Aplicabilidade da arquitetura de Micro-Frontends - Estudo de caso de uma aplicação de venda de ingressos

Fillipe Gomes[1], Ivairton Monteiro Santos[1]

[1]Instituto de Ciências Exatas e da Terra – Campus Universitário do Araguaia – UFMT – Barra do Garças/MT

fillipe484@gmail.com, ivairton@ufmt.br

Abstract. *This work aims to evaluate the applicability of the micro-frontends architecture together with the development of progressive web applications, using a ticket sales platform as an example. The architecture promotes code decoupling by dividing the application into smaller parts. A prototype was implemented to demonstrate the main features of this approach. From this, its advantages and challenges were analyzed, offering an introduction to the topic to assist in decision-making regarding software architecture, serving as a reference for the development of future projects.*

Resumo. *Este trabalho busca avaliar a aplicabilidade da arquitetura de micro-frontends em conjunto com a construção de aplicações web progressivas, utilizando como exemplo uma plataforma de venda de ingressos. A arquitetura promove o desacoplamento do código, dividindo a aplicação em partes menores. Um protótipo foi implementado para demonstrar os principais recursos dessa abordagem. A partir disso, foram analisadas suas vantagens e desafios, oferecendo uma introdução ao tema para auxiliar na tomada de decisões sobre arquitetura de software, servindo de referência para o desenvolvimento de futuros projetos.*

1. Introdução

O crescimento constante das empresas e equipes de desenvolvimento de software no cenário mundial vem transformando cada vez mais a forma como as aplicações são desenvolvidas. As aplicações frontend da forma como conhecemos estão passando por grandes mudanças e dando espaço a novas abordagens, como a de arquitetura de frontends distribuídos. Essa abordagem pode vir a trazer grandes benefícios de gerenciamento de times de desenvolvedores e de código, levando a forma como aplicações frontend são feitas para um novo patamar.

A arquitetura de micro-frontends visa trazer maior flexibilidade, transparência e separação de responsabilidades, ao mesmo tempo em que traz maior flexibilidade para que múltiplos times possam trabalhar em paralelo e ainda compartilhando recursos da aplicação [AMAZON,2023; GEERS, 2022]. Nesse contexto, deve se ter em mente que novas possibilidades também trazem dúvidas e incertezas.

Devido ao tema micro-frontends ser relativamente recente, tendo surgido em meados de 2016, o conteúdo a respeito do tema ainda se encontra escasso ou disperso na Internet. Nesse sentido, este trabalho busca levantar e apresentar informações a respeito do padrão de arquitetura de micro-frontends, explorando vários de seus cenários e

aplicabilidades, demonstrando também pontos de preocupação ou dificuldades envolvidos no processo de implementação de tal arquitetura. A partir disso, busca-se disponibilizar um cenário real de uso, como guia para futuros desenvolvimentos que se beneficiem de tal arquitetura.

2. Metodologia

Com o objetivo de investigar a viabilidade e os casos de uso da arquitetura de micro-frontends, este trabalho optou por analisar cenários de aplicações que pudessem aproveitar dos recursos da arquitetura de micro-frontends. A partir da investigação foi escolhida a implementação de um protótipo como prova de conceito de uma plataforma de venda de ingressos para eventos. A intenção é explorar e demonstrar a ampla gama de recursos oferecidos por essa arquitetura. O desenvolvimento do projeto foi dividido em etapas, para contribuir para o entendimento e a execução do mesmo, sendo elas:

a) *Levantamento de requisitos*: Nesta etapa foram coletados requisitos para definir um modelo de negócios que pudesse se beneficiar da arquitetura de micro-frontends.

b) *Processo de design de Experiência do Usuário (UI/UX)*: Após a definição do projeto foram aplicadas estratégias de design de experiência do usuário (UX) para criar um protótipo estruturado, utilizando de recursos e princípios de design.

c) *Modelagem da arquitetura*: Concepção e design da arquitetura para atender às necessidades específicas do projeto.

d) *Gerenciamento de falhas e observabilidade*: Esta etapa envolveu a implementação de medidas para garantir a operação do sistema, mesmo que de forma parcial em momentos de falhas e proporcionar transparência em relação a possíveis erros, além das etapas necessárias para entrega e publicação.

e) *Infraestrutura*: Esta etapa envolveu todo o processo relacionado às etapas necessárias para integração e entrega da aplicação.

2.1 Levantamento de requisitos

Durante o levantamento de requisitos, foram buscadas referências acadêmicas e analisadas aplicações existentes no mercado para identificar dificuldades e cenários onde a arquitetura de micro-frontends pudesse ser aplicada [HARRIS, 2023, KAZMAN & CLEMENTS, 2023]. Foi fundamental entender o contexto atual das principais plataformas de venda de ingressos online, escolhido como tema para demonstrar a viabilidade da arquitetura.

Neste processo foram definidos os seguintes requisitos essenciais:

- Autenticação do usuário;
- Listagem e possibilidade de busca por eventos/shows;
- Seleção do tipo de ingresso desejado;
- Realização do pagamento e finalização da compra;

2.2 Processo de design de Experiência do Usuário (UI/UX)

A partir do levantamento de requisitos e sua análise foram estabelecidas metodologias e processos de design para garantir a construção de uma experiência mais satisfatória

durante o uso da aplicação. Foram levadas em consideração boas práticas de UX, como usabilidade, tipologias, hierarquia visual, acessibilidade e navegação do usuário.

Nesta etapa foram elaborados todos os diagramas de fluxo e interações da aplicação, seguindo para elaboração dos protótipos de interface, com a diagramação, tipografia, definição do padrão de cores e planejamento dos componentes visuais reutilizáveis. A Figura 1 ilustra uma amostra do protótipo de baixa fidelidade.

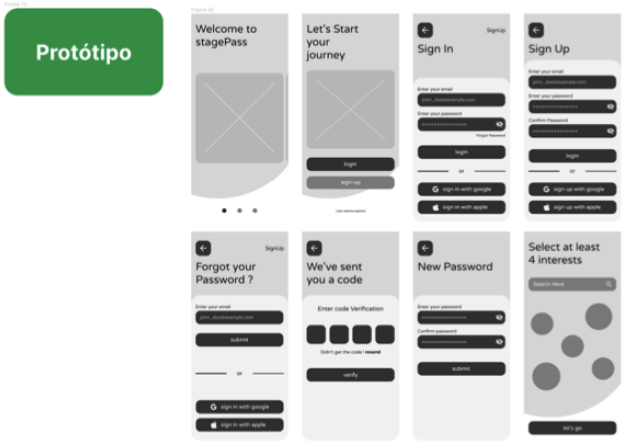

Figura 1. Exemplo de trecho dos arquivos de protótipo de baixa fidelidade.

Durante todo o processo foi utilizado de metodologias de *design thinking*[1] para iteração e desenvolvimento. Dessa forma se divide então os objetivos e hipóteses, pesquisa e análise, ideias e protótipos, testes e consequentemente por fim iterando consecutivamente, seguindo a ideia de diamante duplo[2].

2.3 Modelagem da arquitetura

A arquitetura adotada segue os padrões de design de micro-frontends [LEWIS & FOWLER, 2014; GEERS, 2022], dividindo os módulos conforme as camadas de domínio da aplicação. Essa abordagem permite o desenvolvimento separado dos micro-frontends por funcionalidades específicas, com foco em uma estruturação vertical. Isso melhora a organização e o compartilhamento de informações entre os componentes, resultando em um ambiente mais eficiente e coeso para o desenvolvimento e operação, favorecendo o trabalho em times.

Embora mantenha algum nível de acoplamento, a escolha pela arquitetura modularizada baseada em camadas de domínio oferece mais autonomia às equipes e controle sobre suas funcionalidades. O uso de um modelo híbrido entre micro-frontends horizontais e verticais, ou apenas horizontais, poderia aumentar a complexidade do sistema [GEERS, 2022; AMAZON, 2023].

A Figura 2 ilustra a arquitetura e principais módulos do sistema. Nela pode-se observar a divisão dos módulos por camada de domínio, como algumas de suas possíveis interações e divisões. As setas tracejadas informam as possíveis conexões

[1] Design thinking se refere a uma abordagem de pensamento criativo para organizar e gerar ideias. Mais Informações em: https://rockcontent.com/br/blog/design-thinking/

[2] Diamante duplo pode ser definido como um dos métodos de Design Thinking possíveis. Mais Informações em: https://www.euax.com.br/2020/12/duplo-diamante/

entre os micro-frontends. Por exemplo, o ‘core’ pode se comunicar com o design system (osiris-ui), e com o ‘util-state’. Além também de micro-frontends específicos poderem se comunicar com essas mesmas camadas, ou com outros micro-frontends. As setas bidirecionais indicam relação de obrigatoriedade (‘core’), onde para quaisquer das aplicações se comunicarem, necessitam da aplicação ‘core’, que gere todas.

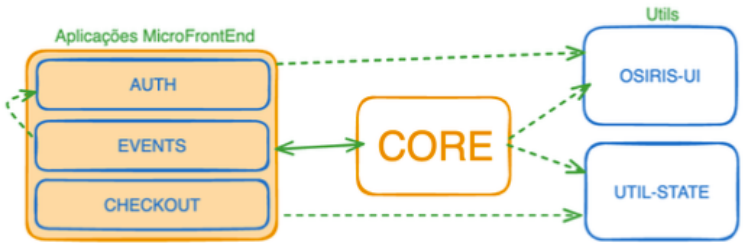

Figura 2. Disposição dos módulos da arquitetura exemplo (protótipo da plataforma de venda de ingressos).

Cada módulo da arquitetura tem as seguintes funcionalidades:

Core: Módulo central do sistema, orquestrador, responsável por gerenciar e inicializar todos os micro-frontends presentes na aplicação e também gerir funcionalidades de uma aplicação PWA.

Aplicações micro-frontend: Pode-se definir como aplicações micro-frontend toda e qualquer aplicação que fornece funcionalidades e aspectos visuais de determinada camada de domínio da aplicação. Como demonstração, foram implementados os módulos para autenticação de usuários, processo de pagamento e visualização de eventos/shows.

Utils: Abriga todas as aplicações utilitárias que contribuem para as demais aplicações existentes. Neste projeto foram definidas as aplicações utilitárias Osiris-UI (define componentes e recursos visuais reutilizáveis) e Util-State (gerenciador global para integração de dados entre módulos).

2.4 Gerenciamento de falhas e observabilidade

O gerenciamento de falhas é essencial para garantir a estabilidade do sistema, especialmente quando os módulos são desenvolvidos separadamente, como na arquitetura de micro-frontends. Cada micro-frontend deve ser capaz de lidar com suas próprias falhas sem impactar os demais módulos [LEWIS & FOWLER, 2014].

Estratégias de monitoramento são essenciais para garantir a observabilidade do sistema, incluindo o rastreamento de transações, monitoramento de desempenho, rastreamento de exceções, e geração de logs e métricas. Esses dados permitem análises mais detalhadas para assegurar a operabilidade e aprimorar a plataforma.

2.5 Infraestrutura

Todos os serviços criados foram divididos em repositórios *git* individualizados, garantindo assim uma maior separação e controle via multi-repos. O motivo principal para isso se deu ao fato de como é relevante que qualquer aplicação atual que seja gerida por múltiplos membros, seja gerida num sistema de controle de versionamento de código, assim podendo seguir estruturas de *branches*, *pull requests*, separação de ambientes de homologação, *staging* e produção.

Foi implementado um sistema de controle de *commits* para organizar os repositórios de cada módulo e garantir que todos os desenvolvedores seguissem um padrão no processo de publicação e atualização. Assim, foram criadas *pipelines* com fluxo de integração contínua para automatizar o processo de publicação de novas versões da aplicação, sem a necessidade de interferência humana no processo.

3. Resultados

A partir dos fundamentos e metodologia apresentados, foi implementado um protótipo de uma plataforma de venda de ingressos, chamada StagePass, com base nos fundamentos e metodologia da arquitetura de micro-frontends. O protótipo foi desenvolvido como uma aplicação PWA.

Para a implementação desse projeto foi utilizado o framework Single-SPA, ao invés do Module Federation (contexto mais difundido), para gerenciar os micro-frontends. Além disso, optou-se pela abordagem de desenvolvimento vertical e pela utilização da biblioteca React. Outras tecnologias acessórias também foram utilizadas.

Para desacoplar os micro-frontends presentes no sistema implementado foi adotado o padrão de carregamento dinâmico de módulos (recomendado pelo Single-SPA).

Devido ao fato do Single-SPA incorporar a biblioteca SystemJS, torna-se possível o aproveitamento da estrutura de *importmaps*. A aplicação ‘core’ ficou responsável por registrar os micro-frontends existentes no projeto e resolver dinamicamente os módulos com base em seus caminhos especificados no arquivo JSON, fornecido pelo *importmap*.

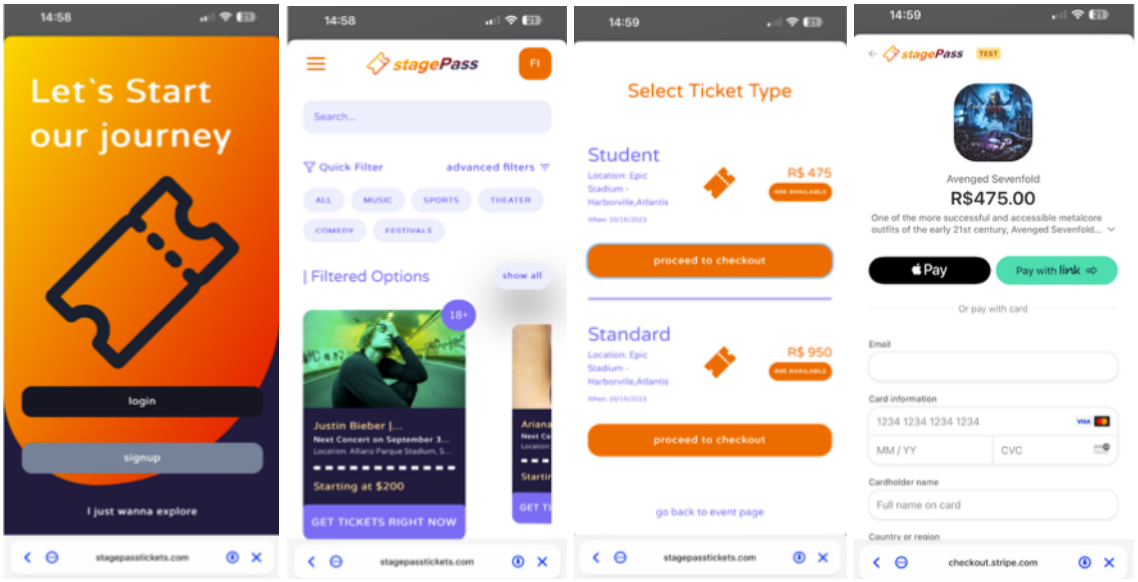

Figura 3. Exemplos de algumas das telas da aplicação desenvolvida.

Na aplicação ‘core’ encontra-se o registro de todas as aplicações micro-frontends, assim como as configurações básicas de rotas que determinam quais micro-frontends devem ser instanciados em determinados momentos. Com exceção do ‘core’, toda a estrutura da aplicação é derivada do *base-application-template*, um *template* criado para servir de referência para a criação de novos micro-frontends.

São vários os detalhes técnicos inerentes ao projeto que não serão possíveis de serem detalhados, desde o funcionamento do PWA, a implementação do *service-worker*, a implementação do repositório de componentes Osiris-UI, como também do repositório Util-State e todo o processo de compartilhamento de estados entre componentes da aplicação; além da implementação do monitoramento de log e falhas, onde foi usada a ferramenta Sentry[3] e da infraestrutura adotada, baseada nos serviços Route 53, Certificate Manager, S3, Cloudfront da AWS. A Figura 3 apresenta algumas das telas da aplicação desenvolvida.

5. Conclusões

Neste trabalho, foi desenvolvido um protótipo de uma plataforma de venda de ingressos online, utilizando a arquitetura de micro-frontends e funcionalidades de aplicações PWA. O protótipo demonstrou os principais mecanismos e interações no contexto de micro-frontends, oferecendo uma experiência de uso integrada, onde o usuário interage com a aplicação sem notar que se trata de aplicações distintas e independentes.

A divisão da arquitetura em camadas de domínio e a organização em múltiplos repositórios estabeleceram responsabilidades claras entre os módulos do sistema, promovendo um alto grau de desacoplamento. O protótipo demonstrou a comunicação entre diferentes aplicações e explorou recursos como modularização e integração com serviços de terceiros, evidenciando a viabilidade e relevância da arquitetura de micro-frontends no desenvolvimento de aplicações web complexas e escaláveis..

Para aprimorar a arquitetura, é importante implementar sistemas de testes, além de desenvolver pipelines automatizados que evitem publicações com problemas. Ferramentas de análise de código, como o SonarQube, podem ser utilizadas para identificar más práticas. Outras melhorias incluem aumentar a cobertura de componentes documentados no Storybook e refatorar os pipelines automatizados para otimizá-los, utilizando técnicas de geração de artefatos e cache.

O código fonte de todo o sistema desenvolvido está disponível em: https://github.com/orgs/stagepasstccdemo/repositories

Referencias

AMAZON. The Difference Between Monolithic and Microservices-architecture. 2023b. Disponível em: https://aws.amazon.com/pt/compare/the-difference-between-monolithic-and-microservi ces-architecture/

GEERS, Michael. What are Micro Frontends? 2022. Disponível em: https://micro-frontends.org/

HARRIS, Chandler. Microservices vs monolithic architecture (Atlassian) Disponível em: https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-m onolith.

KAZMAN, Rick; CLEMENTS, Paul. Software Architecture in Practicing. Addison Wesley, 2023. 3 p.

LEWIS, James; FOWLER, Martin. Microservices: a definition of this new architectural term. a definition of this new architectural term. 2014. Disponível em: https://martinfowler.com/articles/microservices.html

[3] Sentry se trata de uma plataforma para gerenciamento e monitoramento de erros e carga das aplicações. Mais informações em: https://sentry.io