ACCORSI AIRLINES: UM MODELO ARQUITETURAL DIRIGIDO A EVENTOS E CIENTE DE CONTEXTO PARA COMPRA DE PASSAGENS AÉREAS

Pedro Henrique Accorsi[1] Kleinner Farias[2]

Resumo:

O setor de venda de passagens aéreas é fundamental para as conexões globais, sendo crucial a eficiência e adaptabilidade desses sistemas. Eles integram uma grande quantidade de informações, desde preferências individuais dos passageiros até restrições operacionais das companhias aéreas. Com a crescente diversidade de canais de venda e demanda por experiências personalizadas, esses sistemas precisam ser ágeis, inteligentes e capazes de se ajustar dinamicamente às demandas do mercado. Contudo, há uma lacuna na literatura atual em relação a estudos e propostas de arquiteturas que abordem conjuntamente a ciência de contexto do usuário e a flexibilidade arquitetural. Isso resulta em sistemas de venda de passagens aéreas com recomendações inadequadas e arquiteturas inflexíveis, afetando sua capacidade de adaptação às demandas específicas de cada usuário e integração com tecnologias emergentes. Este trabalho, portanto, apresenta Accorsi Airlines, uma proposta de arquitetura de software orientada a eventos e sensível ao contexto para venda de passagens aéreas. A abordagem proposta foi avaliada através do desenvolvimento de um protótipo funcional, a partir do qual 34 desenvolvedores de software puderam avaliar, com um questionário e entrevistas semi estruturadas, tanto a arquitetura quanto o protótipo em si. Os resultados obtidos com 34 desenvolvedores revelam insights valiosos quanto à aceitação e utilidade da arquitetura proposta, mostrando que todos os participantes demonstraram concordância total em relação à percepção de utilidade da proposta, fornecendo perspectivas promissoras para sua aplicação prática.

Palavras-chave: Arquitetura dirigida a eventos; Microsserviços; Arquitetura; Contexto

Abstract:

The airline ticket sales sector is fundamental for global connections, and the efficiency and adaptability of these systems is crucial. They integrate a large amount of information, from individual passenger preferences to airline operational restrictions. With the growing diversity of sales channels and demand for personalized experiences, these systems need to be agile, intelligent and capable of dynamically adjusting to market demands. However, there is a gap in the current literature regarding studies and architecture proposals that jointly address user context science and architectural flexibility. This results in airline ticket sales systems with inadequate recommendations and inflexible architectures, affecting their ability to adapt to each user's specific demands and integrate with emerging technologies. This work, therefore, presents Accorsi Airlines, an event-oriented and context-sensitive software architecture proposal for selling airline tickets. The proposed approach was evaluated through the development of a functional prototype, from which 34 software developers were able to evaluate, with a questionnaire and semi-structured interviews, both the architecture and the prototype itself. The results obtained with 34 developers reveal valuable insights regarding the acceptance and usefulness of the pro-

[1]Graduando em de Ciência da Computação pela Unisinos. Email: pedrohaccorsi@gmail.com

[2]Possui doutorado em Informática pela Pontifícia Universidade Católica do Rio de Janeiro (2012), mestrado em Ciência da Computação pela Pontifícia Universidade Católica do Rio Grande do Sul (2008), graduação em Ciência da Computação pela Universidade Federal de Alagoas (2006) e em Tecnologia da Informação pelo Instituto Federal de Alagoas. Email: kleinnerfarias@unisinos.br

posed architecture, showing that all participants demonstrated complete agreement regarding the perceived usefulness of the proposal, providing promising prospects for its practical application.

Keywords: Event-driven architecture, Microservices, Architecture; Context

1 INTRODUÇÃO

O setor de venda de passagens aéreas representa não apenas a transação comercial entre usuários e companhias aéreas, mas também desempenha um papel fundamental na facilitação de conexões globais. No contexto de um mundo cada vez mais interconectado, onde a mobilidade é essencial, a eficiência e a adaptabilidade dos sistemas de venda de passagens aéreas são cruciais (JURCA; HELLMANN; MAURER, 2014). Esses sistemas funcionam como a espinha dorsal que integra e coordena uma quantidade excepcional de informações, desde as preferências individuais dos passageiros até as restrições operacionais das companhias aéreas. Além disso, a crescente diversidade de canais de venda e o aumento da demanda por uma experiência personalizada exigem sistemas que sejam ágeis, inteligentes e capazes de se ajustar dinamicamente às demandas em constante evolução do mercado (MIRAZ; ALI; EXCELL, 2021).

Porém, a literatura atual apresenta uma lacuna de estudos e propostas de arquiteturas que abordem a ciência de contexto do usuário e flexibilidade arquitetural, conjuntamente. Esta deficiência tem consequências diretas na funcionalidade dos sistemas de venda de passagens aéreas, resultando em recomendações inadequadas (SHANI; GUNAWARDANA, 2010) e arquiteturas inflexíveis (LI et al., 2022). A falta de sensibilidade ao contexto do usuário impacta diretamente na capacidade desses sistemas de se adaptarem dinamicamente às demandas específicas de cada usuário. Por outro lado, a rigidez arquitetural também emerge como um desafio, dificultando a integração desses sistemas com tecnologias emergentes e impedindo a evolução necessária para acompanhar as mudanças no ambiente operacional e nas preferências dos usuários.

Estudos recentes destacam essa problemática. (LAIGNER et al., 2020) reporta um estudo empírico no qual se constatou que a adoção da arquitetura dirigida a eventos melhorou a manutenção e o isolamento de falhas em um sistema que foi refatorado após passar anos evoluindo, dando origem a um código grande, complexo e obsoleto, exigindo um processo de manutenção caro, onde acrescentar novas funcionalidades se tornava algo impraticável. Ao passo que (STEFANIDI et al., 2021) aborda interfaces de usuário cientes de contexto, capazes de adaptar onde e quando um determinado conteúdo aparece na tela, porém é incapaz de introduzir comportamentos diferentes e únicos por usuário, isto é, ir além de apenas mostrar ou não uma informação, mas sim de entregar uma experiência diferente e personalizada.

Este artigo, portanto, apresenta a Accorsi Airlines, uma proposta de arquitetura de software orientada a eventos e sensível ao contexto, adaptada especificamente para o domínio de vendas de passagens aéreas. Essa arquitetura oferece aos desenvolvedores uma vantagem substancial ao apresentar uma estrutura genérica, facilitando sua implementação em diversas tecnologias.

Além disso, a avaliação da Accorsi Airlines envolveu o desenvolvimento de um protótipo funcional, que utiliza efetivamente o contexto do usuário para acionar eventos de negócios em conjunto com eventos técnicos comuns. Esta abordagem ressalta a adaptabilidade e a capacidade de resposta da arquitetura, demonstrando seu potencial para lidar com as complexidades inerentes ao domínio de vendas de passagens aéreas.

Adicionalmente, a avaliação da Accorsi Airlines adotou uma abordagem qualitativa, aplicando o Questionário de Aceitação de Tecnologia (TAM) e realizando entrevistas. Os resultados obtidos com 34 desenvolvedores revelam insights valiosos quanto à aceitação e utilidade da arquitetura proposta. Dos participantes, 58,2% expressaram total concordância em relação à facilidade de compreensão da arquitetura, enquanto 47,06% indicaram total concordância com a facilidade de domínio da mesma. Além disso, todos os participantes demonstraram concordância total em relação à percepção de utilidade da arquitetura proposta. Esses resultados evidenciam uma tendência favorável quanto à aceitação e utilidade da arquitetura proposta, destacando sua viabilidade e potencial impacto positivo tanto na indústria quanto na academia. A alta taxa de concordância relativa à compreensão, domínio e utilidade da arquitetura oferece perspectivas promissoras para sua aplicação prática, assim como para futuras investigações e desenvolvimentos na área de engenharia de software.

O estudo está dividido conforme a seguinte estrutura: a Seção 2 conterá o referencial teórico, com os principais conceitos para entendimento do estudo proposto; a Seção 3 abordará os trabalhos relacionados, explorando o processo de seleção utilizado e também realizando um comparativo destes com o presente; a Seção 4 descreverá a abordagem proposta para o desenvolvimento do estudo; a Seção 5 traz a avaliação juntamente com os resultados obtidos; e, por fim, a Seção 6 traça algumas conclusões e trabalhos futuros.

2 REFERENCIAL TEÓRICO

2.1 Arquitetura de Microsserviços

A ascensão da computação em nuvem, com seus inúmeros benefícios, levou muitas empresas a se realocarem para a nuvem para acompanhar a expansão do mercado. Esse movimento massivo destacou uma grande questão que já havia sido observada anteriormente, a questão da elasticidade em aplicações monolíticas. Uma arquitetura monolítica se refere a aplicações que têm todo o seu código e funcionalidades interligadas (GOEL; NAYAK, 2019). Quando pequena, essa arquitetura possui diversas vantagens em termos de desenvolvimento, entrega e escalabilidade. Contudo, à medida que cresce em tamanho e complexidade, esses benefícios podem se transformar em obstáculos.

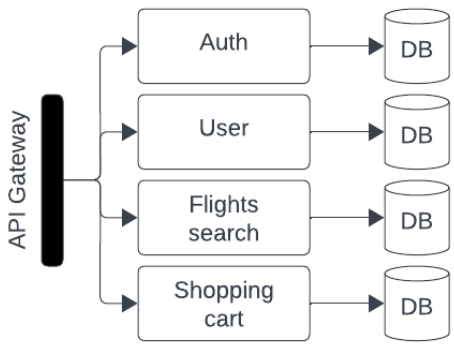
A computação em nuvem opera sob demanda, o que significa que, ao escalar uma grande aplicação monolítica, recursos serão alocados a todas as suas camadas, gerando custos e complexidades desnecessárias, uma vez que apenas uma pequena parte dela deveria receber mais

recursos. Para minimizar esse desperdício, o conceito de aplicações nativas de nuvem foi criado. Essas aplicações devem ser construídas usando containers, onde cada bloco de código é isolado um do outro e geralmente representado por uma máquina virtual. Além disso, devem seguir uma arquitetura baseada em microsserviços (AROUK; NIKAEIN, 2020).

A arquitetura de microsserviços nasceu com a intenção de descrever projetos de software como conjuntos de serviços com implementações distintas e independentes (LEWIS; FOWLER, 2020), com controle descentralizado. Uma aplicação que implementa tal arquitetura deve ser dividida em pequenos serviços, cada um com sua funcionalidade, seguindo o princípio da responsabilidade única. Essa arquitetura possui um limite muito claro e é geralmente projetada para que as comunicações entre os microsserviços sejam feitas por meio de API ou mensageria. Dessa maneira, os vários microsserviços atuam juntos para formar uma aplicação completa, como ilustrado na Figura 1.

Cada estilo arquitetônico tem suas vantagens e desvantagens que devem ser consideradas de acordo com o problema a ser resolvido. Isso também se aplica aos microsserviços, que, embora tenham conceitos e ideias muito valiosas, às vezes podem ser uma escolha inferior a uma arquitetura monolítica. No entanto, no caso da computação em nuvem, há uma forte recomendação para o uso de microsserviços (AROUK; NIKAEIN, 2020), pois eles oferecem grandes benefícios, especialmente em relação à manutenção e problemas de desempenho. Como cada microsserviço tem sua funcionalidade, se um problema surgir, sabe-se exatamente onde a manutenção é necessária. Além disso, a escalabilidade é outra grande vantagem de uma arquitetura baseada em microsserviços. Em contraste com as aplicações monolíticas, onde toda a aplicação é escalada quando mais recursos são necessários, em um microsserviço, os recursos são alocados apenas onde realmente são necessários, evitando a criação de recursos ociosos.

Figura 1 – Arquitetura de microsserviços



Fonte: Elaborado pelo autor

2.2 Arquitetura REST

A arquitetura REST trata-se de um conjunto de princípios e definições necessários para a criação de um projeto com interfaces bem definidas (TOTVS, 2020). Em resumo, define-se que uma aplicação (também compreendido como um microsserviço) vai se comunicar com outra aplicação através do protocolo HTTP ou HTTPS, transferindo dados de uma para outra. Nesta arquitetura, a aplicação que realiza a chamada obrigatoriamente possui conhecimento da aplicação que vai receber a chamada, pois é uma conversa síncrona "direta". Isso significa que o fluxo de dados é bem estabelecido e o acoplamento entre as aplicações é direto (aplicação A conhece a aplicação B), mesmo sendo fraco (aplicação A não sabe como funciona aplicação B).

Esse modelo é amplamente utilizado no estilo requisição/resposta síncrona, embora permita uma conversa assíncrona entre duas aplicações, porém não é tão difundida. Nesse uso, uma aplicação chama outra diretamente, porém não aguarda pela sua resposta e continua seu processamento de maneira independente. Finalmente, no que tange sua usabilidade, é uma arquitetura extremamente sólida e simples, não é a toa que é utilizada em basicamente qualquer sistema distribuído.

Diferentemente de arquiteturas dirigidas a eventos, adicionar novos serviços na arquitetura REST, em geral, implica em introduzir um novo método e realizar chamadas pelos serviços que precisam dele (STOPFORD, 2018). Entretanto, a arquitetura REST possui algumas vantagens consideráveis como a simplicidade em implementar, dados (ou estado) residem em apenas um lugar e controle centralizado (STOPFORD, 2018). De forma que neste estudo o principal ponto de comparação entre a arquitetura REST e a orientada a eventos será a modularização. A fim de mensurar a modularização, alguns conceitos de qualidade de software e métricas serão abordados para suporte da análise quantitativa.

2.3 Arquitetura Dirigida a Eventos

No modelo baseado em eventos, as aplicações (microsserviços) apenas publicam dados, mas não estão cientes das demais aplicações do sistema, ou seja, não sabem quem vai reagir/consumir aqueles dados, o que promove a separação da computação e publicação de eventos de qualquer processamento subsequente (FIEGE; MÜHL; FREILING, 2002) Isso significa que o acoplamento é fraco e indireto, pois uma aplicação nem conhece a outra. Isto permite serviços serem facilmente adicionados ao sistema, no modo *plug and play* (plugável), em que ele se conecta aos fluxos de eventos e é executado quando seus critérios são atendidos (STOPFORD, 2018).

Embora a arquitetura dirigida a eventos traga muitos benefícios, ela tem alguns desafios a serem enfrentados. Por exemplo, manter a consistência dos dados é mais difícil do que na arquitetura REST, pois aqui caso um consumidor de um evento falhe durante seu processo, não é possível avisar ao produtor do que ocorreu, o que pode levar a inconsistência de dados onde

metade do processo foi concluído e outra metade não (GöRDESLI; NASAB; VAROL, 2022). Além disso, por mais que uma arquitetura dirigida a eventos traga benefícios em escalabilidade e modelagem de software, ela pode, ao mesmo tempo, implicar em uma performance menos otimizada em relação ao mesmo sistema implementado de forma monolítica (CABANE; FARIAS, 2023).

2.4 Ciência de Contexto

O termo contexto possui um significado intuitivo para os seres humanos, mas sua definição pode ser vaga e mal definida. Além disso, os papéis do contexto vêm de diferentes campos, como literatura, filosofia, linguística e ciência da computação, cada um propondo sua própria visão de contexto. Mas, de maneira geral, se refere ao que cerca o centro de interesse. Além disso, os contextos fornecem fontes adicionais de informações relacionadas a "quem, onde, quando e o quê" (TSIBIDIS; ARVANITIS; BABER, 2008) e aumentam a compreensão.

(DEY; ABOWD; SALBER, 2001) forneceu uma categorização para dados de contexto, que é usada por muitos trabalhos. Os autores argumentaram que aplicativos sensíveis ao contexto devem entender a situação dos usuários e seus arredores, e adaptar o comportamento do aplicativo de acordo com esta informação. Assim, eles propuseram quatro categorias básicas para modelar o contexto, que são: (1) identidade; (2) localização; (3) tempo; e (4) atividade. Esses tipos de contexto não respondem apenas as questões de quem, onde, quando e o quê, mas também agem como ponteiros para outras fontes de informação contextual. Em geral, isso significa informações como o status, identidade e localização espaço-temporal de pessoas, grupos e objetos físicos e computacionais.

Um sistema de software que é ciente de contexto significa que ele possui a capacidade de entender e levar em consideração o contexto ou situação em que seus usuários estão operando. Isso envolve a coleta, interpretação e utilização de informações relevantes para melhor atender às necessidades e expectativas dos usuários em um determinado momento. Aqui estão alguns aspectos-chave do que isso implica:

- **Coleta de informações contextuais**: O sistema coleta dados e informações sobre o ambiente, o histórico de interações do usuário e outros fatores que possam influenciar as necessidades do usuário. Isso pode incluir informações como localização, preferências, histórico de uso e até mesmo dados demográficos.

- **Compreensão do contexto**: O sistema utiliza uma ou mais técnicas para entender o contexto das interações do usuário. Isso pode incluir a interpretação do texto, voz ou gestos do usuário, bem como a análise de dados contextuais e usabilidade do sistema.

- **Personalização**: Com base no contexto do usuário, o sistema adapta suas respostas, funcionalidades e recomendações para atender às necessidades específicas do usuário na-

quele momento. Isso pode incluir a apresentação de informações relevantes, a sugestão de ações apropriadas e a personalização da interface do usuário.

- **Antecipação de necessidades**: O sistema tenta antecipar as necessidades dos usuários com base no contexto. Isso pode envolver a oferta de informações relevantes antes mesmo que o usuário as solicite, agindo proativamente para atender às expectativas.

- **Aprimoramento da experiência do usuário**: Ao ser ciente de contexto, o sistema busca melhorar a experiência do usuário, tornando-a mais conveniente, eficiente e relevante.

3 TRABALHOS RELACIONADOS

A pesquisa pelos trabalhos relacionados foi realizada em repositórios digitais como Google Scholar e Scopus (Elsevier). Grande parte dos trabalhos selecionados foram resultados de pesquisas pelo termo "*context-aware event driven architecture with microservices*", em português lê-se "arquitetura orientada a eventos ciente de contexto com microsserviços". A metodologia adotada para analisar os trabalhos relacionados tem sido previamente validada em estudo previamente publicados na literatura (RUBERT; FARIAS, 2022), (JÚNIOR; FARIAS, 2021), (URDANGARIN; FARIAS; BARBOSA, 2021), (VIEIRA; FARIAS, 2020).

3.1 Análise dos trabalhos relacionados

(D'AVILA; BARBOSA; OLIVEIRA, 2020). Este estudo propõe o SW-Context, um modelo focado em definir informações de contexto para software e armazenar essas informações em histórico contextual. O modelo proposto fornece informações qualitativas sobre código-fonte e questões de design para apoiar as atividades diárias dos desenvolvedores de software, melhorando assim a sua consciência da situação. O SW-Context foi avaliado através de um estudo de caso na indústria e teve resultados encorajadores, que mostram o potencial de implementação do SW-Context em um cenário mais amplo de manutenção de software. Entretanto, por mais que SW-Context seja internamente modularizado, isto é, dividido em pequenas funcionalidades, não deixa de ser um monolito, falhando num desacoplamento real. Consequentemente, não se beneficia de comunicação via eventos, pois não existem diferentes sistemas conversando.

(LAIGNER et al., 2020). O artigo discute a substituição de um sistema legado de big data monolítico por uma arquitetura baseada em microsserviços e orientada a eventos. Os autores relatam sua experiência e os desafios enfrentados durante a transição. Eles descobriram que a arquitetura resultante permitiu uma manutenção mais fácil e o isolamento de falhas, mas a variedade de tecnologias e o fluxo complexo de dados foram percebidos como desvantagens. Porém, por mais que a migração tenha sido um sucesso, o projeto não se beneficia de dados contextuais para se adaptar em usuário.

(HENNING; HASSELBRING, 2023). Uma avaliação da escalabilidade de frameworks de mensageria para arquiteturas baseada em microsserviços e comunicação via eventos. O estudo avalia cinco frameworks modernos de mensageria, incluindo Apache Flink, Apache Kafka Streams, Apache Samza, Hazelcast Jet e o Apache Beam SDK, em relação à sua escalabilidade usando um método sistemático. O estudo constata que todos os frameworks avaliados exibem escalabilidade aproximadamente linear, desde que recursos suficientes na nuvem sejam provisionados. No entanto, os frameworks mostram diferenças consideráveis na taxa em que os recursos precisam ser adicionados para lidar com o aumento da carga. Além disso, o estudo observa que não há um framework claramente superior, mas a classificação dos frameworks depende do caso de uso. Entretanto, o trabalho não desenvolveu nenhuma aplicação real, e por conta disso, não haviam dados contextuais para serem utilizados, indo por um caminho diferente do presente trabalho.

(LAIGNER et al., 2021). O artigo propõe um conjunto de recursos para sistemas de banco de dados a fim de lidar com os requisitos de gerenciamento de dados de um sistema baseado em microsserviços. Os autores realizaram uma revisão sistemática da literatura de artigos representativos que relatam a adoção de microsserviços, analisaram uma série de aplicações baseadas em microsserviços de código aberto, e conduziram uma pesquisa online para validar os resultados das etapas anteriores com as percepções e experiências de mais de 120 profissionais e pesquisadores experientes. Por meio desse processo, eles conseguiram categorizar o estado da arte do gerenciamento de dados em microsserviços e observar vários desafios fundamentais que não podem ser resolvidos apenas pelas práticas de engenharia de software, mas requerem suporte em nível de sistema para aliviar o peso imposto aos profissionais. O artigo discute as deficiências dos sistemas de banco de dados de ponta em relação aos microsserviços e conclui propondo um conjunto de recursos para sistemas de banco de dados orientados a microsserviços. Entretanto, da mesma forma que o trabalho anterior, este não utiliza dados contextuais para fim algum, de forma que não aborda todas as ideias do presente trabalho.

(ZHELEV; ROZEVA, 2019). Discussão quanto aos desafios de escolher a arquitetura adequada para projetos de big data, que frequentemente precisam processar milhares de mensagens por segundo. Os autores sugerem que soluções que implementam microsserviços e arquitetura orientada a eventos podem oferecer escalabilidade e extensibilidade para a aplicação necessária. Eles destacam as vantagens e os problemas potenciais relacionados ao processamento de fluxo de dados em big data. O artigo também lista as desvantagens da arquitetura de microsserviços, como o custo inicial de implementação, e ressalta a importância da tolerância a falhas. Os autores concluem que, embora muitos aplicativos bem-sucedidos tenham começado como monolíticos, a arquitetura de microsserviços funciona bem para aplicativos complexos que requerem escalabilidade fácil e comunicação mais rápida entre os módulos do aplicativo. Mas, por mais que tenha havido sucesso em criar uma aplicação e deixá-la totalmente desacoplada, ainda assim ela é incapaz de utilizar dados contextuais para se adaptar ao uso.

(RAHMATULLOH et al., 2022). O artigo discute os desafios enfrentados em sistemas ba-

seados em microsserviços, como a comunicação entre serviços usando métodos síncronos como REST APIs, e a necessidade de escalabilidade e desempenho. A solução proposta é integrar a tecnologia de contêiner com a arquitetura orientada a eventos, para lidar com a comunicação interna entre os microsserviços. O estudo constatou que a orientação e eventos resultou em tempos de resposta mais rápidos e menor taxa de erros, embora tenha utilizado mais recursos de CPU em comparação com a arquitetura baseada em REST APIs. A conclusão é que a tecnologia de contêiner integrada à eventos pode melhorar o desempenho e a escalabilidade em sistemas baseados em microsserviços.

(ORTIZ et al., 2019). Apresenta a criação de uma arquitetura escalável para fornecer ações em tempo real com ciência de contexto com base no processamento preditivo de streaming de dados. Para isso, foi implementado uma arquitetura baseada em microsserviços. Como resultado, a arquitetura foi capaz de fornecer previsões confiáveis por meio de análise preditiva e técnicas de processamento de eventos complexos, permitindo a notificação de informações relevantes com ciência de contexto. Além disso, ela foi refatorada para seguir um padrão de arquitetura de microsserviços, melhorando significativamente sua manutenção e evolução. O desempenho da arquitetura foi avaliado por meio de um estudo de caso de qualidade do ar.

3.2 Análise comparativa dos trabalhos relacionados

Critério de Comparação. Foram definidos cinco Critérios de Comparação (CC) para realizar a comparação de similaridades e diferenças entre o trabalho proposto e os artigos selecionados. Sua comparação é crucial para auxiliar na identificação de oportunidades de pesquisa utilizando critérios objetivos, ao invés de subjetivos. Os critérios são descritos a seguir:

- **Estudo Empírico (CC1):** pode ser entendida como aquela em que é necessária comprovação prática de algo, especialmente por meio de experimentos ou observação de determinado contexto para coleta de dados em campo;
- **Análise de Modularização (CC2):** compreende a capacidade de decomposição da aplicação em programas menores com interfaces padronizadas, oferecendo maior capacidade de gerenciamento no desenvolvimento da aplicação;
- **Arquitetura dirigida a eventos (CC3):** composta por diversos componentes onde a comunicação entre produtores e consumidores ocorre por meio de eventos;
- **Arquitetura de microsserviço (CC4):** cada módulo da aplicação é totalmente independente de outros (*standalone*), cada um contém uma suíte de serviços autônomos, comunicando-se por meio de uma API;
- **Contexto de aplicação (CC5):** define se o estudo aborda a implementação de uma aplicação real.

- **Ciência de Contexto (CC6):** se a aplicação em questão utiliza dados contextuais para seu funcionamento, como localização ou idade do usuário

Tabela 1 – Análise comparativa dos Trabalhos Relacionados selecionados

Trabalho Relacionado	**Critério de Comparação**					
	CC1	**CC2**	**CC3**	**CC4**	**CC5**	**CC6**
Trabalho Proposto	●	●	●	●	●	●
(LAIGNER et al., 2020)	●	○	●	●	●	○
(D'AVILA; BARBOSA; OLIVEIRA, 2020)	●	◐	○	○	●	●
(HENNING; HASSELBRING, 2023)	●	◐	●	●	○	○
(LAIGNER et al., 2021)	●	●	◐	●	◐	○
(ORTIZ et al., 2019)	●	●	◐	●	◐	◐
(ZHELEV; ROZEVA, 2019)	◐	●	●	●	◐	○
(RAHMATULLOH et al., 2022)	●	●	●	◐	○	○

Legenda: ● Atende Completamente ◐ Atende Parcialmente ○ Não Atende

Fonte: Elaborado pelo autor.

Oportunidades de pesquisa. A Tabela 1 apresenta a comparação dos estudos selecionados, evidenciando o não atendimento de todos os critérios. A seguir, as oportunidades observadas a partir das comparações: (1) apenas o trabalho proposto atende todos os critérios de comparação definidos; (2) poucos estudos utilizam dados contextuais para seu funcionamento; (3) poucos estudos realmente desenvolvem algo real, que não seja puramente a caráter de teste ou demonstração. Portanto, a seguinte oportunidade de pesquisa foi identificada: desenvolver uma arquitetura baseada em microsserviços e comunicação via eventos para implementar uma aplicação real que se beneficia de dados contextuais para seu funcionamento. No caso, o sistema real alvo será um sistema de compra de passagens aéreas.

4 ABORDAGEM PROPOSTA

Esta Seção apresenta Accorsi Airlines, uma proposta de arquitetura de software baseado em microsserviços e comunicação via eventos, que se beneficia de dados contextuais para seu funcionamento. A Seção 4.1 apresenta as definições base para o projeto, detalhando o que compreende e o que não compreende o estudo. Em seguida, a Seção 4.2 apresenta mais informações sobre o módulo de contexto, seguidos pela Seção 4.3 e 4.4 onde é discutido a arquietura proposta e a implementação, sussessivamente. Tal abordagem foi baseada em orientações empíricas amplamente conhecidas (WOHLIN et al., 2012) e em estudos anteriores publicados (D'AVILA; BARBOSA; OLIVEIRA, 2020).

4.1 Decisões de Projeto

Tomando como referências sistemas de compras de viagens aéreas convencionais, especificamente Latam Airlines, Gol e Azul, uma ampla gama de funcionalidades é oferecida para

proporcionar aos usuários uma experiência completa e personalizada. Essas funcionalidades incluem desde a autenticação e registro de usuários, até a pesquisa detalhada de voos, exibição de resultados, reserva de passagens, gerenciamento de carrinho de compras, processamento de pagamentos, gerenciamento de reservas e notificações por e-mail. Além disso, características como programas de fidelidade, informações sobre aeroportos e suporte multilíngue complementam a experiência. Traduzindo em uma lista, chega-se nos seguintes itens:

1. **Autenticação e Registro:** Registro de novos usuários ou autenticação de usuários existentes.
2. **Gerenciamento de Perfis:** Permite aos usuários atualizar suas informações pessoais, preferências de viagem e detalhes de contato.
3. **Pesquisa de Voos:** Oferece uma maneira de procurar voos com base em critérios como datas, destinos e preferências de classe.
4. **Exibição de Resultados:** Apresenta aos usuários uma lista de voos disponíveis com detalhes como horários, escalas e preços.
5. **Reserva de Voos:** Permite aos usuários selecionar voos desejados e adicioná-los ao carrinho de compras.
6. **Carrinho de Compras:** Mostra aos usuários os voos selecionados e o preço total, permitindo edições e remoções.
7. **Pagamento e Checkout:** Seleção de métodos de pagamento, como cartões de crédito, PayPal, etc.
8. **Gerenciamento de Reservas:** Acesso a detalhes das reservas, incluindo itinerários, números de voo e informações sobre assentos.
9. **Notificações por E-mail:** Comunicações via e-mail.
10. **Check-in Online:** Possibilidade de fazer check-in online antes do voo.
11. **Informações de Aeroportos:** Detalhes sobre os aeroportos de origem e destino, incluindo mapas, direções e serviços disponíveis.
12. **Programa de Fidelidade:** Inscrição em programas de fidelidade para acumular milhas e obter benefícios.
13. **Atendimento ao Cliente:** Páginas de suporte com perguntas frequentes, informações de contato e chat ao vivo.
14. **Idiomas e Moedas:** Suporte a diferentes idiomas e moedas para atender a um público global.
15. **Ofertas e Promoções:** Exibição de ofertas especiais, descontos e promoções atuais.
16. **Informações sobre Bagagem:** Diretrizes de bagagem, políticas de restrições e custos adicionais.

No entanto, no sistema a ser descrito no presente trabalho, optou-se por considerar um subset específico dessas funcionalidades para focar nos requisitos centrais do projeto. Isso inclui funcionalidades essenciais como autenticação, pesquisa de voos, carrinho de compras

e um sistema de comunicação baseado em eventos. Embora algumas funcionalidades robustas sejam deixadas de lado, o objetivo é criar um sistema eficiente, escalável e principalmente capaz de gerar e utilizar dados de contexto para lidar com os aspectos centrais da compra de passagens aéreas, enquanto mantem-se a flexibilidade para expansões futuras. Traduzindo em uma lista, chega-se nos seguintes itens:

1. Autenticação e Gerenciamento de Usuários
2. Pesquisa de Voos
3. Carrinho de Compras
4. E-mail

4.2 Modelo de Contexto

Dentro da arquitetura proposta para o sistema de compra de passagens aéreas, reconhece-se que a experiência do usuário vai além das funcionalidades transacionais básicas. A inclusão de um módulo especializado para capturar e analisar dados contextuais do usuário vem justamente para isso. Este módulo tem a finalidade de coletar informações sobre o comportamento, histórico de ações e preferências do usuário durante sua jornada. Ao entender o contexto de cada usuário, o sistema poderá tomar decisões proativas, como enviar lembretes pertinentes. Um exemplo prático desse cenário é quando um usuário adiciona uma passagem ao carrinho de compras e permanece inativo por um intervalo específico; nesse caso, o módulo reconhecerá o contexto e iniciará um processo para enviar um lembrete por e-mail, incentivando a conclusão da compra.

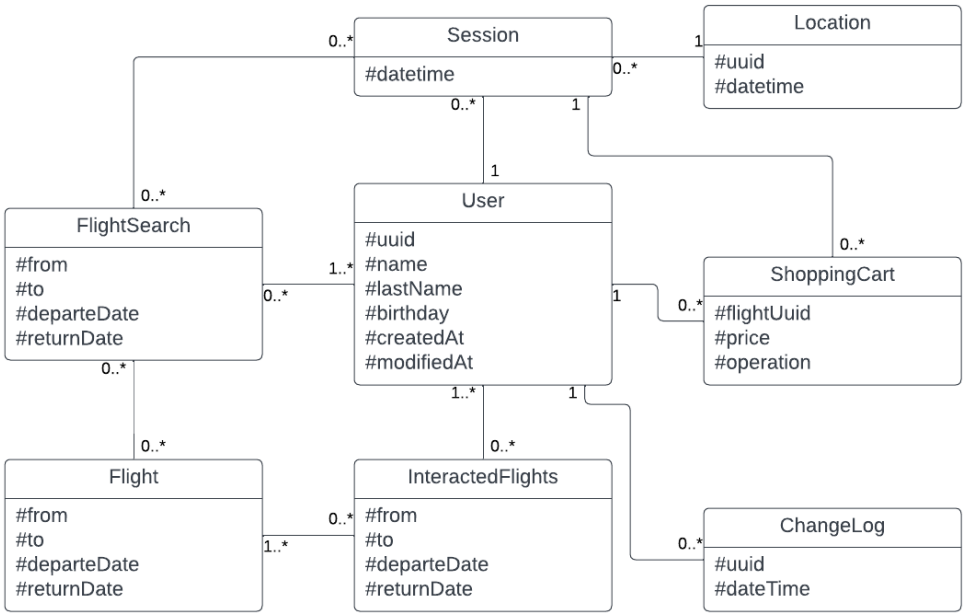
O módulo de dados contextuais foi concebido para ser altamente versátil e adaptável. Ele será capaz de processar diversos eventos, tais como ações do usuário, mudanças de preferências e histórico de compras anteriores. Por meio destas, o sistema é capaz de identificar padrões relevantes e comportamentos que possam indicar oportunidades de interação. A coleta e interpretação desses dados contextuais permitirão que o sistema compreenda as necessidades do usuário de forma mais precisa, enriquecendo sua experiência de compra.

A introdução deste módulo visa não apenas aprimorar as funcionalidades transacionais, mas também criar uma experiência envolvente e proativa. O sistema não se limitará apenas a fornecer informações, mas também agirá com base em insights contextuais para atender às expectativas do usuário de maneira mais relevante e pontual. Isso representa uma abordagem mais abrangente para a compra de passagens aéreas, em que as decisões são embasadas na compreensão do contexto do usuário, elevando a experiência a um novo patamar de personalização e atenção aos detalhes.

Como já apresentado no Referencial Teórico na Seção 2.4, há quatro categorias básicas para modelar o contexto, que são: (1) identidade; (2) localização; (3) tempo; e (4) atividade (DEY; ABOWD; SALBER, 2001). Baseado nas mesmas, foi elaborada a Figura 2 para mostrar o modelo de contexto da Accorsi Airlines, onde as entidades chave e as relações entre as mesmas

são expressadas. O modelo foi organizado em torno das quatro dimensões abordadas anteriormente: identidade, localização, tempo e atividade. A seguir, descreve-se cada um dos membros do modelo.

Figura 2 – Accorsi Airlines: Modelo de Contexto



Fonte: Elaborado pelo autor

- *Session*: identifica os acessos do usuário ao sistema, englobando horário e duração das interações;
- *Location*: identifica de onde o usuário está acessando o sistema;
- *User*: identifica o próprio usuário, a entidade ao qual o modelo se revolve.
- *ShoppingCart*: representa o carrinho de compros que um usuário interage durante uma seção, incluindo informações sobre adições e remoções de voos;
- *ChangeLog*: representa todas as mudanças no perfil do usuário;
- *InteractedFlights*: dado uma pesquisa de voos, representa todos com os quais o usuário interagiu;
- *Flight*: representa um voo, com origen, destino, e datas;
- *FlightSearch*: representa uma pesquisa por voos, englobando vários resultados abaixo de si;

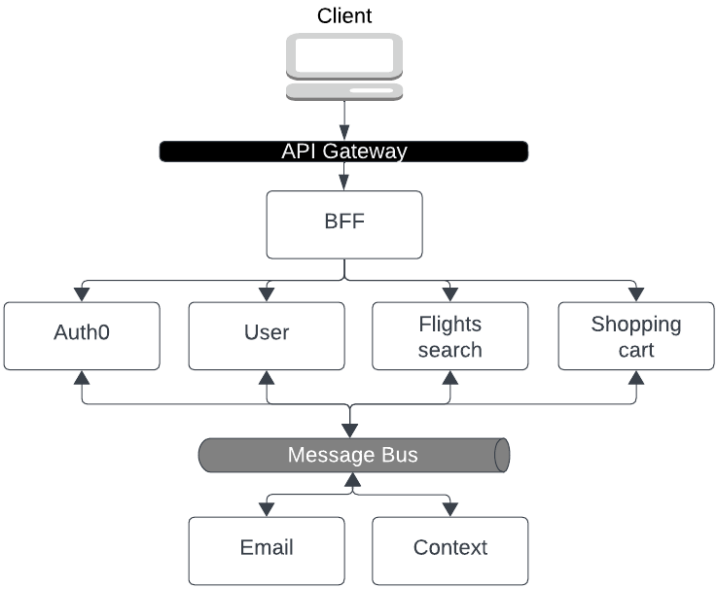
4.3 Arquitetura proposta

Para detalhar a arquitetura do sistema, foram desenhados alguns diagramas, cada qual com seu propósito, de forma que tudo seja explicado e se obtenha material o suficiente para uma

avaliação de eficácia a posteriori, como uma implementação da mesma e entrevistas com outros desenvolvedores.

Diagrama de Arquitetura de Alto Nível: A Figura 3 representa a ideia geral do sistema. Mediante um cliente, são realizadas *requests* (requisições) para um único serviço de backend, comumente referido como BFF ou *backend for frontend*. Sua função é atuar como um gateway entre o cliente e o sistema, de forma que o cliente não tenha conhecimento do sistema de backend inteiro, mas apenas de um. Em seguida, posicionado entre o cliente e o BFF, situa-se um API Gateway. Sua função primordial é a administração, proteção, otimização e regulação do fluxo de requests entre os clientes e os diferentes serviços subjacentes. A comunicação entre o BFF e os demais serviços de backend precisa ser via REST APIs, uma vez que são chamadas síncronas vindas de uma interação do usuário. Finalmente, os serviços de backend vão estar em conversa constante com o Message Bus, de forma que possam consumir e publicar mensagens diretamente ao mesmo. Com isso, todos os microsserviços tem a possibilidade de reagir a quaisquer mensagens enviadas pelos demais, caso seja de interesse para a regra de negócio do mesmo.

Figura 3 – Accorsi Airlines: Diagrama de Interação de Serviços



Fonte: Elaborado pelo autor

Diagrama Arquitetural do módulo de Contexto: A Figura 4 mostra o diagrama arquitetural do módulo de Contexto do Accorsi Airlines, onde encontram-se três grupos. Em verde, estão os *topic subscribers*. São responsáveis por receber todos os eventos emitidos pelos demais serviços que compõem o sistema. Na imagem estão exemplificados os seguintes tópicos:

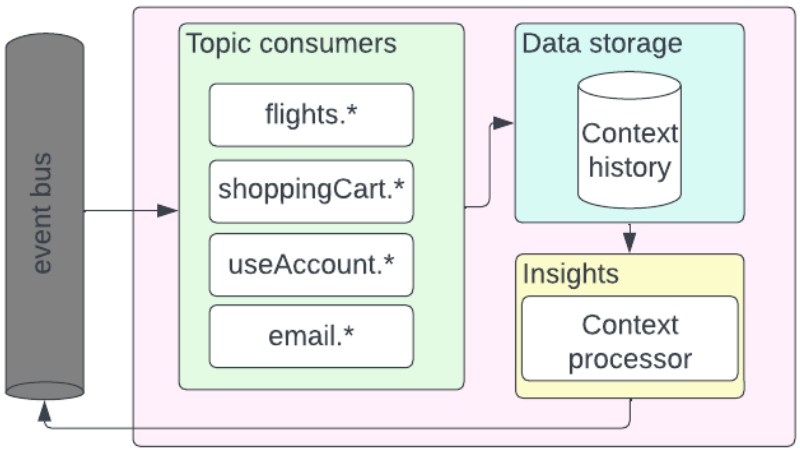
- *flights.**: eventos emitidos pelo serviço de voos, composto por *flights.search*, expondo exatamente as buscas de voos que o usuário fez, e também *flights.searchResults*, com a

lista de resultados obtidas pela pesquisa. Esses eventos são de extrema relevância pois permitem determinar o interesse do usuário, quais destinos ele tem interesse, quais as datas, e inclusive quais os preços que mais o atraem.

- *userAccount.**: eventos emitidos pelos serviço de usuários, especificamente *userAccount.login* e *userAccount.signup*. Com estes eventos, é possível inferir a frequência de acesso do usuário no sistema, os horários de uso, e, tão importante quanto os demais, entender a quanto tempo que o usuário não utiliza o sistema.
- *email.**: eventos emitidos pelo serviço de e-mails, composto apenas por *email.sent* sinalizando quando um e-mail é enviado ao usuário. A partir deste evento, é possível determinar se o usuário está recebendo comunicações em excesso (ou em falta) do sistema, o que é muito relevante quando se considera emails como uma forma de *call to action* (chamada para ação) do usuário, isto é, convidá-lo a usar o sistema.
- *shoppingCart.**: eventos emitidos pelo serviço de carrinho de compras, composto por *shoppinCart.add, shoppingCart.remove, shoppingCart.buy*, essenciais para somar com eventos de busca de voos para determinar exatamente o que o usuário procura e acha relevante, por exemplo principais horários de voos de interesse e variação de preço.

Em seguida, em azul, há o *data storage*, representando o banco de dados que vai armazenar todas as informações que cada um dos *topic subscribers* achar relevante. A partir delas que, em amarelo, o módulo interno *Insights* vai atuar, lendo os dados obtidos através do comportamento dos usuários, processando-os e tomando decisões. Essas decisões significam emitir mais eventos, porém agora para os demais serviços do sistema, para que possam agir de acordo.

Figura 4 – Accorsi Airlines: Diagrama arquitetural do módulo de Contexto



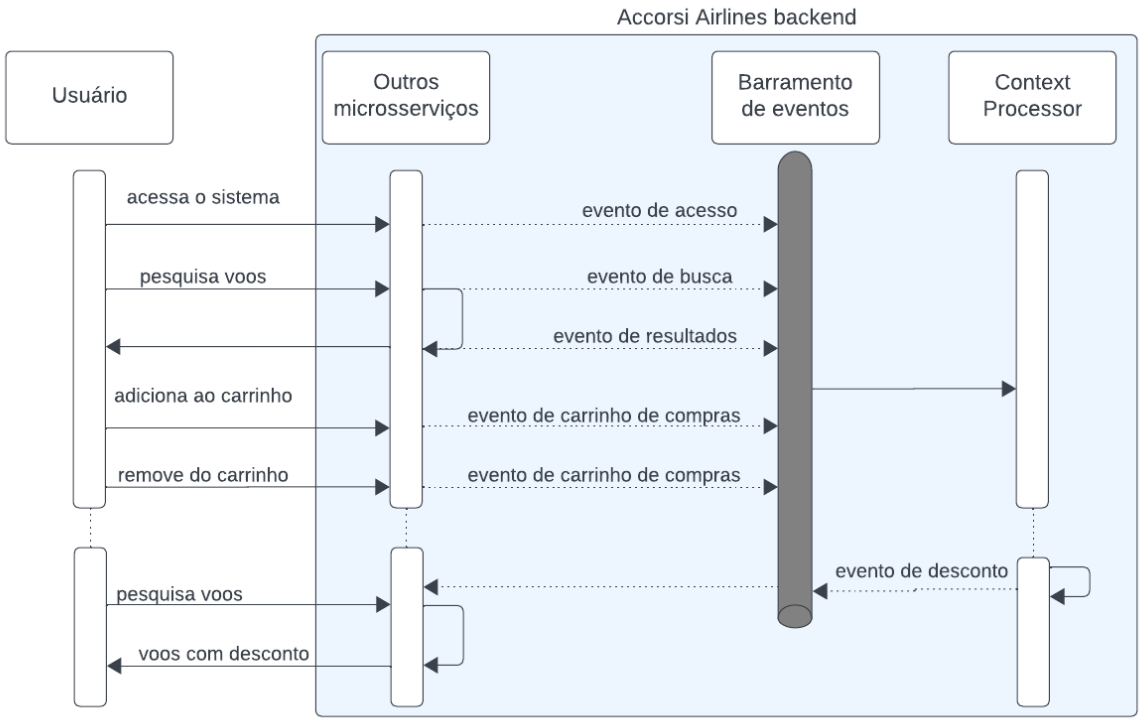
Fonte: Elaborado pelo autor

Fluxo de eventos: A Figura 5 representa como o fluxo de eventos vai funcionar, a partir das interações do usuário para com o sistema. Em suma, cada interação acaba atingindo um

microsserviço diferente no backend, como o de usuários, o de voos, o de carrinho de compras, assim por diante. Por sua vez, cada um desses microsserviços vai emitir um evento para o barramento, e seguir seu fluxo de funcionamento padrão.

Em seguida, esses eventos são todos capturados pelo microsserviço de contexto, que inicialmente apenas captura e armazena. Eventualmente, de tempos em tempos todo o contexto capturado recentemente é processado, de forma a obter informações sobre o usuário e o sistema, para que ações possam ser determinadas. À exemplo, caso o usuário faça uma quantidade pré determinada de acesso, pesquise pelos mesmos voos, e não efetive uma compra, o microsserviço de contextos vai gerar a ação de automaticamente efetivar um desconto para o usuário na próxima pesquisa de voos. Isso também acontece via eventos, onde o microsserviço de contextos emite isso ao barramento, e o sistema responsável captura e lida com a informação nova.

Figura 5 – Accorsi Airlines: Fluxo de eventos



Fonte: Elaborado pelo autor

4.4 Aspectos de Implementação

Sistema de Mensageria: A escolha do Apache Kafka como o meio comum de comunicação entre os microsserviços neste projeto é baseada em diversos fatores estratégicos e técnicos. Em primeiro lugar, o Kafka é conhecido por sua escalabilidade e capacidade de lidar com um grande volume de eventos em tempo real. Isso é essencial para garantir que os microsservi-

ços possam se comunicar de forma eficiente, independentemente do crescimento do sistema. Além disso, o Kafka oferece uma arquitetura de mensagens distribuída altamente confiável. Ele garante a entrega de mensagens mesmo em situações de falha, o que é crucial para manter a integridade das operações do sistema. A capacidade de armazenar eventos por um período configurável também é valiosa para a recuperação de dados e análises retroativas. Outro aspecto relevante é a capacidade de Kafka de suportar a diversidade tecnológica dos microsserviços. Como cada microsserviço pode ser desenvolvido em uma linguagem e utilizar um banco de dados diferente, o Kafka atua como um intermediário neutro, permitindo a comunicação eficiente entre essas entidades heterogêneas. Em resumo, a escolha do Apache Kafka como tecnologia central de comunicação se baseia em sua escalabilidade, confiabilidade e capacidade de acomodar a diversidade tecnológica dos microsserviços, tornando-o a única tecnologia que merece um planejamento antecipado neste contexto.

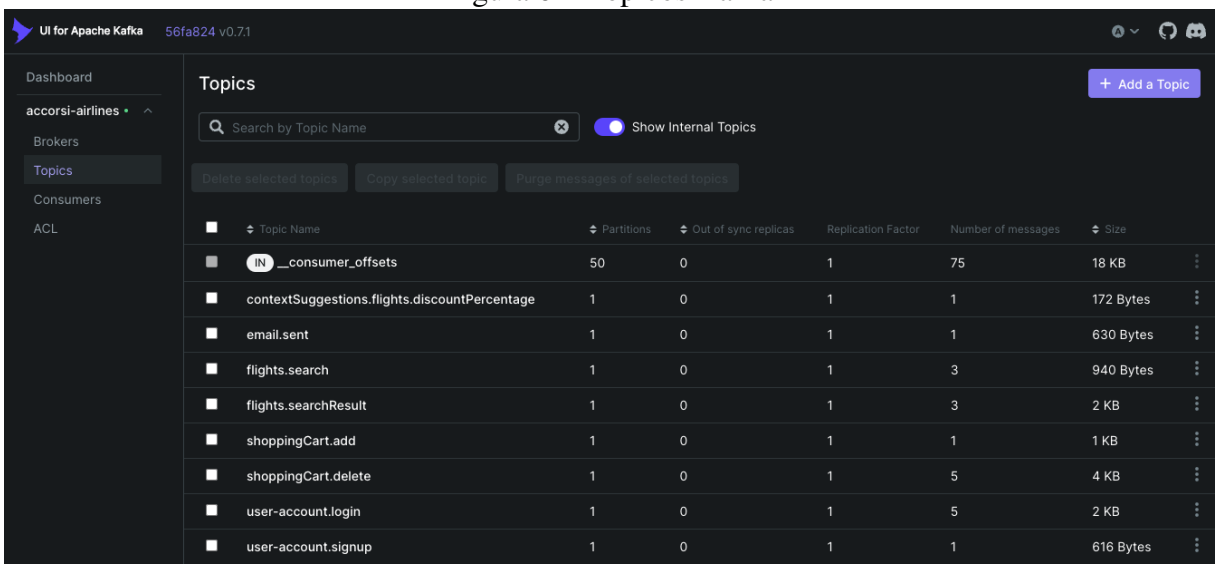
Para auxiliar na visualização do fluxo de eventos, tanto para os pesquisadores mas especialmente ao demonstrar para os participantes da pesquisa posteriormente, foi utilizada uma *UI* (interface de usuário), demonstrada nas Figuras 6 e 7. Essa interface se conecta a um cluster Kafka e é capaz de exibir informações relevantes sobre o mesmo, como quais os brokers ativos, a lista de consumidores e todos os tópicos em funcionamento. É capaz de mostrar também a quantidade de mensagens que cada tópico possui, o tamanho em bytes, e permite a criação dessas entidades via UI, para que alguns testes sejam mais simples de conduzir.

A Figura 6 mostra os tópicos presentes no broker, ou seja, uma lista com todos os eventos emitidos pelo sistema, previamente explicados na Seção 4.3. Essa lista é automaticamente populada sempre que um tópico ainda não listado recebe uma mensagem. Ou seja, basta algum microsserviço enviar uma mensagem, que a mesma pode ser conferida através da UI. Foi adotada uma nomenclatura em que o prefixo do tópico corresponde ao microsserviço que o gerou, para facilitar a leitura. Na figura, pode-se visualizar eventos provindos dos microsserviços de usuários, emails, voos e carrinho de compra, ou seja, capturando praticamente qualquer interação do usuário com o sistema.

Já a Figura 7 representa os consumidores, ou seja, quem está inscrito para "ouvir" cada um dos tópicos mencionados acima. Novamente, essa lista é atualizada automaticamente sempre que um novo subscriber entra no cluster. Diferente dos tópicos, os consumidores não possuem um nome específico, porém em contrapartida possuem um grupo. Este grupo também segue a nomenclatura onde o prefixo se refere ao microsserviço que o implementa, ou seja, na imagem podemos ver consumidores provenientes do microsserviço de contextos, voos e usuários.

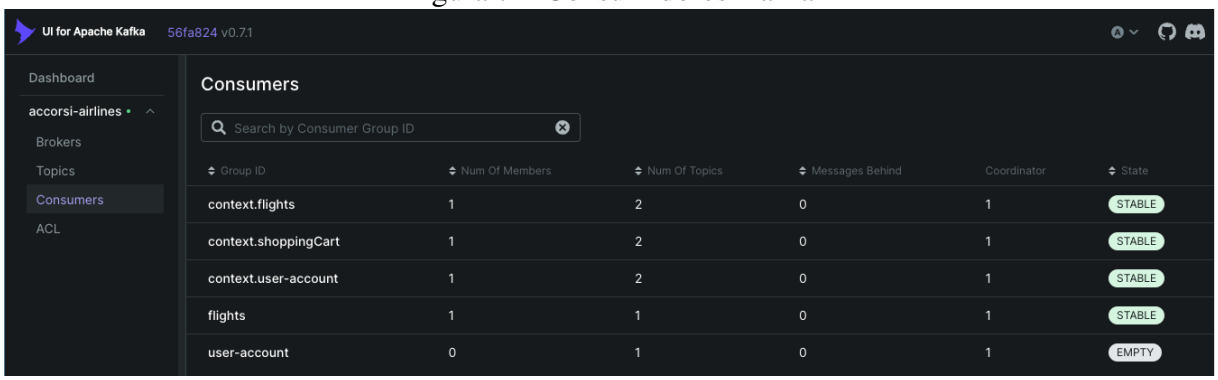
Microsserviço de Contextos: Por mais que seja o coração do sistema, este microsserviço foi pensado para ser razoavelmente simples. Como já mencionado é composto por três módulos internos: armazenamento de dados, insights e consumidores. Para o armazenamento de dados, foi utilizado um banco de dados MySQL. A escolha de um banco relacional se dá porque os dados a serem armazenados nesse sistema são naturalmente relacionais, uma vez que apenas em conjunto possuem valor, ou seja, utilizar documentos não estruturados (noSQL) não traria

Figura 6 – Tópicos Kafka



Fonte: Elaborado pelo autor

Figura 7 – Consumidores Kafka



Fonte: Elaborado pelo autor

benefício algum. Já em relação ao MySQL em si, MySQL é o segundo RDBMS (sigla em inglês para sistema de gerenciamento de banco de dados relacionais) mais popular (IT, 2023), e causa disso é seu amplo uso no mercado, estando presente na infraestrutura dos maiores softwares da indústria, como Spotify, Netflix, Facebook e Booking.com (CASTRO, 2021). Estes fatos motivaram a escolha.

Já no que tange o código, compreendido pelo módulo de insights e consumidores, foi implementado utilizando TypeScript, com o runtime Bun, exclusicamente pela familiaridade dos pesquisadores com a tecnologia. No módulo de insights, para fazer uma análise de todo o contexto obtido dos usuários, foi implementado apenas um insight, de maneira que fosse possível provar/testar a funcionalidade. Na figura 7 podemos ver uma versão em pseudocódigo da função responsável. Essa função tem o objetivo de buscar todos os usuários que fizeram login no sistema nos últimos 15 dias, e para cada um deles, verificar todos os voos que os mesmos adicionaram ou removeram do carrinho de compras. Caso seja determinado que o usuário acessou

o sistema pelo menos 3 vezes, e adicionou e removeu do carrinho o mesmo voo pelo menos 3 vezes, então é disparado um evento de desconto, de forma que na próxima pesquisa por voos deste usuário, ele encontre um preço menor, na tentativa de converter um potencial comprador. Porém, vale mencionar que caso um usuário receba este desconto, há um controle para não repeti-lo pelos próximos 15 dias, para que não seja possível explorar essa funcionalidade.

Com essa estrutura básica, busca-se demonstrar que o sistema tem capacidade de extrair, processar e reagir à maneira que o usuário utiliza o sistema, pois ela consegue 1) extrair informações do usuário e 2) processar essas informações coerentemente para 3) gerar algum valor de volta para o usuário, de maneira exclusiva.

Figura 8 – Função de Insight em pseudocódigo

```
FUNCTION processContextIndefinitely

    FETCH users, numberOfLogins FROM DB.USER_LOGIN      WHERE timestamp <= 15 days ago
    FETCH usersProcessed        FROM DB.USERS_PROCESSED WHERE timestamp <= 15 days ago

    SET userIdsToNotRun FROM usersProcessed.userId

    FOR EACH user IN users DO

        IF user.userId IS IN userIdsToNotRun OR user.numberOfLogins IS LESS THAN 3 THEN
            CONTINUE to next user
        ENDIF

        SET removedFlightsCount TO 0

        FETCH carts FROM DB.SHOPPING_CART WHERE userId = user.userId AND timestamp <= 15 days ago

        SET repetitiveFlights FROM ALL DUPLICATE OCCURENCES OF Origin and Destiny IN carts

        FOR EACH flight IN repetitiveFlights DO
            IF flight.wasRemoved THEN increment removedFlightsCount BY 1
        ENDFOR

        IF removedFlightsCount > 3 THEN
            PUBLISH MESSAGE with user.userId and discountPercentage of 5
            INSERT user.userId, current timestamp INTO DB.USERS_PROCESSED
        ENDIF

    ENDFOR

END FUNCTION
```

Fonte: Elaborado pelo autor

Demais serviços: Como mencionado no item 4.3, foram desenvolvidos mais 5 microsserviços para representar o backend inteiro da aplicação:

1. *auth*: assinar e verificar JWT
2. *user-account*: login e signup
3. *flights*: pesquisa de voos por datas
4. *shopping-cart*: carrinho de compras
5. *email*: notificações via e-mail

Assim como o microsserviço de contexto, esses 5 também foram desenvolvidos em TypeScript rodando em Bun. Os serviços *user-account*, *flights* e *shopping-cart* possuem bancos de dados próprios, isolados e acessíveis apenas pelo serviço. Não obstante, todos eles estão integrados à rede de mensageria, tanto publicando como consumindo diferentes eventos para seu funcionamento pleno. Finalmente, os 5 serviços expõem algumas APIs REST para chamadas síncronas entre si, visto que por ser um sistema distribuído, algumas tarefas demandam conversas síncronas entre diferentes serviços, o que não é possível com troca de mensagens.

5 AVALIAÇÃO

Nesta Seção apresenta-se o desenvolvimento da avaliação do protótipo do backend da Accorsi Airlines. A Seção 5.1 descreve o processo de avaliação implantado. A Seção 5.2 apresenta o processo de seleção dos participantes. Na Seção 5.3 é desenvolvido o questionário avaliativo da aplicação. Na Seção 5.4 são analisados os dados quantitativos e qualitativos obtidos através dos questionários aplicados. Na Seção 5.5 são analisados pontos de discussão. Na Seção 5.6 são demonstradas as implicações obtidas através dos resultados analisados.

5.1 Processo de Avaliação

Para ilustrar o processo de avaliação, foi criada a Figura 8. Nela se encontra detalhada cada uma das 4 fases executadas para a obtenção dos dados, sendo descritas abaixo:

Fase 1: Seleção dos participantes: Nesta fase inicia-se o processo de avaliação da aplicação definindo os perfis dos participantes (dados de entrada). Como pré-requisito, os participantes devem ter experiência profissional em desenvolvimento de software e serem capazes de utilizar algum REST Client (visto que o protótipo representa um backend). Obtendo por fim a lista de participantes (dados de saída) que irão realizar a avaliação na fase seguinte.

Fase 2: Execução do protótipo. Nesta etapa, o protótipo é disponibilizado aos participantes através da máquina dos pesquisadores, pois exige uma configuração prévia. Os participantes efetuam algumas interações com o sistema, realizando desde a criação de uma conta até a pesquisa de voos, adicionar e remover do carrinho de compras, etc, observando quaisquer inconsistências e, especialmente, se o sistema se adaptou ou não baseado no uso do usuário.

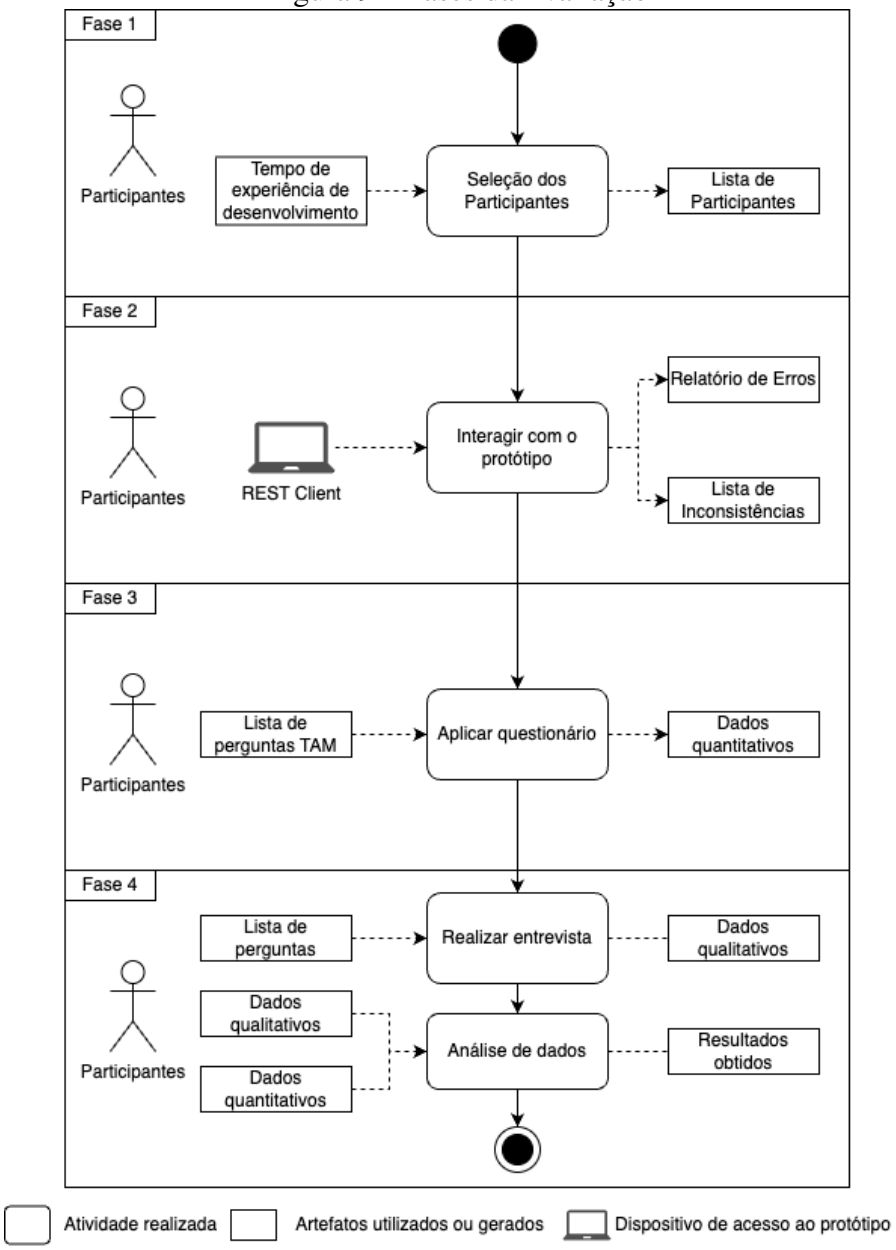
Fase 3: Aplicação dos questionários. Nesta fase o participante é direcionado a responder a uma lista de perguntas (dados de entrada) utilizando a metodologia de aplicação de questionário TAM, referente a experiência de usabilidade, comportamento e importância da ferramenta. Esta etapa da avaliação tem como intuito obter dados quantitativos.

Fase 4: Entrevistas e Análise dos Dados. Nesta fase são realizadas as entrevistas, estas ocorrem com um participante de cada perfil, extraindo as impressões obtidas ao interagir com Accorsi Airlines. Ao fim da aplicação das entrevistas, são gerados dados qualitativos de saída.

Ao final das quatro fases, são analisadas as métricas e impressões obtidas na fase de apli-

cação do questionário TAM e das entrevistas realizadas. Iniciando assim o processo de análise dos dados qualitativos e quantitativos (dados de entrada) assim gerando os resultados finais.

Figura 9 – Fases da Avaliação



Fonte: Elaborado pelo autor

5.2 Seleção dos Participantes

Esta Seção trabalha no levantamento das informações para determinar a seleção dos participantes a avaliarem a aplicação. Como pré-requisito, todos os participantes devem atuar como desenvolvedores de software profissionalmente, e serem capazes de utilizar algum REST Client para interagir com o sistema. Desta forma, foram selecionados três tipos de perfis, baseado na

experiência anterior com desenvolvimento e arquitetura de sistemas.

Desenvolvedores com menos de 2 anos de experiência: Esses desenvolvedores representam a perspectiva dos iniciantes na indústria de desenvolvimento de software. Eles podem oferecer insights sobre a acessibilidade e a facilidade de aprendizado da arquitetura proposta, já que estão mais próximos da fase de aprendizado e adaptação. Suas experiências podem revelar se o sistema é intuitivo para novos desenvolvedores e se a documentação/explicação é adequada.

Desenvolvedores com mais de 2 e menos de 5 anos de experiência: Esse grupo representa desenvolvedores intermediários, que já têm alguma experiência, mas ainda estão se desenvolvendo. Eles podem fornecer feedback valioso sobre as funcionalidades e da arquitetura de software empregada. Esses desenvolvedores podem identificar desafios que enfrentam ao lidar com sistemas complexos e sugerir melhorias com base em suas experiências passadas.

Desenvolvedores com mais de 5 anos de experiência: Os desenvolvedores com mais de 5 anos de experiência são especialistas na área e têm uma visão aprofundada do desenvolvimento de software. Eles podem oferecer insights críticos sobre a eficiência, desempenho e segurança da arquitetura de software proposta. Além disso, sua experiência pode revelar oportunidades para otimizar o sistema e torná-lo mais robusto.

Em conjunto, esses três grupos de desenvolvedores representam uma amostra diversificada. Suas opiniões e experiências coletivas podem ajudar a identificar áreas de aprimoramento, garantir que o sistema atenda a uma ampla gama de necessidades e assegurar sua viabilidade a longo prazo no mercado de desenvolvimento de software.

5.3 Questionários

Para avaliar o protótipo, foi aplicado um questionário dividido em duas partes, cada uma com um propósito específico. Inicialmente, a primeira parte visa identificar o perfil dos participantes, levando em consideração suas experiências anteriores com arquitetura de software. A seguir, a segunda parte tem como objetivo avaliar qualitativamente o protótipo. Abaixo segue uma descrição mais detalhada de ambas.

Pesquisa de Perfil. Conforme previamente planejado, dividimos os participantes em três categorias. Nessa etapa do questionário, procuramos coletar informações que são fundamentais para entender o histórico de cada participante. Isso inclui idade, tempo de experiência profissional em desenvolvimento, conhecimento em arquitetura distribuída e orientada a eventos. Para tal, algumas das informaçõescoletadas são a idade, grau de escolaridade, se já desenhou algum sistema anteriormente, entre outras.

Pesquisa TAM. Esta parte do questionário tem como objetivo validar qualitativamente o protótipo usando o Modelo de Aceitação de Tecnologia (TAM). Os tópicos abordados incluem avaliar a usabilidade da ferramenta, medindo a facilidade de interação e aprendizado, a utilidade da aplicação em relação ao seu propósito, a correspondência do comportamento do software e, por fim, se a aplicação atende às expectativas dos usuários. Para coletar as opiniões, utilizamos

a metodologia da escala Likert, onde os participantes classificaram suas avaliações como "Concordo Totalmente", "Concordo Parcialmente", "Neutro", "Discordo Parcialmente" ou "Discordo Totalmente".

5.4 Análise de Resultados

A Tabela 2 e 3 apresentam os resultados obtidos na pesquisa de perfil e TAM, apontando as características e opiniões de cada participante. Os dados foram coletados no período de 29 de outubro até 4 de novembro de 2023, contabilizando um total de 34 participantes. Abaixo se encontra uma análise detalhada em relação aos dois grandes focos da pesquisa, o perfil e as perguntas TAM.

Resultados da pesquisa de Perfil: Pode-se observar que a maioria dos entrevistados possui entre 25 e 30 anos (29,4%), porém é uma amostra praticamente igual à de pessoas entre 20 e 24, que representam 26,5% dos candidados, e que coincidendemente também é a mesma representação dos maiores de 30, também com 26,5%. Em relação à escolaridade, 28 dos 34 participantes, representando 82,4% do total, possuem ou estão cursando sua graduação. Quanto à experiência profissional em desenvolvimento de software, 29,4% possuem menos de 2 anos, 41,2% possuem de 2 a 6 anos de experiência, e 29,4% representam os profissionais com mais de 6 anos de indústria. Já no que diz respeito à experiência em modelagem de software, os valores são menores, com mais de 40% dos participantes tendo menos de 2 anos de experiência, e apenas 30% com mais de 5 anos. Finalmente, dos 34 entrevistados, 23 possuíam experiência prévia com algum sistema de mensageria, representando 67,6% do total.

Resultados da pesquisa TAM: A Tabela 3 representa os resultados obtidos nas questões focada em aplicar as questões TAM para uma avaliação em relação à arquitetura e ao protótipo proposto pelos pesquisadores. Por ser uma sequência de perguntas TAM, foram abordados tópicos em relação à facilidade de uso, percepção de utilidade e intenção de comportamento. Sobre a facilidade de uso, 58,2% dos participantes concordam totalmente que foi fácil entender a arquitetura proposta, enquanto 29,41% concordaram parcialmente e 11,76% se mostraram neutros. Em relação à maestria da arquitetura, 47,06% concordam totalmente que é fácil de masterizá-la, ao passo que 35,29% e 17,65% concordam parcialmente e se demonstraram neutros, respectivamente. Em relação à percepção de utilidade, todos os entrevistados concordaram total (70,59%) ou parcialmente (29,41%) que o protótipo se mostrou satisfatório e é sim capaz de extrair e processar corretamente dados contextuais em relação ao usuário e a forma que o mesmo utiliza o sistema, gerando um impacto positivo nos participantes. Finalmente, no que tange a intenção de comportamento, todos os participantes concordaram que o sistema serve sim como uma base completa para uma aplicação real, porém, 6 se mostraram neutros em relação a utilizar ou não a mesma de maneira produtiva.

Com isso, Accorsi Airlines comporta-se adequadamente mediante a intenção do seu desenvolvimento. Contemplando os quesitos necessários para a extração e processamento de dados

Tabela 2 – Resultados da pesquisa de Perfil

Pergunta	Resposta	Quantidade	Porcentagem
Qual sua idade?	Menos de 20	6	17,6
	De 20 a 24	9	26,5
	De 25 a 30	10	29,4
	Mais de 30	9	25,5
Qual sua maior escolaridade (completa ou em andamento)?	Ensino médio	2	5,9
	Graduação	28	82,4
	Mestrado	4	11,8
	Doutorado	0	0
Quanto tempo de experiência profissional você tem em desenvolvimento de software?	Menos de 2	15	44,1
	De 2 a 4	7	20,6
	De 5 a 6	4	11,8
	De 7 a 8	4	11,8
	Mais de 8	4	11,8
Você já utilizou sistemas de mensageria?	Sim	23	67,6
	Não	11	32,4
Como você avalia seu conhecimento em relação a arquitetura de microsserviços baseada em eventos?	Nenhum conhecimento		
	1	2	5,9
	2	8	23,5
	3	8	23,5
	4	4	17,6
	5	5	29,4
	Consigo criar um sistema		

contextuais de usuários, tornando possível uma experiência exclusiva para cada um.

Resultado das Entrevistas: Para expandir e entender melhor os resultados da pesquisa TAM, 15 participantes foram selecionados de maneira aleatória para uma entrevista em relação ao protótipo estudado. Busca-se desta forma, contrastar ou reafirmar os resultados obtidos na aplicação do questionário TAM.

Ao serem questionados sobre as impressões que obtiveram ao executar a aplicação, os participantes relataram em unanimidade que, por mais que tenha sido relativamente fácil e intuitivo, a falta de uma *UI* (interface de usuário) foi um empecilho no pontapé inicial para as interações com o sistema, pois não havia um fluxo claro a ser seguido visto que os mesmos estavam interagindo com um REST Client, e não com uma tela agradável e apresentável em relação ao projeto. Essa opinião se mostrou muito mais forte para os entrevistados com menor experiência em desenvolvimento, que consequentemente tiveram um pouco mais de dificuldade no entendimento.

Sobre a funcionalidade de contextos, todos os entrevistados tiveram uma ótima impressão em relação a como o sistema funciona e das possibilidades que isso traz, porém, se mostraram um pouco decepcionados com o fato de que as "reações" ao contexto do usuário não eram dinâmicas, ou seja, havia um set de possibilidades pequeno para demonstrar e testar. Por mais que compreendam que o sistema demonstra todas as capacidades para realizar isso, ficou-se com a impressão de "quero mais" neste aspecto.

Na percepção dos participantes, o maior ganho que o sistema traz é justamente a ideia de

Tabela 3 – Resultados da pesquisa TAM

	Concordo totalmente	Concordo parcialmente	Neutro	Discordo parcialmente	Discordo totalmente
Percepção de facilidade de uso					
A arquitetura proposta é fácil de entender	20	10	4		
A arquitetura proposta é fácil de masterizar	16	12	6		
Percepção de utilidade					
O protótipo apresentado é capaz de extrair dados contextuais em relação ao usuário	24	10			
O protótipo apresentado é capaz de processar dados contextuais em relação ao usuário e se adaptar (entregar uma experiência específica)	23	11			
Intenção de comportamento					
O protótipo apresentado serve como base para desenvolver uma aplicação real e completa	24	10			
Utilizaria Accorsi Airlines para comprar viagens aéreas	22	6	6		

entender o contexto do usuário e adaptar seu funcionamento para melhorar a experiência pessoal de cada usuário, porém comentaram que num caso de uso real, seria difícil atrair usuários apenas com esse diferencial em relação aos grandes websites de viagens aéreas, como Gol, Azul e Latam por exemplo.

5.5 Limitações

Este projeto propôs uma arquitetura de software para um sistema ciente de contexto, de forma que adapte seu funcionamento de maneira para cada usuário. Por mais que a proposta e o protótipo tenham sido capazes de atingir este objetivo, ficou claro que uma extração de contexto ótima necessita um sistema maior, com um volume expressivo de informações, tanto referente ao próprio usuário como também a forma com que ele utiliza o sistema. A segunda limitação é a falta de uma UI dedicada. Isso abriria portas para uma UX (experiência de usuário) realmente única e diferenciada, de forma que trabalhasse em conjunto com todo o backend proposto, entregando uma experiência totalmente diferente da comum quando se trata de sites de viagens aéreas. Para subsidiar tais limitações, novos estudos se mostram necessários, em especial no que tange UI/UX, uma vez que envolve conhecimentos para além de arquitetura de software e backend de sistemas, se mostrando um desafio relevante e grande para o futuro.

6 CONCLUSÃO E TRABALHOS FUTUROS

Este artigo apresentou Accorsi Airlines, um modelo arquitetural para um sistema de viagens aéreas ciente de contexto, baseado em microsserviços, capaz de se beneficiar de comunicação via eventos para funcionar plenamente. Tal arquitetura foi composta por 6 microsserviços,

que simplificam o funcionamento geral de um site de viagens aéreas. Somados à eles, foi desenvolvido um microsserviço especialmente para lidar com o contexto de cada usuário, a forma que ele utiliza o sistema e suas características pessoais, de forma que insights puderam ser tomados para modificar o comportamento do sistema exclusivamente para aquele usuário. Isso foi possível pois cada um dos microsserviços emitia eventos para as mais diversas ações, os quais eram armazenados pelo serviço de mensageria Kafka, posteriormente sendo consumidos pelo microsserviço de contextos, onde os insights aconteciam.

A arquitetura e o protótipo foram testados e validados por um total de 34 desenvolvedores de software, dos quais 29,4% representam desenvolvedores com menos de 2 anos de experiência, 41,2% com mais de 2 e menos de 6 anos de experiência, e 29,4% com mais de 6 anos de experiência. Baseado neste grupo, 88,23% tiveram facilidade para entender a arquitetura, e todos concordaram que o protótipo se mostrou satisfatório e capaz de extrair e processar corretamente dados contextuais em relação ao usuário.

Em adição, as entrevistas com os quinze participantes proporcionaram insights valiosos sobre o protótipo estudado. Ficou evidente que, embora tenham achado a aplicação relativamente fácil de usar, a ausência de uma interface de usuário clara foi uma barreira inicial, especialmente para os entrevistados com menos experiência em desenvolvimento. Além disso, embora tenham elogiado a funcionalidade de contextos, desejaram maior dinamicidade nas "reações" ao contexto do usuário. A capacidade do sistema de compreender e adaptar-se ao contexto do usuário foi destacada como seu principal benefício, mas surgiu a preocupação de que, em um cenário de uso real, atrair usuários apenas com essa característica pode ser desafiador diante da concorrência com os principais sites de viagens aéreas. Portanto, essas entrevistas forneceram informações cruciais para refinamentos e aprimoramentos futuros do protótipo. Em relação aos insights, seria interessante uma abordagem similar ao que foi proposto por (ZHELEV; ROZEVA, 2019), utilizando inteligência artificial para produzir insights. Já quanto à UI, um estudo à parte apenas de UI/UX poderia ser conduzido, introduzindo essa nova experiência ao sistema.

Referências

AROUK, O.; NIKAEIN, N. Kube5g : A cloud-native 5g service platform. 2020.

CABANE, H.; FARIAS, K. On the impact of event-driven architecture on performance: An exploratory study. **Future Generation Computer Systems**, Elsevier, 2023.

CASTRO, S. **5 Reasons Why MySQL Is Still The Go-To Database Management System**. 2021. Disponível em: https://www.jobsity.com/blog/5-reasons-why-mysql-is-still-the-go-to-database-management-system. Disponível em: <https://www.jobsity.com/blog/5-reasons-why-mysql-is-still-the-go-to-database-management-system>. Acesso em: 02 nov. 2023.

D'AVILA, L. F.; BARBOSA, J. L. V.; OLIVEIRA, K. S. F. de. Sw-context: a model to improve developers' situational awareness. p. 535–543, 2020.

DEY, A. K.; ABOWD, G. D.; SALBER, D. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications, human–computer interaction. 2001.

FIEGE, L.; MÜHL, G.; FREILING, F. Modular event-based systems. **The Knowledge Engineering Review**, v. 17, p. 359 – 388, 2002.

GOEL, D.; NAYAK, A. Reactive microservices in commodity resources. 2019.

GöRDESLI, M.; NASAB, A.; VAROL, A. Handling rollbacks with separated response control service for microservice architecture. 2022.

HENNING, S. ren; HASSELBRING, W. Benchmarking scalability of stream processing frameworks deployed as event-driven microservices in the cloud. 2023.

IT, S. **DB-Engines Ranking**. 2023. Disponível em: https://db-engines.com/en/ranking. Disponível em: <https://db-engines.com/en/ranking>. Acesso em: 02 nov. 2023.

JÚNIOR, E.; FARIAS, K. Modelgame: A quality model for gamified software modeling learning. In: **15th Brazilian Symposium on Software Components, Architectures, and Reuse**. [S.l.: s.n.], 2021. p. 100–109.

JURCA, G.; HELLMANN, T. D.; MAURER, F. Integrating agile and user-centered design. **2014 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC ST)**, 2014.

LAIGNER, R. et al. From a monolithic big data system to a microservices event-driven architecture. **2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)**, p. 213–220, 2020.

LAIGNER, R. et al. Data management in microservices: State of the practice, challenges, and research directions. 2021.

LEWIS, J.; FOWLER, M. **Microservices**. 2020. Disponível em: <https://martinfowler.com/articles/microservices.html>. Acesso em: 25 oct. 2023.

LI, R. et al. Understanding software architecture erosion: A systematic mapping study. **Journal of Software: Evolution and Process**, 2022.

MIRAZ, M. H.; ALI, M.; EXCELL, P. S. Adaptive user interfaces and universal usability through plasticity of user interface design. **Computer Science Review**, 2021.

ORTIZ, G. et al. Real-time context-aware microservice architecture for predictive analytics and smart decision-making. 2019.

RAHMATULLOH, A. et al. Event-driven architecture to improve performance and scalability in microservices-based systems. 2022.

RUBERT, M.; FARIAS, K. On the effects of continuous delivery on code quality: A case study in industry. **Computer Standards & Interfaces**, Elsevier, v. 81, p. 103588, 2022.

SHANI, G.; GUNAWARDANA, A. Evaluating recommendation systems. In: **Recommender Systems Handbook**. [S.l.]: Springer, Boston, MA, 2010. p. 257–297.

STEFANIDI, Z. et al. Real-time adaptation of context-aware intelligent user interfaces, for enhanced situational awareness. **2021 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC ST)**, 2021.

STOPFORD, B. **Designing Event-Driven Systems**. [S.l.]: O'Reilly Media, Incorporated, 2018.

TOTVS. **Arquitetura REST: Saiba o que é e seus diferenciais**. 2020. Disponível em: https://www.totvs.com/blog/developers/rest/. Disponível em: <https://www.totvs.com/blog/developers/rest/>. Acesso em: 23 mar. 2023.

TSIBIDIS, G.; ARVANITIS, T. N.; BABER, C. The what, who, where, when, why and how of context-awareness. 2008.

URDANGARIN, R. G.; FARIAS, K.; BARBOSA, J. Mon4aware: A multi-objective and context-aware approach to decompose monolithic applications. In: **XVII Brazilian Symposium on Information Systems**. [S.l.: s.n.], 2021. p. 1–9.

VIEIRA, R. D.; FARIAS, K. Usage of psychophysiological data as an improvement in the context of software engineering: A systematic mapping study. In: **XVI Brazilian Symposium on Information Systems**. [S.l.: s.n.], 2020. p. 1–8.

WOHLIN, C. et al. **Experimentation in software engineering**. [S.l.]: Springer Science & Business Media, 2012.

ZHELEV, S.; ROZEVA, A. Using microservices and event driven architecture for big data stream processing. 2019.