *host* com uma simples atualização de página no navegador do usuário.

[7] https://www.embrapa.io/about/

3.6 Resultados da Integração

Na Figura ?? pode ser visto que ao acessar a Plataforma +Precoce, os arquivos iniciais ***remoteEntry.js*** sendo carregados. Já na Figura ?? a funcionalidade Cronograma é exibido em tela, e os arquivos carregados diretamente do pluginGantt. Um exemplo do *host* manager, pode ser visto na Figura ?? onde o mesmo carrega o arquivo de inicialização ***remoteEntry.js*** do pluginFlow, e quando exibido o Fluxograma, os arquivos necessários para exibir o componente do pluginFlow são carregados, na Figura ??.

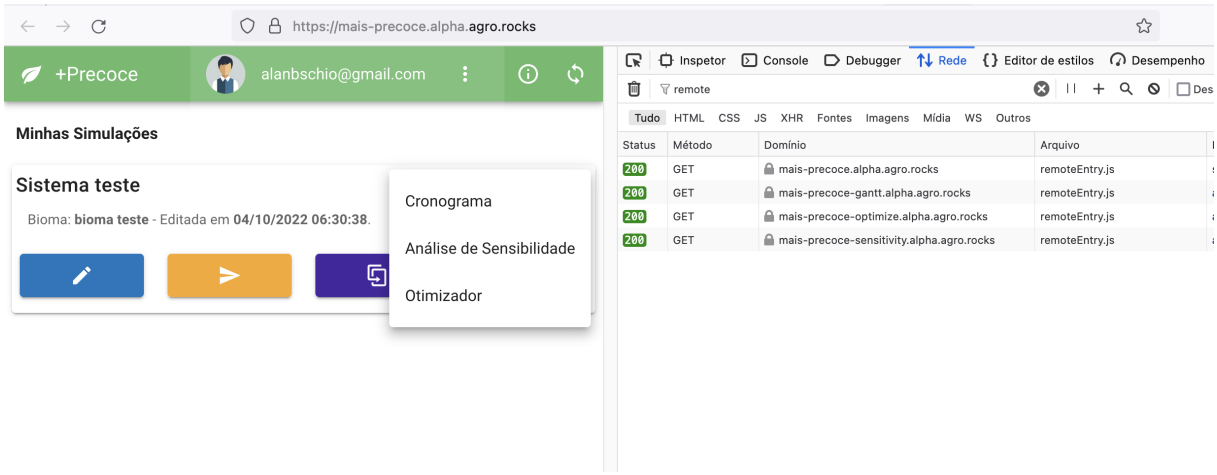


Figura 25 – *Host* pwa carregando arquivos dos seus *remotes*. Fonte: Autor

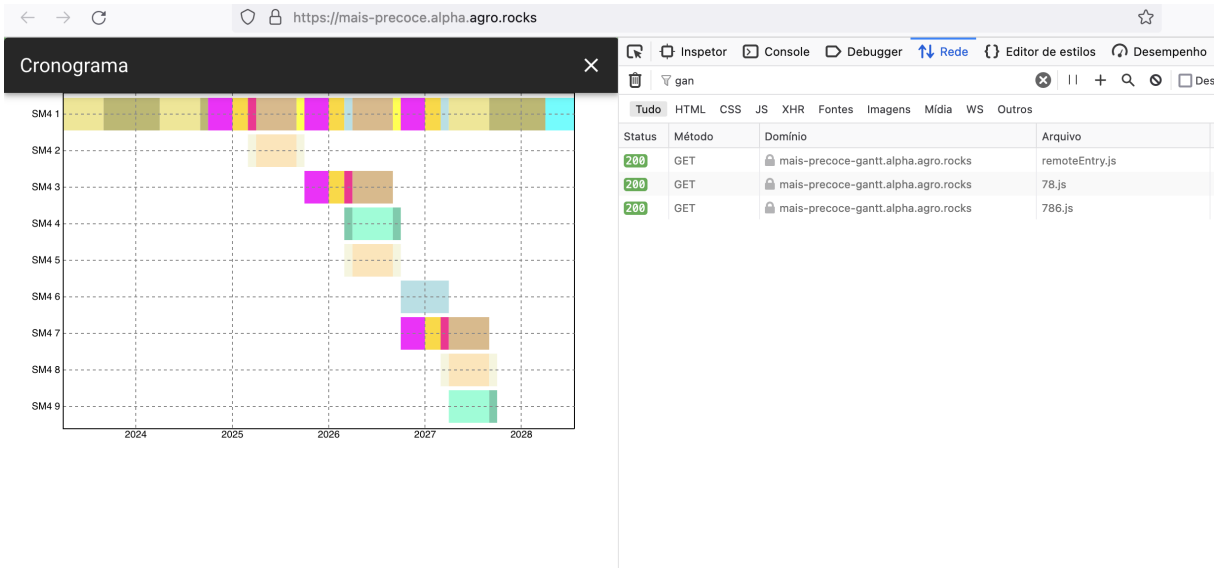


Figura 26 – *Host* pwa exibindo componente carregado do *remote* pluginGantt. Fonte: Autor

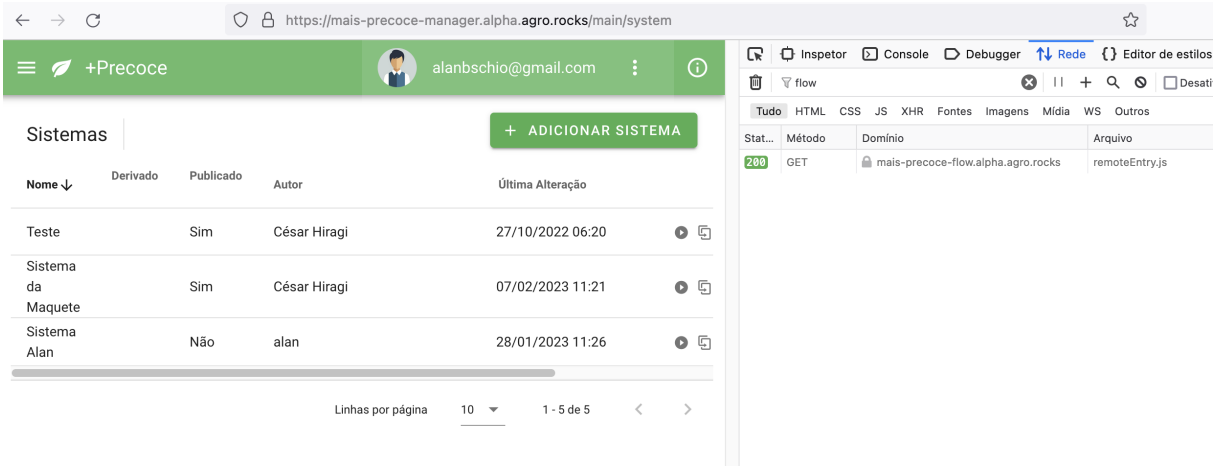


Figura 27 – *Host* manager exibindo arquivos de inicialização do *remote* pluginFlow. Fonte: Autor

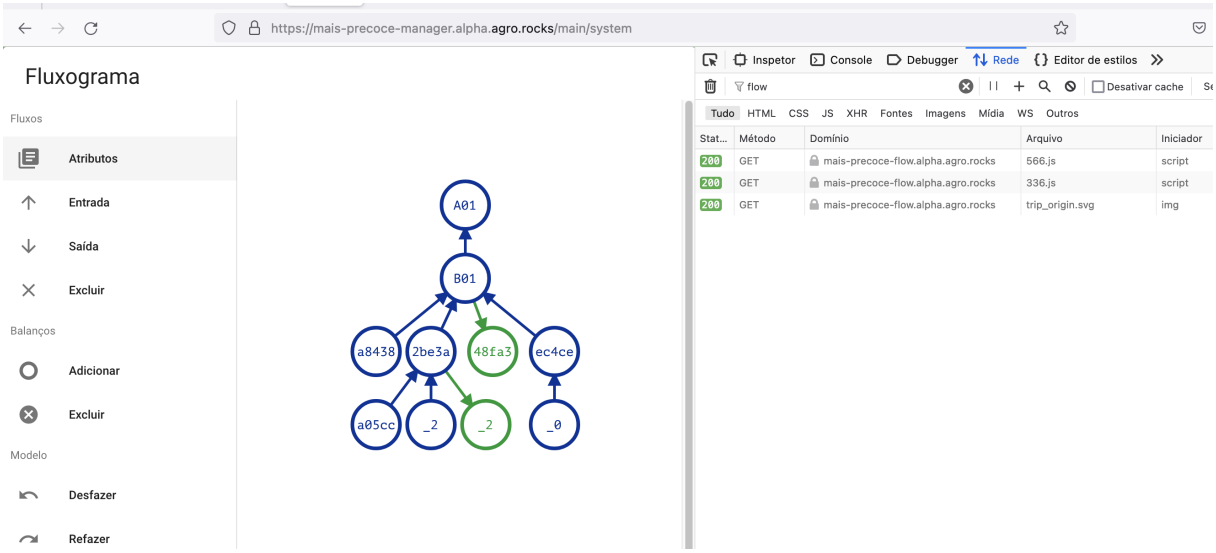


Figura 28 – *Host* manager exibindo componente carregado do *remote* pluginFlow. Fonte: Autor

4 Controle de Indisponibilidade

A implementação da integração utilizando o ModuleFederationPlugin já sana o problema de unificar os projetos em um único sistema. Contudo a P+P possui duas necessidades, que não são contempladas pela implementação original da ferramenta e que demandaram desenvolvimento.

A primeira necessidade: caso algum *remote* esteja com problemas e precise ser desativado, ou ainda não estiver *online*, essa funcionalidade não deve ser disponibilizada ao usuário. Da maneira como o ModuleFederationPlugin atua, seria necessário toda vez, realizar alguma alteração no *host* para que isso não impactasse o mesmo. Além de um segundo fator: o carregamento dos arquivos *remoteEntry* é síncrono, o ModuleFederationPlugin foi desenvolvido síncrono por padrão, de maneira que ao realizar o carregamento inicial do *host*, todos os *remotes* são acessados pelo carregamento dos arquivos **remoteEntry.js**, assim se um *remote* estiver *offline*, toda a plataforma possui seu comportamento comprometido, pois ao ser retornado um *status* de arquivo não encontrado, é disparada uma erro e o restante da validação do código não é realizada. A segunda necessidade: a disponibilização de funcionalidades sem conexão, ou disponibilização *offline*. Ao ser instalada em um celular ou computador, a P+P possui a funcionalidade de ser utilizada sem conexão, e quando o usuário estiver novamente *online*, a mesma sincroniza as informações. O ModuleFederationPlugin utiliza o carregamento dinâmico em tempo real, como descrito no apêndice **??** na seção **??**, isso significa que de fato os componentes dos *remotes* só terão seu código unificado ao *host*, quando o usuário decidir usar as funcionalidades. Ao ser instalada por mais que o registro das funcionalidades tenha sido feito com sucesso, os arquivos necessário para as mesmas ainda não se encontram unidos ao código.

Para solucionar esses problemas foi criado o ModuleFederationEnhancedPlugin, que é baseado no ModuleFederationPlugin. Nele são implementadas as soluções de inicialização assíncrona e disponibilização offline.

4.1 Plugin ModuleFederationEnhancedPlugin

Desenvolvido pelo autor deste trabalho, o ModuleFederationEnhancedPlugin (MFEP) é um *plugin* para o *webpack*, de código aberto, seguindo as mesmas premissas do ModuleFederationPlugin. O desenvolvimento do MFEP se dá por englobar todas as funcionalidades e capacidades do MFP pela utilização da técnica de herança da orientação a objeto. Neste *plugin* foi utilizada da palavra reservada *extends* do JavaScript para definir que o MFEP herda tudo o que o MFP possui, veja o código inicial:

```
1 const ModuleFederationPlugin =
```

```
2    require("webpack").container.ModuleFederationPlugin;
3  class ModuleFederationEnhancedPlugin extends ModuleFederationPlugin {
4    constructor(options) {
5      super(options);
6      this.options = options;
7    }
8  }
9  module.exports = ModuleFederationEnhancedPlugin;
```

4.2 Inicialização Assíncrona

Primeiramente foi realizada a implementação da inicialização assíncrona, assim, quando um *remote* estiver indisponível ou *offline* a inicialização do *host* não será afetada. Para isso é necessário utilizar a abordagem que encadeia o carregamento de um *remote* dentro de uma *Promise*[1] que age como um *Proxy*[2] entre a requisição feita pelo *host* e o carregamento do *remote* de fato. O *webpack* é preparado para receber essa abordagem[3], portanto quando configurado um *remote* no lugar de colocar a URL que o mesmo está, deve-se colocar um *script* como no exemplo a baixo:

```
1  promise new Promise(resolve => {
2    const remoteURL = 'http://localhost:3002/remoteEntry.js'
3    const script = document.createElement('script')
4    script.src = remoteURL
5    script.onload = () => {
6      const asyncRemote = {
7        get: (request) => window.remoteName.get(request),
8        init: (arg) => {
9          try {
10           return window.remoteName.init(arg)
11         } catch(e) {
12           console.log('remote container already initialized')
13         }
14       }
15     }
16     resolve(asyncRemote)
17   }
18   document.head.appendChild(script);
19 })
```

Com essa abordagem, ao inicializar o *host*, em vez de esperar a resposta da requisição para a URL do *remote*, o *webpack* chama a função passada e continua a execução do restante do arquivo. Contudo, apesar de não interromper a execução e inicialização do *host*, ao tentar se usar o componente do *remote* uma exceção de erro é disparada. Como estão sendo utilizadas funções de importação de código, lidar com exceções disparadas durante a importação pode ser um tanto quanto trabalhosa e de difícil compreensão para desenvolvedores, por isso, o *plugin* abstrai toda essa complexidade e em vez de retornar o componente, quando

[1] https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Promise
[2] https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Proxy
[3] https://webpack.js.org/concepts/module-federation

não é carregado retorna um valor nulo, ficando muito mais fácil pro desenvolvedor realizar validação simples como por exemplo *componenteCronogramaOnline = Cronograma !== null*. Primeiro o *plugin* desenvolvido transforma a *string* com a URL do *remote* em um objeto, por exemplo um *remote* **"pluginGantt@pluginGantt.com.br/remoteEntry.js"** se torna um objeto com: *name* de valor **pluginGantt**; *url* de valor **pluginGantt.com.br**; *remoteEntry* de valor **remoteEntry.js**, e em seguida passa esse objeto para a função ***dynamicRemote*** que utiliza da implementação de *promise* indicada pelo *webpack* e adiciona um passo importante, quando o carregamento falhar, é retornada uma instância com funções sem retorno. Assim quando o *webpack* tentar importar o componente que obteve falha no carregamento inicial, ele recebera a instância nula.

```
1  const dynamicRemote = (remote) => {
2   return `(resolve) => {
3    const script = document.createElement("script");
4    script.src = "${remote.url}/${remote.remoteEntry}";
5    script.onload = () => {
6     const module = {
7      get: (request) => window["${remote.name}"].get(request),
8      init: (arg) => {
9       try {
10     return window["${remote.name}"].init(arg);
11      } catch (e) {
12     console.log("Problem loading remote ${remote.name}", e);
13      }
14     },
15    };
16    resolve(module);
17   };
18   script.onerror = () => {
19    const module = {
20      get: () => () => {},
21      init: () => () => {},
22     };
23    resolve(module);
24   }
25   document.head.appendChild(script);
26  }`;
27 };
```

Outro ponto importante que precisa ser alterado é quando a criação de *chunks*, que são parte da técnica do *webpack* de agrupar códigos para uma otimização no carregamento[4]. Como os projetos utilizam o *framework* Vue.js, a compilação não é feita diretamente pelo *webpack*, primeiro ela é passada pelo VueCLI[5], que possui algumas configurações padrão que não condizem com as necessárias para o carregamento assíncrono. Por isso foi criada a função ***setAsyncConfig***, que atualiza as configurações para valores assincronos, veja a implementação:

```
1  const setAsyncConfig = (compiler) => {
2    if (!isVue(compiler)) return compiler;
3    if (!compiler.options.remotes) {
```

[4] https://webpack.js.org/guides/code-splitting/
[5] https://cli.vuejs.org/

```
4       if (compiler.options.optimization)
5         compiler.options.optimization.splitChunks = false;
6       else compiler.options.optimization = { splitChunks: false };
7     } else {
8       const { optimization } = compiler.options;
9       if (optimization.splitChunk) {
10        if (optimization.splitChunk.cacheGroups) {
11          if (optimization.splitChunk.cacheGroups.common) {
12            optimization.splitChunk.cacheGroups.common.chunks = "async";
13          }
14          if (optimization.splitChunk.cacheGroups.defaultVendors) {
15            optimization.splitChunk.cacheGroups.defaultVendors.chunks = "async";
16          }
17        }
18      }
19    }
20    return compiler;
21  };
```

Ambas as funções são utilizadas no arquivo ***ModuleFederationEnhancedPlugin.js***, a primeira após a inicialização do *plugin*, para converter os *remotes* de *stirng* para o uso da abordagem de *Promise*. A segunda é utilizada dentro da função *apply* que é executada no momento da compilação do código, ficando o arquivo ModuleFederationEnhancedPlugin da seguinte maneira:

```
1  const ModuleFederationPlugin =
2    require("webpack").container.ModuleFederationPlugin;
3    const DefaultAsync = require("./src/DefaultAsync");
4    const convertStringToObject = (remote) => {
5      const parts = remote.split("@");
6      const urlParts = parts[1].split("/");
7      const remoteEntry = urlParts.pop();
8      const url = urlParts.join("");
9      return {
10       name: parts[0],
11       url,
12       remoteEntry,
13     };
14   };
15 class ModuleFederationEnhancedPlugin extends ModuleFederationPlugin {
16   constructor(options) {
17     super(options);
18     this.options = options;
19     Object.keys(remotes).forEach(remoteKey => {
20     options.remotes[remoteKey] = `promise new Promise(${DefaultAsync.dynamicRemote(
           convertStringToObject(options.remotes[remoteKey])).toString()})`
21     })
22    }
23    apply(compiler) {
24     DefaultAsync.setAsyncConfig(compiler);
25     super.apply(compiler);
26   }
27 }
28 module.exports = ModuleFederationEnhancedPlugin;
```

Com a implementação do *plugin* pronta, o mesmo foi disponibilizado no NPM[6] que é

[6] https://www.npmjs.com/

um registro *online* de pacotes e dependências, foi necessário realizar a instalação do ModuleFederationEnhancedPlugin em todos os projetos, tanto *remotes* quanto *hosts*, utilizando o código **npm i @schirrel/module-federation-enhanced-plugin –save-dev**. Após a instalação, foram atualizadas duas linhas de código nos arquivos ***vue.config.js*** dos projetos:

```
1  const ModuleFederationPlugin = webpack.container.ModuleFederationPlugin;
2  e
3  new ModuleFederationPlugin({
```

dão lugar as linhas:

```
1  const ModuleFederationEnhacedPlugin = require("@schirrel/module-federation-enhanced-
       plugin");
2  e
3  new ModuleFederationEnhacedPlugin({
```

Ambas as funcionalidades possuem testes unitários implementadas para garantir seu funcionamento. A Figura **??** exibe a P+P funcionando normalmente mesmo com os dois *remotes* estando indisponíveis.

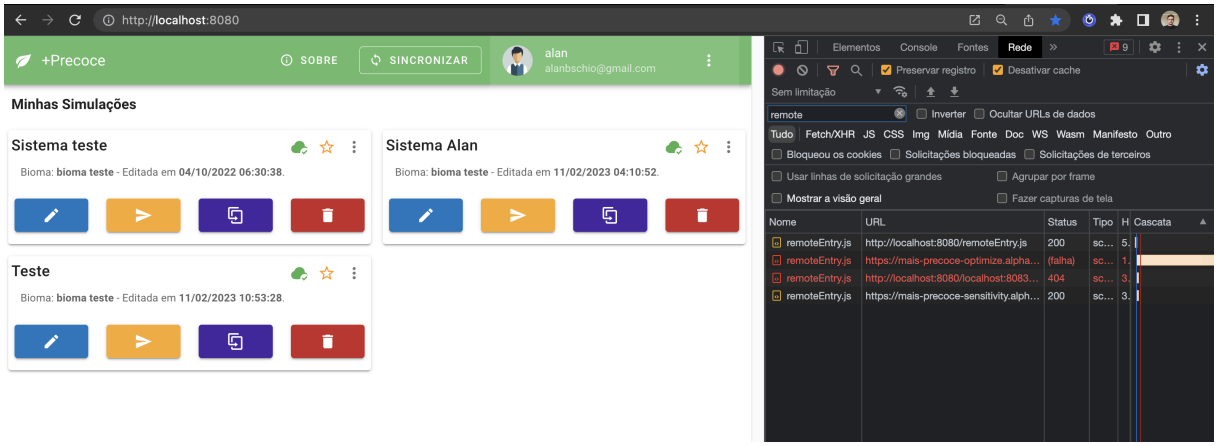


Figura 29 – *Host* sem prejuízo de inicialização mesmo com *remotes offline*. Fonte: Autor

Com isso os dois *hosts* tiveram seu código atualizado para utilizarem a validação de *remote* disponível. No projeto manager foi criada a variável **flowRemoteOnline** dentro da propriedade *data* do componente ***CrudSystem.vue*** e adicionado o código abaixo na função *mounted*:

```
1     mounted () {
2       this.flowRemoteOnline = !!FlowTree
3     },
```

Em seguida se utilizou a variável no código HTML do componente, fazendo a validação de exibição do botão utilizando a diretiva **v-if** do Vue.js no **v-tooltip**, então a funcionalidade de Fluxograma não é mais exibida na lista de opções como mostrado na Figura **??**, e código a baixo:

```
1  <v-tooltip bottom :open-on-focus="false" :open-on-click="false"
```

```
2       v-if="flowRemoteOnline">
3     <template v-slot:activator="{ on, attrs }">
4       <v-icon small class="mr-2" @click="flowForm(item)" v-bind="attrs" v-on="on">
5         mdi-play-circle
6       </v-icon>
7     </template>
8     <span>Abrir o Fluxograma</span>
9   </v-tooltip>
```

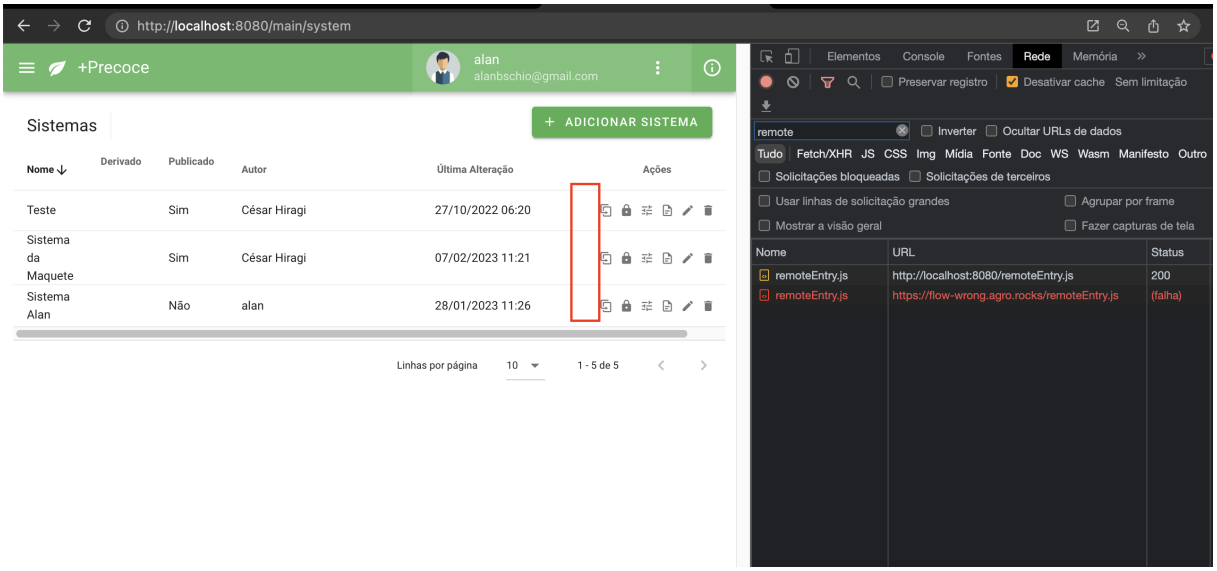


Figura 30 – *Remotes* pluginFlow *offline* e *host* manager não disponibilizando sua funcionalidade. Fonte: Autor

No pwa foi atualizado o componente ***SimulationCardMenu.vue***, que agora realiza o preenchimento da lista de opções do menu de acordo com a disponibilidade do *remote*. Foi necessário somente atualizar o código *JavaScript* do componente e para o preenchimento da lista. A Figura **??** mostra o pwa exibindo somente o *remote* que foi carregado corretamente, abaixo segue o código atualizado:

```
1  import Optimization from 'pluginOptimize/Optimization'
2  import Cronograma from 'pluginGantt/GanttChart'
3  import Sensibilidade from 'pluginSensitivity/SensitivityAnalysis'
4  export default {
5     data () {
6       return {
7         menus: [],
8         moduleToShow: null,
9       }
10    },
11    async mounted () {
12      if (Cronograma) {
13        this.menus.push({name: 'Cronograma', component: Cronograma })
14      }
15      if (Sensibilidade) {
16        this.menus.push({ name: 'Analise de Sensibilidade', component: Sensibilidade })
17      }
18      if (Optimization) {
19        this.menus.push({ name: 'Otimizador', component: Optimization })
20      }
```

```
21        }
22    }
```

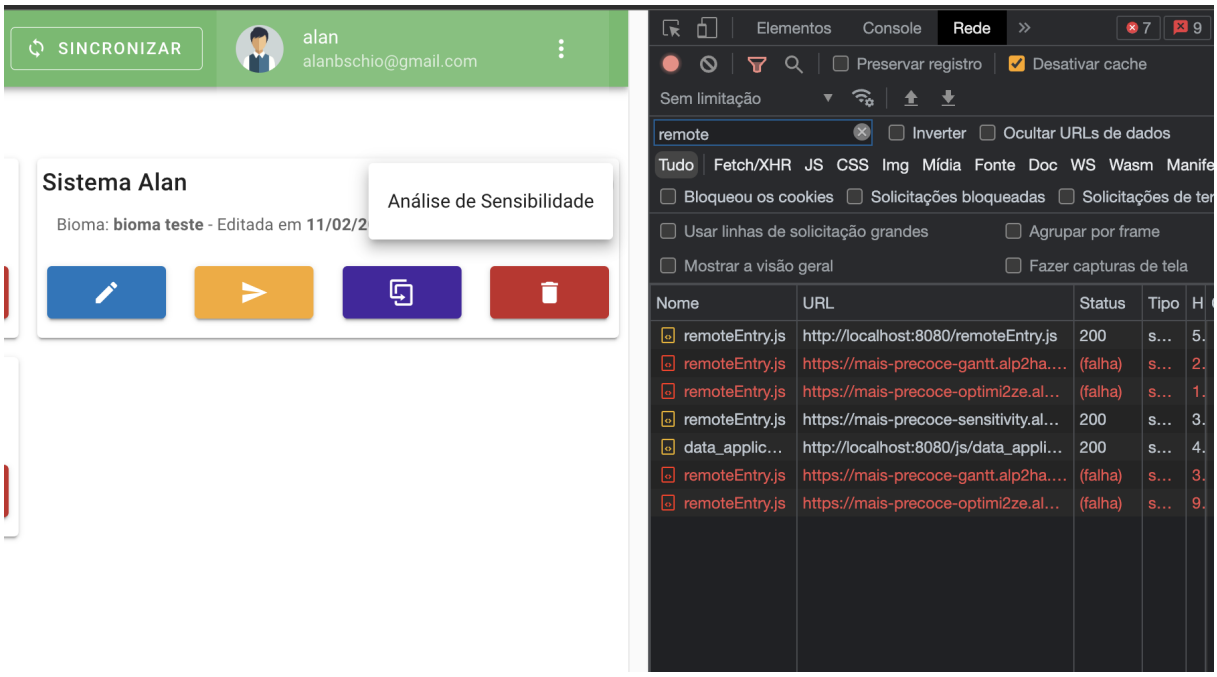


Figura 31 – *Remotes offline* e *host* pwa exibindo somente funcionalidade do *remote* disponível. Fonte: Autor

4.3 Disponibilização Offline

Para que se possa realizar a disponibilização *offline* dos arquivos, é necessário que se tenha mapeados o nome de todos os *remotes* e arquivos que cada um expõe, que por ventura podem ser utilizados pelo *host*. Caso essa tarefa ficasse a cargo do desenvolvedor, ela poderia ser trabalhosa e repetitiva. Contudo já existe um lugar onde ambas as informações são adicionadas obrigatoriamente: ***vue.config.js***, na configuração do MFP. Porém este arquivo existe e só é acessível em tempo de compilação de código. E as informações acima citada, são necessárias em tempo de execução. Foi então realizado o desenvolvimento dentro do MFEP para que ele gere essa listagem e a inclua como um dos itens expostos do *remote* e *host* automaticamente.

Inicialmente durante a fase de estudo sobre a implementação da funcionalidade, suas possibilidades e empecilhos, foi criada uma *issue*[7] no repositório oficial de exemplos do uso de MFP, repositório mantido e que tem grande parte das dúvidas respondidas pelo principal criador do MFP. Ao discutir possibilidades da disponibilização e o motivo de ser necessário, chegou-se ao entendimento de que a listagem de *exposes* e *remotes* adicionados a configuração do MFP era de fato algo útil para a necessidade da disponibilização, mas não somente para esta. O principal mantenedor e criador do MPF, Zackary, inclusive

[7] https://github.com/module-federation/module-federation-examples/issues/1323

sugeriu, na mesma *issue*, que fosse realizada uma tentativa de adicionar essa funcionalidade ao *webpack*, o que foi feito após criar uma *issue*[8] no repositório oficial do *webpack* e logo em seguida realizada a solicitação de adição de código[9], popularmente conhecida como PR. Ao analisar a implementação sugerida, o criador do *webpack* Tobias Koppers, informou que apesar de parecer necessária ao MFP a funcionalidade não aparentava fazer tanto efeito ao *webpack* no geral. Por fim Tobias ajudou mostrando uma maneira mais simplificada de realizar a listagem do que havia sido sugerida pelo autor deste trabalho.

No MFEP foi criado o arquivo ***GenerateModuleNameList.js***, que gera a listagem de projetos baseada nas informações adicionadas na propriedade *exposes* das configurações do *plugin*, essa listagem é criada com nome de *moduleNameList* e dispõe a listagem a ser utilizada junto aos *exposes*, para ser utilizada pelo *host* que utilizar o *remote*. Implementação da listagem:

```
1  const GenerateModuleNameList = (options) => {
2      const exposedKeys = Object.keys(options.exposes);
3      if (exposedKeys.length) {
4        return {
5          "./moduleNameList": `data:application/json,${JSON.stringify(exposedKeys)}`,
6        };
7      }
8      return {};
9    };
```

Da mesma maneira foi criado o arquivo ***GenerateRemoteUrlMap.js***, que gera a lista de *remotes*, com suas respectivas URLs e que disponibilizará a lista com nome de *remoteUrlMap* para ser unida a propriedade *exposes*. Esta funcionalidade é injetada dentro das informações do *host* para que o mesmo use-a para ter conhecimento geral de todos *remotes* que ele possuem em tempo de execução. A implementação da geração da listagem no arquivo se dá por:

```
1  const generateRemotesList = (options) => {
2    return Object.keys(options.remotes).map((remoteName) => {
3      const remote = options.remotes[remoteName];
4      return { [remoteName]: remote.split("@")[1] }
5    });
6  };
7  const GenerateRemoteUrlMap = (options) => {
8    if (options.remotes) {
9      return {
10       "./remoteUrlMap": `data:application/json,${JSON.stringify(
11           generateRemotesList(options)
12            .reduce((obj, item) => Object.assign(obj, { ...item }), {})
13       )}`,
14     };
15   } else return {};
16 };
```

[8] https://github.com/webpack/webpack/issues/15229
[9] https://github.com/webpack/webpack/pull/15357

Ambos os arquivos são importados no arquivo principal do *plugin*, ***ModuleFederationEnhancedPlugin.js***, e seus valores se unem com o valor já existente na propriedade *exposes*. A seguir o código adicionado ao arquivo:

```
1  const GenerateRemoteUrlMap = require("./src/GenerateRemoteUrlMap");
2  const GenerateModuleNameList = require("./src/GenerateModuleNameList");
3
4  class ModuleFederationEnhancedPlugin extends ModuleFederationPlugin {
5    constructor(options) {
6      super(options);
7
8      options.exposes = {
9        ...options.exposes,
10       ...GenerateModuleNameList(options),
11       ...GenerateRemoteUrlMap(options),
12     };
13   }
14 }
```

Para realizar o carregamento dinâmico baseado nas duas listagens, foi criado também um arquivo utilitário, chamado *helper* dentro do repositório do MFEP, para abstrair a complexidade. Assim os *host* só precisam realizar a invocação da função e utiliza-lá. O *helper* ganhou o nome de **@schirrel/module-federation-enhanced-plugin-helper** para uma melhor separação de responsabilidade, e também para poder ser utilizado sem a necessidade do vínculo com o *plugin*, se necessário. O *helper* disponibiliza a função *loadRemoteAndModules* que deve receber como propriedade o valor que foi definido na propriedade *name* utilizado na configuração do MFEP no arquivo ***vue.config.js*** do *host*. A função *loadRemoteAndModules* inicialmente utiliza a função *loadRemotes* utilizando o *name* do *host* para carregar a listagem de *remotes* que o mesmo possui através do arquivo *remoteUrlMap* criado pelo MFEP. A função *loadRemotes* realiza a comunicação inicial com o *remote* e obtém do mesmo a listagem de *modules*, que são os arquivos expostos pelo *remote*. Esta lista de *remotes* e seus respectivos *modules* é devolvida ao *loadRemoteAndModules* que invoca a função *loadModules* repassando estas informações. Então dentro da *loadRemoteAndModules* é invocada a *loadModules*, recebendo a listagem anterior. A *loadModules* por sua vez faz o carregamento de todos os arquivos disponíveis por cada *remote*. A implementação dos arquivos utilizados pelo *helper* se encontram no apêndice **??**. Assim fica a cargo do *host* realizar a instalação do *helper* utilizando o comando **npm i @schirrel/module-federation-enhanced-plugin-helper**, importar e utilizar a função *loadRemoteAndModules*.

Durante o desenvolvimento foi identificado esse carregamento dinâmico baseado em listagem, não é possível de ser feito utilizando a função *import* usada anteriormente. Uma vez que a mesma não aceita que seja passada como propriedade uma variável. Para sobressair a esse problema foi criado o arquivo ***getModule.js***, que pode ser visto no apêndice **??**, onde os arquivos são carregados utilizando a funcionalidades do *webpack* que são mais trabalhosas de se utilizar.

No pwa foi criado o arquivo ***src/helpers/module-federation/index.js*** que possui o seguinte código:

```
1 import { loadRemoteAndModules } from '@schirrel/module-federation-enhanced-plugin-
      helper'
2
3 export const loadAllRemotesToSaveOffline = () => {
4   // same name as in vue.config.js
5   loadRemoteAndModules('pwa')
6 }
```

Em seguida foi adicionado ao arquivo ***App.vue*** a importação e utilização desse arquivo, dentro da função *mounted*, do ciclo de vida do Vue.js, ficando da maneira a seguir:

```
1 import { loadAllRemotesToSaveOffline } from '@/helpers/module-federation'
2 export default {
3   mounted () {
4     loadAllRemotesToSaveOffline()
5   }
6 }
```

Agora ao ser inicializado o P+P já carrega todas as informações prévia que possam ser necessárias. O projeto já possui configuração de armazenamento dos arquivos em cache para utilização *offline*, com isso, ficou a cargo deste trabalho, apenas garantir que os mesmos sejam carregados.

4.4 Resultados do Controle de Disponibilidade

Após a implementação e utilização da mesma, pode ser ver na Figura **??** que ao ser carregado o *host* pwa, agora carrega todas os arquivos necessários para a funcionalidade dos *remotes*, por exemplo o pluginGantt. Em seguida é possível ver na Figura **??** que com a internet do computador desligada, as opções no menu continuam aparecendo e é possível selecionar o Cronograma. E ao selecionar o Cronograma, pode ser visto na Figura **??** o funcionamento do componente totalmente offline.

O plugin ModuleFederationEnhancedPlugin está disponível online no Github pelo URL https://github.com/schirrel/ModuleFederationEnhancedPlugin. E seus dois pacotes disponível, *plugin* e *helper* podem ser encontrados na listagem de pacotes NPM pelas URLs https://www.npmjs.com/package/@schirrel/module-federation-enhanced-plugin e https://www.npmjs.com/package/@schirrel/module-federation-enhanced-plugin-helper.

4.5 Considerações Finais

As funcionalidades implementadas no plugin MFEP possuem testes unitários para garantir a sua funcionalidade, o desenvolvimento dos testes unitários foi feito realizando a

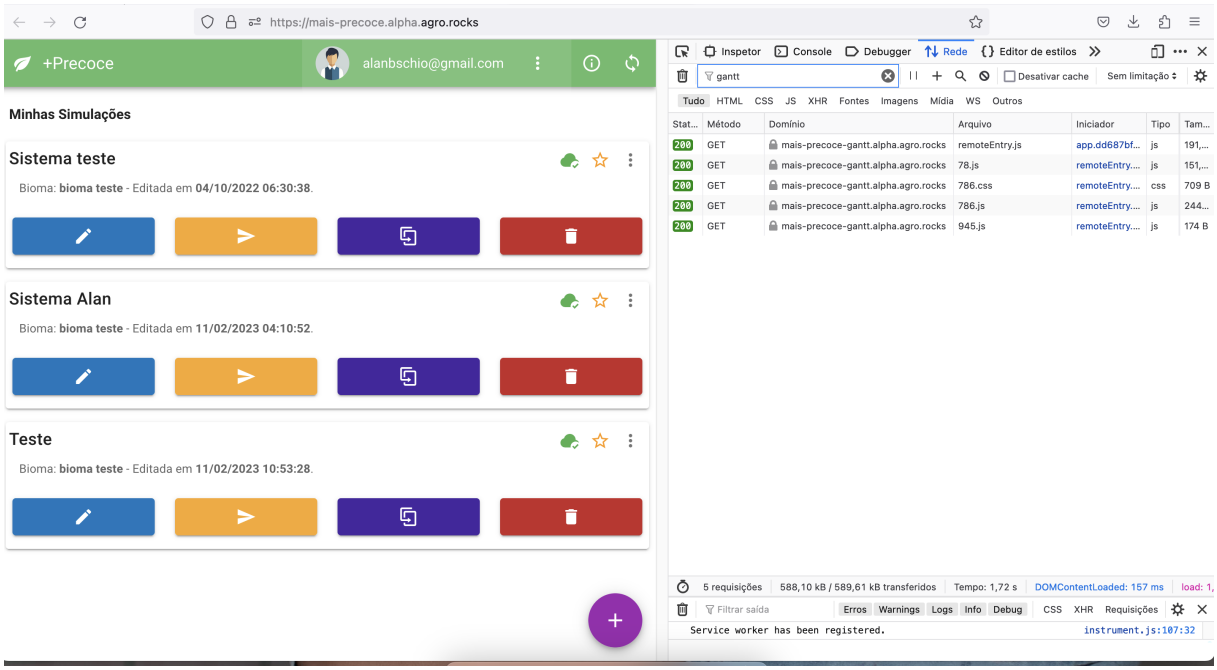


Figura 32 – *Host* pwa carregando todos os arquivos do *remotes* pluginGantt para uso *offline*. Fonte: Autor

biblioteca AVA[10], a escolha dessa biblioteca se da pela sua implementação simplificada e como a mesma possui um ótimo detalhamento de erros em caso de falha de teste. De acordo com o relatório[11] de cobertura de código o testes cobrem mais de 90% dos códigos

Durante o desenvolvimento do *plugin* foi possível entender um pouco mais de como o *webpack* e o Module Federation trabalham nas suas camadas mais inferiores. Ao implementar funcionalidades e sobrescrever outras, foi possível ter um conhecimento melhor de sua complexidade e de como ele abstrai o desenvolvimento, facilitando a vida do desenvolvedor.

Outro ponto interessante de se destacar é que foi possível ter algumas conversas por Github[12], Twitter, mensagem privada, e E-mail com o principal criador do ModuleFederationPlugin, Zackary Jackson, onde o mesmo ajudou em como implementar algumas partes das funcionalidades. Esse tipo de comunicação só foi possível pois o *webpack* e o Module Federation são ferramentas de código aberto, onde a comunidade de desenvolvimento é engajada em aprender e evoluir o projeto, bem como passar seu conhecimento e ajudar outras pessoas. As implementações do *plugin* e o que ele se propõe corrigir, o autor deste trabalho foi convidado[13] a fazer parte do projeto que esta em desenvolvimento e que será a próxima versão do ModuleFederationPlugin, já tendo algumas funcionalidades

[10] https://github.com/avajs/ava
[11] https://app.codecov.io/gh/schirrel/ModuleFederationEnhancedPlugin
[12] https://github.com/module-federation/module-federation-examples/issues/1323
[13] https://github.com/module-federation/module-federation-examples/issues/1323#issuecomment-1053342919

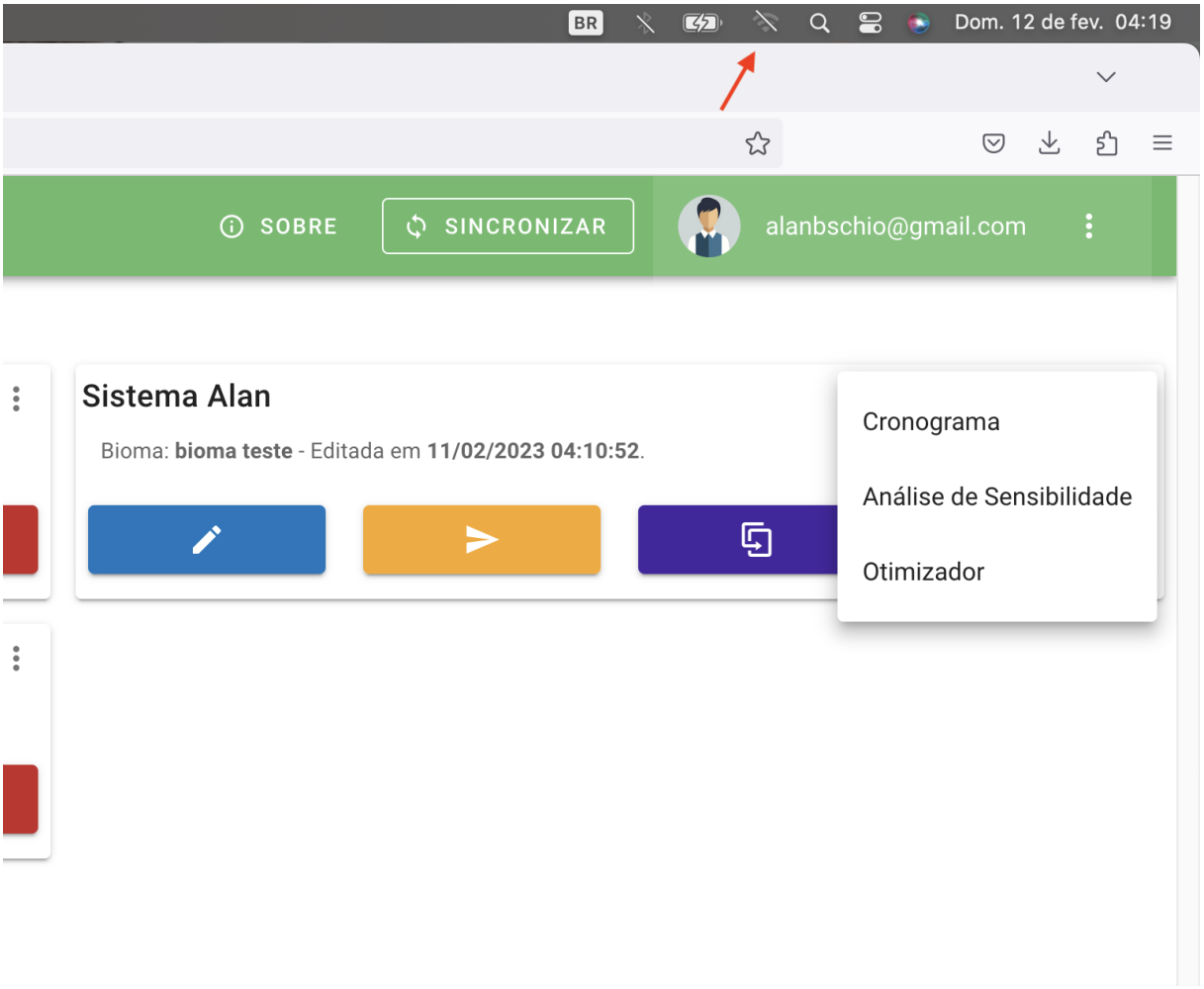


Figura 33 – *Host* pwa sendo utilizado sem internet. Fonte: Autor

como a listagem de remotes e *modules*[14,15] incorporadas ao código da nova versão. Com isso, pode-se ter esperança de uma parte dos desenvolvimentos realizados aqui, possam a vir fazer parte das funcionalidades do próprio MFP, sem a necessidade de um *plugin* intermediário, no caso o MFEP.

[14] https://github.com/module-federation/enhanced/issues/1
[15] https://github.com/module-federation/enhanced/pull/2

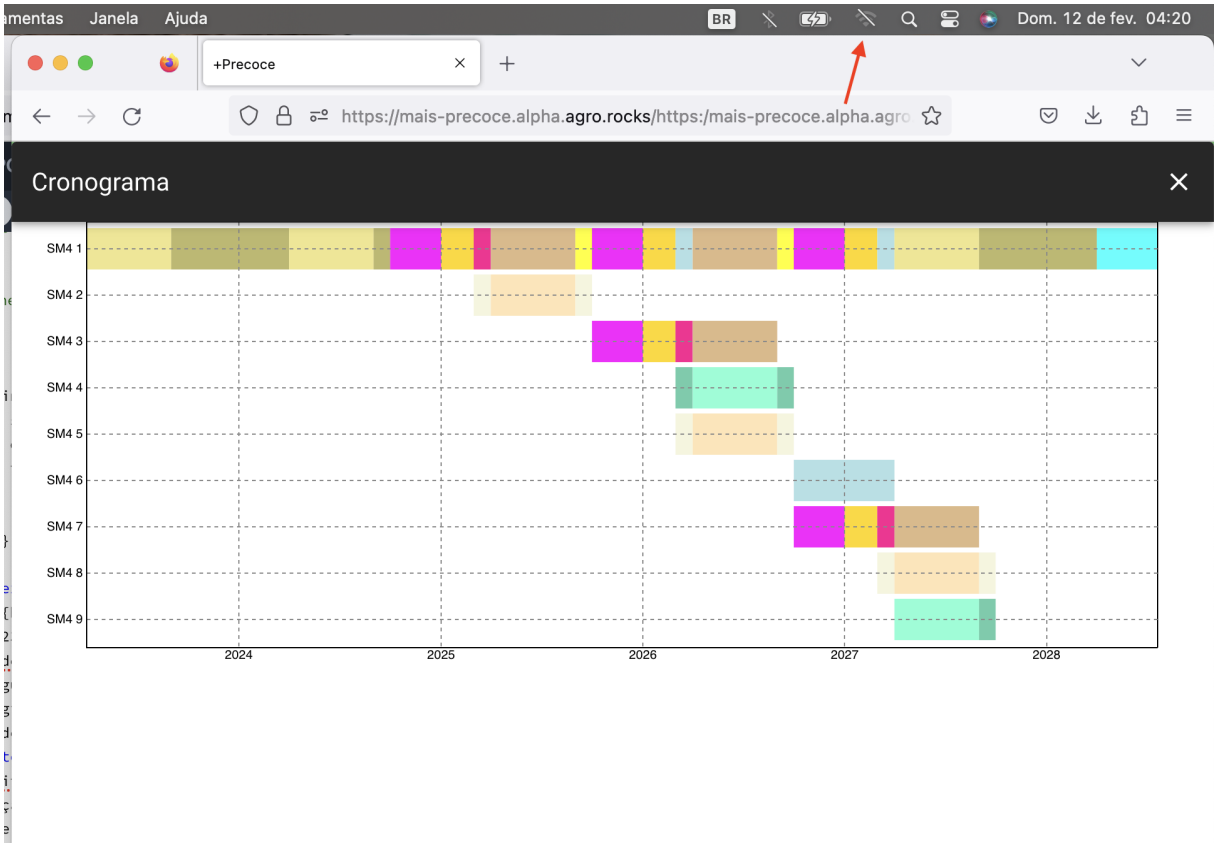


Figura 34 – *Host* sendo utilizado sem internet e exibindo funcionalidade de Cronograma do *remote* pluginGantt. Fonte: Autor

5 Melhorias no Módulo Fluxograma

Apesar de todos os *remotes* estarem integrados ao seus respectivos *hosts* e exibirem as funcionalidades, estas não estavam prontas para uso final, pois não utilizam os dados originadas da plataforma e repassados pela integração, todos utilizam dados fictícios. Isso acontece devido alteração do formato e estrutura de dados que a P+P utiliza ser diferente da esperada entre os módulos, devido a decisões internas da empresa sobre a estrutura de dados após o desenvolvimento dos módulos. Para demonstrar uma integração completa de funcionalidade, foi decidido realizar a adaptação do formato e estrutura de dado no módulo Fluxograma.

A alteração do formato de dados realizado pela empresa, permitiu simplificar o processo anterior, pois o novo formato de dado traz todos as informações necessárias sobre um nó ou fluxo. Sendo somente necessária uma conversão simples de lista em árvore. Ao aplicar correções sobre o uso das informações e processo de conversão da lista para o objeto do D3, entendeu-se que para uma melhor manutenção futura, uma reestruturação da utilização de arquivos era necessária. O que antes era um único arquivo *D3Tree.js* contento regras de desenho de gráfico, regras de interface, regras de utilização e interação com o gráfico, conversão de objeto, desconversão de objeto, montagem de novo objeto, agora se tornará um arquivo com responsabilidades apenas gráficas: desenho de gráfico e regras de utilização e interação com o gráfico. Optou-se por realizar a separação de responsabilidade em demais arquivos que dispõe da funcionalidade. As funções responsáveis por administrar e manipular a listagem de dados não são mais do escopo da biblioteca gráfica. Assim, toda a regra de negócio por trás da adição, edição e alteração de um nó, fluxo ou balanço foi isolada para fora do componente de interface. Para realizar esse isolamento atendeu-se a um outro requisito existente no projeto pluginFlow e que até então não havia sido possível de realizar: utilização de *CoffeeScript* junto ao projeto. O *CoffeeScript* é uma linguagem de programação que compila para JavaScript que visa por facilitar e simplificar o uso de JavaScript[1] por meio de uma sintaxe mais concisa. Para isso foi adicionado ao projeto a dependências 'coffee-loader' e adicionada a configuração no arquivo ***vue.config.js*** o seguinte código:

```
1    module: {
2      rules: [
3      {
4        test: /\.coffee$/,
5        use: [
6        {
7          loader: 'coffee-loader',
8          options: { sourceMap: true }
9        }]
```

[1] https://coffeescript.org/

```
10        }]
11    }
```

Os arquivos com as novas funcionalidades estão na pasta */coffee*, e seus arquivos precisam possuí ao seu final a linha de código *module.exports = { }*, onde dentro das chaves **{}** estarão os nomes das funções, variáveis ou constantes que possam ser utilizadas em outros arquivos. Para utilizar essas funções em outros arquivos *CoffeeScript* deve-se usar a função *require*, por exemplo:

```
1    const funcionalidade = require("arquivoDeFuncionalidade.coffee").funcionalidade
2    const constanteStatus = require("arquivoDeFuncionalidade.coffee").constanteStatus
3    // ou
4    const funcionalidades = require("arquivoDeFuncionalidades.coffee")
5    funcionalidades.minhaFuncionalidade()
6    funcionalidades.constanteStatus
```

Já para arquivos *JavaScript* deve-se usar a funcionalidade *import* , por exemplo:

```
1    import { funcionalidade, constanteStatus } from "arquivoDeFuncionalidade.coffee"
2    // ou
3    import funcionalidades from "arquivoDeFuncionalidades.coffee"
```

Com a utilização de arquivos *CoffeeScript* foi possível realizar melhorias de interface e regra de negócio:

Toda regra de negócio ou processamento de dado ou informação é realizado agora em arquivos dedicados. Isso beneficia o desenvolvimento pois se tem um melhor entendimento do arquivo e suas funções, bem como facilita a análise do código para se realizar manutenções.

Com as adaptações realizadas, o usuário que utilizar o Gestor e clicar pela ação "Abrir o Fluxograma", como mostrado Figura **??** da seção **??**, será exibido o Fluxograma contendo as informações que o Gestor obteve da base de dados e repassou para o componente. Todas as ações realizadas no Fluxograma surtem efeito no dado, como por exemplo quando o usuário editar e salvar o Fluxograma, este é devolvido para o Gestor que atualiza essa informação na base de dados. Por fim, é toda uma nova funcionalidade completa integrada ao sistema final.

Resultados e Conclusão

Neste presente trabalho foi desenvolvido a integração entre seis projetos por meio da abordagem de Micro Frontends usando a ferramenta Webpack ModuleFederationPlugin. Foi desenvolvido o plugin auxiliar ModuleFederationEnhacedPlugin que se baseia no ModuleFederationPlugin e adiciona mais funcionalidades necessárias para os módulos integrados. Foi desenvolvido por padrão *remotes* carregados de forma assíncrona, para que sua indisponibilidade não afete a inicialização de seu *host*, com isso também foi desenvolvido uma maneira de permitir que seu *host* possa lidar com a essa indisponibilidade do *remote* sem que haja exceções ou erros no código. Foi desenvolvido o ModuleFederationEnhacedPluginHelper que possibilita o carregamento de todo o conteúdo necessário para a funcionalidade disposta em um *remote* de uma única vez, permitindo que o *host* que utilize a técnica de PWA para funcionamento *offline* consiga disponibilizar também as funcionalidades dos *remotes*.

Como pode ser visto na seção **??** a integração apresenta bons resultados na disponibilização de funcionalidades de maneira independente e a parte do código principal da plataforma. Tendo os demais projetos se juntando em tempo de execução, no próprio navegador do usuário. Com a implementação da integração, o processo de disponibilidade sobre uma correção ou adição de funcionalidade em um módulo se torna mais rápido, uma vez que somente aquele determinado projeto afetado é que precisa passar por todo o processo de geração do artefato final (*build*) e por todo o fluxo de disponibilização. Como os projetos possuem seus escopos menores e delimitados, tem assim menos códigos, o que torna sua compilação mais rápida e consequentemente sua disponibilização. Outro ganho é quanto a realização de novos desenvolvimentos, ao serem realizados novos módulos, estes podem sem integrar desde o começo a plataforma, sem a preocupação de quebra no código principal ou de efeito colateral, assim ao fim de um desenvolvimento ou entrega de um novo trabalho, pode-se finalizar já entregando a funcionalidade nova para o usuário, diferente das entregas anteriores de módulos da plataforma. Por último, o Anexo **??** traz a documentação de como configurar novos projetos para que possam ser integrados na P+P, além de ter adicionar a documentação interna da Embrapa um guia de utilização do MF, MFP e MFPE nos projetos, bem como a passagem de conteúdo também por meio de reuniões com os responsáveis técnicos e demais desenvolvedores.

Além dos resultados em código, se obtêm também o resultado de contribuição tecnológica tanto para a Embrapa, ao fornecer uma nova arquitetura distribuída e modularizada para seus projetos *frontend*, quando para o Programa de Pós graduação Mestrado Profissional em Computação Aplicada, por ser o primeiro trabalho a trazer a inovação da abordagem de Micro Frontends para ser exploradas e trabalhos de demais colegas e

soluções de problemas.

Trabalhos Futuros

No que tange aspectos técnicos da integração, o presente trabalho desenvolveu toda a estrutura necessária para a integração dos módulos atuais e futuros e resolveu todos os problemas que foram encontrados durante o desenvolvimento, ficando a cargo de novos trabalhos as funcionalidades ou problemas que possam ser identificados no futuro.

Contudo, afim de deixar a utilização de *remotes* mais dinâmica e com menos necessidade de alteração em seu *host*, existem a possibilidade da padronizações na exportação dos módulos. Por exemplo, tanto para a Aplicação Portal quanto Gestor, os componentes dos *remotes* foram utilizados em menus. Além de importar o componente do *remote* foram realizadas codificações com valores fixos para definir o texto apresentado no menu ou seu ícone. Esses valores fazem mais sentido de serem trazidos junto ao componente importado, uma vez que o *remote* é o dono da funcionalidade e deve ditar sua definição de aparência no menu. Por exemplo onde foi realizado o uso de:

```
1    this.menus.push({name: 'Cronograma', component: Cronograma })
```

sendo substituído por

```
1    this.menus.push(CronogramaMenuItem)
```

Tais definições não foram realizadas neste presente trabalha por conta de seu foco na estrutura técnica da integração e por ter sido optado por realizar o minimo possível de alteração nos módulos que são *remotes*.

Outra possibilidade é a adição da configuração do uso de Module Federation ao projeto padrão utilizado pela empresa, ou a criação de um novo projeto padrão para ser utilizado em projetos que se integrem. A Embrapa utiliza em sua plataforma *Embrapa.io* o conceito de projeto padrão chamado de *boilerplate*, um trabalho interessante seria incorporar nesses *boilerplate* o desenvolvimento realizado para a funcionalidade de monitoramento, o Sentry, e realizar o mesmo tipo de desenvolvimento para a funcionalidade de analise Matomo que foi incorporada nas ultimas semanas pela equipe. Também seria interessante a criação de *boilerplate* específicos para *hosts* e *remotes*.

Uma oportunidade de extrema grandeza se dá maior teste e validação de comportamento do MF com as funcionalidades de PWA, entendendo se existe demais desenvolvimentos necessários para a utilização de outras funcionalidades progressivas. Entender como *push notifications* e como o acesso de funcionalidades do dispositivo se comportam ao utilizarem códigos distribuídos por exemplo.

Fica a cargo da Embrapa definir se realizará uma internalização do código do

ModuleFederationEnhancedPlugin[2], que hoje tem código aberto e livre. Caso a decisão de internalização seja tomada, fica a cargo de trabalhos futuros a atualização de todos os projetos para que utilize a internalizada dentro dos arquivos *package.json*, bem como se realizará a alteração da biblioteca de testes unitários para a ferramenta utilizada internamente, o Jest.

Por último, e não menos importante, o acompanhamento do plugin *https://github.com/module-federation/enhanced*, que é a versão oficial de melhorias para o MF, que já possui algumas das funcionalidades criadas no MFEP, para que no momento que este obter a funcionalidade de carregamento assíncrono de remotes poder realizar a migração para o mesmo.

Ficam a cargos de trabalhos futuros as adequações relativas a adaptação de estrutura do dado, dos mesmos moldes do realizado no capítulo **??** nos demais projetos dos módulos: pluginGantt, pluginOptimize e pluginSensitify. Para que suas funcionalidades possam de fato ser utilizadas na P+P. Tais ajustes não foram efetuados no presente trabalho pois precisam de validação quanto a regras de negócio dos módulos, bem como o foco do trabalho foi a do desenvolvimento de partes técnicas da integração, como falado anteriormente.

[2] https://github.com/schirrel/ModuleFederationEnhancedPlugin

Referências

ABREU, R. A. F. de. *Interface gráfica para a Plataforma +Precoce: Uma visão em Gantt Chart*. Dissertação (Mestrado) — Universidade Federal de Mato Grosso do Sul, 2020. Citado 8 vezes nas páginas 13, 1, 5, 6, 7, 17, 20 e 73.

BASSO, T. *Plataforma +precoce: Simulador de sistemas de produção e novilho precoce*. Dissertação (Mestrado) — Universidade Federal de Mato Grosso do Sul, 2018. Citado 6 vezes nas páginas 13, 1, 2, 5, 6 e 17.

BOST, B. *Micro-frontend Architectures on AWS*. 2021. Disponível em: <https://aws.amazon.com/blogs/architecture/micro-frontend-architectures-on-aws/>. Citado na página 10.

CARDOSO, E. M. *Análise de Sensibilidade na Plataforma +Precoce*. Dissertação (Mestrado) — Universidade Federal de Mato Grosso do Sul, 2021. Citado 7 vezes nas páginas 13, 1, 5, 7, 8, 17 e 32.

CNN. *Procura por profissionais de tecnologia cresce 671% durante a pandemia*. 2021. Disponível em: <https://www.cnnbrasil.com.br/business/procura-por-profissionais-de-tecnologia-cresce-671-durante-a-pandemia/>. Citado na página 1.

DENNING, J. *A step-by-step guide to single-spa*. 2016. Disponível em: <https://canopytax.github.io/post/a-step-by-step-guide-to-single-spa>. Citado na página 13.

ENDEAVOUR. *Escalabilidade: saiba quão longe sua ideia pode ir*. 2015. Disponível em: <https://endeavor.org.br/estrategia-e-gestao/escalabilidade/>. Citado na página 1.

ENZ, L. de G. *Estudo para simulação a eventos discretos e contínuos para aplicações na pecuária*. Dissertação (Mestrado) — Universidade Federal de Mato Grosso do Sul, 2019. Citado 5 vezes nas páginas 13, 1, 5, 6 e 17.

GEERS, M. *Micro Frontends in action*. [S.l.]: Manning Publications Co., 2020. Citado 4 vezes nas páginas 13, 10, 11 e 12.

JACKSON, C. *MicroFrontends*. 2019. Disponível em: <https://martinfowler.com/articles/micro-frontends.html>. Citado 2 vezes nas páginas 9 e 10.

JACKSON, Z. *Webpack 5 Module Federation: A game-changer in JavaScript architecture*. 2020. Disponível em: <https://medium.com/swlh/webpack-5-module-federation-a-game-changer-to-javascript-architecture-bcdd30e02669>. Citado na página 15.

JACKSON, Z.; HERRINGTON, J. *Practical module federation*. Shopify, 2020. Disponível em: <https://module-federation.myshopify.com/products/practical-module-federation>. Citado 9 vezes nas páginas 14, 10, 13, 17, 65, 66, 67, 68 e 71.

LAURILA, S. *An Exploratory Study of Micro Frontends*. Dissertação (Mestrado) — Department of Computer and Information Science, 2020. Citado na página 15.

MEZZALIRA, L. *Building Micro-Frontends*. 1. ed. [S.l.]: O'Reilly Media, 2021. 337 p. ISBN 9781492082965. Citado 2 vezes nas páginas 12 e 13.

MONTELIUS, A. *An Exploratory Study of Micro Frontends*. Dissertação (Mestrado) — Department of Computer and Information Science, 2021. Citado na página 10.

MORAIS, J. de S. *Integração do Modelo AMPL a Plataforma Computacional de Pecuária +Precoce*. Dissertação (Mestrado) — Universidade Federal de Mato Grosso do Sul, 2021. Citado 5 vezes nas páginas 13, 1, 5, 7 e 17.

PAVLENKO, A. et al. Micro-frontends: application of microservices to web front-ends. *Journal of Internet Services and Information Security*, v. 10, p. 49?66, May 2020. Citado 4 vezes nas páginas 8, 9, 10 e 12.

PELTONEN, S.; MEZZALIRA, L.; TAIBI, D. Motivations, benefits, and issues for adopting micro-frontends: A multivocal literature review. *Information and Software Technology*, v. 136, p. 106571, 2021. ISSN 0950-5849. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0950584921000549>. Citado na página 10.

ROSSOUW, M. *Webpack 5 and Module Federation - A Microfrontend Revolution*. 2020. Disponível em: <https://dev.to/marais/webpack-5-and-module-federation-4j1i>. Citado na página 13.

SILVA, C. H. *Micro frontends - Uma abordagem de microservices para o front-end*. 2019. Disponível em: <https://www.infoq.com/br/presentations/micro-frontends-microservice-front-end/?utm_source=qconsp2019&utm_medium=palestra>. Citado na página 13.

SIMEONOVA, A. *Introduction to Luigi*. 2020. Disponível em: <https://developers.sap.com/tutorials/luigi-getting-started.html>. Citado na página 13.

STEYER, M. *Import Maps Ð The Next Evolution Step for Micro Frontends?* 2022. Disponível em: <https://www.angulararchitects.io/aktuelles/import-maps-the-next-evolution-step-for-micro-frontends-article/>. Citado na página 13.

THOUGHTWORKS. *Technology Radar - Micro Frontends*. 2016. Disponível em: <https://www.thoughtworks.com/pt/radar/techniques/micro-frontends>. Citado 2 vezes nas páginas 9 e 13.

WEBPACK. *Chanelog Verson 5, Module Federation*. 2020. Disponível em: <https://github.com/webpack/changelog-v5/blob/master/README.md#module-federation>. Citado na página 15.

Apêndices

APÊNDICE A – ModuleFederationPlugin

Neste apêndice mostraremos como utilizar o ModuleFederationPlugin bem como explicando cada etapa do processo e funcionamento do mesmo.

A.0.1 Configuração

Vamos mostrar como de fato é facilitado o procedimento de configuração de um projeto *host* e *remote* utilizando como base projetos VueJS criados através da *vue-cli*. Para isso executamos os comandos abaixo para criação de cada:

- *vue create projeto-host*
- *vue create projeto-remote*

Estes comando geraram duas pastas de projeto: projeto-host e projeto-remote **??**.

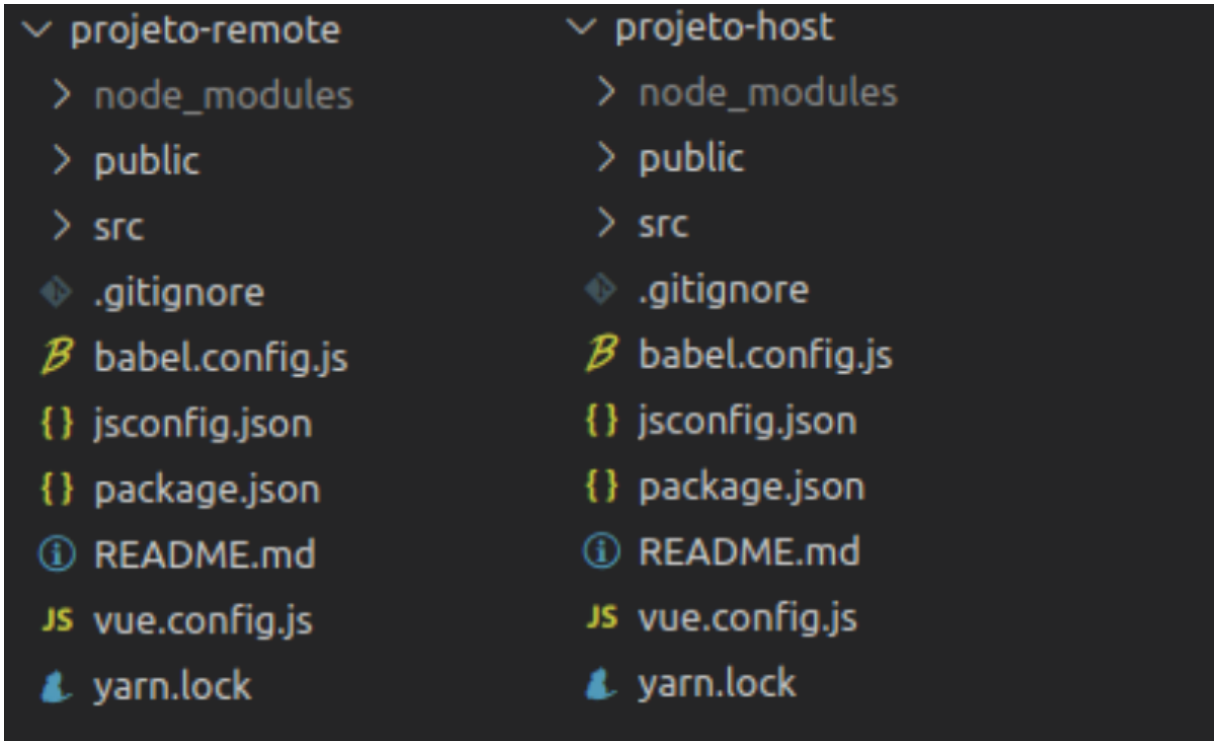


Figura 35 – Pastas dos projetos *host* e *remote*. Fonte: Autor

As configurações do webpack se encontram comumente dentro do arquivo *Webpack.config.js*, porém como é utilizada de uma arquitetura VueJS, a configuração terá lugar no arquivo *vue.config.js* dentro da propriedade *configureWebpack*, onde tudo dentro do mesmo será repassado ao webpack de maneira a se que equivale ao arquivo *webpack.config.js*.

A.0.2 Dependências

Antes de se começar a desenvolver é necessário instalar e atualizar algumas dependências do projeto.

webpack-cli: Primeiramente é indicado instalar o *webpack-cli* global, pois será usada a configuração customizada para utilização do MFP. Será possível instalá-lo localmente em seu projeto, porém indica-se a instalação global umas vez que se tratando de Micro Frontends é comum o uso de vários projetos, desta maneira a dependência se torna disponível a todos.

webpack: Esta é a dependência mais importante pois é quem disponibiliza a funcionalidade do MF, deve se instalar no mínimo a sua primeira *release* da versão 5, 5.0.0, até a data deste trabalho a versão mais atual se encontra *5.58.2*.

@vue/cli-service: Para que o VueJS consiga se comunicar e usufruir dos benefícios do webpack na versão 5, é necessário que o serviço VueJS que se encarrega de geração e *build* e execução de código esteja no mínimo em sua versão, até a data deste trabalho a versão mais atualizada é a *5.0.0-beta.6*. Com isso, se existir alguma outra dependência de serviços VueJS em seu projeto, prefixo **@vue/** é necessário que a mesma também seja atualizada para a versão mais recente e posterior a 5.

A.0.3 Configuração

Primeiramente é necessário importar webpack e seu plugin MFP em seu arquivo *vue.config.json*. Bem como definiremos a propriedade *publicPath*, e a configuração do webpack em *configureWebpack*. Adicionaremos o *plugins* **ModuleFederationPlugin** a propriedade *plugins*, como mostrado no código abaixo:

```
1   const webpack = require("webpack");
2   const ModuleFederationPlugin = webpack.container.
       ModuleFederationPlugin;
3   module.exports = {
4     publicPath: 'http://localhost:3000',
5     configureWebpack: {
6       plugins: [
7       new ModuleFederationPlugin({
8         name: "nomeDoModule",
9         filename: "remoteEntry.js",
10        exposes:{
11          "./Componente": "./caminho/Componente.vue",
12        },
13        remotes:{
14          remote1: "remote1@http://localhost:8081/remoteEntry.js"
15        },
16        shared: {
```

```
17            vue: {
18              singleton: true,
19              eager: true
20            },
21            d3: {
22              requiredVersion:"3.0.0",
23            }
24          },
25        }),
26        ],
27      }
28    };
```

Agora será explicado cada propriedade da configuração:

- **publicPath**: é o caminho sendo executado e acessível, o webpack levará em consideração na hora de importar o arquivo com o conteúdo compartilhado.
- **configureWebpack.plugins**: São onde estarão disponíveis os plugins de webpack utilizados.
- **ModuleFederationPlugin**: Instância do *plugins* MFP.
- **ModuleFederationPlugin.name**: Cada módulo, *host* ou remoto, deve ser nomeado, este nome é muito importante pois é tratado como um escopo do projeto. Em caso de módulos remote, este nome é utilizado no *host* e deve ser idêntico ao definido aqui.
- **ModuleFederationPlugin.filename**: O MFP gera um arquivo que dentro de sua estrutura é chamado manifesto JavaScript, especializado para os módulos remotos exportados e quaisquer dependências compartilhadas (**??**).
- **ModuleFederationPlugin.exposes**: Esta propriedade é definida nos *remotes* para identificar quais arquivos de sua base de código estão sendo compartilhados, podem ser arquivos, componentes, páginas, como da preferência do desenvolvedor.
- **ModuleFederationPlugin.remotes**: Esta propriedade é definida no *host* para identificar quais *remotes* estão sendo utilizados, importados, nesta propriedade são utilizadas as propriedades *publicPath*, *name* e *filename* definidos na configuração do remote.
- **ModuleFederationPlugin.shared**: Aqui são definidas as dependências compartilhadas entre *host* e *remotes* e bem como também são adicionadas as versões de cada, caso uma dependências não tenha versão definida ou com a propriedade **singleton**

com valor **true**, isso indica que *remote* e *host* utilizam de dependência, caso esta esteja definida, indica que será isolada, para que não cause conflito entre *host* e *remote*. A propriedade **eager** informa ao MFP que essa dependências é obrigatória para o carregamento de *host* ou *remote*, assim o webpack se encarrega de obrigatoriamente carregar esta dependência antes de iniciar o carregamento do projeto. Por último, a propriedade **requiredVersion** define a versão a ser utilizada para uma dependência, em caso de *remotes* compartilharem a biblioteca em versões diferentes.

Agora nas figuras **??** e **??** é possível ver como ficaram as configurações dos projetos *host* e *remote*

projeto-host > JS vue.config.js > <unknown> > configureWebpack > plugins > remotes > projetoRemote

```
const webpack = require("webpack");
const ModuleFederationPlugin = webpack.container.ModuleFederationPlugin;
module.exports = {
 publicPath: 'http://localhost:4000',
 configureWebpack: {
  plugins: [
   new ModuleFederationPlugin({
    name: "projetoHost",
    filename: "remoteEntry.js",
    remotes:{
     projetoRemote: "projetoRemote@http://localhost:3000/remoteEntry.js"
    },
    shared: {
     vue: {
      singleton: true,
     },
    },
   }),''
  ],
 }
};
```

Figura 36 – Configuração webpack do projeto *host*. Fonte: Autor

Além da configuração do webpack existe apenas uma adaptação de código e que possui praticamente zero esforço. Por se tratar de uma funcionalidade que se baseia em importes dinâmicos é necessário que o arquivo inicial, conhecido na área como *entry point* esteja utilizando o chamado *dynamic import*, ou seja, que o ponto de entrada do projeto seja um arquivo carregando todo o demais projeto utilizando a abordagem de *import()*. Para alcançar esse objetivo dentro das pastas *src* dos projetos, figura **??**, tem os arquivos **main.js**, estes arquivos devem ser renomeados para **bootstrap.js** (**??**), e crie um novo arquivo chamado **main.js** contendo apenas a linha abaixo, em ambos:

```
1    import("./bootstrap");
```

Apenas essas modificações já tornam necessário o compartilhamento de código dinâmico e em tempo de execução entre os *remotes* e seu *host*, fazendo com que a

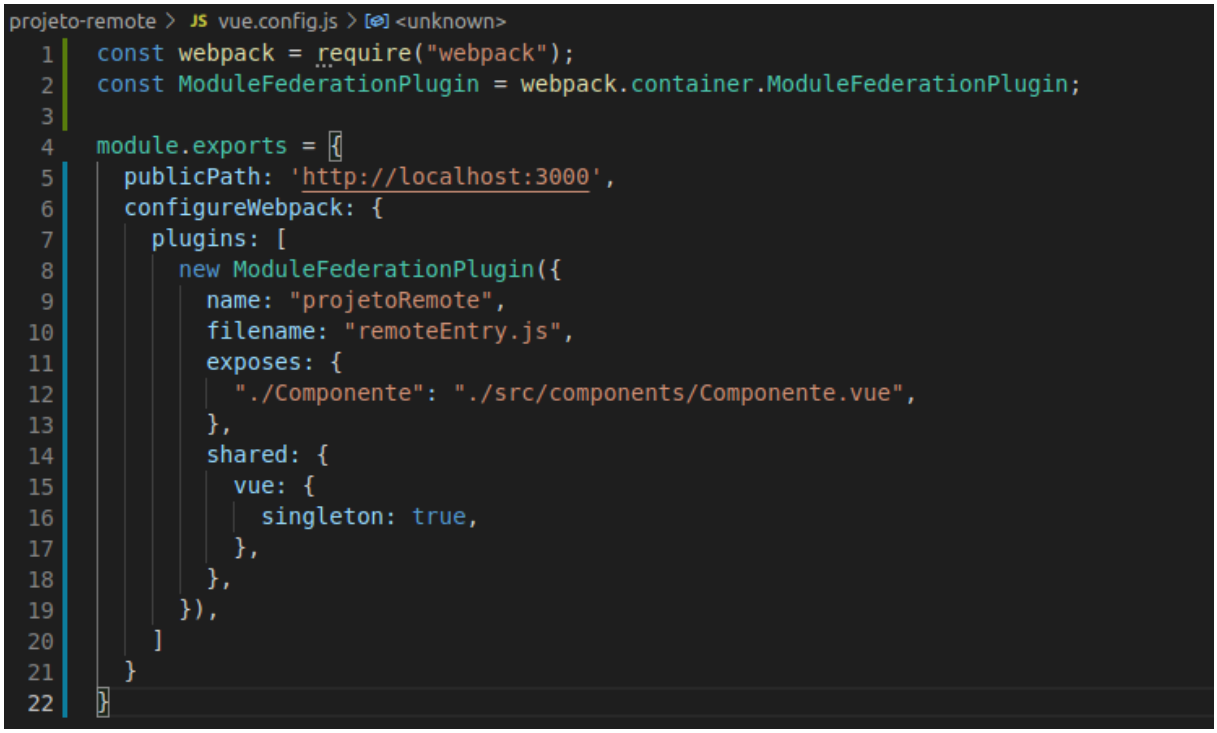


Figura 37 – Configuração webpack do projeto *remote*. Fonte: Autor

abordagem de Micro Frontends aconteça de maneira indolor a quem desenvolve e utiliza dessa abordagem.

A.0.4 Funcionamento

O **ModuleFederationPlugin** (MFP) é nada mais nada menos do que um *atalho* para o uso de outros três *plugins* inter-relacionados: **ContainerPlugin**, **ContainerReferencePlugin**, **SharePlugin** (**??**). O MFP se transforma em um ponto de configuração simplificada e unificada, onde o mesmo se encarrega de dividir as informações e repassar para os outros, que podem inclusive se for da vontade do desenvolvedor utilizá-los manualmente sem o auxilio do MFP.

- **ContainerPlugin**: Este *plugins* gerencia a exportação dos módulos remotos definidos na propriedade de *exposes* da configuração. Ele também cria o arquivo de entrada remota (remoteEntry.js), que é chamado de ?escopo? internamente. Se seu módulo é apenas um remote, este é o único *plugins* de que você precisa (**??**).
- **ContainerReferencePlugin**: Este é o *plugins* utilizado pelo *host* para gerenciar os *remotes* em sua configuração (**??**).
- **SharePlugin**: Este *plugins* gerencia a parte compartilhada da configuração. Ele lida com todos os requisitos de versão dos pacotes compartilhados. Compartilhado também é conhecido como ?substituições? tanto interna quanto externamente. Isso é usado tanto pelo *remote* quanto pelo *host* (**??**).

Podemos dizer que de maneira simplificada MFP fornece uma maneira do webpack identificar se o que esta sendo utilizado por um *host* deve ser carregado de uma dependência interna, de algum arquivo do projeto ou se deve ser realizada uma requisição HTTP para o link do *remote* que foi adicionado na configuração, para incorporar o código. É como se o arquivo que esta de forma distribuída em um outro domínio, fizesse parte da pasta original acessada no *host*,

A.0.5 Compilação

(**??**) cita que a melhor maneira de um entendimento mais profundo sobre como o webpack realiza o *build* que compila dos artefatos, para isso deve executar seu comando com o parâmetro *–mode development*, como pode ser visto na Figura **??**, são gerados alguns arquivos, mas para nós o importantes são:

- 1º *dist/remoteEntry.js*,
- 2º *dist/js/src_components_Componente_vue.js*,
- 3º *dist/js/node_modules_vue_dist_vue_runtime_esm_js.js*;

O arquivo **remoteEntry**, por mais que seja o ponto de partida ele apenas possui uma camada de abstração de que especifica e informa ao *host* o que o *remote* possui disponível, é este que traduz ao *host* que quando for solicitada a importação do **Componente.vue**, devem ser realizadas requisições HTTP para os arquivos que de fato possuem esse componente. Assim ao criar para instância será invocado o arquivo *dist/js/src_components_Componente_vue.js* e o navegador faz avaliação desse arquivo, incorporando ao código já existente. Caso o*remote* possua alguma dependência que não deve ser compartilhada ou que não existe no *host*, o arquivo *dist/js/-node_modules_vue_dist_vue_runtime_esm_js.js* também será solicitado para que injete no *host* tudo que é necessário para o **Componente.vue** funcionar corretamente.

```
 DONE  Compiled successfully in 877ms

  File                                          Size                Gzipped

  dist/js/node_modules_vue_dist_vue_runtime_esm_js.js    230.50 KiB          62.50 KiB
  dist/js/app.js                                78.82 KiB           9.74 KiB
  dist/js/src_components_Componente_vue.js      43.46 KiB           6.87 KiB
  dist/remoteEntry.js                           17.39 KiB           4.54 KiB
  dist/js/chunk-vendors.js                      14.71 KiB           4.39 KiB

  Images and other types of assets omitted.

 DONE  Build complete. The dist directory is ready to be deployed.
 INFO  Check out deployment instructions at https://cli.vuejs.org/guide/deployment.html

Done in 4.16s.
➜  projeto-remote git:(master) ✗
```

Figura 38 – *Build* do projeto *remote*. Fonte: Autor

É importante entender que esses arquivos são uma fusão dos pacotes JavaScript necessários para executar o próprio aplicativo, bem como os pacotes JavaScript necessários para os módulos remotos. Muitas vezes, esses dois se sobrepõem. Por exemplo, os mesmos pacotes configuráveis do fornecedor referenciados pelos módulos remotos são usados pelo próprio aplicativo.

A.0.6 Execução

Quando o **remoteEntry.js** é carregado pelo navegador, ele registra uma variável global com o nome especificado na propriedade *name* na configuração do MFP. Esta variável contém duas coisas, uma função *get* que retorna qualquer um dos módulos remotos e uma função *override* que gerencia todos os pacotes compartilhados.

Por exemplo, você tem um módulo com o nome **remoto1** que expõe um módulo de tempo de execução denominado **Componente**. Depois de carregar o código, você seria capaz de abrir o console e inspecionar **window.remoto1** e ver que ele tem duas funções, *get* e *override*.

Com a função *get*, pode-se obter o **Componente**, assim:

```
1    window.remoto1.get('Componente')
```

Isso retorna uma promessa, que quando resolvida dá a você uma *factory*. Pode-se invocar isso assim:

```
1    window.remoto1.get('Componente').then(factory => console.log(
        factory()));
```

Isso exibirá por meio de **console.log** o módulo conforme definido na implementação do **Componente**. Se **Componente** exportar o padrão, chamado de *default*, o valor retornado de *factory()* será um objeto com uma propriedade chamada *default*. Se o módulo tiver outras exportações nomeadas, elas também serão anexadas ao objeto.

Um efeito colateral de executar a *factory* é de também carregar quaisquer pacotes compartilhados exigidos pelo módulo remoto. O webpack lida bem com carregamento apenas de pacotes compartilhados necessários para dar suporte a cada módulo. Por exemplo, se o componente A importa Vue e Vuetify, mas o componente B apenas importa Vue, ao executar a *factory* em B traria apenas Vue. Isso significa que o MFP consegue criar escopos compartilhados e isolados. Por utilizar de uma ferramenta de compilação, webpack, ao ser gerado o artefato final de um módulo, ao informar a versão de uma determinada dependência, a ferramenta consegue:

- **a)** Criar um escopo isolado daquela dependência dentro do código compilado e do escopo global de uma página;

- **b)** Utilizar o escopo isolado de algum outro módulo, caso os dois possuam a mesma versão de dependências;

Com a criação de um escopo isolado, se permite que diversos módulos coexistam sem que suas bibliotecas em versões diferentes causem impacto nos demais, e também garante uma melhor performance e desempenho de código, pois quando existirem mesmas versões de dependências, o primeiro módulo a ser carregado traz esta dependências, e os demais não possuem a necessidade de realizar o carregamento da mesma, causando uma economia de requisição e processamento.

Por exemplo, se temos um *host* que está acessível pela URL ***www.mfe-host.com.br***, e um *remote* na URL ***www.mfe-remote.com.br***. No momento que o *host* acessar o arquivo de definição remoteEntry do *remote*, a requisição HTTP será realizada para ***www.mfe-remote.com.br/remoteEntry.js***, este arquivo informara ao *host* quais os componentes/arquivos que caso sejam importados, devem ser buscados lá no link do *remote*. Usando como exemplo as Figuras **??** e **??**, quando o desenvolvedor utilizar a importação,

```
1    import Componente from "projetoRemote/Componente"
```

em vez de carregar o arquivo de componente ***js/src_components_Componente_vue.js*** da URL ***www.mfe-remote.com.br***, o webpack utilizará o nome de arquivo que foi retornado pelo ***www.mfe-remote.com.br/remoteEntry.js***, realizando então requisição HTTP para a URL ***www.mfe-remote.com.br/js/src_components_Componente_vue.js***, e fara a avaliação desse código, o incorporando ao código rodando no navegador. De maneira muito simplificada, podemos dizer que o MFP faz com que o webpack acredito que ainda esta carregando os arquivos sempre de um mesmo lugar, de uma mesma pasta, mas que por debaixo dos panos, ele esta carregando os arquivos de outros lugares online, dos *remotes*.

Além disso, o MF é inteligente o suficiente para resolver pacotes compartilhados entre remotos. Por exemplo, um aplicativo *host* chamado **home** consome dois remotes: **nav** e **search**. Tanto a navegação quanto a pesquisa exigem Vuetify, mas a **home** não. Qualquer que seja o remoto carregado primeiro, carregará sua versão do Vuetify e, se o outro módulo remoto for carregado, ele usará o Vuetify já carregado, desde que seus parâmetros de versão sejam satisfeitos. Importante de lembrar: Importações circulares e aninhamento de *remotes* são suportados. Por exemplo, o App A importa um módulo do App B, o App B importa um módulo exportado do App C, que importa um módulo de exposição do App A. Isso é ilustrado na figura **??**. O webpack resolverá essas importações circulares de maneira correta e eficiente.

Figura 39 – Importação circular. Fonte: (**??**)

APÊNDICE B – Atualizações e Adaptações dos projetos

B.1 plugin-sensitivity

Neste módulo foram feitas atualizações e remoção de dependências, como:

B.1.1 Dependências

Removidas

Foram removidas por desuso as bibliotecas chart.js, vue-chartjs, @fortawesome/fontawesome-free e material-design-icons-iconfont **Atualizadas**

- axios **de** 0.19.2 **para** 0.27.0
- dexie **de** 3.0.3 **para** 3.2.2
- vuetify **de** 2.2.15 **para** 2.6.0

B.2 plugin-optimize

Desenvolvido por (**??**) em seu trabalho, foram feitas adaptações devido a atualizações de dependências. Nas sessões abaixo será explicado como ocorreu.

B.2.1 Dependências

Removidas

Foram removidas por desuso as bibliotecas vue-mathjs, circular-dependency-plugin.

Atualizadas

- mathjs **de** 8.0.1 **para** 10.4.0
- vue **de** 2.6.11 **para** 2.6.14
- vuetify **de** 2.4.0 **para** 2.6.0
- @vue/test-utils **de** 1.0.3 **para** 1.3.0

B.2.2 Adaptação

Com a atualização da biblioteca de testes *@vue/test-utils* foi necessário criar o arquivo de configuração **jest.config.js** contendo o *preset @vue/cli-plugin-unit-jest* para a configuração.

Nos arquivos de testes unitário do diretório *test/unit/components* foram removidas a precondição **context** ficando apenas a chamada direta do teste com o método **it**. Nos arquivos de testes unitário do diretório *tests/e2e* foram alteradas algumas classes utilizadas para se obter os elementos da tela, por conta da atualização da dependência Vuetify.

B.3 plugin-flow

B.3.1 Dependências

Foram atualizadas as dependências:

- vue **de** 2.6.10 **para** 2.6.14
- vuetify **de** 1.5.18 **para** 2.6.0
- vue-template-compiler **de** 2.6.10 **para** 2.6.14

B.3.2 Adaptação

Com a atualização da biblioteca *Vuetify* se fez necessário criar o arquivo *src/plugins/vuetify.js* para configuração da biblioteca, também foram encontrados componentes foram renomeados, a baixo os componentes que atualizamos:

- **de** v-list-tile **para** v-list-item
- **de** v-list-tile-action **para** v-list-item-action
- **de** v-list-tile-content **para** v-list-item-content
- **de** v-list-tile-title **para** v-list-item-title

No arquivo TreeModal.vue estava faltando a declaração da variável *modalModel* na propriedade *data* do componente. Foi então criada a variável *showModal* que faz mais jus a utilização da variável: dizer se o modal esta ou não sendo exibido.

APÊNDICE C – Criação do projeto plugin-gantt

Foi criado utilizando modelo de projeto da Embrapa Foram utilizadas as dependências:

- d3, versão 3.0.0
- d3-selection, versão 3.0.0
- d3-tip, versão 0.9.1
- vue, versão 2.6.11

APÊNDICE D – Dependências que utilizam o webpack

As seguintes dependências foram atualizadas para que o Vue.js utilizasse do webpack na versão 5, sem que isso acarretasse em qualquer problema de compilação de seus componentes:

- @vue/cli-plugin-eslint: versão 5.0.0
- @vue/cli-plugin-pwa: versão 5.0.0
- @vue/cli-plugin-router: versão 5.0.0
- @vue/cli-service: versão 5.0.0

D.1 Compoentente CardSimulationMenu

```
<template>
  <div>
    <v-menu bottom left v-if="menus.length">
      <template v-slot:activator="{ on, attrs }">
        <v-btn icon v-bind="attrs" v-on="on">
          <v-icon>mdi-dots-vertical</v-icon>
        </v-btn>
      </template>
      <v-list>
        <v-list-item
          :key="menu.name"
          v-for="menu in menus"
          @click="select(menu)"
        >
          <v-list-item-title>{{ menu.name }}</v-list-item-title>
        </v-list-item>
      </v-list>
    </v-menu>
    <v-dialog fullscreen v-model="showModuleDialog">
      <v-card class="flexcard">
        <v-toolbar dark>
          <v-toolbar-title v-if="moduleToShow">{{
            moduleToShow.name
          }}</v-toolbar-title>
          <v-spacer></v-spacer>
          <v-btn icon dark @click="close">
            <v-icon>mdi-close</v-icon>
          </v-btn>
        </v-toolbar>
        <component v-if="moduleToShow" :is="moduleToShow.component" :simulation="
            simulation" v-on:close="close" @error="error"/>
      </v-card>
```

```
32      </v-dialog>
33
34      <message-wrapper ref="message" />
35    </div>
36  </template>
37
38  <script>
39  import Optimization from 'pluginOptimize/Optimization'
40  import Cronograma from 'pluginGantt/GanttChart'
41  import Sensibilidade from 'pluginSensitivity/SensitivityAnalysis'
42  import MessageWrapper from '@/components/MessageSnack.vue'
43
44  export default {
45    name: 'SimulationCardMenu',
46    components: {
47      MessageWrapper
48    },
49    props: {
50      simulation: {
51        required: true
52      }
53    },
54    data () {
55      return {
56        menus: [{
57          name: 'Cronograma',
58          component: Cronograma
59        },{
60          name: 'Analise de Sensibilidade',
61          component: Sensibilidade
62        },{
63          name: 'Otimizador',
64          component: Optimization
65        }
66      ],
67        showModuleDialog: false,
68        moduleToShow: null,
69      }
70    },
71    methods: {
72      select (menuItem) {
73        this.$emit('select', { item: menuItem, simulation: this.simulation })
74        this.moduleToShow = menuItem
75        this.showModuleDialog = true
76      },
77      close () {
78        this.showModuleDialog = false
79        this.moduleToShow = null
80      },
81      error (message) {
82        this.close()
83        this.$refs.message.open(message, 'error')
84      }
85    }
86  }
87  </script>
88  <style>
89  .flexcard {
90    display: flex;
91    flex-direction: column;
```

```
92      height: 100%;
93    }
94    .flexcard .v-toolbar {
95      flex: 0;
96    }
97    </style>
```

APÊNDICE E – Helper de carregamento dinâmico de módulos

Arquivo principal dynamicLoad.js:

```
1  /* eslint-disable */
2  import { getModule } from "./getModule";
3
4  const loadModule = async (props) => {
5    const { remote, url, module } = props;
6
7    if (!remote || !url || !module) {
8      throw new Error(
9        "ModuleFederationEnhancedPlugin Helper: `remote`, `url` and `module` are
             required to load a module."
10     );
11   }
12   try {
13     const Module = getModule({ remote, module, url });
14     return Module;
15   } catch (e) {
16     return null;
17   }
18 };
19
20 const loadRemotes = async (hostName) => {
21   const remoteUrlMapFactory = await window[hostName].get("./remoteUrlMap");
22   const remoteMap = remoteUrlMapFactory();
23   const remotePromises = [];
24   Object.keys(remoteMap).forEach(async (key) => {
25     remotePromises.push(
26       new Promise(async (resolve) => {
27         const moduleMap = await loadModule({
28           remote: key,
29           url: remoteMap[key],
30           module: "./moduleNameList",
31         });
32         resolve({
33           remote: key,
34           url: remoteMap[key],
35           modules: moduleMap || [],
36         });
37       })
38     );
39   });
40
41   const remotes = await Promise.all(remotePromises);
42   return remotes;
43 };
44
45 const loadModules = async (remotes) => {
46   const modulesLoaded = [];
47   const modulesObject = {};
48   remotes.forEach((remote) => {
```

```
49      remote.modules.forEach(async (module) => {
50        modulesLoaded.push(
51          new Promise(async (resolve) => {
52            const moduleLoaded = await loadModule({
53              remote: remote.remote,
54              url: remote.url,
55              module: module,
56            });
57            if (moduleLoaded) {
58              const moduleName = module.replace("./", "");
59              modulesObject[moduleName] = {
60                remote: remote.remote,
61                moduleName: module.replace("./", ""),
62                module: moduleLoaded,
63              };
64            }
65            resolve();
66          })
67        );
68      });
69    });
70
71    await Promise.all(modulesLoaded);
72    return modulesObject;
73  };
74
75  export async function loadRemoteAndModules(hostName) {
76    const remotes = await loadRemotes(hostName);
77    const modules = await loadModules(remotes);
78    return modules;
79  }
```

Arquivo getModule.js:

```
1  /* eslint-disable */
2
3  import { injectScript } from "@module-federation/utilities";
4
5  const isObject = (module) =>
6    module instanceof Object && !(module instanceof Array);
7
8  const isExportDefault = (module) =>
9    Object.keys(module)?.length === 1 && Object.keys(module)[0] === "default";
10
11 export const getModule = async ({ remote, module, url }) => {
12   if (!remote || !module) {
13     throw new Error(
14       "ModuleFederationEnhancedPlugin Helper: `remote` and `module` are required to
           load a module."
15     );
16   }
17   try {
18     const container =
19       window[remote] ||
20       (await injectScript({
21         global: remote,
22         url: url,
23       }));
24
25     const factory = await container.get(module);
```

```
26        const Module = factory();
27        if (isObject(Module)) {
28          return isExportDefault(Module) ? Module.default : Module;
29        }
30
31        return Module;
32      } catch (e) {
33        return null;
34      }
35    };
```

Anexos

ANEXO A – Manual de integração em novos Módulos

1. Instale o plugin ModuleFederationEnhancedPlugin:

```
npm i @schirrel/module-federation-enhanced-plugin
```

2. No arquivo ***vue.config.js***: 2.1 Importe o plugin:

```
const ModuleFederationEnhacedPlugin = require("@schirrel/module-federation-enhanced-plugin");
```

2.2 Na configuração altere a propriedade *publicPath* para utilizar **process.env.VUE_APP_URL**
2.3 Na configuração é necessário que existe a propriedade *configureWebpack*, caso ela não exista, a adicione como um objeto vazio, deve ficar por exemplo:

```
module.exports = defineConfig({
  publicPath: process.env.VUE_APP_URL,
  configureWebpack: {
  }
})
```

2.4 Na propriedade *configureWebpack* adicione o ModuleFederationEnhacedPlugin na propriedade *plugins*, caso a mesma não exista, crie como um *array* vazio, **plugins:** []. A configuração do ficará como:

```
plugins: [
 new ModuleFederationEnhacedPlugin({
  name: 'nomeDoRemoteOuHost',
  filename: 'remoteEntry.js',
  exposes: {},
  remotes: {},
  shared: {
   vue: {
    eager: true,
    singleton: true,
    requiredVersion: dependencies.vue
   },
   vuetify: {
    eager: true,
    singleton: true,
    requiredVersion: dependencies.vuetify
   },
  },
 }),
]
```

altere a propriedade *name* de acordo com o nome do projeto. Caso o seu projeto compartilhe outras dependências, as adicione a propriedade *shared*. Utilize a propriedades *exposes* caso seu projeto exporte algum componente ou arquivo, e utilize a propriedade *remotes* caso seu projeto importe componentes ou arquivos de outros projetos.

ANEXO B – DECLARAÇÃO DE AUTORIA E RESPONSABILIDADE

Alan Balen Schio, residente e domiciliado na cidade de Coxim, Estado do Mato Grosso do Sul, portador do RG de nº 32903944 e CPF nº 037.764.351-35, declaro que a Dissertação apresentada, com o título "Integração de Módulos Utilizando Micro Frontends na Plataforma +Precoce" é de minha autoria e assumo a total responsabilidade pelo seu conteúdo e pela originalidade do texto. Declaro que identifiquei e referenciei todas as fontes e informações gerais que foram utilizadas para construção do presente texto. Declaro também que este artigo não foi publicado, em parte, na íntegra ou conteúdo similar em outros meios de comunicação, tendo sido enviado com exclusividade para Faculdade de Computação(FACOM) da Universidade Federal de Mato Grosso do Sul (UFMS).

Campo Grande, 13 de Março de 2023.

Alan Balen Schio
RA 202100799