

**FACULDADE DE TECNOLOGIA E CIÊNCIAS SOCIAIS APLICADAS – FATECS
CURSO**

LEONARDO ANDRADE ROSIGNOLI PEREIRA
21709613

**DESENVOLVIMENTO DE UM ASSISTENTE PESSOAL E
GERENCIADOR DE CALENDÁRIOS EMPRESARIAL UTILIZANDO
SWIFT E JAVA**

BRASÍLIA
2023

LEONARDO ANDRADE ROSIGNOLI PEREIRA

**DESENVOLVIMENTO DE UM ASSISTENTE PESSOAL E
GERENCIADOR DE CALENDÁRIOS EMPRESARIAL UTILIZANDO
SWIFT E JAVA**

Trabalho de Conclusão de Curso (TCC) apresentado
como um dos requisitos para a conclusão do curso de
Engenharia de Computação do UniCEUB– Centro
Universitário de Brasília

Orientador (a): [Francisco Javier de Obaldia Diaz](#)

BRASÍLIA
2023

LEONARDO ANDRADE ROSIGNOLI PEREIRA

TEMA DO TRABALHO

Trabalho de Conclusão de Curso (TCC) apresentado como um dos requisitos para a conclusão do curso de Engenharia de Computação do UniCEUB – Centro Universitário de Brasília

Orientador (a): [Francisco Javier de Obaldia Diaz](#)

Brasília, 2023.

BANCA EXAMINADORA

Nome e titulação.
Orientador (a)

Nome e titulação.
Examinador (a)

Nome e titulação.
Examinador (a)

DESENVOLVIMENTO DE UM ASSISTENTE PESSOAL E GERENCIADOR
DE CALENDÁRIOS EMPRESARIAL UTILIZANDO SWIFT E JAVA.
DEVELOPMENT OF A PERSONAL ASSISTANT AND CORPORATE
CALENDAR MANAGER USING SWIFT AND JAVA.

Leonardo Andrade Rosignoli Pereira¹, Francisco Javier de Obaldia Díaz²

RESUMO

No mundo empresarial de hoje, gerenciar várias obrigações, compromissos e prazos é uma tarefa que pode exigir muito esforço e coordenação, esse desafio é ainda mais amplificado pela necessidade de navegar por várias aplicações de calendário, o que pode levar a uma agenda fragmentada e ineficiente. Pensando nisso, esse artigo apresenta o aplicativo 'Calendar Man', uma aplicação inovadora feita para simplificar a interação do usuário com seu calendário. Ele tem inteligência artificial e se integra facilmente com várias plataformas de calendário que já usamos, como o Calendário iOS e o Outlook, com o objetivo principal de criar uma ferramenta que torne mais fácil e eficiente o gerenciamento de eventos.

O processo de desenvolvimento deste projeto foi dividido em 2 sub-projetos, sendo eles desenvolvimento front-end e desenvolvimento back-end . O desenvolvimento front-end foi executado utilizando SwiftUI, enquanto o último foi desenvolvido usando aplicações Java/Spring, aderindo ao padrão de micro-serviços.

O artigo aprofunda-se nos desafios encontrados durante o processo de desenvolvimento, principalmente relacionados à integração total da IA com a aplicação. Ele também destaca que a aplicação está atualmente em seus estágios iniciais, indicando que a recepção do usuário e melhorias potenciais são áreas de consideração futura.

Palavras-chave: Desenvolvimento de Software. Inteligência Artificial. Gerenciamento de Calendários.

Abstract: In today's business world, managing multiple obligations, commitments, and deadlines can be a task that requires significant effort and coordination. This challenge is further amplified by the need to navigate various calendar applications, which can lead to a fragmented and inefficient schedule. With this in mind, this article introduces the 'Calendar Man' app, an innovative application designed to simplify user interaction with their calendar. It features artificial intelligence and easily integrates with various calendar platforms we already use, such as iOS Calendar and Outlook, with the primary goal of creating a tool that makes event management easier and more efficient.

The development process of this project was divided into two sub-projects: front-end development and back-end development. Front-end development was executed using SwiftUI, while the latter was developed using Java/Spring applications, adhering to the microservices pattern. The article delves into the challenges encountered during the development process, primarily related to the full integration of AI with the application. It also highlights that the application is currently in its early stages, indicating that user reception and potential improvements are areas for future consideration.

keywords: Software development. Artificial intelligence. Calendar Management.

¹ UniCEUB, Leonardo Andrade Rosignoli Pereira.

² UniCEUB, Francisco Javier de Obaldía Díaz.

³ UniCEUB, primeiro examinador.

⁴ UniCEUB, segundo examinador.

SUMÁRIO

1 INTRODUÇÃO	6
2 REVISÃO BIBLIOGRÁFICA	6
2.1 A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT	6
2.2 Prompt Programming for Large Language Models: Beyond the Few-Shot Paradigm	7
2.3 AWS Well-Architected Framework	7
2.4 Microservices - a definition of this new architectural term	8
3 METODOLOGIA DO TRABALHO	8
3.1. FRONT-END	8
3.1.1. PADRÕES DO PROJETO	8
3.1.2. AUTENTIFICAÇÃO	9
3.1.3 INTEGRAÇÃO COM PLATAFORMAS EXTERNAS	9
3.1.4. INTEGRAÇÃO COM CHAT BOT	10
3.2.1. METODOLOGIA DE DESENVOLVIMENTO	10
3.2.2. MICRO SERVIÇO CALENDAR-EVENT-MANAGER	11
3.2.3. MICRO SERVIÇO CHAT-WORKFLOW-MANAGER	12
3.3. INFRAESTRUTURA	14
3.3.1. HOSPEDAGEM DOS MICRO SERVIÇOS(Elastic Compute Cloud)	14
3.3.2. DEPLOY DOS MICRO SERVIÇOS	14
3.4. BANCO DE DADOS	15
4. APRESENTAÇÃO E ANÁLISE DOS RESULTADOS	16
5. CONSIDERAÇÕES FINAIS	21

1 INTRODUÇÃO

Com o aumento constante de responsabilidades e atividades em nosso cotidiano, o gerenciamento de tempo se tornou uma parte crucial na vida das pessoas. E para otimizar o tempo de seus funcionários, empresas de grande porte começaram a utilizar ferramentas de calendários para organizar reuniões entre seus colaboradores. No entanto, com o avanço recente na área de Inteligência artificial, surge uma lacuna nessa esfera tecnológica do qual o aplicativo apresentado neste artigo, 'Calendar Man', se propõe a preencher.

Este estudo apresenta então o desenvolvimento de 'Calendar-Man', um assistente de gerenciamento de calendário inovador que emprega a Inteligência Artificial para permitir a interação do usuário com seu próprio calendário em linguagem natural e oferecer sugestões otimizadas para o agendamento de eventos, sejam eles pessoais ou profissionais.

A principal diferença de 'Calendar Man' para os aplicativos de gerenciamento de calendário já existentes é sua IA integrada. O usuário é capaz de interagir com a IA em linguagem natural, fazendo com que a sua experiência seja mais intuitiva. Além disso, a IA é equipada para analisar o calendário do usuário e sugerir horários para próximos eventos, uma funcionalidade que faz com que o usuário possa apenas dar uma ideia geral de quando quer marcar um evento e receber uma proposta personalizada sobre qual o melhor horário.

Outro diferencial deste projeto é a capacidade do aplicativo de se integrar com calendários mais populares atualmente em uso como o *Outlook* e *iOS Calendar*. Isso facilita a mudança de plataforma do usuário, sendo que com apenas alguns cliques ele consegue importar todos seus calendários de diferentes fontes e migrar para essa nova aplicação.

Em resumo, este artigo busca explorar o desenvolvimento e as funcionalidades do

aplicativo 'Calendar Man', evidenciando a aplicação da IA em assistentes de gerenciamento de calendário e consolidando sua relevância no campo dos aplicativos de tecnologia.

2 REVISÃO BIBLIOGRÁFICA

2.1 *A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT*

Este estudo demonstra formas de melhorar a interação com modelos de linguagem como o *Chat GPT* através de frases preparadas, uma estratégia que foi utilizada no desenvolvimento desta aplicação. Usando essas orientações, foram criados "padrões de prompts", que no contexto deste aplicativo são instruções que ajudam a guiar a inteligência artificial na gestão do calendário do usuário.

Esses padrões de prompts não só podem ser empregados para uma variedade de funções, mas também podem ser aplicados em diversos contextos. Nesse contexto, exemplos relevantes de padrões de prompts incluem "*The Meta Language Creation Pattern*" para adaptar a linguagem ao back-end, fazendo com que a resposta sempre venha em um formato especializado, o "*The Question Refinement Pattern*" para fazer com que a inteligência artificial faça mais perguntas ao usuário e marque eventos mais certos.

O objetivo desses padrões de prompts é facilitar e tornar mais eficiente a interação do usuário com a inteligência artificial. Eles fornecem um caminho para formular perguntas de forma a obter as respostas mais úteis e relevantes.

Ao analisar os resultados desta pesquisa, pôde-se chegar à conclusão de que os padrões de prompts podem melhorar significativamente a forma como interagimos com os modelos de linguagem. No entanto, com o avanço constante que está acontecendo na área, é necessário atualizar e aprimorar continuamente esses padrões para

acompanhar as melhorias.

2.2 Prompt Programming for Large Language Models: Beyond the Few-Shot Paradigm

Este artigo examina o desempenho do GPT-3 usando diferentes tipos de prompts. Verificou-se que o GPT-3 tem melhor desempenho com mais exemplos, sugerindo que esses exemplos ajudam a instruir a tarefa a ser feita e reforça os padrões dados. A diferença em eficácia dos prompts com ou sem exemplos foi demonstrada em uma tarefa de tradução. Os resultados mostraram que os prompts sem exemplos podem igualar e até superar o desempenho padrão de prompts com poucos exemplos.

O artigo também discute as limitações dos exemplos mostrando que ao invés de melhorar a performance, eles podem contaminar o objetivo principal e acabar piorando o desempenho. Enfim, aborda o uso de "metaprompts", que são prompts gerais que quando combinados com informações adicionais se transformam em prompts específicos.

2.3 AWS Well-Architected Framework

O artigo 'AWS Well-Architected Framework', da Amazon, orienta na criação de sistemas seguros, eficientes e economicamente viáveis. Ele segmenta o tema em cinco pilares: Excelência Operacional, Segurança, Confiabilidade, Eficiência de Performance e Otimização de Custo. (*Amazon Web Services, AWS Well-Architected Framework. 2023.*)

Excelência Operacional se divide nos seguintes princípios:
configurações de ambiente em código, incrementar com mudanças pequenas, frequentes e reversíveis, sempre refinar os procedimentos de operação, antecipar a falha. (e se preparar) e aprender através de erros operacionais.

Segurança:

- Implementar uma forte base de identidade (princípio do privilégio mínimo)
- Rastreabilidade: Monitorar, alertar e auditar ações ocorridas dentro do aplicativo.
- Segurança em todas as camadas da aplicação
- Automatizar melhores práticas de segurança
- Criptografar dados (em trânsito e guardados)
- Remover acesso direto aos dados.
- Preparar-se para caso de falha na segurança.

Confiabilidade: Aqui, o artigo detalha como um sistema deve recuperar-se de problemas, como inconsistências de configuração ou falhas de rede. Os pontos principais são:

Recuperação automática de falhas, testar os procedimentos de recuperação, escalar horizontalmente para aumentar a disponibilidade agregada da carga de trabalho. *e.g. substitua um recurso grande por vários recursos pequenos para reduzir o impacto de uma única falha na workload geral*, parar de tentar adivinhar a capacidade, fazer alterações na infraestrutura por meio de automações.

Eficiência de Performance: Esta seção trata da utilização eficiente dos recursos computacionais, evitando desperdícios. Pontos principais:

facilitar a implementação de tecnologia, delegando tarefas complexas ao seu fornecedor de nuvem, tornar-se global em minutos, usar arquiteturas sem servidor, experimentar com mais frequência, entender como os serviços de nuvem são consumidos e usar sempre a abordagem tecnológica mais alinhada às suas metas.

Otimização de Custo: Finalmente, o artigo discute como entregar valor ao negócio de forma econômica, incluindo a compreensão e controle dos gastos. Princípios:

Investir em gerenciamento financeiro na nuvem/otimização de custos, adotar um modelo de consumo, avaliar eficiência geral e por último, analisar e atribuir despesas.

Em resumo, este artigo foi considerado como um guia para projetar e gerenciar os micro serviços usados nesta aplicação, que são disponibilizados através da plataforma de nuvem da AWS.

2.4 *Microservices - a definition of this new architectural term*

No artigo "*Microservices*" ,Martin Fowler e James Lewis definem micro serviços como um jeito de desenvolver programas que dividem a aplicação em partes pequenas chamadas de serviços. Esses serviços ajudam as empresas a melhorar e acelerar seus sistemas e também são mais baratos para alterar, já que são desacoplados uns aos outros.

Eles compararam isso com a arquitetura monolítica, onde tudo é estruturado em uma única parte e chegam a conclusão que microsserviços são mais fáceis de mudar e consertar. Eles falaram sobre características dos micro serviços, como serem organizados em volta das habilidades do negócio, ter endpoints inteligentes e pipelines simples, e se preparar para falhas.

Mas usar micro serviços tem suas dificuldades. Já que são necessários monitoramento e implantação rápida de aplicativos. Além de que o gerenciamento de micro serviços pode se tornar algo complexo ao depender do tamanho geral da aplicação.

Com exemplos fáceis de entender, o artigo é bom para quem quer aprender sobre microsserviços. James Lewis e Martin Fowler mostram os dois lados. Assim, o texto de James Lewis e Martin Fowler se destaca como uma leitura essencial para quem deseja implementar este tipo de arquitetura de software, apresentando uma visão equilibrada e completa da arquitetura de microsserviços.

3 METODOLOGIA DO TRABALHO

Tem-se como objetivo deste projeto, desenvolver um sistema aplicativo funcional,

o qual foi dividido nas seguintes etapas com o propósito de aplicar melhores práticas no ramo de engenharia de software e arquitetura.

3.1. FRONT-END

Para a parte visual desta aplicação (*front-end*), foi utilizada a linguagem de programação *Swift*, junto a seu próprio *framework* de ferramentas, *SwiftUI*.

3.1.1. PADRÕES DO PROJETO

O projeto foi desenvolvido em um repositório open-source, no qual novas funcionalidades foram desenvolvidas em branches separadas, e quando prontas, margeadas novamente a branch principal.

Para manter a organização do projeto dito que a quantidade de linhas de código e complexidade geral aumenta com o tempo de projeto, foi feita uma estrutura organizacional de pacotes da seguinte maneira:

- *services*: Contém toda a parte do código que lida com chamadas externas ao back-end, integrações e autenticação.
- *utils*: Contém métodos e classes utilitários gerais utilizados pela aplicação, por exemplo, contém uma classe *DateUtils*, que é utilizada apenas para converter formato de data para o utilizado internamente.
- *components*: Utilizado para componentes visuais utilizados pela aplicação como Botões, ou animações.
- *views*: Esse é o pacote principal, no qual ficam as páginas da aplicação, onde são usados os componentes e os clientes, cada "view" representa uma página diferente na aplicação.
- *resources*: Contém todos os recursos da aplicação, como as constantes utilizadas e os segredos (que não são enviados ao repositório).
- *models*: Essa pasta tem os modelos que são necessários para fazer as chamadas para as API's externas.

Além desses pacotes já descritos, projetos em *Swift* também contêm uma estrutura padrão chamada "*Assets*", no qual são guardados os recursos visuais da aplicação, como cores do projeto, logo da aplicação e imagens utilizadas no programa.

3.1.2. AUTENTIFICAÇÃO

Como sistema de providenciamento de autenticação, foi tomada uma decisão estratégica de utilizar um provedor externo ao invés de desenvolver uma própria solução, neste caso foi utilizada a plataforma Auth0, devido a sua facilidade de integração com a linguagem utilizada no desenvolvimento e seu baixo custo. Essa decisão foi tomada através de vários fatores que serão justificados a seguir.

Primeiramente e principalmente, foi reconhecida a complexidade e consumo de tempo que levaria para desenvolver um sistema próprio. Para desenvolver um sistema seguro, confiante e eficiente seria necessário um banco de dados que mantivesse informações criptografadas e toda uma infraestrutura a suportando, com mais micro-serviços no back-end especializados para esse sistema. Agora, ao optar pela utilização da plataforma *Auth0*, pode-se evitar esses problemas e focar diretamente nas funcionalidades da aplicação.

Segundo, essa plataforma também oferece escalabilidade e flexibilidade. Enquanto a aplicação cresce, o processo de autenticação pode precisar mudar e esta plataforma oferece maneiras simples de planejar tais mudanças. Essa funcionalidade levaria muito tempo a ser planejada em uma solução personalizada.

E por último, a plataforma providencia documentação simples e suporte. O que significa que caso dificuldades surgissem, poderia-se obter ajuda através de documentações técnicas.

Em conclusão, a decisão de utilizar um provedor externo foi tomada com o objetivo de garantir segurança, economizar recursos e manter a flexibilidade para escalonamento da

aplicação. Ao terceirizar o esse processo, foi possível concentrar em entregar as funcionalidades que fazem deste aplicativo único.

3.1.3 INTEGRAÇÃO COM PLATAFORMAS EXTERNAS

Como dito anteriormente, essa aplicação se conecta com outros calendários. Porém, cada plataforma possui seus modelos e regras diferentes, e para fazer com que todos funcionem juntos, os eventos de cada um são simplificados em um único modelo usado por toda a aplicação, esse modelo contém a seguinte estrutura.

Figura 1. Estrutura dos eventos no front-end.

```
struct Event: Codable, Identifiable {  
    let id: String  
    let title: String  
    let startDate: String  
    let endDate: String  
}
```

Fonte: Autor

No qual o *id* é um identificador único, usado para evitar duplicações, o Título e datas de começo e fim. Além disso, também é enviado um identificador único do próprio usuário que o vincula a seus eventos às requisições ao *back-end*.

Para fazer cada tipo de integração, houve uma abordagem necessária diferente, primeiro integrado foi o calendário *iOS*, o mais simples de todos dito que a linguagem de programação utilizada foi inventada para interagir diretamente com este sistema. Com a ajuda do *framework EventKit*, foi necessário simplesmente pedir autorização do usuário para acesso a seu calendário e em seguida, buscar os dados e modelá-los de acordo com as necessidades da aplicação.

A próxima plataforma integrada foi a do *Outlook*, no qual o processo foi um pouco mais complexo. Para fazer essa integração, foram feitos os seguintes passos.

1. Criar uma conta no *Azure Entra* e registrar o aplicativo.

2. Adicionar permissões de leitura de calendário de usuários no *Azure*.
3. Instalar a biblioteca de autenticação da microsoft (*MSAL*) na aplicação.
4. Definir URL do aplicativo. Quando os usuários se autenticam, o *Azure AD B2C* envia um código de autorização para o aplicativo usando o *URI* de redirecionamento configurado no registro do aplicativo.
5. O usuário se autentica e dá permissão de acesso de seu calendário à aplicação.

Após essas etapas serem concluídas, pode-se interagir com o calendário através do *Azure GRAPH*, um *gateway* de dados da *Microsoft* que providencia maneiras simples de interação com seus serviços, neste caso o calendário.

3.1.4. INTEGRAÇÃO COM *CHAT BOT*

Esta aplicação contém uma funcionalidade de *ChatBot*, na qual o usuário consegue através de linguagem natural marcar novos eventos em seu calendário.

Para o *front-end* foi feita a seguinte tela.

Figura 2. Demonstração de conversa com *chatbot*.



Fonte: Autor

Na qual as mensagens que o usuário envia ficam em azul e as do sistema ficam em amarelo. (a lógica por trás de tal funcionalidade é explicada na seção 3.2.3.)

3.2. BACK-END

3.2.1. METODOLOGIA DE DESENVOLVIMENTO

Para o *back-end*, foi definido que seria utilizada a arquitetura em micro-serviços, devido a sua capacidade de escalonamento, em que cada micro-serviço tem uma única função especializada.

Os micro-serviços aqui apresentados foram desenvolvidos com Java 17 e o *framework Spring*.

Para manter a qualidade do código, foram utilizados alguns conceitos e ferramentas, uma delas foi o *sonarQube*, uma plataforma de código aberto amplamente utilizada para realizar análises de código estáticas e contínuas, com o objetivo de identificar problemas de qualidade, manutenibilidade e segurança no código-fonte de um projeto. As métricas utilizadas nestes projetos foram as seguintes, no qual assim que identificado o problema, era corrigido em sequência:

- Bugs: Erros no código que podem causar falhas ou comportamento incorreto.
- Vulnerabilidades: Problemas de segurança que podem ser explorados por invasores.
- Code Smells: Problemas no código que podem indicar más práticas de programação ou complexidade desnecessária.
- Duplicações: Partes do código que são duplicadas e podem ser refatoradas para melhorar a manutenção.

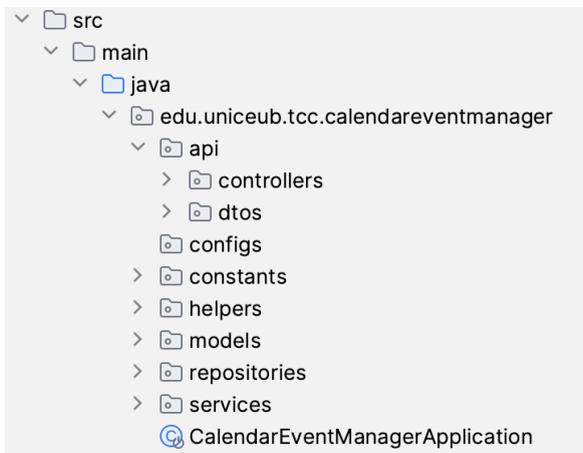
3.2.2. MICRO SERVIÇO *CALENDAR-EVENT-MANAGER*

Esse é o micro-serviço principal desta aplicação, responsável por gerenciar o calendário do usuário. É utilizado toda vez

que um usuário precisa interagir com seu calendário através de chamadas de API. Também é utilizado pelo assistente virtual para obter informações a respeito do calendário do usuário.

Para a estrutura deste projeto, foi usado o modelo MVC. O padrão MVC (*Model-View-Controller*) é um design pattern amplamente utilizado na arquitetura de aplicações web, que divide a aplicação em três componentes principais: Model, View e Controller. (*Anshul vyas, MVC Pattern, 2018.*)

Figura 3. Estrutura de pacotes para o projeto em back-end.



Fonte: Autor

Model: O *Model* representa a camada de negócios e os dados da aplicação. É responsável por armazenar e gerenciar os dados, assim como executar as regras e lógicas específicas do sistema. Neste contexto é representado através dos pacotes *model* e *repositories*, que são responsáveis por manter os modelos dos dados que vão para o banco de dados e por interagir com o mesmo através de requisições.

View: A *View* é a camada de apresentação da aplicação que lida com a interação do usuário. Ela exibe os dados do *Model* e coleta as entradas do usuário. No *Spring*, as views geralmente são desenvolvidas utilizando tecnologias como *JSP* (*Java Server Pages*), *Thymeleaf*, ou mesmo integração com frameworks modernos de *front-end*. Já no caso dessa aplicação, é representado pelo *front-end* em *Swift*.

Controller: É o componente intermediário que lida com a comunicação entre a *Model* e

a *View*. Ele processa as solicitações do usuário capturadas pela *View*, interage com o *Model* para obter ou atualizar dados e, em seguida, retorna a resposta adequada para a *View*. No *Spring*, as classes *Controller* são reconhecidas por classes com as meta anotações *@Controller* ou *@RestController* e os métodos são mapeados para endpoints HTTP usando anotações como *@GetMapping* ou *@PostMapping*. Neste projeto, tais classes estão dentro do pacote 'controllers'.

Junto do pacote de controllers, encontra-se outro, chamado 'dtos' (*Data Transfer Object*) que simplesmente serve como um padrão de design usado para transferir dados entre diferentes camadas de uma aplicação, neste caso, entre a camada *Controller* e a camada de serviço.

Além desses pacotes, temos alguns outros que são apenas utilitários, utilizados em várias as camadas da aplicação. Mas outro pacote muito importante é o 'service', responsável por conter a lógica de negócios da aplicação e intermediar a comunicação entre a camada *Controller* e a camada de repositório. Suas principais funções são:

1. Encapsular a lógica de negócios: A camada de serviço contém as regras e processos específicos da aplicação e garante a correta execução dessas operações antes de interagir com os dados.
2. Orquestrar as chamadas de repositório: A camada de serviço é responsável por coordenar as interações com a camada de acesso a dados (repositório), garantindo a correta recuperação, manipulação e persistência dos dados.
3. Facilitar a reutilização e a manutenção: A camada de serviço permite o reaproveitamento de código ao abstrair comportamentos comuns, facilitando a manutenção e a escalabilidade do projeto.

3.2.3. MICRO SERVIÇO CHAT-WORKFLOW-MANAGER

A aplicação proporciona ao usuário a possibilidade de conversar com um assistente virtual, e através de linguagem natural, checar e marcar novos eventos em seu calendário. Por esse motivo, foi criado este novo micro-serviço, separando sua responsabilidade para ser apenas lidar com essas conversas.

Esta aplicação adere aos mesmos padrões arquitetônicos mencionados anteriormente, embora incorpore alguns conceitos novos. Para possibilitar a integração do assistente virtual com nosso *back-end*, foi imprescindível desenvolver métodos especializados. Estes funcionam como um portal de acesso para o assistente virtual e são passados como parâmetros durante as conversas. Isso permite que o modelo de linguagem entenda suas capacidades dentro do contexto da aplicação.

Segue um exemplo ilustrando os logs da aplicação (simplificados para melhor visualização). Estes foram gerados ao realizar as solicitações apresentadas na Figura 2, que demonstra uma interação com o *Chatbot*:

```
1. Sending message to OpenAI:
ChatMessage(role=user, content=Olá! )
Received response from OpenAI:
2. ChatMessage(role=assistant, content=Olá! Como posso ajudar você hoje?)
3. Sending message to OpenAI:
ChatMessage(role=user, content=Marque 2 reuniões de 30 minutos começando às 2 da tarde amanhã, com 20 minutos de distância entre elas. )
4. Received response from OpenAI:
ChatMessage(role=assistant,
functionCall=ChatFunctionCall(name=get_current_date, arguments={}))
5. Assistant action required. Executing assistant action function get_current_date.
6. Sending message to OpenAI:
ChatMessage(role=function, content={
"currentDate" : "2023-11-08T18:16:17.081670006"
}, name=get_current_date, functionCall=null)
7. Received response from OpenAI:
ChatMessage(role=assistant,
functionCall=ChatFunctionCall(name=schedule_ev
```

```
ent, arguments={"title":"Reunião 1","startDate":"2023-11-09T14:00:00","endDate":"2023-11-09T14:30:00"}))
```

Assistant action required. Executing assistant action function schedule_event.

Sending message to OpenAI:

```
ChatMessage(role=function, content=null,
name=schedule_event, functionCall=null)
```

Received response from OpenAI:

```
ChatMessage(role=assistant,
functionCall=ChatFunctionCall(name=schedule_event, arguments={"title":"Reunião 2","startDate":"2023-11-09T14:50:00","endDate":"2023-11-09T15:20:00"}))
```

Assistant action required. Executing assistant action function schedule_event.

8. *Sending message to OpenAI:*

```
ChatMessage(role=function, content=null,
name=schedule_event, functionCall=null)
```

Received response from OpenAI:

```
ChatMessage(role=assistant, content=As duas reuniões foram agendadas com sucesso. Segue abaixo os detalhes:
```

1. Reunião 1:

- *Data: 09 de novembro de 2023*

- *Horário de início: 14:00*

- *Horário de término: 14:30*

2. Reunião 2:

- *Data: 09 de novembro de 2023*

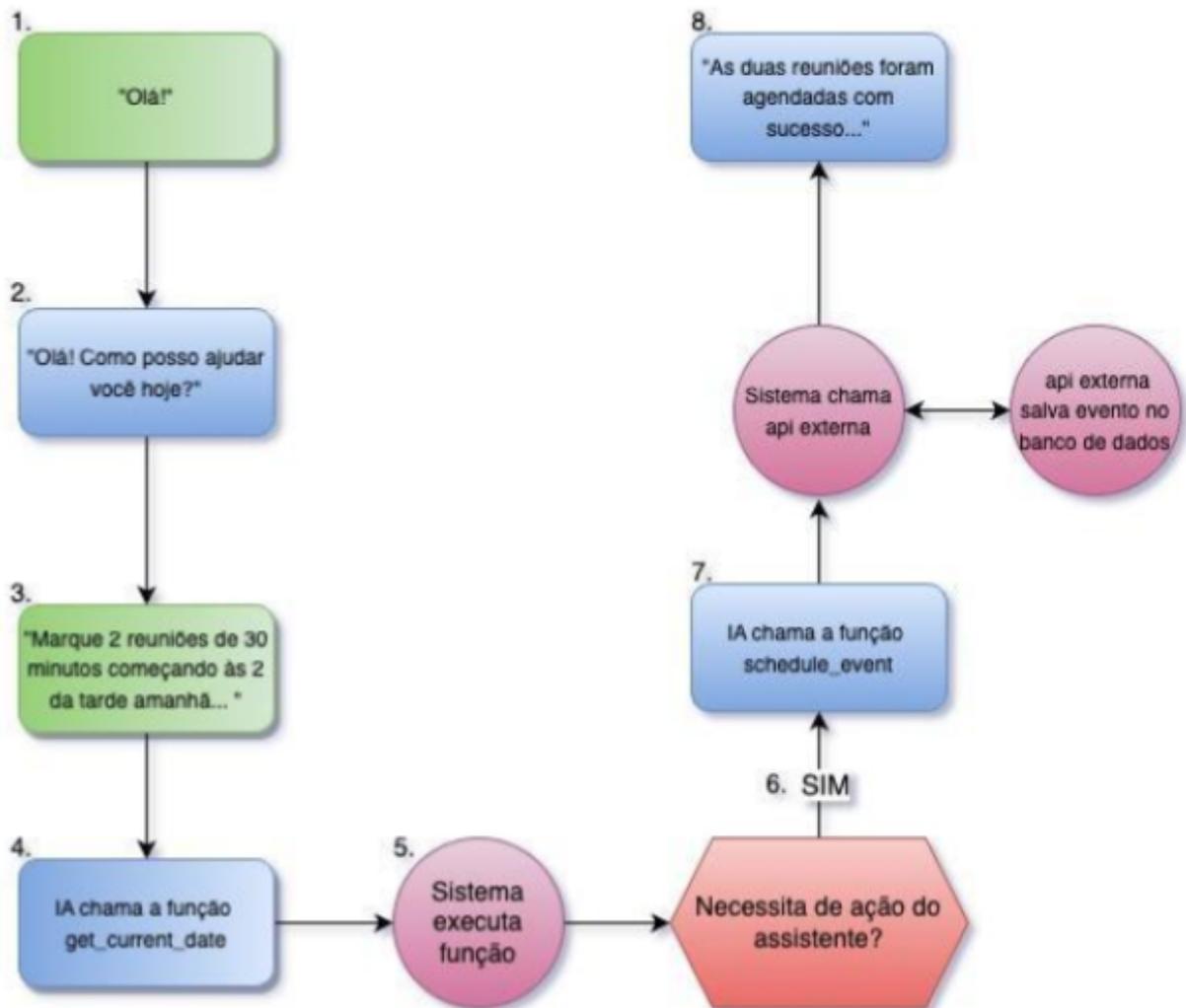
- *Horário de início: 14:50*

- *Horário de término: 15:20*

Certifique-se de estar disponível no horário determinado para cada uma das reuniões.)

Com esses logs da aplicação, pode-se ser ilustrado da seguinte forma:

Figura 04. Diagrama de fluxo do exemplo demonstrado



Fonte: Autor

1. O usuário manda mensagem dizendo olá;
2. O assistente virtual responde;
3. O usuário faz sua requisição, pedindo para marcar suas reuniões;
4. Ao perceber que o usuário fez uma solicitação, ele começa a interagir com o *back-end*, chamando as funções especializadas, feitas para que ele consiga interagir com o sistema da aplicação. A primeira chamada que ele faz é para perguntar qual o dia de hoje, com a função "get_current_date".

Nota: Durante a fase de desenvolvimento, observou-se que o *ChatGPT* ocasionalmente apresentava dificuldades em compreender

corretamente conceitos temporais, como "hoje" ou "amanhã". Tal incompatibilidade resultava em marcações de eventos em datas imprecisas. Para mitigar esse problema, implementou-se uma instrução inicial no sistema que orientava: "*Sempre assuma que você desconhece a data atual e solicite-a*".

Após a introdução desta instrução, o *Chat GPT* demonstrou uma compreensão aprimorada do contexto temporal, reduzindo significativamente os erros na marcação de eventos.

5. O sistema detecta que o assistente virtual está realizando uma atividade que requer uma ação do assistente, conhecida como chamada de função no código. Essa chamada é definida

como uma lista de ações que determinam se o próximo passo a ser executado vem da ação do usuário ou do próprio assistente.

6. Então após executar a função solicitada, sua resposta é reenviada ao sistema com a data atual.
7. Com a informação da data atual, o assistente pode fazer uma nova requisição para marcar os eventos de acordo com as solicitações do usuário. Essa chamada é enviada para a API "*calendar-event-manager*", responsável por salvar os eventos no banco de dados.
8. A resposta da API é devolvida e interpretada pelo assistente, que então notifica o usuário de seus eventos marcados.

3.3. INFRAESTRUTURA

A infraestrutura desta aplicação foi toda feita na rede *AWS*, devido aos seguintes motivos.

- Primeiramente, a familiaridade com a plataforma foi um fator decisivo, dado que o responsável pelo projeto já possuía experiência prévia com a mesma.
- Em segundo lugar, a *AWS* oferece uma ótima estrutura de custos, que inclui "*tiers*" de serviços gratuitos e uma política de preços transparente. Isso permitiu a implantação de toda a infraestrutura necessária sem incorrer em custos excessivos. A tabela a seguir apresenta uma visão geral dos gastos na plataforma durante o período de desenvolvimento do projeto (valores em USD).

Tabela 1. Despesas incorridas ao utilizar a plataforma AWS.

Mês	Custo (USD)
Setembro	0.35\$
Outubro	0.06\$
Novembro	0.03\$ (ESTIMADO)

Fonte: Autor

- **Confiabilidade:** Por fim, a confiabilidade da *AWS* foi outro fator importante na decisão. A plataforma tem um histórico comprovado de fornecimento de serviços confiáveis, com um Acordo de Nível de Serviço (SLA) garantindo um tempo de atividade de 99,99% para a maioria dos serviços. Isso assegura que o projeto esteja sempre disponível para os usuários, minimizando o tempo de inatividade e melhorando a experiência do usuário (*Amazon Compute Service Level Agreement, 2022*).

3.3.1. HOSPEDAGEM DOS MICRO SERVIÇOS(*Elastic Compute Cloud*)

O *Amazon Elastic Compute Cloud (EC2)* é um componente central dos serviços de computação oferecidos pela *Amazon Web Services (AWS)*. Trata-se de um serviço que proporciona capacidade computacional escalável na nuvem, eliminando a necessidade de investir em hardware e reduzindo o tempo necessário para obter e iniciar novas instâncias de servidor para minutos. O *EC2* permite aos usuários alugar máquinas virtuais, conhecidas como instâncias, onde podem rodar suas próprias aplicações.

3.3.2. DEPLOY DOS MICRO SERVIÇOS

O processo para fazer o deploy da aplicação a essa instância foi feito com as seguintes etapas:

- **Modificação da Aplicação:** A primeira etapa envolve quando há uma alteração no código fonte do micro-serviço, seja para adição de novas funcionalidades, correção de bugs, ou a realização de melhorias no desempenho ou na segurança da aplicação.

- **Compilação do Código:** Após a realização das alterações necessárias, o código da aplicação deve ser compilado. Isso é feito para verificar se o código é sintaticamente correto e se pode ser convertido em código de máquina para execução. No caso dessas aplicações, foi utilizado o gerenciador de pacotes Maven.
- **Geração do Arquivo .jar:** Uma vez que o código tenha sido compilado com sucesso, o próximo passo é gerar um arquivo *JAR (Java ARchive)*. Este

que continua a funcionar mesmo quando a conexão *SSH* é encerrada. Dentro da sessão "screen", o comando "*java -jar*" é usado para iniciar a aplicação, apontando para o arquivo *JAR* que foi transferido para a instância.

Dito isso, ambas as aplicações foram hospedadas em apenas 1 instância *EC2*, para que houvesse diminuição de custos e também por que mesmo com ambas aplicações na mesma máquina, os recursos do container utilizados são mínimos, com

Figura 05: Utilização do container das aplicações (08/11/23 - 15/11/23)



Fonte: Autor

arquivo contém todas as classes compiladas, juntamente com seus recursos associados, que são necessários para executar a aplicação.

- **Conexão ao EC2 através de SSH:** A instância *EC2*, na qual a aplicação será implantada, é acessada através do protocolo *SSH (Secure Shell)*. Este protocolo é utilizado para estabelecer uma conexão segura com a instância *EC2*, permitindo a transferência segura do arquivo *JAR* para a instância.
- **Execução da Aplicação através do Comando "screen":** Finalmente, a aplicação é executada na instância *EC2*. Isso é feito utilizando o comando "screen", que permite a execução da aplicação em uma sessão

cerca de 15 de uso. (Figura 05)

3.4. BANCO DE DADOS

Para o banco de dados, foi decidido que seria utilizado o banco *AtlasDB*, desenvolvido pelos criadores do *MongoDB*, que é um serviço de banco de dados multi-cloud versátil projetado para simplificar a implantação e o gerenciamento de bancos de dados em vários provedores de nuvem. Este serviço oferece aos usuários a flexibilidade para selecionar sua infraestrutura de nuvem, que neste caso foi a *AWS*.

Alguns dos benefícios específicos oferecidos por esse serviço são descritos a seguir, com exemplos de como tais recursos foram usados.

Simplificação do Gerenciamento de Banco de Dados: O MongoDB Atlas simplifica a implantação, configuração e gerenciamento de bancos de dados, reduzindo assim a carga administrativa e permitindo que fosse possível se concentrar apenas no processo de desenvolvimento do aplicativo, sem que fosse necessário perder tempo com configuração de ambiente.

Escalabilidade: O *MongoDB Atlas* oferece escalabilidade automática, que pode ser inestimável para projetos com necessidades de dados variáveis ou em rápido crescimento.

Segurança: O *MongoDB Atlas* incorpora robustas medidas de segurança, incluindo criptografia de ponta a ponta, isolamento de rede e controles de acesso baseados em funções. Essas características podem ajudar a garantir a integridade e confidencialidade dos dados. Neste projeto, como o banco de dados é acessado apenas por uma API (*calendar-event-manager*), qualquer requisição que venha por outro endereço de IP que não seja o da máquina da API pré-definida ou do desenvolvedor, é bloqueada (Figura 06). O IP da máquina da aplicação é representado pela descrição *EC2*.

banco de dados localize dados sem ter que escanear cada documento em uma coleção, melhorando assim os tempos de resposta das consultas e reduzindo a carga no sistema.

No contexto do projeto em questão, índices foram criados para ordenar pelos campos *StartDate* e *EndDate* na coleção de eventos dos usuários. Isso significa que, para cada documento na coleção, os valores de *startDate* e *endDate* são armazenados em uma entrada de índice. Essa estrutura permite que o *MongoDB* percorra rapidamente o índice e encontre os documentos que correspondem a uma consulta nesses campos, melhorando significativamente o desempenho de consultas que ordenam ou filtram por *startDate* e *endDate*.

Este *index* foi criado por ser sempre utilizado, para buscar por eventos de um dia específico, ou pelo próprio *chatbot* para buscar informações do calendário do usuário em um certo intervalo de tempo.

Também é importante notar que, embora os índices proporcionem benefícios substanciais em termos de desempenho de operações de leitura, eles têm certos custos. Índices podem consumir quantidades significativas de espaço de armazenamento,

Figura 06. Lista de acessos permitidos pelo Atlas DB.

Network Access

IP Access List Peering Private Endpoint

[+ ADD IP ADDRESS](#)

You will only be able to connect to your cluster from the following list of IP Addresses:

IP Address	Comment	Status	Actions
[REDACTED]	EC2	Active	EDIT DELETE
[REDACTED]	Home IP address	Active	EDIT DELETE
[REDACTED]	College IP Address	Active	EDIT DELETE

Fonte: Autor

Índices: No *MongoDB*, os índices são um recurso crítico que aprimora o desempenho de consultas ao banco de dados, fornecendo caminhos eficientes para os dados. Eles funcionam de forma semelhante ao índice de um livro, permitindo que o mecanismo de

e também podem adicionar sobrecarga às operações de gravação, pois cada operação de inserção ou atualização também deve atualizar os índices associados. Por este motivo, foram criados apenas esses índices já citados.

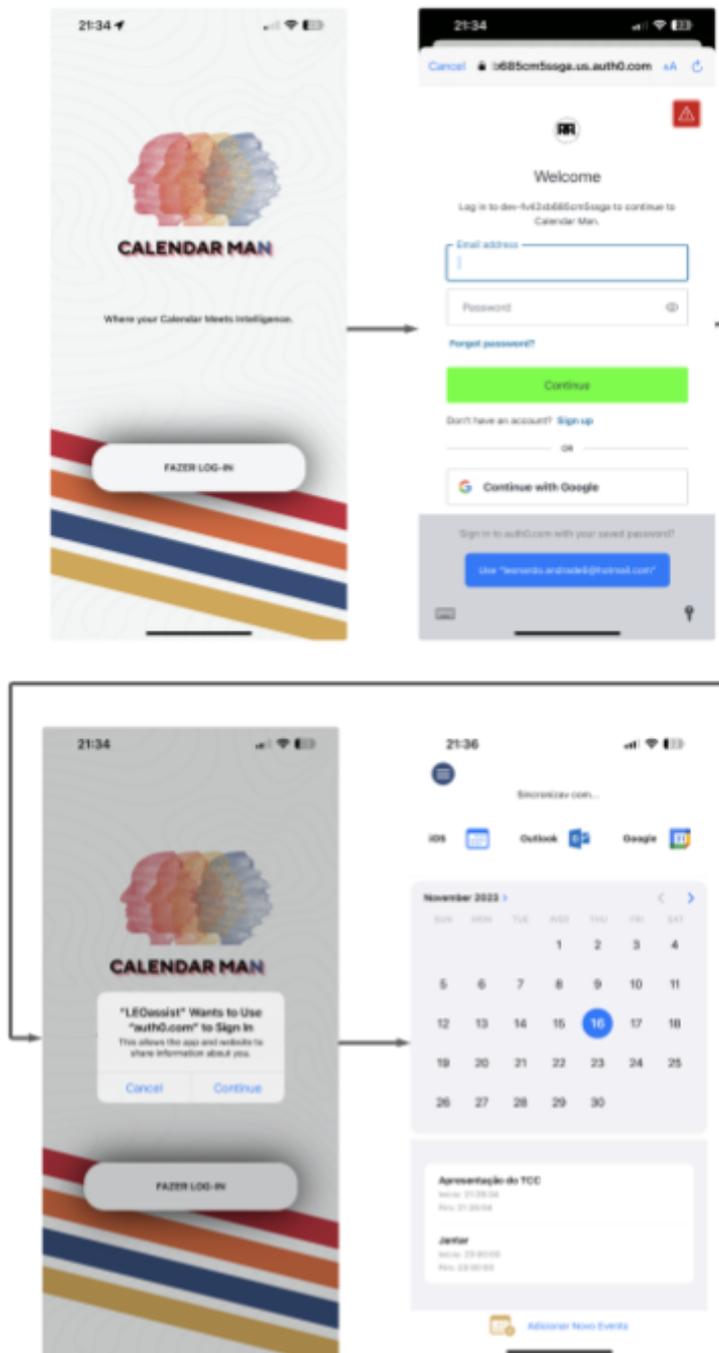
4. APRESENTAÇÃO E ANÁLISE DOS RESULTADOS

O resultado final deste artigo é demonstrado através de sequências de figuras que evidenciam a funcionalidade da aplicação.

A Figura 07 a seguir demonstra como é

feito o processo de login, no qual o usuário é redirecionado ao site de autenticação e após ser concluída com sucesso, volta à página inicial da aplicação, a do seu calendário, em que o usuário pode sincronizar seus calendários.

Figura 07: Fluxo de login da aplicação

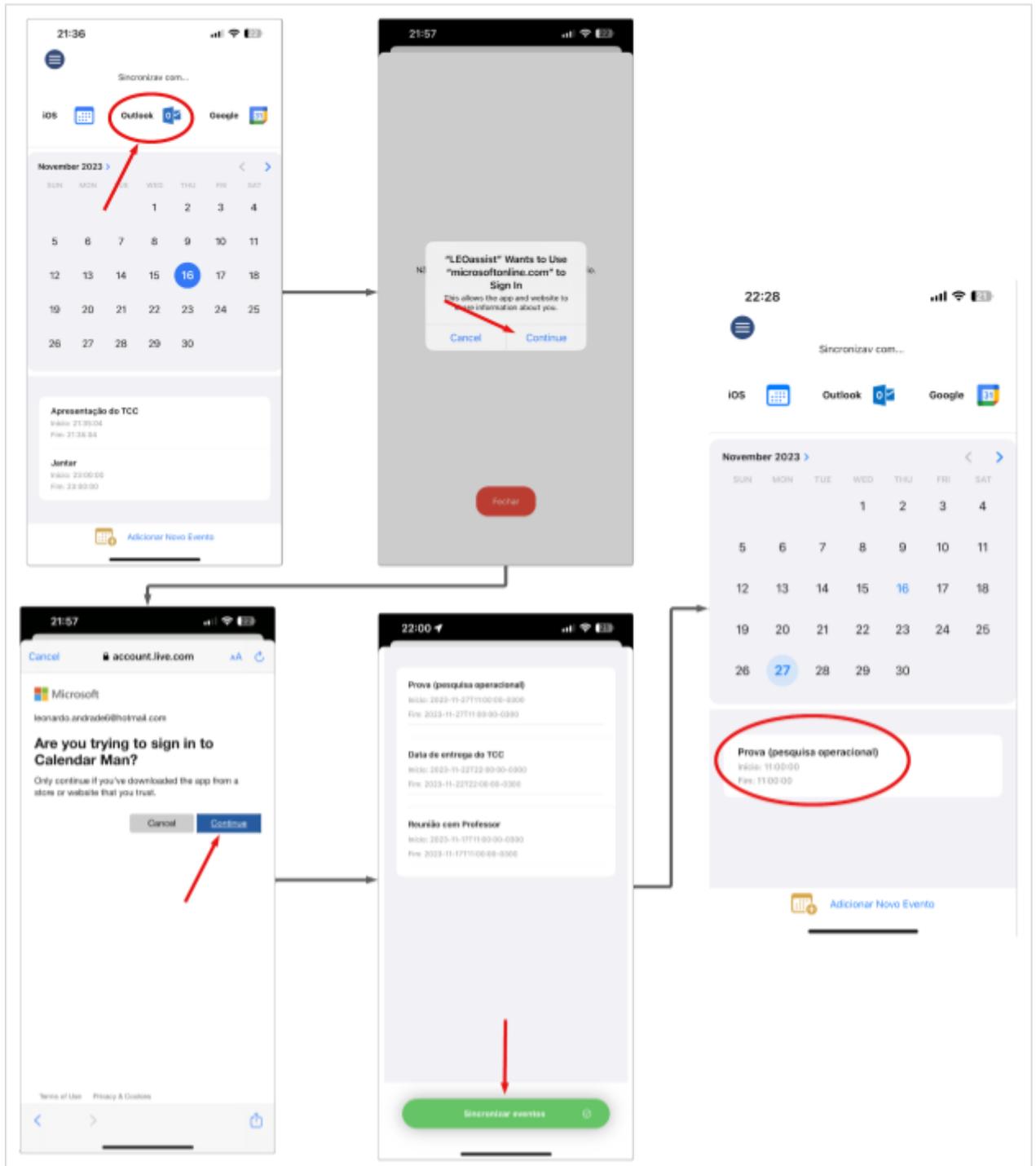


Fonte: Autor

A Figura 08 ilustra a integração com o calendário do *Outlook*. Neste processo, o usuário inicia a ação ao clicar para efetuar login. Em seguida, recebe uma notificação informando que será redirecionado para o site do *Outlook*. Uma vez que a autenticação é concluída, é apresentada uma lista dos

eventos programados para o próximo mês. Ao selecionar a opção "sincronizar", o calendário do usuário é atualizado para refletir os eventos listados, resultando na completa integração com o sistema da aplicação.

Figura 08: Fluxo de integração com o calendário Outlook

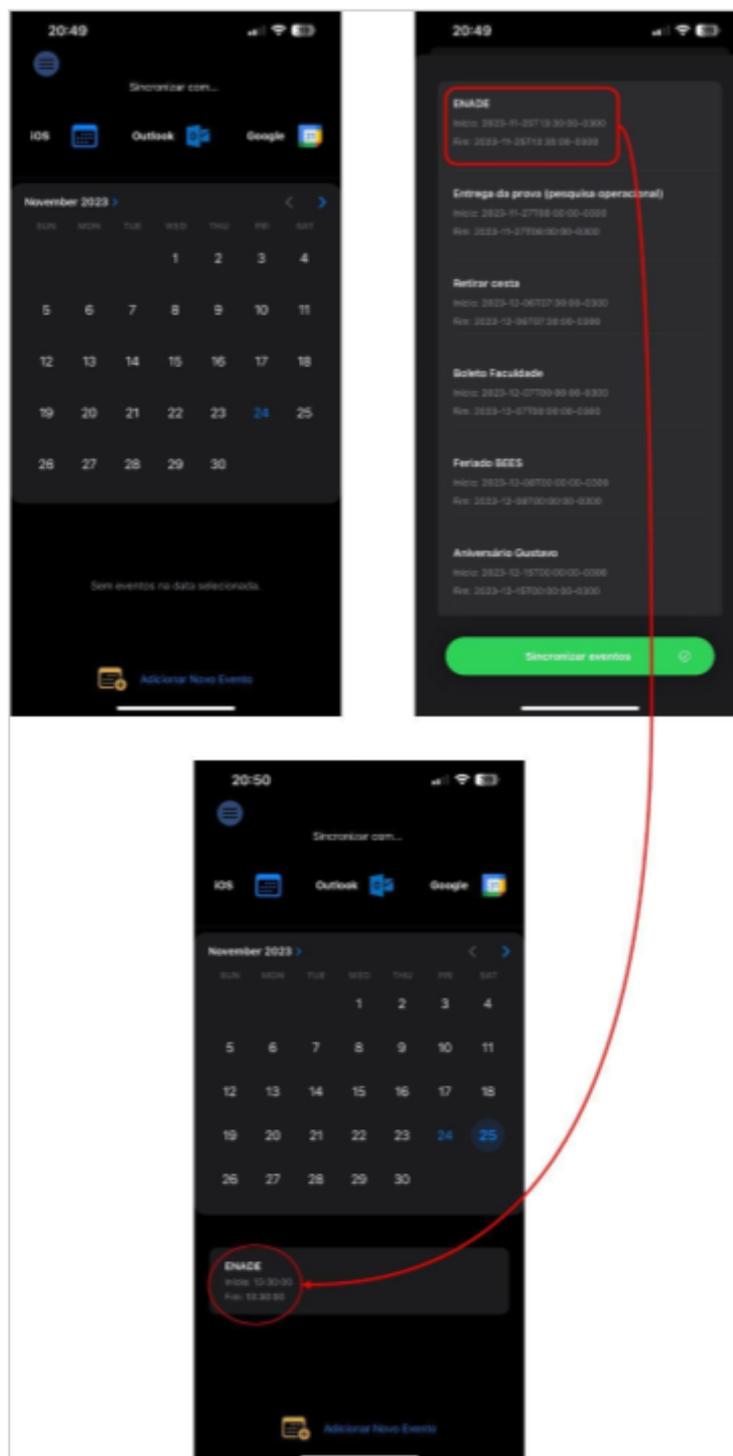


Fonte: Autor

Ao seleccionar a opção "sincronizar", o A integração com o calendário do *iOS* é demonstrada a partir da Figura 09, como pode-se perceber, a interface é a mesma utilizada com o calendário do *Outlook*, devido ao fato de que esse calendário pertence ao mesmo sistema operacional para o qual a linguagem (*SwiftUI*) é programada,

nota-se que não há necessidade de redirecionar o usuário para um site externo, o usuário apenas precisa autorizar uma vez o acesso à seu calendário. Após isso, o calendário do usuário é atualizado para refletir os eventos listados, resultando na completa integração com o sistema da aplicação.

Imagem 09: Fluxo de integração com iOS (Modo noturno)

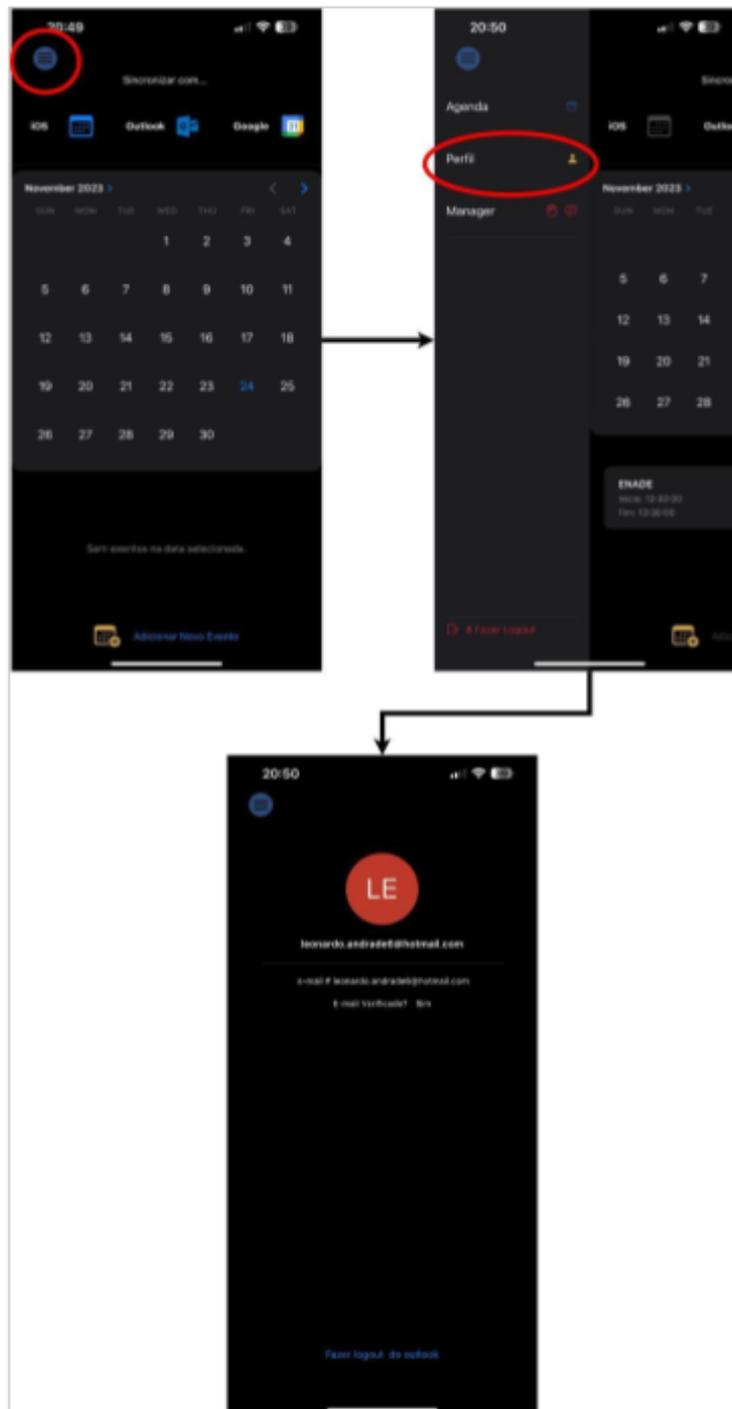


Fonte: Autor

Na figura 10, é demonstrado uma tela secundária, na qual o usuário consegue ver as informações do seu perfil, preenchidas na hora da inscrição do produto. Essas informações são extraídas através de um *JWT (Json Web Token)*, que é um padrão aberto (*RFC 7519*) que define uma maneira compacta e segura de transmitir informações

entre partes como um objeto *JSON*. Essas informações podem ser verificadas e confiáveis porque são digitalmente assinadas.

Figura 10. Fluxo para visualização de perfil do usuário



5. CONSIDERAÇÕES FINAIS

Este artigo apresentou o desenvolvimento do '*Calendar Man*', um assistente de gerenciamento de calendário inovador. A aplicação, construída com a interação de tecnologias como *Swift*, *SwiftUI* e *Java/Spring*, teve como objetivo proporcionar ao usuário uma interface intuitiva para o gerenciamento de eventos de calendário, empregando Inteligência Artificial para otimizar o agendamento.

Durante o desenvolvimento do projeto, foram destacados desafios importantes, principalmente aqueles relacionados à integração da IA com a aplicação. A necessidade de aprimorar os algoritmos de modelagem de linguagem para uma interação mais efetiva e a necessidade de melhorar a interpretação de contextos temporais foram algumas das questões que surgiram. A solução para esses problemas foi obtida através da utilização de "padrões de prompts", que se mostraram uma ferramenta eficaz para aprimorar a interação do usuário com a IA.

Além disso, a necessidade de integração com diferentes plataformas de calendário apresentou um conjunto de desafios. Cada plataforma tem suas próprias especificidades e, portanto, exigiu um conjunto único de soluções para sua integração adequada. A solução finalmente adotada envolveu a simplificação dos eventos de cada calendário em um único modelo usado em toda a aplicação.

O projeto '*Calendar Man*' encontra-se atualmente em seus estágios iniciais de lançamento e, como tal, a recepção do usuário e as melhorias potenciais são áreas para futura consideração. A integração de mais plataformas de calendário, aprimoramentos na interação de linguagem natural e melhorias na eficiência da IA são algumas das áreas identificadas para desenvolvimento adicional.

Em resumo, o desenvolvimento de '*Calendar Man*' ressalta o potencial das aplicações de IA na facilitação e melhoria do

gerenciamento de calendários. A aplicação serve não apenas como uma solução prática para o gerenciamento de eventos, mas também como um estudo de caso valioso sobre a interação entre IA e gerenciamento de calendários. A aplicação abre caminho para futuras pesquisas e desenvolvimentos nesta área emergente de estudo.

Outro ponto importante a ser destacado é que o de implantação manual, embora eficaz, pode ser aprimorado através da automação, utilizando ferramentas de Integração Contínua/Entrega Contínua (CI/CD), como *Jenkins*, *CircleCI* ou a própria *AWS CodePipeline*, para automatizar as etapas de compilação, teste e implantação.

Por fim, é importante ressaltar que este artigo reflete apenas o estado atual de desenvolvimento do '*Calendar Man*'. O campo da IA está em constante evolução e, como tal, o '*Calendar Man*' continuará a evoluir e se adaptar para se manter relevante e eficaz no futuro.

REFERÊNCIAS

Cosmina, **I. Pivotal Certified Professional Core Spring 5 Developer Exam: A Study Guide Using Spring Framework 5**, Apress; 2nd ed. edition (11 de dezembro, 2019).

Jules White, Quchen Fu, Sam Hays, Michael Sandborn, Carlos Olea, Henry Gilbert, Ashraf Elnashar, Jesse Spencer-Smith, and Douglas C. Schmidt, "**A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT**". Vanderbilt University, Tennessee. <<https://arxiv.org/abs/2302.11382>>

Amazon Web Services, **Abordagem da Amazon Web Services à resiliência operacional no setor financeiro e além**. (Abril de 2021, https://docs.aws.amazon.com/whitepapers/latest/aws-operational-resilience/customers-can-achieve-and-test-resiliency-on-aws.html?did=wp_card&trk=wp_card)

Amazon Web Services, AWS Well-Architected Framework. 10 de Abril, 2023. (https://docs.aws.amazon.com/wellarchitected/latest/framework/welcome.html?did=wp_card&trk=wp_card)

AWS Legal Team. **Contrato de nível de serviço do**

Amazon Compute. Amazon Web Services, 2018.

Jones, M., Bradley, J., & Sakimura, N. (2015). RFC 7519: **JSON Web Token (JWT)**. IETF Tools. <https://tools.ietf.org/html/rfc7519>

Vyas, Anshul. *MVC PATTERN*. Medium, 2018.

Gupta. M. *OCA Java SE 8 Programmer I Certification Guide First Edition*, 9 de dezembro, 2016.

MongoDB, Modernize with MongoDB Atlas and AWS. AtlasDB platform. (<https://www.mongodb.com/collateral/modernize-mongodb-atlas-aws>)

VEIGA, Ilma Passos Alencastro (Organização), [et. al.]. *Orientações institucionais para a elaboração de trabalho de conclusão de curso de graduação* – Brasília : UniCEUB, 2018.

Laria Reynolds, Kyle McDonell. *"Prompt Programming for Large Language Models: Beyond the Few-Shot Paradigm"*. <https://arxiv.org/abs/2102.07350>