

UNIVERSIDADE FEDERAL DO PARANÁ

SABRY INACIO RAFRAFI

UMA AVALIAÇÃO INICIAL DO POTENCIAL DE LLMs OPEN-SOURCE NO SUPORTE A  
PROCESSOS DE ELICITAÇÃO DE REQUISITOS DE SOFTWARE

CURITIBA PR

2025

SABRY INACIO RAFRAFI

UMA AVALIAÇÃO INICIAL DO POTENCIAL DE LLMS OPEN-SOURCE NO SUPORTE A  
PROCESSOS DE ELICITAÇÃO DE REQUISITOS DE SOFTWARE

Trabalho apresentado como requisito parcial à conclusão do Curso de Bacharelado em Ciência da Computação, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Computação*.

Orientador: Andrey Ricardo Pimentel.

CURITIBA PR

2025

# Ficha catalográfica

Substituir o arquivo `0-iniciais/catalografica.pdf` pela ficha catalográfica fornecida pela Biblioteca da UFPR (PDF em formato A4).

## **Instruções para obter a ficha catalográfica e fazer o depósito legal da tese/dissertação (contribuição de André Hochuli, abril 2019. Links atualizados Wellton Costa, Nov 2022):**

1. Estas instruções se aplicam a dissertações de mestrado e teses de doutorado. Trabalhos de conclusão de curso de graduação e textos de qualificação não precisam segui-las.
2. Verificar se está usando a versão mais recente do modelo do PPGInf e atualizar, se for necessário (<https://gitlab.c3sl.ufpr.br/maziero/tese>).
3. conferir o *checklist* de formato do Sistema de Bibliotecas da UFPR, em <https://bibliotecas.ufpr.br/servicos/normalizacao/>
4. Enviar e-mail para "referencia.bct@ufpr.br" com o arquivo PDF da dissertação/tese, solicitando a respectiva ficha catalográfica.
5. Ao receber a ficha, inseri-la em seu documento (substituir o arquivo `0-iniciais/catalografica.pdf` do diretório do modelo).
6. Emitir a Certidão Negativa (CND) de débito junto a biblioteca, em <https://bibliotecas.ufpr.br/servicos/certidao-negativa/>
7. Avisar a secretaria do PPGInf que você está pronto para o depósito. Eles irão mudar sua titulação no SIGA, o que irá liberar uma opção no SIGA pra você fazer o depósito legal.
8. Acesse o SIGA (<http://www.prppg.ufpr.br/siga>) e preencha com cuidado os dados solicitados para o depósito da tese.
9. Aguarde a confirmação da Biblioteca.
10. Após a aprovação do pedido, informe a secretaria do PPGInf que a dissertação/tese foi depositada pela biblioteca. Será então liberado no SIGA um link para a confirmação dos dados para a emissão do diploma.

# Ficha de aprovação

Substituir o arquivo 0-iniciais/aprovacao.pdf pela ficha de aprovação fornecida pela secretaria do programa, em formato PDF A4.

*A minha querida mãe Vainy...*

## **AGRADECIMENTOS**

Agradeço acima de tudo à minha mãe, Vainy Inacio Silva, por todo o amor, apoio e carinho. Agradeço também ao meu amigo Celso Kovalek, por todo o auxílio até aqui.

Estendo também meus agradecimentos a todos os professores do Departamento de Informática da UFPR e, em especial, aos professores Lucas Ferrari de Oliveira, Elias P. Duarte Jr., Elenice Novak, André Grégio, Carlos Alberto Maziero, Eduardo Cunha de Almeida, Bruno Müller Junior, Marcos Didonet Del Fabro, Letícia Mara Peres, Wagner Zola, Mael Sachine, Eduardo Hoefel, Raul Prado Raya, Michele Nogueira Lima, José Renato Barbosa, Iomara Scandelari Lemos e Paulo Justiniano Junior por todo o conhecimento adquirido.

Agradeço em especial, não apenas pelo ensino, ao professor Andrey Ricardo Pimentel por toda a orientação dada a este trabalho e também aos professores Marco Zanata Alves e Natasha Valentim pelo auxílio na reta final da graduação.

## SUMÁRIO

<b>1</b>	<b>INTRODUÇÃO</b>	<b>7</b>
<b>2</b>	<b>CONCEITOS FUNDAMENTAIS</b>	<b>9</b>
2.1	LLMS	9
2.1.1	O Que São?	9
2.1.2	Arquitetura	9
2.1.3	Quantização	10
2.2	REQUISITOS DE SOFTWARE	11
2.2.1	Importância	11
2.2.2	Tipos de Requisitos	13
<b>3</b>	<b>TRABALHOS RELACIONADOS</b>	<b>15</b>
<b>4</b>	<b>PROPOSTA</b>	<b>17</b>
4.1	REQUISITOS DE SISTEMA	17
4.1.1	Hardware utilizado	17
4.1.2	Python	17
4.1.3	LangChain	17
4.1.4	LangGraph	17
4.1.5	Chainlit	18
4.1.6	Ollama	18
4.2	IMPLEMENTAÇÃO	18
4.2.1	Modelo	18
4.2.2	Limite de Tokens	19
4.2.3	Prompt de Sistema Customizado	19
4.2.4	Processamento do Estado das Mensagens	20
4.2.5	Início da Sessão de Chat	20
4.2.6	Processamento das Mensagens do Chat	21
4.3	EXECUÇÃO	22
<b>5</b>	<b>EXPERIMENTO</b>	<b>23</b>
5.1	DEFINIÇÃO	23
5.2	CONFIGURAÇÃO DO CHATBOT	23
5.3	ENTREVISTA	24
5.4	GERAÇÃO DO RESUMO DA ENTREVISTA	25
5.5	GERAÇÃO DE HISTÓRIA DE USUÁRIO	26
5.6	FORMULÁRIO TAM	26
5.7	REALIZAÇÃO DO EXPERIMENTO	28

5.8	RESULTADOS OBTIDOS . . . . .	28
5.8.1	Entrevista . . . . .	28
5.8.2	Geração do Resumo da Entrevista . . . . .	29
5.8.3	Geração da História de Usuário. . . . .	29
5.8.4	Formulário TAM . . . . .	29
<b>6</b>	<b>CONCLUSÕES . . . . .</b>	<b>31</b>
	<b>REFERÊNCIAS . . . . .</b>	<b>32</b>
	<b>APÊNDICE A – CONTEÚDO DO CASO COMPLETAMENTE BEM-SUCEDIDO NO EXPERIMENTO. . . . .</b>	<b>37</b>
A.1	ENTREVISTA . . . . .	37
A.2	GERAÇÃO DE RESUMO . . . . .	40
A.3	GERAÇÃO DE HISTÓRIA DE USUÁRIO . . . . .	42
	<b>APÊNDICE B – CONTEÚDO DAS HISTÓRIAS DE USUÁRIO GERADAS NOS DEMAIS CASOS DO EXPERIMENTO.. . . . .</b>	<b>45</b>
B.1	CASOS PARCIALMENTE BEM-SUCEDIDOS. . . . .	45
B.1.1	Primeiro Caso. . . . .	45
B.1.2	Segundo Caso. . . . .	47
B.2	CASO FALHO . . . . .	49

# 1 INTRODUÇÃO

A Engenharia de Requisitos (ER) é uma fase crítica do processo de desenvolvimento de software, que inclui a etapa de elicitação de requisitos de software. Porém, apesar de sua importância, ER permanece sendo uma fase desafiadora devido às complexidades de comunicação e incerteza nas etapas iniciais do processo de desenvolvimento (Arora et al., 2023; Pressman e Maxim, 2016).

Além disso, organizações e equipes frequentemente negligenciam ou não compreendem sua importância e impacto no sucesso de projetos. Algumas razões para a falta de recursos e esforços investidos em ER incluem restrições de orçamento e tempo, ausência de treinamento e habilidades adequados, e uma ênfase em abordagens "implementation-first". Estes fatores conduzem a desafios consideráveis em etapas posteriores de desenvolvimento à medida em que requisitos incorretos, incompletos ou inconsistentes se tornam cada vez mais difíceis de se resolver e atender (Arora et al., 2023).

Desde a proposição da arquitetura Transformer de redes neurais por Vaswani et al. (2017), os modelos grandes de linguagem (LLMs) tem se provado promissores em diversas áreas, incluindo processamento de linguagem natural. LLMs são modelos avançados de inteligência artificial (IA) projetados para processar e gerar linguagem humana por meio do reconhecimento e aprendizado de padrões e estruturas em porções vastas de dados textuais. Estes modelos permitiram avanços significativos em tarefas de processamento de linguagem natural (PLN) e provam ser particularmente habilidosos em lidar com desafios complexos baseados em linguagens (Arora et al., 2023). LLMs, como os da série Generative Pre-trained Transformer (GPT) da OpenAI<sup>1</sup> ou da série Qwen<sup>2</sup> da Alibaba Cloud<sup>3</sup>, compreendem e geram linguagem humana ao prever a próxima palavra mais provável em uma dada sequência de palavras. O ChatGPT da OpenAI, construído a partir dos avanços dos LLMs, é um exemplo de plataforma de chatbot projetada para facilitar conversas interativas e dinâmicas baseadas em texto. Quando um usuário fornece ao ChatGPT ou ao Qwen algum prompt como input, os modelos processam o prompt e geram uma resposta contextualmente apropriada com base nos padrões aprendidos durante a fase de treinamento do modelo.

Uma grande maioria de requisitos funcionais ou não funcionais são especificados por meio de linguagem natural (LN). Logo, LLMs possuem o potencial de serem um "divisor de águas" no campo de ER. Isto poderia se dar pela automação e simplificação de variadas tarefas cruciais ou pela assistência na abordagem de desafios de ER, como os mencionados anteriormente. Seja com foco voltado para geração de código baseada em LLMs ou para programação por desenvolvedores humanos, elicitar requisitos de sistema de maneira clara e inequívoca é fundamental, o que ressalta a crescente importância de ER na atual era de engenharia de software (Arora et al., 2023).

Muito embora a grande maioria dos LLMs de maior popularidade atualmente sejam de formato "closed source", isto é, em que o acesso aos dados de treinamento ou ao código do modelo é restrito (como é o caso das séries Claude, GPT e Gemini, respectivamente, da Anthropic, OpenAI e Google), este trabalho tem como objetivo geral construir um chatbot para testar o

---

<sup>1</sup>OpenAI: <https://openai.com/pt-BR/about/>

<sup>2</sup>Qwen: <https://qwen.org/>

<sup>3</sup>Alibaba Cloud: [https://www.alibabacloud.com/en/about?\\_p\\_lc=1&spm=a3c0i.14327653.9556232360.35.454b188fyIJJaY3](https://www.alibabacloud.com/en/about?_p_lc=1&spm=a3c0i.14327653.9556232360.35.454b188fyIJJaY3)

potencial de um modelo open source para apoio os processos de entrevistas de stakeholders e produção de Histórias de Usuário que ilustrem requisitos encontrados na etapa de entrevistas.

Derivados deste objetivo geral, este trabalho tem como objetivos específicos a implementação de um chatbot baseado em um LLM open source; o estabelecimento da configuração necessária para gerenciamento de contexto durante o uso do chatbot; a definição seguida da realização de um experimento prático (contando com participantes humanos) envolvendo o uso do chatbot para simular entrevistas de stakeholders para um software hipotético, gerar resumos dessas entrevistas e, a partir desses resumos, Histórias de Usuário; e a avaliação (por parte dos participantes) dos resultados deste experimento. O processo de definir este experimento necessariamente inclui como sub-objetivos a definição de prompts de sistema específicos para cada etapa do experimento.

Tendo isto em vista, tanto os testes de implementação do chatbot deste trabalho quanto o experimento envolvendo o mesmo foram realizados tendo como base o modelo Qwen2.5 (Qwen et al., 2025).

Como resultados deste trabalho, tem-se uma forma de implementar um chatbot baseado em LLMs através de frameworks e bibliotecas completamente open source e, a partir do exemplo mais bem sucedido do experimento (incluído como Apêndice deste trabalho), insights sobre como certos padrões de prompts de usuário e sistema podem conduzir os LLMs a melhores resultados na automatização de processos de requisitos de software.

O Capítulo 2 proverá os conceitos fundamentais nos quais este trabalho se fundamenta, como as definições de requisitos e LLMs. O Capítulo 3 abordará alguns trabalhos e artigos similares que se relacionam e servem de motivação a este trabalho. O Capítulo 4 apresentará em detalhes a implementação e o funcionamento do chatbot desenvolvido ao longo deste trabalho. O Capítulo 5 descreverá tanto o experimento proposto quanto os resultados obtidos. Por fim, o Capítulo 6 apresentará as conclusões gerais do trabalho e sugestões de possíveis futuros trabalhos.

## 2 CONCEITOS FUNDAMENTAIS

### 2.1 LLMS

#### 2.1.1 O Que São?

Os grandes modelos de linguagem ("large language models" em inglês, ou LLMs) são uma categoria de modelos de redes neurais treinados em vastas quantidades de dados, tornando-os capazes de entender e gerar linguagem natural e outros tipos de conteúdo para executar uma grande variedade de tarefas. Os LLMs funcionam como enormes máquinas de previsão estatísticas que prevêm repetidamente a próxima palavra em uma sequência, aprendendo padrões em seus dados de treinamento e gerando, então, uma linguagem que segue esses padrões. Eles representam um grande salto na forma como seres humanos interagem com a tecnologia por serem capazes de lidar com linguagem humana não estruturada em escala. Enquanto mecanismos de pesquisa tradicionais e outros sistemas programados usavam algoritmos específicos para corresponder palavras-chave, os LLMs capturam contexto, nuances e raciocínios mais profundos. Uma vez treinados, esses modelos podem se adaptar a vários cenários que envolvem interpretação de texto, como (Stryker, 2026):

- Resumir artigos;
- Depurar código;
- Redigir documentos jurídicos;
- Traduzir textos;
- Responder perguntas, etc.

Eles são o apogeu de décadas de progresso na áreas de PLN e aprendizado de máquina, e são amplamente usados em empresas, com organizações realizando grandes investimentos em inúmeras funções de negócios e casos de uso (Stryker, 2026).

Eles evoluíram a partir de abordagens anteriores estatísticas e de redes neurais recorrentes ("recurrent neural networks" ou RNNs) (Vaswani et al., 2017). Avaliações de benchmarks para LLMs tem evoluído de avaliações linguísticas restritas para avaliações abrangentes e multitarefas (Wang et al., 2019; Hendrycks et al., 2021).

Além disso, cientistas de dados costumam usar técnicas de ajuste fino ("fine-tuning" em inglês) para alinhamento, um processo que consiste em tornar as saídas de um modelo úteis, seguras e consistentes com os valores humanos (Stryker, 2026).

Ainda, enquanto o pré-treinamento provê ao modelo um conhecimento geral, o ajuste fino permite adaptar um modelo de propósitos gerais a tarefas específicas, como sumarização, classificação ou suporte ao cliente, produzindo saídas mais próximas de exemplos fornecidos por seres humanos, exigindo muito menos recursos do que o treinamento a partir do zero (Stryker, 2026).

#### 2.1.2 Arquitetura

Os LLMs são geralmente baseados na arquitetura Transformer e, em um alto nível, pode-se dizer que seu funcionamento se baseia no uso de mecanismos de atenção global para

capturar dependências globais entre entradas e saídas, permitindo maior paralelização durante a fase de treinamento (Vaswani et al., 2017).

Modelos mais conhecidos baseados nesta arquitetura incluem, além do GPT (Generative Pre-trained Transformer), o BERT (Bidirectional Encoder Representations from Transformers), e fazem uso de encoders e decoders, para transformar os inputs em embeddings e gerar texto, respectivamente (Devlin et al., 2019).

Pode-se dizer que a grande inovação trazida pela arquitetura Transformer, quando esta foi introduzida em 2017, foi a substituição da recorrência pela autoatenção (Vaswani et al., 2017). A aprendizagem por reforço ("reinforcement learning"), principalmente através dos algoritmos de gradiente de política, foi empregada para ajustar os LLMs a certos comportamentos desejados além da previsão bruta do próximo token (Christiano et al., 2023). A aprendizagem por reforço a partir do feedback humano ("reinforcement learning from human feedback" ou RLHF) aplica esses métodos para otimizar uma política, o que tem sido crítico para alinhar as respostas dos modelos às expectativas dos usuários, aprimorando factualidade e performance de tarefas, e reduzindo respostas ofensivas (Ouyang et al., 2022).

Cada LLM tem um limite de memória chamado janela de contexto, que determina quantos tokens ele consegue processar ao mesmo tempo. Os primeiros LLMs tinham janelas pequenas, mas os modelos mais recentes suportam janelas de centenas de milhares de tokens, permitindo casos de uso como resumir longos documentos, prestar assistência em grandes bases de código ou mesmo manter longas conversas coerentes com usuários (Stryker, 2026).

### 2.1.3 Quantização

Eficiência é um motivo crucial para o uso de métodos de quantização para rodar LLMs em sistemas que não possuem altos recursos computacionais. Com esses métodos, é possível usar esses modelos quantizados em aplicações do mundo real em dispositivos como laptops, tablets e smartphones (Clark, 2025).

Tal como a compressão de imagem funciona reduzindo a sua resolução para que ela ocupe menos espaço, a quantização tem como base representar os valores dos pesos e ativações de um modelo em formatos de dados menores do que o formato original em que este foi treinado (Valenzuela, 2024).

As técnicas de quantização podem ser classificadas em (Clark, 2025):

- PTQ (Post-Training Quantization): quantiza após o treinamento de um LLM. Tem como estratégia principal a calibração com uma escala e um zero-point. É usada quando não há como re-treinar o modelo;
- QAT (Quantization-Aware Training): integra o processo de quantização ao ajustar os pesos no processo de treinamento. É usada quando se quer a melhor acurácia possível com quantidades baixas de bits de precisão.

Pode-se destacar as seguintes técnicas de quantização mais utilizadas para reduzir o tamanho de LLMs (Talamadupula, 2024):

- QLoRA: quantiza os pesos em formato NF4 ("4-bit NormalFloat") e os fatores de escala de cada bloco em 8 bits. Em um modelo de 65 bilhões de parâmetros, isso levaria a aproximadamente 3 GB de economia. É muito usado em ajustes finos com poucos recursos.

- GPTQ: mistura pesos em INT4 com ativadores em FP16 (floats representados por 16 bits). Permite rodar modelos de 70 bilhões de parâmetros em uma única GPU, não requer re-treino e representa um bom ponto de trade-off.
- GGML/GGUF: biblioteca que prioriza a performance em CPUs, oferecendo k-quants (Q2\_K, Q5\_0, Q8\_0, etc). Por exemplo, Q5\_0 tem 5 bits de precisão para todos os pesos e Q2\_K tem 4 bits para os maiores e 2 bits para os menores.
- AWQ: mantém menos de 1% em FP16 e quantiza o restante em INT3 (inteiros de 3 bits) ou INT4. Focando em pesos "mais importantes" para minimizar perda de acurácia, é possível reduzir memória de modelos de larga escala em aproximadamente 70%.

Jin et al. (2024) propõe uma avaliação abrangente de LLMs instruction-tuned (mais usados em cenários reais) e suas versões quantizadas, testando estratégias como quantização em 4 bits e 2 bits em modelos de diferentes tamanhos (7, 14 e 72 bilhões de parâmetros). Os resultados mostram que:

- A quantização em 4 bits preserva desempenho próximo ao original em diversas tarefas (entendimento, geração de texto, conhecimento e alinhamento);
- O método SpQR supera GPTQ em quantização extremamente compacta (2 bits), destacando-se na eficiência.

Kurtic et al. (2024) avalia o desempenho de modelos quantizados em benchmarks reais, que simulam cenários práticos como interações em chat e geração de código. Três benchmarks foram destacados:

- Arena-Hard: uma extensão automatizada do Chatbot Arena que avalia respostas a prompts complexos, com alta correlação (89%) com preferências humanas. Os resultados mostram que modelos quantizados (de 8 bits e 4 bits do Llama 3.1) alcançam pontuações próximas aos originais (BF16, ou seja, números em ponto flutuante de 16 bits), com intervalos de confiança sobrepostos, indicando mínimo impacto na precisão;
- HumanEval e HumanEval+: testam a capacidade de geração de código. Os modelos quantizados recuperam até 99,9% (8 bits) e 98,9% (4 bits) da precisão em tarefas simples e complexas, confirmando sua eficácia em cenários de programação.

Com essa economia em custos de memória e processamento, a quantização não apenas permite trazer LLMs para sistemas mais próximos ao nível de consumidor como também permite um custo energético muito menor (Mukherjee, 2024).

## 2.2 REQUISITOS DE SOFTWARE

### 2.2.1 Importância

Os requisitos de software são essenciais para o desenvolvimento de qualquer sistema, pois são a base para a prototipação, design, testes e validação em todas as etapas do desenvolvimento. O processo de elicitação de requisitos é fundamental para o bom andamento de um projeto pois melhora a comunicação entre stakeholders, permite uma gestão mais eficiente de expectativas e reduz riscos de falhas no projeto (Sommerville, 2019).

Assim, um dos maiores desafios enfrentados por desenvolvedores, product owners e product managers é a definição dos requisitos de software: entender o que é essencial, quais funcionalidades deve ser priorizadas e como garantir a satisfação do usuário final. Requisitos de software são especificações que definem as funcionalidades e restrições de um software (Sommerville, 2019).

A clareza que esses requisitos proporcionam é o que permite uma boa comunicação (Sommerville, 2019). Por exemplo: um aplicativo de uma instituição financeira, voltado ao cliente, deve ser capaz de realizar transações via PIX, pagar boletos e fazer investimentos. Esses são 3 requisitos do produto digital que permitem aos usuários a realização de tais funções.

A engenharia de requisitos envolve três atividades fundamentais. São elas (Sommerville, 2019):

- Descoberta dos requisitos por meio da interação com stakeholders (elicitação e análise);
- Conversão desses requisitos em uma forma padrão (especificação);
- Averiguação de que os requisitos realmente definem o sistema que o cliente quer (validação).

A elicitação de requisitos de software é uma etapa que envolve identificar as necessidades e expectativas dos stakeholders. É importante que sejam utilizados métodos eficientes durante a coleta desses requisitos, para, então, garantir a confiabilidade e a adequação às demandas do sistema. Existem duas abordagens fundamentais para a elicitação de requisitos (Sommerville, 2019):

- Entrevistas com stakeholders: envolve conversas diretas com pessoas envolvidas ou interessadas no projeto para reunir informações com maior profundidade;
- Observações ou etnografia: envolve observar os usuários em seus ambientes de trabalho.

Esse levantamento dos requisitos é um processo contínuo e colaborativo que envolve diversos stakeholders, incluindo clientes, usuários finais, gerentes de equipe, engenheiros de software e analistas de negócios (Sommerville, 2019).

A documentação de requisitos de software de forma clara e precisa é essencial para garantir o sucesso de qualquer produto. Afinal, a documentação detalhada serve como um guia para todas as etapas do desenvolvimento. Para isso, algumas técnicas são bem úteis, como diagramas de casos de uso, especificações tabulares, estruturadas ou em linguagem natural (Sommerville, 2019).

Um bom gerenciamento de requisitos de software inclui o controle das mudanças, onde qualquer alteração proposta é registrada (Sommerville, 2019). Para facilitar esse processo, podem ser utilizadas ferramentas de gerenciamento como Jira (Atlassian, 2025).

A validação de requisitos de software envolve a confirmação de que as especificações documentadas estão corretas, completas e compreendidas por todos os envolvidos. As técnicas que podem ser usadas para essa validação são (Sommerville, 2019):

- Revisões: envolvem a análise dos requisitos por uma equipe de stakeholders e especialistas, por exemplo, os tech leads, garantindo que todos os aspectos sejam confirmados e aceitos;
- Prototipagem: consiste em criar modelos simplificados do sistema, que permitem aos usuários interagir e fornecer feedback antes da construção completa do produto. Figma e Adobe XD são ferramentas bastante usadas para isso;

- Gerações de casos de teste: utilizam testes desenvolvidos a partir dos requisitos de usuário.

Ao contrário do que muitos pensam, os processos de engenharia de requisitos não se dão apenas no início de um projeto. O motivo disto é que os requisitos de um software estão sempre mudando, uma vez que os problemas que eles tratam não conseguem ser totalmente definidos. Isso torna requisitos de software obrigatoriamente incompletos. A compreensão das partes envolvidas em um projeto de software evolui, logo os requisitos deste devem evoluir para refletir essas mudanças de compreensão (Sommerville, 2019).

A engenharia de requisitos, embora fundamental, enfrenta vários desafios. Um dos principais é a comunicação eficaz entre as partes interessadas e os desenvolvedores. Muitas vezes, há uma lacuna entre o que os usuários precisam e como os desenvolvedores interpretam essas necessidades. A complexidade de integrar novas tecnologias em sistemas existentes também apresenta desafios significativos (Sommerville, 2019).

## 2.2.2 Tipos de Requisitos

### 2.2.2.1 *Requisitos Funcionais (RF)*

Os requisitos funcionais definem o que o sistema deve fazer e especificam funcionalidades e comportamentos esperados (Sommerville, 2019). Alguns exemplos comuns são:

- Autenticação de usuários;
- Gestão e consulta de dados por um usuário ou administrador;
- Processamento de pagamentos;
- Emissão de relatórios;
- Atualização de estados ("em processamento" para "concluído").

### 2.2.2.2 *Requisitos Não Funcionais (RNF)*

Os requisitos não funcionais definem restrições de tempo, restrições sobre o processo de desenvolvimento e restrições impostas por padrões. Frequentemente, aplicam-se ao sistema como um todo ao invés de apenas às suas características individuais (Sommerville, 2019).

Podem ser subdivididos em (Sommerville, 2019):

- Requisitos de produto: desempenho, confiabilidade, segurança, usabilidade;
- Requisitos organizacionais: processos operacionais, processos de desenvolvimento, ambiente;
- Requisitos externos: regulações, legislações, ética.

Pode-se apontar os seguintes exemplos de requisitos não funcionais (Sommerville, 2019):

- Em expediente normal, o sistema não pode permanecer fora do ar por mais de 5 anos;
- Os usuários do sistema devem se autenticar com o cartão de identificação;

- O sistema deve implementar providências para garantir a privacidade do paciente, conforme estabelecido na LGPD.

Alguns exemplos de métricas para requisitos não funcionais são (Sommerville, 2019):

- Velocidade: tempo de resposta;
- Tamanho: KBs ou quantidade de RAM necessária;
- Facilidade de uso: tempo de treinamento;
- Confiabilidade: tempo médio até uma falha;
- Robustez: tempo de reinício após a falha;
- Portabilidade: percentual de dependência do sistema-alvo.

### 2.2.2.3 *Outros Tipos*

Pode-se também classificar os requisitos de software como requisitos de domínio (referentes às regras específicas de um setor).

Os requisitos de domínio estão associados a padrões específicos do domínio de aplicação do software. Eles são moldados pelo contexto em que o software será utilizado e podem incluir regulamentações legais, padrões da indústria ou práticas específicas do negócio (Sommerville, 2019).

### 3 TRABALHOS RELACIONADOS

A partir da revisão de literatura realizada por Marques et al. (2024), é possível apontar vários estudos que se conectam profundamente com a ideia do uso de LLMs na área de engenharia de requisitos de software e, por consequência, com a proposta e o experimento realizados neste trabalho. Ainda que tal revisão, em sua maior parte, dedique sua atenção a estudos relacionados ao uso do modelo "closed-source" ChatGPT da OpenAI, pode-se destacar os seguintes estudos:

- Liu e Reddivari (2022) destacam a importância da engenharia de requisitos no ciclo de vida de desenvolvimento de software, discutem o papel da inteligência artificial na melhoria da qualidade dos requisitos e sugerem o uso de técnicas avançadas de aprendizado de máquina e processamento de linguagem natural para atender aos requisitos textuais, tendo como ponto forte do estudo a identificação da lacuna de conhecimento sobre como a IA deve ser incorporada aos processos de requisitos de software para garantir uma representação de alta qualidade. Porém, o texto não fornece detalhes concretos sobre resultados e contribuições específicas.
- Abdelfattah et al. (2023) propõe o uso do ChatGPT para criar diagramas de arquitetura de software, tendo como ponto forte do estudo a criação de exercícios práticos que focam no entendimento dos princípios básicos da engenharia de requisitos.
- Arora et al. (2023) apresenta uma análise SWOT para pesquisa e desenvolvimento usando LLMs para engenharia de requisitos.
- Belzner et al. (2023) revisa as oportunidades de uso de LLMs em engenharia de requisitos, design de sistema, geração de código e de testes, revisão de código e processos de software.
- Bencheikh e Höglund (2023) mostram a eficácia do ChatGPT na geração de requisitos, mas enfatizam o papel humano no processo.
- El-Hajjami et al. (2024) avalia o GPT-3.5-turbo e o GPT-4 em classificação de requisitos comparados a métodos como SVMs ("support vector machines") e LSTM ("long short-term memory"). A conclusão dos autores é que não há uma única melhor técnica para todas as classificações porque a melhor técnica varia dependendo da classificação específica.
- Kutzner e Gröpler (2023) exploram a geração de texto com IA para criar requisitos e especificações funcionais para a produção de documentação. O ponto forte do estudo é o reconhecimento da importância de uma documentação clara.
- Ronanki et al. (2023) avalia o potencial do ChatGPT para apoiar os requisitos de software, tendo como ponto forte a participação ativa de experts na área.
- Rasheed et al. (2023) demonstra como múltiplos agentes GPT podem realizar de forma independente várias tarefas de engenharia de software, análise de requisitos, design, geração de código, debugging e gerenciamento de manutenção de software.

- Sridhara et al. (2023) usa o ChatGPT para investigar tarefas diárias de engenharia de software tais como resolver ambiguidades em requisitos de software, sugerir nomes de métodos, priorizar casos de testes, revisar código e resumir logs.
- Subahi (2023) mapeia requisitos de software usando o modelo de linguagem BERT em conjunto com um grande conjunto de dados.
- Wang et al. (2023) cria um framework chamado Chatcoder para refinar requisitos por meio de conversa com LLMs.
- White et al. (2023b; 2023a) cataloga padrões de design de engenharia de prompt para várias tarefas de engenharia de software.
- Zhang et al. (2023) avalia recuperação zero-shot de requisitos do ChatGPT em 2 tarefas de análises de requisitos sobre 4 datasets.
- Fantechi et al. (2024) detecta variabilidade de requisitos em documentos em linguagem natural usando o GPT-3.5 e o Microsoft Bing.
- Luitel et al. (2024) usa BERT para gerar previsões para preencher slots mascarados em requisitos.
- Oswal et al. (2024) converte requisitos não estruturados em Histórias de Usuários, usando uma abordagem baseada em IA, e enfatiza que a geração manual dessas Histórias é um processo laborioso e desafiador.
- Waseem et al. (2024) avalia o ChatGPT como um bot de desenvolvimento de software em diferentes fases do ciclo de vida de desenvolvimento de software.
- Yeow et al. (2024) avalia perguntas geradas pelo ChatGPT-3.5 para elicitación de requisitos.

## 4 PROPOSTA

### 4.1 REQUISITOS DE SISTEMA

#### 4.1.1 Hardware utilizado

O chatbot foi implementado e testado no sistema operacional Ubuntu (24.04.3), usando como GPU uma NVIDIA RTX 4070 Ti Super, com 16 GB de VRAM e como CPU um AMD Ryzen 5 7600, com 32 GB de RAM DDR5.

#### 4.1.2 Python

O desenvolvimento do chatbot foi feito na versão 3.12 da linguagem Python em um ambiente virtual `.venv` criado com:

```
python3 -m venv .venv

# para ativar o ambiente virtual
source .venv/bin/activate

# para sair do ambiente
deactivate
```

`venv`, ao ser utilizado para criar uma instalação Python isolada, permite gerenciar instalações separadas de pacotes para projetos diferentes. Logo, ao alternar entre diferentes projetos, pode-se criar para cada projeto um ambiente virtual independente dos demais. Esta abordagem, que é recomendável quando se trabalha com pacotes de terceiros (o que é o caso aqui), traz como benefício a preservação da instalação Python geral do sistema.

Todos os demais módulos utilizados para o desenvolvimento do chatbot foram instalados dentro do ambiente virtual `.venv` criado.

#### 4.1.3 LangChain

LangChain é um framework de código aberto (licença MIT) que facilita a integração de LLMs em sistemas e aplicações. Portanto, ele possui muitos casos de uso em comum com os próprios modelos de linguagem, como análise e resumo de documentos, chatbots e análise de código (LangChain, 2026a; langchain ai, 2026a).

Sua versão para Python é construída a partir do pacote `langchain-core`, que contém as interfaces e abstrações principais usadas em todo o ecossistema LangChain. Essas abstrações foram projetadas para serem tão simples e modulares quanto possível, permitindo que qualquer provedor possa implementar as interfaces necessárias para serem facilmente usadas por todo o ecossistema (PyPI, 2025a).

Ambos os pacotes Python foram instalados no ambiente `.venv` com:

```
pip install langchain langchain-core
```

#### 4.1.4 LangGraph

Biblioteca pertencente ao ecossistema LangChain, LangGraph é um framework de código aberto (licença MIT) de orquestração de baixo nível e um ambiente de execução para criar,

gerir e implementar agentes de IA de longa duração e com gerenciamento de estados (LangChain, 2026b; langchain ai, 2026b).

Sua versão para Python foi instalada no ambiente `.venv` com:

```
pip install langgraph>0.2.27
```

#### 4.1.5 Chainlit

Chainlit é um pacote Python de código aberto (licença Apache-2.0) projetado para a construção de interfaces de usuário de IA conversacional interativas e semelhantes a interfaces como as do ChatGPT (Chainlit, 2025d).

O pacote foi instalado no ambiente `.venv` com:

```
pip install chainlit
```

#### 4.1.6 Ollama

Ollama é uma plataforma de código aberto (licença MIT) projetada para executar LLMs localmente, permitindo que os usuários gerem texto, obtenham assistência em codificação, resumam conteúdo, criar embeddings etc, com os modelos providos pela plataforma, como Qwen, Phi e Mistral, em formato quantizado (geralmente Q4\_K\_M) (Ollama, 2025c).

O cliente Ollama foi instalado com:

```
curl -fsSL https://ollama.com/install.sh | sh
```

##### 4.1.6.1 Interface para o LangChain

Para que o LangChain consiga interagir com o serviço Ollama, faz-se necessário o pacote `langchain_ollama` (licença MIT), que provê a infraestrutura requerida para realizar a integração (PyPI, 2025b).

O pacote foi instalado no ambiente `.venv` com:

```
pip install -qU langchain_ollama
```

## 4.2 IMPLEMENTAÇÃO

### 4.2.1 Modelo

O chatbot usa um LLM pré-baixado pelo cliente Ollama que é carregado pelo método `OllamaLLM` do pacote `langchain_ollama` (LangChain, 2025d). Para baixar algum modelo presente na plataforma Ollama, roda-se o comando `ollama pull`, seguido do nome do modelo desejado:

```
ollama pull "nome do modelo"
```

Para saber os modelos disponíveis para download através do cliente Ollama, assim como suas variações de tamanho (baseado em número de parâmetros), deve-se consultar o site da plataforma. Por exemplo, se um usuário deseja baixar a versão de 20 bilhões de parâmetros do modelo `gpt-oss` presente na plataforma, deve-se executar o seguinte comando:

```
ollama pull gpt-oss:20b
```

Na implementação do chatbot, a chamada do modelo se dá pela criação de um objeto da classe `OllamaLLM`:

```
...
llm = OllamaLLM(model="gpt-oss:20b", streaming=True)
...
```

O parâmetro opcional `streaming=True` permite imprimir como saída os tokens à medida que vão sendo gerados, tornando a interação com os modelos mais responsiva.

#### 4.2.2 Limite de Tokens

Através da função `trim_messages` (LangChain, 2025c) do pacote `langchain_core`, o chatbot pode limitar o tamanho do histórico de conversa com um LLM à uma certa quantidade máxima de tokens ou mensagens:

```
...
trimmer = trim_messages(
    max_tokens = 65536, # o modelo gpt-oss permite até um contexto de
                        # até 128k tokens

    strategy="last",
    token_counter=llm,
    include_system=True,
    allow_partial=False,
    start_on="human",
)
...
```

O parâmetro `max_tokens = 65536` da função recebe, como um inteiro, o limite máximo de tokens das mensagens cortadas a ser considerado (que, no caso acima, é 65.536). Para cada modelo a ser utilizado, é imprescindível consultar os limites de janela de contexto no site da plataforma ou, no caso do modelo já ter sido baixado pelo cliente Ollama, executando o comando `ollama show "nome do modelo"`.

O parâmetro `strategy="last"` determina que o contexto contenha apenas as últimas mensagens da conversa, dentro do limite `max_tokens`. O parâmetro `token_counter=llm` recebe a função (ou, no caso, o modelo) responsável por contar os tokens de uma mensagem.

O parâmetro `include_system=True` determina que o histórico da conversa fornecido ao modelo inclua a `SystemMessage`, que contém instruções especiais para o modelo. O conteúdo dessas instruções será melhor detalhado na subseção seguinte.

O parâmetro `allow_partial=False` recebe, como booleano, o valor que decide se mensagens parciais devem ser consideradas dentro da janela de contexto. No caso, a janela conserva apenas mensagens completas.

Por fim, o parâmetro `start_on="human"` determina que o histórico de conversa guardado comece com uma `HumanMessage`, que contém o prompt do usuário do chatbot.

#### 4.2.3 Prompt de Sistema Customizado

Na implementação do chatbot da proposta, customiza-se as instruções de sistema contidas na `SystemMessage` através da criação de um objeto da classe `ChatPromptTemplate` (LangChain, 2025e):

```
...
prompt_template = ChatPromptTemplate.from_messages(
    [
        (
            "system",
            "You are a helpful AI bot.",
        ),
        MessagesPlaceholder(variable_name="messages"),
    ]
)
```

```
)
...
```

O método `from_messages` permite a criação de um padrão de prompt de conversa a partir de diferentes formatos de mensagens. No caso, instancia-se `prompt_template` com uma lista contendo, como primeiro valor, uma tupla no formato `("system", "You are a helpful AI bot.")`, definindo assim o conteúdo da `SystemMessage`.

Como segundo valor dessa lista, instancia-se um objeto da classe `MessagePlaceholder`, que conterá uma lista das mensagens do histórico da conversa sob a chave `variable_name="messages"`.

#### 4.2.4 Processamento do Estado das Mensagens

No chatbot da proposta, a função `call_model` é responsável por processar o estado atual das mensagens, preparar um prompt para o LLM e obter do mesmo uma resposta:

```
...
def call_model(state: MessagesState):
    trimmed_messages = trimmer.invoke(state["messages"])
    prompt = prompt_template.invoke({"messages": trimmed_messages})
    response = llm.invoke(prompt)
    return {"messages": response}
...
```

Ela tem como único parâmetro o objeto `state: MessagesState`, que inclui todas as mensagens trocadas até o momento, representando o estado atual do chat.

Nesta função, o `trimmer` configurado anteriormente é chamado para remover mensagens antigas, garantindo que o número total de tokens não exceda o limite configurado. Em seguida, as mensagens "trimadas" são passadas como um dicionário sob a chave `"messages"` como parâmetro da chamada do `prompt_template` (também definido anteriormente) que criará o prompt completo com as instruções de sistema.

O `llm` configurado anteriormente processa então o prompt e gera uma resposta que, por sua vez, é retornada pela função.

#### 4.2.5 Início da Sessão de Chat

Na implementação da proposta, a função `on_chat_start` (Chainlit, 2025b) é chamada sempre que uma nova sessão de chat se inicia, sendo responsável por configurar todo o workflow do chat:

```
...
@cl.on_chat_start
async def on_chat_start():
    workflow = StateGraph(state_schema=MessagesState)

    workflow.add_edge(START, "model")
    workflow.add_node("model", call_model)
    memory = MemorySaver()
    app = workflow.compile(checkpointer=memory)
    cl.user_session.set("app", app)
...
```

O decorador `@cl.on_chat_start`, fornecido pelo framework Chainlit, define que a função deve ser chamada quando uma sessão de chat começa.

Uma vez que a função começa, é criado um objeto da classe `StateGraph`, que representa um grafo cujos nós se comunicam por leitura e escrita em um estado compartilhado, com o esquema `state_schema=MessagesState`, especificando que o estado do chat é representado por uma lista de mensagens (LangChain, 2025b). Em seguida, é adicionada uma aresta ao grafo, conectando o nó `START` ao nó `"model"` que, por sua vez, é associado à função `call_model` descrita anteriormente. Isso significa que, quando uma nova sessão começa, o workflow inicia no nó `"model"` e a função `call_model` é chamada para processar mensagens e obter respostas do LLM.

É criado, então, um objeto `MemorySaver` para salvar o estado do workflow (LangChain, 2025a). Ele funciona armazenando checkpoints usando um `defaultdict`. O workflow é, então, compilado em um objeto `CompiledStateGraph` com o salvador de memória, tornando-se então uma aplicação executável (LangChain, 2025b).

Por fim, a aplicação é armazenada em uma variável interna `"app"` de sessão do Chainlit, permitindo então que ela seja recuperada e utilizada em outras áreas do código, como na função `on_message`, que será detalhada a seguir.

#### 4.2.6 Processamento das Mensagens do Chat

No chatbot da proposta, a função `on_message` (Chainlit, 2025c) é chamada sempre quando uma nova mensagem do usuário é recebida em uma sessão de chat e é responsável por processar a mensagem, invocar o LLM (através do workflow criado anteriormente) para obter uma resposta e retornar essa resposta ao usuário:

```
...
@cl.on_message
async def on_message(message: cl.Message):
    app = cl.user_session.get("app")
    query = message.content
    input_messages = [HumanMessage(query)]
    msg = cl.Message(content="")

    for part in app.invoke({"messages": input_messages}, config)
    ["messages"][-1].content:
        await msg.stream_token(part)
```

O decorador `@cl.on_message`, fornecido pelo framework Chainlit, define que a função deve ser chamada quando uma mensagem do usuário é recebida durante uma sessão de chat.

A função começa com a aplicação compilada do workflow definido anteriormente sendo recuperada. Em seguida, a mensagem do usuário que será processada pelo modelo é obtida a partir do parâmetro `message: cl.Message`. Com ela, uma `HumanMessage` é instanciada. Cria-se então uma mensagem vazia, usando `cl.Message(content=)`. Esta mensagem vazia será usada para armazenar e transmitir a resposta para a interface do usuário.

Por fim, a aplicação é invocada, tendo como parâmetros a `HumanMessage` com a mensagem do usuário e uma variável de configuração que define um `thread_id` como no exemplo a seguir:

```
config = {"configurable": {"thread_id": "abc123"}}
```

A invocação da aplicação executa o workflow definido em `on_chat_start`, obtendo assim uma resposta do modelo de linguagem, que consiste em uma lista de mensagens. A última

mensagem da lista é a resposta do modelo à última mensagem do usuário. Tal resposta é então transmitida para o usuário por meio de um laço que itera sobre cada `part` do conteúdo da resposta. Dentro desse laço, chamamos a função `await msg.stream_token(part)` para transmitir a função token a token, em tempo real, tornando a experiência de uso do chatbot mais responsiva (Chainlit, 2025a).

### 4.3 EXECUÇÃO

Concluída a implementação e instalados os requisitos de sistema (o que inclui o download de algum LLM via Ollama), inicia-se a execução do chatbot da proposta via terminal, dentro do ambiente virtual Python configurado, com o seguinte comando:

```
chainlit run "nome_do_arquivo.py"
```

O programa em execução automaticamente redirecionará o usuário para a interface web gerenciada pelo Chainlit no navegador padrão do sistema do usuário, no endereço: <http://localhost:8000/>.

## 5 EXPERIMENTO

### 5.1 DEFINIÇÃO

O experimento foi dividido em 4 etapas:

- 1. A simulação de uma curta entrevista, através do chatbot proposto, em que o LLM escolhido é instruído a atuar como um engenheiro de software que entrevistará os participantes humanos do experimento a respeito de funcionalidades interessantes para um software fictício.
- 2. A geração de um resumo da entrevista resultante da etapa 1, através do mesmo chatbot, onde o LLM é instruído a resumir o conteúdo do prompt do usuário.
- 3. A geração de uma História de Usuário, a partir do resumo obtido na etapa 2, através do mesmo chatbot.
- 4. O preenchimento de um formulário, baseado na teoria do Technology Acceptance Model (TAM), no qual os participantes do experimento avaliarão a utilidade e facilidade de uso percebidas a partir de suas experiências nas etapas anteriores.

Cada uma das etapas listadas acima será detalhada nas seções seguintes deste capítulo.

### 5.2 CONFIGURAÇÃO DO CHATBOT

Após seu desenvolvimento, o chatbot foi testado inicialmente com 3 LLMs open source:

- Mistral, um modelo de 7 bilhões de parâmetros distribuído com a licença Apache 2.0 (Team, 2023). Sua versão disponibilizada na plataforma Ollama tem quantização Q4\_K\_M, possui uma janela de contexto máxima de 32K tokens e um tamanho de 4.4GB (Ollama, 2025b).
- Qwen2.5, em sua versão de 14 bilhões de parâmetros distribuída com a licença Apache 2.0 (Team, 2024). Sua versão na Ollama tem quantização Q4\_K\_M e possui uma janela de contexto máxima de 32K tokens e um tamanho de 9.0GB (Ollama, 2024).
- Command R7B, um modelo de 7 bilhões de parâmetros distribuído com a licença Creative Commons Attribution-NonCommercial 4.0 International Public. Sua versão na Ollama tem quantização Q4\_K\_M, possui uma janela de contexto máxima de 8K tokens e um tamanho de 5.1GB (Ollama, 2025a).

Nestes testes, tanto o Mistral quanto o Command R7B entenderam erroneamente as instruções para entrevistar o usuário e, de imediato, responderam eles próprios as perguntas de base sugeridas para serem feitas ao usuário. O Qwen2.5 conseguiu manter o papel de entrevistador por 2 prompts de usuário antes de começar a responder ele próprio os questionamentos.

Logo, o LLM open-source escolhido para o experimento foi o Qwen2.5, criado pela Alibaba Cloud.

O download do modelo foi feito via cliente Ollama com o seguinte comando no terminal:

```
ollama pull qwen2.5:14b
```

No código do chatbot, a chamada do modelo se dá de maneira análoga ao exemplo mostrado no capítulo anterior:

```
...
llm = OllamaLLM(model="qwen2.5:14b", streaming=True)
...
```

Nas etapas 1 a 3 do experimento, a janela de contexto usada foi no limite máximo suportado pelo modelo:

```
...
trimmer = trim_messages(
    max_tokens = 32768, # limite máximo de contexto do modelo
    strategy="last",
    token_counter=llm,
    include_system=True,
    allow_partial=False,
    start_on="human",
)
...
```

### 5.3 ENTREVISTA

Para cada participante do experimento, a entrevista consiste na interação entre o participante e o LLM, através da interface do chatbot. O cenário hipotético que se deseja simular aqui é a fase de entrevistas de stakeholders para extrair requisitos e insights preliminares a respeito de um sistema fictício chamado "Calendarium", que seria um serviço online de calendários e agendas, semelhante ao Google Agenda (Google, 2025).

Para isso, é necessário instruir o modelo a atuar como o engenheiro de software que conduzirá a entrevista. Para isso, faz-se o uso dos Prompts de Sistemas Customizados, discutidos no capítulo anterior, da seguinte forma:

```
...
prompt_template = ChatPromptTemplate.from_messages(
    [
        (
            "system",
            ## caso entrevista:
            "Você é um engenheiro de software, em fase de elicitação "
            + "de requisitos para o seguinte sistema: Calendarium é um "
            + "serviço de agenda e calendário online. Com ele é "
            + "possível gerenciar eventos e compromissos, compartilhar "
            + "a programação com outras pessoas, agregar à sua agenda "
            + "diversas agendas públicas, entre outras "
            + "funcionalidades. Atenda o usuário da melhor forma "
            + "possível em português.",
        ),
        MessagesPlaceholder(variable_name="messages"),
    ]
)
...
```

Dando início à execução do chatbot, a entrevista deve sempre começar com o seguinte prompt de usuário, fornecida ao modelo pelo supervisor do experimento:

```
"Olá! Gostaria que você me entrevistasse a respeito do
Calendarium, a fim de elicitar requisitos e funcionalidades
desejáveis para esse sistema. Faça uma ou 2 perguntas por vez
se possível."
```

A partir da primeira resposta do modelo, a entrevista deve se estender por, no mínimo, mais 2 prompts do participante ou, no máximo, mais 6 prompts do participante, que deve responder aos questionamentos do modelo à respeito do Calendarium. A partir do momento em que o modelo responder ao 6º prompt do participante ou este decidir não mais continuar a interação após seu 2º prompt (o que vier primeiro), a entrevista deve ser encerrada com a seguinte prompt de usuário, também fornecida pelo supervisor:

"Por favor, liste todas as características de sistema desejáveis para o Calendarium identificadas até então na nossa conversa."

O supervisor deve então salvar toda a entrevista, desde o primeiro prompt fornecido por ele até a resposta do modelo ao seu prompt de encerramento, em um arquivo .txt, antes de encerrar o chatbot via terminal. Além disso, o supervisor deve editar o arquivo com o único propósito de facilitar a identificação de quais partes da conversa são prompts de usuário ou respostas do modelo. Antes do início da próxima etapa do experimento, o arquivo .txt deve conter o conteúdo completo da entrevista no seguinte formato:

```

USUÁRIO:
Olá! Gostaria que você me entrevistasse a respeito do Calendarium, a
fim de elicitare requisitos e funcionalidades desejáveis para esse
sistema. Faça uma ou 2 perguntas por vez se possível.

ENGENHEIRO:
[*resposta do modelo*]

USUÁRIO:
[*prompt do participante*]

ENGENHEIRO:
[*resposta do modelo*]

USUÁRIO:
[*prompt do participante*]

ENGENHEIRO:
[*resposta do modelo*]

...

USUÁRIO:
Por favor, liste todas as características de sistema desejáveis para o
Calendarium identificadas até então na nossa conversa.

ENGENHEIRO:
[*resposta do modelo*]

```

#### 5.4 GERAÇÃO DO RESUMO DA ENTREVISTA

Para cada entrevista, deseja-se gerar um resumo da mesma, através do chatbot da proposta. Para isto, o LLM deve receber uma nova instrução de sistema através do Prompt de Sistema Customizado:

```

...
prompt_template = ChatPromptTemplate.from_messages(
    [

```

```

    (
        "system",
        ## caso resumidor:
        "Você é um resumidor de entrevistas. Forneça um resumo "
        + "detalhado das entrevistas fornecidas pelo usuário, em "
        + "até cerca de 800 palavras.",
    ),
    MessagesPlaceholder(variable_name="messages"),
]
)
...

```

Dando início à execução do chatbot, a interação consiste no supervisor fornecer ao modelo o conteúdo formatado da entrevista obtida na etapa anterior como prompt de usuário. A interação deve ser encerrada após a resposta do modelo.

O supervisor deve então salvar apenas esta resposta em um arquivo .txt, antes de encerrar o chatbot via terminal.

## 5.5 GERAÇÃO DE HISTÓRIA DE USUÁRIO

Para cada resumo obtido na etapa anterior, deseja-se gerar uma História de Usuário a partir do mesmo, através do chatbot da proposta. Logo, é necessário instruir o LLM a atuar como uma espécie de "storyteller" que gerará a História. Nesta etapa do experimento, usamos o seguinte Prompt de Sistema Customizado:

```

...
prompt_template = ChatPromptTemplate.from_messages(
    [
        (
            "system",
            ## caso storyteller:
            "Você é um storyteller. Para cada resumo de entrevista "
            + "fornecido, responda com uma História de Usuário baseada "
            + "em tal resumo, em até cerca de 800 palavras. "
            + "O propósito dessa história é que ela possa ser "
            + "analisada por engenheiros de software para extrair "
            + "requisitos de sistema.",
        ),
        MessagesPlaceholder(variable_name="messages"),
    ]
)
...

```

Dando início à execução do chatbot, a interação consiste no supervisor fornecer ao modelo o conteúdo do resumo obtido na etapa anterior como prompt de usuário. A interação deve ser encerrada após a resposta do modelo.

O supervisor deve então salvar apenas esta resposta em um arquivo .txt, antes de encerrar o chatbot via terminal.

## 5.6 FORMULÁRIO TAM

Cada participante do experimento deve então responder a um formulário, montado e disponibilizado via Google Forms (Google, 2026), que consiste em uma adaptação da teoria

Technology Acceptance Model (TAM), para avaliar a utilidade e a facilidade de uso percebidas ("perceived usefulness" e "perceived ease-of-use", respectivamente) nas suas experiências com o chatbot da proposta nas etapas anteriores. Antes do participante responder o formulário, o supervisor do experimento deve fornecer a ele uma cópia dos arquivos .txt obtidos nas 3 etapas anteriores, para consulta caso o participante julgue necessário.

Inicialmente, o formulário deve apresentar ao participante uma breve descrição da teoria do TAM. Na seção seguinte do formulário, o participante deve se identificar e informar sua área de atuação. Em seguida, o participante deverá avaliar a utilidade percebida do chatbot, respondendo 5 questões:

- 1. Usar este chatbot no meu trabalho me permitiria cumprir tarefas mais rapidamente?
- 2. Usar este chatbot melhoraria a performance no meu trabalho?
- 3. Usar este chatbot no meu trabalho aumentaria minha produtividade?
- 4. Usar este chatbot faria meu trabalho mais fácil?
- 5. Eu acharia este chatbot útil no meu trabalho?

As questões são respondidas de forma objetiva, em uma escala linear de respostas de 1 a 7, sendo:

- 1. Extremamente improvável
- 2. Bastante improvável
- 3. Ligeiramente improvável
- 4. Indiferente
- 5. Ligeiramente provável
- 6. Bastante provável
- 7. Extremamente provável

Além disso, esta seção do formulário deve fornecer uma definição de "performance" e "produtividade" para que o participante se atente, durante o preenchimento das questões.

Em seguida, o participante deverá avaliar a facilidade de uso percebida do chatbot, respondendo outras 5 questões, cujas respostas se dão na mesma escala linear da seção anterior:

- 1. Aprender a usar este chatbot seria fácil para mim?
- 2. Eu acharia fácil fazer este chatbot fazer o que eu quero?
- 3. Minha interação com este chatbot seria clara e compreensível?
- 4. Seria fácil para mim tornar-me habilidoso ao usar este chatbot?
- 5. Eu acharia este chatbot fácil de se usar?

Por fim, na última seção do formulário, o participante é livre para compartilhar ou não suas impressões a respeito do chatbot, assim como sugestões e críticas.

## 5.7 REALIZAÇÃO DO EXPERIMENTO

Além do supervisor, o experimento contou com 4 participantes, que contribuíram para as etapas 1 a 3 (entrevista, geração de resumo e História de Usuário, respectivamente) através de reuniões online síncronas individuais com o supervisor. Para a etapa 4 (formulário TAM), os participantes contribuíram de forma online assíncrona.

Todos os 4 participantes são graduandos do curso de Bacharelado em Ciência da Computação da UFPR, sendo que ao menos 2 destes estagiam na área de Desenvolvimento de Software.

A execução do chatbot durante as etapas 1 a 3 foi realizada nos mesmos SO e hardware em que foram testados ao longo de sua implementação, ou seja, em um sistema operacional Ubuntu (24.04.3), tendo como GPU uma NVIDIA RTX 4070 Ti Super, com 16 GB de VRAM e como CPU um AMD Ryzen 5 7600, com 32 GB de RAM DDR5. Tendo em vista que a versão usada do modelo Qwen2.5 possui um tamanho de 9 GB, o modelo conseguiu ser carregado integralmente na memória da GPU, possibilitando uma velocidade razoável de inferência.

## 5.8 RESULTADOS OBTIDOS

### 5.8.1 Entrevista

Das 4 entrevistas realizadas, 1 foi totalmente bem-sucedida (isto é, manteve-se uma conversa completamente coesa do início ao fim da interação), 2 foram parcialmente bem-sucedidas (ou seja, manteve-se uma conversa coesa até certo ponto da interação, a partir do qual o LLM passou a dar sinais de alucinação) e 1 foi falha (isto é, o modelo, a partir de certo ponto da conversa apresentou alucinações que tornaram impossível continuar a ter uma interação coesa). Os sinais de alucinação encontrados nas entrevistas desta etapa resumem-se ao modelo responder questões referentes ao "Calendarium" formuladas por ele mesmo e imprimir, de forma incoerente, tokens como `<im_start>` e caracteres latinos e chineses.

Ao longo das entrevistas, foi possível constatar que algumas perguntas formuladas pelo modelo a respeito do "Calendarium" recorrentemente surgiam, como por exemplo:

- "Como você gerencia sua agenda atualmente?"
- "Você prefere receber notificações de seus compromissos através do celular, computador ou ambos?"
- "Você gostaria de ter a opção de criar eventos recorrentes automaticamente?"

Observando a entrevista completamente bem-sucedida (incluída no Apêndice A para consideração, assim como seu respectivo resumo gerado na etapa seguinte) em comparação às 3 demais, foi notado que, para cada leva de questões geradas pelo modelo, os prompts-resposta do participante seguiam um formato mais organizado, enumerando qual resposta correspondia à qual pergunta, o que pode ter dado menos margem para alucinação do modelo, garantindo uma conversa coerente por toda a interação.

Apesar disso, mesmo nos cenários de alucinação por parte do modelo, o prompt final desta etapa resultou em uma satisfatória listagem de todos os requisitos mencionados ao longo de toda a entrevista.

### 5.8.2 Geração do Resumo da Entrevista

Dos 4 resumos gerados a partir de suas respectivas entrevistas, os 4 foram bem-sucedidos, ou seja, todos os requisitos de sistema identificados pelo LLM na etapa anterior estavam presentes no resumo gerado.

Analisando os resumos gerados, foi notado que estes se basearam, em grande parte, na listagem de requisitos gerada no final da etapa anterior, repetindo por vezes *ipsis literis* seu conteúdo e por vezes, acrescentando algum contexto oriundo do restante da entrevista. Em 1 dos resumos, o modelo apontou brevemente, antes de listar os requisitos identificados, os interesses e hábitos que o participante alegou possuir ao longo da sua respectiva entrevista. Em outro resumo, o modelo dividiu os requisitos identificados na entrevista em "funcionais" e "não-funcionais", adicionando aos não-funcionais requisitos que não haviam sido abordados explicitamente na entrevista original (como "segurança" e "usabilidade").

### 5.8.3 Geração da História de Usuário

Das 4 tentativas de geração de Histórias de Usuário, 1 foi totalmente bem-sucedida (isto é, o texto gerado apresentou, de forma explícita, formato "como-quero-para", personagem principal e histórias de usuário concisas e acompanhadas de critérios de aceitação), 2 foram parcialmente bem-sucedidas (ou seja, o texto gerado apresentou, de forma explícita, apenas alguns dos elementos mencionados no caso totalmente bem-sucedido) e 1 foi falha (isto é, o texto gerado não explicitou nenhum dos elementos mencionados no caso totalmente bem-sucedido). A História de Usuário completamente bem-sucedida foi incluída para consideração no Apêndice A.

### 5.8.4 Formulário TAM

#### 5.8.4.1 *Utilidade Percebida*

Com relação ao preenchimento da seção referente à avaliação de utilidade percebida no uso do chatbot nas etapas anteriores, os 4 participantes do experimento responderam às questões da seguinte forma,

- Q1.1: 3 consideraram "bastante provável" e 1 considerou "ligeiramente provável" que o uso do chatbot os permitiria cumprir tarefas mais rapidamente.
- Q1.2: 2 consideraram "bastante provável" e 2 consideraram "ligeiramente provável" que o uso do chatbot melhoraria suas performances no trabalho.
- Q1.3: 1 considerou "extremamente provável", 1 considerou "bastante provável" e 2 consideraram "ligeiramente provável" que o uso do chatbot aumentaria suas produtividades.
- Q1.4: 1 considerou "extremamente provável", 2 consideraram "bastante provável" e 1 considerou "ligeiramente provável" que o uso do chatbot faria seus trabalhos mais fáceis.
- Q1.5: 1 considerou "extremamente provável", 2 consideraram "bastante provável" e 1 considerou "ligeiramente provável" que achariam o chatbot útil em seus trabalhos.

#### 5.8.4.2 *Facilidade de Uso Percebida*

Com relação ao preenchimento da seção referente à avaliação de facilidade de uso percebida no uso do chatbot nas etapas anteriores, os 4 participantes do experimento responderam às questões da seguinte forma,

- Q2.1: 2 consideraram "extremamente provável" e 2 consideraram "bastante provável" que aprender a usar o chatbot seria fácil.
- Q2.2: 1 considerou "bastante provável", 2 consideraram "ligeiramente provável" e 1 se considerou "indiferente" com relação a achar fácil fazer o chatbot fazer o que este quer.
- Q2.3: 1 considerou "bastante provável" e 3 consideraram "ligeiramente provável" que suas interações com o chatbot seriam claras e compreensíveis.
- Q2.4: 1 considerou "extremamente provável", 2 consideraram "bastante provável" e 1 considerou "ligeiramente provável" que seria fácil para eles se tornarem habilidosos ao usar o chatbot.
- Q2.5: 1 considerou "extremamente provável" e 3 consideraram "bastante provável" que achariam o chatbot fácil de se usar.

#### 5.8.4.3 *Feedbacks*

Os feedbacks colhidos ao fim do formulário indicaram interesse por parte dos participantes na ideia do chatbot da proposta e que este pode fornecer uma base para elicitação de requisitos e um escopo inicial para um projeto. Os feedbacks também incluíram sugestões para aprimorar a interface, como tornar possível que o entrevistado possa visualizar, durante a conversa, as informações que o chatbot possui.

## 6 CONCLUSÕES

Neste trabalho, foi explorado o potencial de um LLM open-source (Qwen2.5) como possível ferramenta de apoio aos processos de entrevistas de stakeholders, sumarização de entrevistas e produção de Histórias de Usuário através de um experimento, com um chatbot implementado com o modelo escolhido, simulando estas etapas, e o preenchimento de um formulário TAM para avaliação da utilidade e facilidade de uso percebidas no uso do chatbot implementado.

Na etapa de entrevistas, foi constatado que o modelo apresentou dificuldades em manter uma conversa coerente com o participante em certos casos, embora tenha permitido listar requisitos e insights preliminares para o desenvolvimento do software em questão. A etapa de sumarização das entrevistas obtidas mostrou total capacidade do modelo em sintetizar os requisitos identificados em um formato conciso ao mesmo tempo que incorporava certos contextos oriundos da conversa entre os participantes e o modelo. Na etapa de geração de Histórias de Usuário, o modelo apresentou dificuldades em adaptar os resumos gerados em um texto contendo todos os elementos característicos de uma História de Usuário.

As respostas dos participantes colhidas no formulário TAM levaram a constatar que o uso do chatbot muito provavelmente permitiria cumprir tarefas rapidamente, faria seus trabalhos mais fáceis e que os usuários achariam o chatbot útil em seus trabalhos. Além disso, seria também muito provável que usar, aprender a usar e se tornar habilidoso ao usar o chatbot seria fácil. Isso indica uma boa utilidade e facilidade de uso percebidas no chatbot da proposta.

Tudo isso corrobora a existência de um potencial ainda pouco explorado no uso de modelos open-source, apesar de suas atuais limitações, para automatização de certas etapas referentes à elicitación de requisitos na área de Engenharia de Software.

Com base nisso, há razões para apontar como possíveis caminhos a seguir no futuro os seguintes trabalhos:

- A realização do mesmo experimento, porém com uma base significativamente maior de participantes, incluindo participantes de diferentes graus de instrução e áreas do conhecimento;
- O ajuste fino do mesmo modelo usado no experimento ou de algum outro modelo, seja via re-treino ou uso de LoRAs ("low rank adaptation" em inglês, ou adaptação de baixo nível), para especializá-lo em geração de Histórias de Usuário;
- O refinamento e aprimoramento da interface web do chatbot da proposta e da experiência de usuário derivada deste.

## REFERÊNCIAS

- Abdelfattah, A., Ali, N. e Elaziz, M. (2023). Roadmap for software engineering education using chatgpt. In Proceedings of the 2023 International Conference on Artificial Intelligence Science and Applications in Industry and Society (CAISAIS). Acessado em 12/12/2025.
- Arora, C., Grundy, J. e Abdelrazek, M. (2023). Advancing requirements engineering through generative ai: Assessing the role of llms. arXiv:2310.13976. Acessado em 11/07/2025.
- Atlassian (2025). Jira. <https://www.atlassian.com/software/jira>.
- Belzner, L., Gabor, T. e Wirsing, M. (2023). Large language model assisted software engineering: Prospects, challenges, and a case study. In Proceedings of the International Conference on Bridging the Gap between AI and Reality. Acessado em 12/12/2025.
- Bencheikh, L. e Höglund, N. (2023). Exploring the efficacy of chatgpt in generating requirements: An experimental study. Chalmers University of Technology. Acessado em 12/12/2025.
- Chainlit ([2025?]a). Message. <https://docs.chainlit.io/api-reference/message#stream-a-message>. Acessado em 04/12/2025.
- Chainlit ([2025?]b). on\_chat\_start. <https://docs.chainlit.io/api-reference/lifecycle-hooks/on-chat-start>. Acessado em 04/12/2025.
- Chainlit ([2025?]c). on\_message. <https://docs.chainlit.io/api-reference/lifecycle-hooks/on-message>. Acessado em 04/12/2025.
- Chainlit ([2025?]d). Overview. <https://docs.chainlit.io/get-started/overview>. Acessado em 04/12/2025.
- Christiano, P., Leike, J., Brown, T. B., Martic, M., Legg, S. e Amodei, D. (2023). Deep reinforcement learning from human preferences. arXiv:1706.03741. Acessado em 09/01/2026.
- Clark, B. ([2025?]). What is quantization? <https://www.ibm.com/think/topics/quantization>. Acessado em 14/01/2026.
- Devlin, J., Chang, M.-W., Lee, K. e Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv:1810.04805. Acessado em 09/01/2026.
- El-Hajjami, A., Fafin, N. e Salinesi, C. (2024). Which ai technique is better to classify requirements? an experiment with svm, lstm, and chatgpt. arXiv:2311.11547. Acessado em 12/12/2025.
- Fantechi, A., Gnesi, S. e Semini, L. (2024). Exploring llms' ability to detect variability in requirements. In International Working Conference on Requirements Engineering: Foundation for Software Quality. Acessado em 12/12/2025.
- Google (2025). Google calendar. <https://workspace.google.com/intl/pt-BR/products/calendar/>.
- Google (2026). Google forms. <https://workspace.google.com/intl/pt-BR/products/forms/>.

- Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D. e Steinhardt, J. (2021). Measuring massive multitask language understanding. *arXiv:2009.03300*. Acessado em 08/01/2026.
- Jin, R., Du, J., Huang, W., Liu, W., Luan, J., Wang, B. e Xiong, D. (2024). A comprehensive evaluation of quantization strategies for large language models. *arXiv:2402.16775*. Acessado em 14/01/2026.
- Kurtic, E., Marques, A., Kurtz, M. e Alistarh, D. (2024). We ran over half a million evaluations on quantized llms—here’s what we found. <https://developers.redhat.com/articles/2024/10/17/we-ran-over-half-million-evaluations-quantized-llms#>. Acessado em 14/01/2026.
- Kutzner, T. e Gröpler, J. (2023). Supporting students in the creation of requirements and functional specifications in interdisciplinary software development projects with the help of ai-based text generation tools. [https://accedacris.ulpgc.es/bitstream/10553/128281/1/Supporting\\_students\\_creation.pdf](https://accedacris.ulpgc.es/bitstream/10553/128281/1/Supporting_students_creation.pdf). Acessado em 12/12/2025.
- LangChain (2025a). Checkpointing. <https://reference.langchain.com/python/langgraph/checkpoints/#langgraph.checkpoint.memory.InMemorySaver>. Acessado em 04/12/2025.
- LangChain (2025b). Graphs. <https://reference.langchain.com/python/langgraph/graphs>. Acessado em 04/12/2025.
- LangChain (2025c). Messages. [https://reference.langchain.com/python/langchain/messages/#langchain.messages.trim\\_messages](https://reference.langchain.com/python/langchain/messages/#langchain.messages.trim_messages). Acessado em 04/12/2025.
- LangChain (2025d). Ollama. [https://reference.langchain.com/python/integrations/langchain\\_ollama/#langchain\\_ollama.OllamaLLM](https://reference.langchain.com/python/integrations/langchain_ollama/#langchain_ollama.OllamaLLM). Acessado em 04/12/2025.
- LangChain (2025e). Prompts. [https://reference.langchain.com/python/langchain\\_core/prompts/#langchain\\_core.prompts.chat.ChatPromptTemplate](https://reference.langchain.com/python/langchain_core/prompts/#langchain_core.prompts.chat.ChatPromptTemplate). Acessado em 04/12/2025.
- LangChain (2026a). Langchain overview. <https://docs.langchain.com/oss/python/langchain/overview>. Acessado em 15/01/2026.
- LangChain (2026b). Langgraph overview. <https://docs.langchain.com/oss/python/langgraph/overview>. Acessado em 14/01/2026.
- langchain ai (2026a). langchain. <https://github.com/langchain-ai/langchain>. Acessado em 15/01/2026.
- langchain ai (2026b). langgraph. <https://github.com/langchain-ai/langgraph>. Acessado em 14/01/2026.
- Liu, K. e Reddivari, K. (2022). Artificial intelligence in software requirements engineering: State-of-the-art. In *Proceedings of the IEEE 23rd International Conference on Information Reuse and Integration for Data Science (IRI)*. Acessado em 12/12/2025.

- Luitel, D., Hassani, S. e Sabetzadeh, M. (2024). Improving requirements completeness: automated assistance through large language models. *Requir. En.*, 29(1):73–95.
- Marques, N., Silva, R. R. e Bernardino, J. (2024). Using chatgpt in software requirements engineering: A comprehensive review. *Future Internet*, 16(6):6–11.
- Mukherjee, S. (2024). Understanding model quantization in large language models. <https://www.digitalocean.com/community/tutorials/model-quantization-large-language-models>. Acessado em 14/01/2026.
- Ollama (2024). qwen2.5:14b. <https://ollama.com/library/qwen2.5:14b>. Acessado em 04/12/2025.
- Ollama (2025a). command-r7b:7b. <https://ollama.com/library/command-r7b:7b>. Acessado em 09/01/2026.
- Ollama (2025b). mistral:7b. <https://ollama.com/library/mistral:7b>. Acessado em 09/01/2026.
- Ollama (2025c). Ollama. <https://ollama.org/pt/>. Acessado em 04/12/2025.
- Oswal, J., Kanakia, H. e Suktel, D. (2024). Transforming software requirements into user stories with gpt-3.5-: An ai-powered approach. In Proceedings of the 2024 2nd International Conference on Intelligent Data Communication Technologies and Internet of Things (IDCIoT). Acessado em 12/12/2025.
- Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L., Simens, M., Askell, A., Welinder, P., Christiano, P., Leike, J. e Lowe, R. (2022). Training language models to follow instructions with human feedback. arXiv:2203.02155. Acessado em 09/01/2026.
- Pressman, R. S. e Maxim, B. R. (2016). *Engenharia de Software: Uma Abordagem Profissional, 8ª Edição*, capítulo: 8º, páginas 131–133. AMGH Editora Ltda.
- PyPI (2025a). langchain-core. <https://pypi.org/project/langchain-core/>. Acessado em 04/12/2025.
- PyPI (2025b). langchain-ollama. <https://pypi.org/project/langchain-ollama/>. Acessado em 04/12/2025.
- Qwen, Yang, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Li, C., Liu, D., Huang, F., Wei, H., Lin, H., Yang, J., Tu, J., Zhang, J., Yang, J., Yang, J., Zhou, J., Lin, J., Dang, K., Lu, K., Bao, K., Yang, K., Yu, L., Li, M., Xue, M., Zhang, P., Zhu, Q., Men, R., Lin, R., Li, T., Tang, T., Xia, T., Ren, X., Ren, X., Fan, Y., Su, Y., Zhang, Y., Wan, Y., Liu, Y., Cui, Z., Zhang, Z. e Qiu, Z. (2025). Qwen2.5 technical report. Relatório técnico, Qwen.
- Rasheed, Z., Waseem, M., Kemell, K.-K., Xiaofeng, W., Duc, A. N., Systä, K. e Abrahamsson, P. (2023). Autonomous agents in software development: A vision paper. arXiv:2311.18440. Acessado em 12/12/2025.
- Ronanki, K., Berger, C. e Horkoff, J. (2023). Investigating chatgpt’s potential to assist in requirements elicitation processes. In Proceedings of the 49th Euromicro Conference on Software Engineering and Advanced Applications (SEAA). Acessado em 12/12/2025.

- Sommerville, I. (2019). *Engenharia de software, 10ª Edição*, capítulo: 4th, páginas 85–117. Pearson Education do Brasil Ltda.
- Sridhara, G., G., R. H. e Mazumdar, S. (2023). Chatgpt: A study on its utility for ubiquitous software engineering tasks. arXiv:2305.16837. Acessado em 12/12/2025.
- Stryker, C. ([2026?]). O que é llm (grandes modelos de linguagem)? <https://www.ibm.com/br-pt/think/topics/large-language-models>. Acessado em 08/01/2026.
- Subahi, A. F. (2023). Bert-based approach for greening software requirements engineering through non-functional requirements. <https://doi.org/10.1109/ACCESS.2023.3317798>. Acessado em 12/12/2025.
- Talamadupula, K. (2024). A guide to quantization in llms. <https://syml.ai/developers/blog/a-guide-to-quantization-in-llms/>. Acessado em 14/01/2026.
- Team, M. A. (2023). Mistral 7b. <https://mistral.ai/news/announcing-mistral-7b/>. Acessado em 09/01/2026.
- Team, Q. (2024). Qwen2.5: A party of foundation models. <https://qwenlm.github.io/blog/qwen2.5/>. Acessado em 04/12/2025.
- Valenzuela, A. (2024). Quantization for large language models (llms): Reduce ai model sizes efficiently. <https://www.datacamp.com/tutorial/quantization-for-large-language-models>. Acessado em 14/01/2026.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. e Polosukhin, I. (2017). Attention is all you need. arXiv:1706.03762. Acessado em 06/01/2026.
- Wang, A., Singh, A., Michael, J., Hill, F., Levy, O. e Bowman, S. R. (2019). Glue: A multi-task benchmark and analysis platform for natural language understanding. arXiv:1804.07461. Acessado em 08/01/2026.
- Wang, Z., Li, J., Li, G. e Jin, Z. (2023). Chatcoder: Chat-based refine requirement improves llms' code generation. arXiv:2311.00272. Acessado em 12/12/2025.
- Waseem, M., Das, T., Ahmad, A., Liang, P., Fehmideh, M. e Mikkonen, T. (2024). Chatgpt as a software development bot: A project-based study. arXiv:2310.13648. Acessado em 12/12/2025.
- White, J., Fu, Q., Hays, S., Sandborn, M., Olea, C., Gilbert, H., Elnashar, A., Spencer-Smith, J. e Schmidt, D. C. (2023a). A prompt pattern catalog to enhance prompt engineering with chatgpt. arXiv:2302.11382. Acessado em 12/12/2025.
- White, J., Hays, S., Fu, Q., Spencer-Smith, J. e Schmidt, D. C. (2023b). Chatgpt prompt patterns for improving code quality, refactoring, requirements elicitation, and software design. arXiv:2303.07839. Acessado em 12/12/2025.
- Yeow, J., Rana, M. e Majid, N. (2024). An automated model of software requirement engineering using gpt-3.5. In Proceedings of the 2024 ASU International Conference in Emerging Technologies for Sustainability and Intelligent Systems (ICETSIS). Acessado em 12/12/2025.

Zhang, J., Chen, Y., Niu, N., Wang, Y. e Liu, C. (2023). Empirical evaluation of chatgpt on requirements information retrieval under zero-shot setting. arXiv:2304.12562. Acessado em 12/12/2025.

## APÊNDICE A – CONTEÚDO DO CASO COMPLETAMENTE BEM-SUCEDIDO NO EXPERIMENTO

### A.1 ENTREVISTA

USUÁRIO: Olá! Gostaria que você me entrevistasse a respeito do Calendarium, a fim de elicitare requisitos e funcionalidades desejáveis para esse sistema. Faça uma ou 2 perguntas por vez se possível.

ENGENHEIRO: Olá! É um prazer auxiliar na coleta de requisitos para o Calendarium. Vamos começar com as primeiras perguntas.

- Como você gerencia sua agenda atualmente? Que tipo de recursos e funcionalidades considera essenciais em um serviço de calendário?
- Você compartilha suas agendas ou eventos com outras pessoas regularmente? Se sim, quais são os métodos que utiliza para fazer isso?

USUÁRIO: 1. Gerencio minha agenda manualmente em uma agenda física, marcando nos respectivos dias os compromissos e tarefas determinados. Poder criar tarefas e compromissos e ter lembretes dos compromissos marcados, além de conseguir olhar mais facilmente todos os compromissos da semana/mês (uma espécie de overview), 2. Não muito, normalmente uso a agenda como organizador pessoal mesmo

ENGENHEIRO: Muito bem! Vamos continuar com algumas perguntas adicionais para entender melhor suas necessidades.

- Você prefere receber notificações de seus compromissos através do celular, computador ou ambos? Como você gostaria que essas notificações fossem formatadas (por exemplo, e-mail, aplicativo móvel push notification)?
- Gostaria de ver algum tipo de integração com outros serviços ou plataformas (como Google Calendar, Outlook, etc.) no Calendarium? Se sim, quais delas seriam essenciais para você?

USUÁRIO: 3. Ambos, de preferência, mas prefiro por push do que por email, já que não o acesso com frequência e acho que o polui muito, 4. Sim, para o caso de importar agendas específicas (por ex: feriados do brasil) ou importar calendários que outras pessoas fizeram em outras plataformas (ex: calendário da faculdade feito no Google Agenda)

ENGENHEIRO: Ótimo! Essas informações são muito úteis. Vamos prosseguir com mais algumas perguntas.

- Você gostaria de ter a opção de criar eventos recorrentes automaticamente? Por exemplo, se você tem reuniões semanais em dias específicos e horas.
- Como você vê a possibilidade de visualizar sua agenda compartilhada em um calendário colaborativo com cores diferentes para identificar os compromissos de cada pessoa individualmente?
- Alguma vez você gostaria de incluir detalhes adicionais no evento, como localização ou anexar documentos importantes relacionados ao compromisso?

USUÁRIO: 5. Sim, seria muito útil para mim, pois tenho eventos recorrentes, como aulas da faculdade, que gostaria de ser alertado sobre. Mas, tem alguns eventos recorrentes que não preciso ser alertado sobre, então gostaria de poder desativar as notificações em alguns eventos, , 6. Acho uma feature legal se pudesse diferenciar os diferentes calendários importados, como citado anteriormente, assim fica mais fácil de visualizar que tipo de evento é, , 7. Sim, principalmente a parte de salvar documentos em algum evento de um dia, assim posso me lembrar do que foi passado em uma aula, por exemplo. Acho que também seria útil poder salvar documentos diferentes em um evento recorrente, assim todas as aulas ficam salvas no evento recorrente de aulas, por exemplo

ENGENHEIRO: Muito obrigado pelas suas respostas detalhadas! Elas serão muito úteis para o desenvolvimento do Calendarium. Aqui estão mais algumas perguntas:

- Você gostaria que fosse possível criar eventos em um calendário compartilhado e automaticamente enviar convites aos participantes com a opção de aceitar, talvez ou recusar?
- Como você vê a possibilidade de ter uma funcionalidade para filtrar sua agenda por tipo de evento (por exemplo, trabalho, pessoal, academia) ou por período específico (ex: eventos que ocorrem entre 16h e 18h)?
- Considerando a privacidade, você gostaria de poder definir eventos como públicos ou privados? Ou mesmo ter opções de compartilhamento mais granulares para controlar quem pode visualizar quais eventos?

Sua colaboração está sendo muito valiosa!

USUÁRIO: 8 - Sim, seria bem útil poder mandar isso por email e por uma notificação pra pessoa, assim ela pode ver mais fácil que foi convidada pra uma reunião, por exemplo 9 - Acho muito legal, principalmente a parte de filtrar por horários. 10 - Sim, assim posso deixar meus eventos pessoais privados e só compartilhar os que eu quero. Acho que seria interessante também poder compartilhar por um link um evento que eu criar e a outra pessoa acessar o link e poder entrar no mesmo evento, vendo as anotações, documentos e outras coisas compartilhadas

ENGENHEIRO: Muito obrigado pelas suas respostas detalhadas! Elas estão fornecendo uma visão clara de como você usa um calendário e quais funcionalidades considera importantes. Vou organizar esses requisitos para a equipe do Calendarium. Se precisar adicionar mais alguma coisa ou tiver novas ideias em mente, sinta-se à vontade para compartilhar! Até agora, conseguimos entender que você valoriza:

- Notificações por push e no dispositivo móvel.
- Integração com outros serviços de calendário como Google Calendar e Outlook.
- A capacidade de criar eventos recorrentes automaticamente.
- Visualização de diferentes calendários importados em cores distintas.
- Inclusão de detalhes adicionais nos eventos, incluindo anexos de documentos.
- Envio automático de convites para participantes com opções de resposta.
- Funcionalidades para filtrar a agenda por tipo e período específico.
- Configuração de privacidade e compartilhamento granulares.

Se tiver mais alguma pergunta ou quiser expandir algum desses pontos, estou à disposição!

USUÁRIO: 11 - Gostaria que ele funcionasse bem offline, sincronizando as alterações que faço posteriormente quando tiver acesso a rede, pois nem sempre tenho internet 12 - Gostaria que fosse possível personalizar o aplicativo para deixar ele do meu jeito, podendo mudar as cores dele, por exemplo 13 - Acho que seria legal que as tarefas no aplicativo funcionassem como um checklist, onde posso marcar o que já concluí e ver o que está pendente. Também que pudesse criar subtarefas dentro dessas tarefas, para dividir bem as coisas que tenho que fazer, como listas de compras, por exemplo 14 - Gostaria que tivesse uma versão de computador também, acessível pela web, pois uso diferentes sistemas operacionais no dia a dia e acho mais fácil entrar pelo navegador. 15 - Baseado no que disse acima, poder entrar com a conta Google e salvar tudo com esse login seria o jeito mais fácil pra mim

ENGENHEIRO: Muito obrigado pelas suas novas sugestões! Elas adicionam muita valiosa funcionalidade ao Calendarium. Aqui está um resumo das informações que você forneceu:

- Funcionamento Offline e Sincronização: Você gostaria de usar o calendário offline, com a capacidade de sincronizar as alterações feitas quando conectado à internet novamente.
- Personalização do Aplicativo: Ter opções para personalizar a aparência do aplicativo, como mudar cores e layouts.
- Gerenciamento de Tarefas: A funcionalidade de criar tarefas em formato de checklist com subtarefas para gerenciar melhor as atividades diárias.
- Versão Web do Aplicativo: Ter uma versão web acessível através do navegador, facilitando o uso em diferentes sistemas operacionais e dispositivos.
- Login Integrado ao Google: A capacidade de fazer login com a conta do Google para sincronizar todos os dados facilmente.

Essas informações adicionais serão extremamente úteis na evolução do Calendarium. Se você tiver mais ideias ou detalhes sobre esses pontos, por favor, sinta-se à vontade para compartilhar! Estamos aqui para garantir que o Calendarium atenda às suas necessidades e proporcionar a melhor experiência possível. Obrigado novamente pela sua colaboração!

USUÁRIO: Por favor, liste todas as características de sistema desejáveis para o Calendarium identificadas até então na nossa conversa.

ENGENHEIRO: Com base nas nossas discussões, aqui estão as principais características de sistema que você considerou desejáveis para o Calendarium:

- Notificações Personalizadas
  - Notificações por push em dispositivos móveis.
  - Formatação das notificações como notificações do aplicativo (push notifications) em vez de e-mail.
- Integração com Outros Serviços Integração com outros serviços de calendário, tais como Google Calendar e Outlook.
- Eventos Recorrentes
  - Criação automática de eventos recorrentes.

- Opções para desativar notificações em eventos recorrentes específicos.
- Visualização Colaborativa Visualização compartilhada com cores distintas para identificar compromissos individuais em um calendário colaborativo.
- Detalhes Adicionais nos Eventos Inclusão de detalhes adicionais (como local e documentos anexados) nos eventos.
- Convites Automáticos Envio automático de convites a participantes com opções para aceitar, talvez ou recusar.
- Filtragem da Agenda Funcionalidade para filtrar a agenda por tipo de evento (trabalho, pessoal, academia) e período específico.
- Privacidade e Compartilhamento Granulares
  - Definição de eventos como públicos ou privados.
  - Opções de compartilhamento granulares que permitem controlar a visibilidade do conteúdo para diferentes usuários.
- Funcionamento Offline e Sincronização Funcionalidade offline com sincronização automática das alterações feitas quando o dispositivo estiver conectado à internet novamente.
- Personalização do Aplicativo Opções de personalização, como mudar cores e layouts.
- Gerenciamento de Tarefas Funcionalidade de checklist para tarefas com subtarefas.
- Versão Web do Aplicativo Versão web acessível via navegador.
- Login Integrado Login integrado ao Google ou outros provedores de identidades, permitindo a sincronização fácil dos dados entre dispositivos e serviços.

Estas características abrangem uma variedade de necessidades para gerenciamento de tempo, colaboração, personalização e produtividade. Elas servirão como um ponto de partida robusto para o desenvolvimento do Calendarium, visando atender às suas expectativas e fornecer funcionalidades úteis que irão beneficiar os usuários em geral. Se houver mais detalhes ou características adicionais que você gostaria de ver incluídas no Calendarium, por favor, não hesite em compartilhar!

## A.2 GERAÇÃO DE RESUMO

Resumo da Entrevista sobre o Desenvolvimento do Calendarium Introdução: O objetivo desta entrevista é coletar requisitos e funcionalidades desejáveis para o desenvolvimento de um sistema de calendário chamado Calendarium. O usuário atualmente gerencia sua agenda manualmente em uma agenda física, com a necessidade de um serviço digital mais robusto que ofereça notificações automáticas, integração com outros serviços, visualização colaborativa e gestão de tarefas. Principais Requisitos e Funcionalidades Desejadas:

- Notificações Personalizadas
  - Notificações por push em dispositivos móveis.

- Formatação das notificações como push notifications (em vez de e-mail) para evitar poluição do inbox e garantir rápida acessibilidade.
- Integração com Outros Serviços Integração com serviços como Google Calendar, Outlook, etc., permitindo a importação de agendas específicas ou feriados nacionais.
- Eventos Recorrentes
  - Criação automática de eventos recorrentes (por exemplo, aulas da faculdade).
  - Opções para desativar notificações em certos eventos recorrentes quando necessário.
- Visualização Colaborativa Visualização compartilhada com cores distintas para identificar compromissos individuais em um calendário colaborativo.
- Detalhes Adicionais nos Eventos Inclusão de detalhes adicionais, como localização e anexos (documentos relacionados ao evento).
- Convites Automáticos Envio automático de convites a participantes com opções para aceitar, talvez ou recusar.
- Filtragem da Agenda Funcionalidade para filtrar a agenda por tipo de evento (trabalho, pessoal, academia) e período específico.
- Privacidade e Compartilhamento Granulares
  - Definição de eventos como públicos ou privados.
  - Opções de compartilhamento granulares que permitam controlar a visibilidade do conteúdo para diferentes usuários.
- Funcionamento Offline e Sincronização Funcionalidade offline com sincronização automática das alterações feitas quando o dispositivo estiver conectado à internet novamente.
- Personalização do Aplicativo Opções de personalização, como mudar cores e layouts.
- Gerenciamento de Tarefas Funcionalidade de checklist para tarefas com subtarefas para melhor gerenciamento de atividades diárias.
- Versão Web do Aplicativo Versão web acessível via navegador, facilitando o uso em diferentes sistemas operacionais e dispositivos.
- Login Integrado Login integrado ao Google ou outros provedores de identidade para sincronização fácil dos dados entre dispositivos e serviços.

Conclusão: Este resumo contém as principais funcionalidades desejadas pelo usuário para o Calendarium, destacando a necessidade de um sistema que seja flexível, colaborativo e personalizável. O desenvolvimento do Calendarium visa proporcionar uma experiência integrada e eficaz para gerenciamento de tempo e organização pessoal e profissional. Se houver mais detalhes ou características adicionais que o usuário gostaria de incluir no projeto, a equipe está pronta para incorporá-las na próxima fase de desenvolvimento.

### A.3 GERAÇÃO DE HISTÓRIA DE USUÁRIO

**História do Usuário:** Gerenciamento Eficiente de Compromissos com Calendarium  
**Como:** Maria Silva, uma estudante universitária e funcionária part-time que gerencia múltiplos compromissos pessoais e profissionais. **Quer:** Um sistema de calendário chamado Calendarium que a ajude a organizar sua vida acadêmica e profissional com eficiência e facilidade, oferecendo notificações personalizadas, integração com outros serviços, visualização colaborativa, gerenciamento de tarefas e outras funcionalidades. **Para:** Melhorar a produtividade ao evitar omissões em compromissos importantes, facilitar a comunicação entre colegas e familiares sobre eventos compartilhados e permitir que ela se concentre em suas tarefas diárias sem distrações desnecessárias. **Personagem Principal:** Maria Silva **Background:** Maria é uma estudante universitária do último ano cursando Ciência da Computação. Além disso, trabalha part-time como assistente administrativo em uma pequena empresa de desenvolvimento de software. Ela também participa ativamente de clubes acadêmicos e organiza eventos sociais com amigos. **Problemas Atuais:**

- **Manutenção Manual:** Maria mantém sua agenda em papel, o que é desafiador para gerenciar compromissos múltiplos sem confusão.
- **Comunicação Ineficiente:** Compartilhar e atualizar detalhes de eventos com colegas e familiares é um processo lento e propenso a erros.
- **Falta de Integração:** Não tem uma maneira fácil de integrar seus compromissos acadêmicos e profissionais em um único sistema.

**Necessidades:**

- **Notificações Eficazes:** Precisa ser lembrada de tarefas importantes sem distrações desnecessárias.
- **Visualização Colaborativa:** Deseja visualizar suas atividades em conjunto com os compromissos de outros, para facilitar a comunicação e evitar conflitos de horários.
- **Gerenciamento Flexível:** Necessita de um sistema que permita gerenciar tarefas diárias com facilidade, incluindo subtarefas e anexos relacionados.
- **Integração Ampliada:** Deseja ter um calendário que possa ser integrado com outros serviços existentes como Google Calendar ou Outlook.

**Historias de Usuário 1.** Como Maria Silva, quero receber notificações personalizadas para compromissos importantes no meu calendário, para evitar atrasos e esquecimentos sem poluir minha caixa postal.

- **Epic:** Gestão de Compromissos
- **User Story 1:** Eu como Maria Silva, quero ser informada por push quando um evento importante está prestes a começar ou terminar em meu calendário.
- **Critério de Aceitação:** As notificações devem ser configuráveis para diferentes tipos de eventos e devem ser enviadas diretamente ao dispositivo móvel sem precisar abrir o aplicativo.

2. Como Maria Silva, quero integrar meus compromissos do Google Calendar com meu calendário preferido, para ter uma visão única e consolidada dos meus dias ocupados.

- Epic: Integração de Serviços
- User Story 1: Eu como Maria Silva, quero importar minhas tarefas da faculdade diretamente do Google Calendar ao meu calendário preferido.
- Critério de Aceitação: O sistema deve permitir a sincronização bidirecional com o Google Calendar para garantir que as alterações feitas em um serviço sejam refletidas no outro.

3. Como Maria Silva, quero ver os compromissos dos outros participantes coloridos e distintamente em meu calendário compartilhado, para não ter conflitos de horários quando planejo eventos sociais ou reuniões com o clube acadêmico.

- Epic: Visualização Colaborativa
- User Story 1: Eu como Maria Silva, quero ser capaz de distinguir facilmente os compromissos de outros usuários em um calendário compartilhado através de cores distintas para cada usuário.
- Critério de Aceitação: Cada evento no calendário compartilhado deve ser mostrado com uma cor específica e única atribuída a quem o criou.

4. Como Maria Silva, quero ter detalhes adicionais em meus eventos do calendário como localização e anexos importantes para lembrar-me de onde estarei e dos documentos necessários que preciso levar.

- Epic: Detalhes Adicionais
- User Story 1: Eu como Maria Silva, quero adicionar anexos relevantes a meus eventos do calendário.
- Critério de Aceitação: Os anexos devem ser facilmente acessíveis a partir da visualização do evento e permitir que eu inclua arquivos importantes como documentos, fotos ou URLs.

5. Como Maria Silva, quero enviar convites automáticos para os participantes dos meus eventos, para facilitar a organização de reuniões e eventos sociais sem depender de e-mails ou mensagens individuais.

- Epic: Convites Automáticos
- User Story 1: Eu como Maria Silva, quero enviar convites automaticamente aos participantes do evento.
- Critério de Aceitação: O sistema deve permitir que eu envie convites com opções para responder se aceito, talvez ou recuso a presença.

6. Como Maria Silva, quero ter funcionalidades offline e sincronização automática para garantir que os dados do meu calendário estejam sempre atualizados quando estou sem conexão de internet, como durante viagens ou em áreas remotas.

- Epic: Funcionamento Offline

- User Story 1: Eu como Maria Silva, quero usar o aplicativo offline e ter certeza que as alterações feitas serão sincronizadas automaticamente quando eu tiver conexão de internet novamente.
- Critério de Aceitação: O sistema deve armazenar dados locais em um dispositivo sem conexão de internet e sincronizar automaticamente com o servidor quando a conexão é restabelecida.

7. Como Maria Silva, quero ter uma versão web do calendário para acessar facilmente os meus compromissos e tarefas em qualquer lugar, sem depender apenas dos aplicativos móveis.

- Epic: Versão Web
- User Story 1: Eu como Maria Silva, quero poder acessar meu calendário diretamente via navegador.
- Critério de Aceitação: A versão web deve ser funcional e oferecer a mesma facilidade que o aplicativo móvel.

Essas histórias de usuário fornecem uma visão abrangente das necessidades de Maria Silva no gerenciamento eficiente de seu tempo e compromissos. Elas podem servir como base para os engenheiros definirem os requisitos funcionais do sistema Calendarium, garantindo que seja um produto robusto e fácil de usar para usuários com múltiplas responsabilidades pessoais e profissionais.

## APÊNDICE B – CONTEÚDO DAS HISTÓRIAS DE USUÁRIO GERADAS NOS DEMAIS CASOS DO EXPERIMENTO

### B.1 CASOS PARCIALMENTE BEM-SUCEDIDOS

#### B.1.1 Primeiro Caso

História de Usuário: Gestão Eficiente de Compromissos com o Calendarium  
Personagem Principal:

- Nome: João Silva
- Função: Desenvolvedor Sênior em uma empresa de tecnologia.
- Contexto: João é responsável por gerir múltiplos projetos simultâneos, incluindo reuniões diárias com a equipe e colaboração externa.

Contexto do Problema: João precisa de um calendário que seja não apenas visualmente amigável para marcá-lo de forma eficiente, mas também robusto o suficiente para gerenciar as demandas complexas de seu trabalho. Ele enfrenta desafios com a atualização manual dos compromissos e a falta de uma solução integrada para lembretes e compartilhamento de eventos. História: Como um desenvolvedor sênior, eu quero usar o Calendarium para gerenciar eficientemente minhas reuniões diárias, pair programming e projetos em equipe, para que possa maximizar a produtividade sem precisar lidar com distrações desnecessárias ou conflitos de horário. Recursos Necessários:

- Interface Gráfica Intuitiva
  - Visualizar detalhes como horários, localização (online/físico), participantes e temas das reuniões.
  - Indicar automaticamente conflitos de horário em uma interface clara e fácil de entender.
- Compartilhamento Eficiente de Eventos
  - Compartilhar eventos e agendas com diferentes níveis de acesso (exclusivo, público, convidados).
  - Integrar facilmente o sistema para pair programming e reuniões de desenvolvimento em equipe sem a necessidade de controles manuais excessivos.
- Gerenciamento de Lembretes Separado
  - Criar um sistema de lembretes separado para tarefas diárias ou triviais que não interfiram na visualização dos compromissos importantes.
  - Personalizar facilmente o gerenciamento de notificações, definindo intervalos e tipos de alertas conforme necessário.
- Integração com Serviços Externos

- Integrar automaticamente convites de reunião via e-mail para simplificar o processo de marcação.
- Sincronizar com calendários externos (como Google Agenda) para uma visão geral mais abrangente das tarefas.
- Agendas Públicas com Controle Acessível
  - Criar eventos públicos para toda a organização ou comunidade, mantendo o controle preciso sobre quem pode visualizar e participar.
  - Gerenciar diferentes níveis de acesso aos eventos criados publicamente.
- Personalização Visual e Configurações Flexíveis
  - Ajustar temas visuais conforme preferências individuais.
  - Personalizar configurações de notificação para evitar sobrecarga informativa e garantir a relevância das mensagens recebidas.

#### Critérios de Aceitação:

- Interface Gráfica: O usuário deve ser capaz de visualizar todas as informações necessárias sobre um evento (horário, local, participantes) sem precisar navegar por múltiplas telas.
- Compartilhamento: Deve ser possível compartilhar eventos com diferentes níveis de acesso e recebê-los automaticamente no calendário do receptor.
- Lembretes Diários: O sistema deve permitir a criação e gestão de lembretes separados para tarefas diárias, sem interferir na visualização dos compromissos principais.
- Integração: Convites de reunião enviados via e-mail devem ser automaticamente adicionados ao calendário.
- Agendas Públicas: O usuário deve ter a capacidade de criar eventos públicos com diferentes níveis de acesso, protegendo o conteúdo compartilhado da visibilidade indesejada.
- Personalização Visual: Deve ser possível escolher temas e cores para personalizar a interface do calendário conforme o gosto pessoal.
- Configurações de Notificação Personalizadas: O usuário deve poder definir intervalos específicos, tipos de alertas (sinais sonoros, notificações push) e silenciar temporariamente notificações para momentos cruciais.

**Objetivos do Produto:** Ao implementar esses requisitos no Calendarium, João espera aumentar significativamente sua eficiência ao gerir múltiplas tarefas diárias. Isso inclui um tempo menor gasto em atualizações manuais e uma redução na sobrecarga informativa por meio de notificações personalizadas. **Conclusão:** A implementação do Calendarium com esses recursos permitirá que João, e outros usuários como ele, gerencie efetivamente seu calendário sem comprometer a produtividade ou a segurança dos dados compartilhados.

## B.1.2 Segundo Caso

História de Usuário: Criação de uma Agenda Dinâmica e Colaborativa

Título: Como um profissional ocupado, quero gerenciar minha agenda de maneira eficiente para não perder nenhum detalhe importante ou compromisso. Objetivo do usuário: Gerir eventos recorrentes, configurar lembretes personalizados, visualizar a agenda de forma clara e compartilhar informações com outros usuários de forma segura e integrada às ferramentas que já uso diariamente. Contexto: Maria é uma gerente sênior de marketing em uma empresa multinacional. Ela está constantemente lidando com múltiplas equipes, reuniões remotas e apresentações internacionais. Com a pressão do trabalho e viagens frequentes, Maria precisa de um sistema robusto que permita a ela gerir sua agenda de maneira eficiente, sem perder nenhum detalhe importante ou compromisso. Historias específicas:

- Gerenciamento de Eventos Recorrentes
  - Como Maria, quero poder configurar facilmente eventos recorrentes para reuniões diárias com a equipe e outras atividades frequentes.
  - Caso específico: Se Maria precisa fazer ajustes em um evento recorrente (como alterar o horário ou local de uma reunião semanal), ela deseja ser capaz de aplicar essa mudança automaticamente a todos os eventos futuros.
- Lembretes Personalizados e Automáticos
  - Como Maria, quero poder configurar múltiplos lembretes personalizados para cada evento (por exemplo, um lembrete por email com 3 dias de antecedência e um push notificação no dia anterior).
  - Caso específico: Durante viagens internacionais, quando a mudança de fusos horários pode confundir seus lembretes, Maria gostaria que o sistema enviasse avisos via e-mail e push para garantir que ela não perca nenhuma reunião.
- Visualização da Agenda
  - Como Maria, quero visualizar minha agenda agrupada por dia, semana ou mês para evitar perder detalhes importantes.
  - Caso específico: Quando estiver planejando uma viagem internacional com várias atividades ao longo de vários dias, Maria deve ser capaz de filtrar e ver todas as suas tarefas e reuniões em um só lugar.
- Compartilhamento e Colaboração
  - Como Maria, quero definir diferentes níveis de acesso para usuários que visualizam minha agenda (por exemplo, acesso público para reuniões da empresa, privado apenas para a equipe interna).
  - Caso específico: Quando precisa coordenar uma apresentação com uma equipe internacional, ela deve ser capaz de permitir que os membros da equipe externa vejam e editem seu próprio calendário sem comprometer a privacidade das informações confidenciais.
- Integrações com Outras Ferramentas

- Como Maria, quero sincronizar automaticamente meus eventos com ferramentas como Google Drive e Trello para adicionar anexos diretamente aos eventos (por exemplo, links de apresentações ou documentos).
- Caso específico: Quando planeja uma reunião que requer a discussão de um documento compartilhado em Trello, ela deve ser capaz de abrir o documento diretamente a partir do calendário sem ter que alternar entre aplicativos.
- Widgets para Dispositivos Móveis
  - Como Maria, quero usar widgets no meu smartphone e tablet para visualizar rapidamente minha agenda sem entrar no aplicativo.
  - Caso específico: Ao acordar cedo ou durante uma reunião remota, ela deve ser capaz de verificar rapidamente as próximas tarefas e reuniões usando um widget.
- Adição de Anexos
  - Como Maria, quero adicionar anexos (como links para documentos ou arquivos) diretamente aos eventos do meu calendário.
  - Caso específico: Quando planeja uma apresentação que requer a revisão de vários documentos em Drive e Trello, ela deve ser capaz de enviar os documentos necessários diretamente ao evento correspondente no calendário.

#### Critérios de aceitação:

- Maria deve poder configurar eventos recorrentes sem dificuldades.
- O sistema deve permitir ajustes personalizados nos lembretes e notificações para diferentes tipos de eventos.
- A visualização da agenda deve ser flexível, permitindo filtragem por categorias ou intervalos de tempo.
- Deve existir um mecanismo claro para definir níveis de acesso e colaboração com outros usuários.
- O sistema deve ser capaz de se integrar às ferramentas que Maria já usa regularmente (Google Drive, Trello) sem perder dados ou funcionalidade.
- Widgets devem estar disponíveis para dispositivos móveis, permitindo uma visão rápida da agenda.
- Anexos devem poder ser adicionados diretamente aos eventos via interface de usuário.

Essa história de usuário ilustra as necessidades específicas e desejos do nosso usuário fictício Maria em relação a um sistema de gerenciamento de calendário eficaz, flexível e seguro. Com base nessas demandas, os engenheiros podem extrair requisitos funcionais e não funcionais que serão necessários para desenvolver o Calendarium como uma solução líder no mercado.

## B.2 CASO FALHO

História de Usuário: Controle Granular e Compartilhamento Eficiente de Eventos em Calendarium

Como é a Experiência Atual? Como atualmente, o usuário gerencia seus compromissos pessoais e profissionais usando um calendário digital padrão. No entanto, quando deseja compartilhar eventos com amigos ou colegas de trabalho, ele enfrenta várias limitações:

- **Compartilhamento Simplificado:** O sistema de calendário atual permite apenas a criação de convites genéricos para todos os eventos, sem nenhuma opção de controle granular sobre quem pode visualizar e confirmar presença em quais eventos.
- **Acesso Automático:** Qualquer pessoa convidada tem acesso integral ao calendário compartilhado, o que não é sempre desejável quando apenas algumas pessoas devem ser autorizadas a participar ou apenas ver os detalhes do evento sem fazer alterações.
- **Notificações Desatualizadas:** O sistema atual de notificação envia lembretes padrão para todos os eventos compartilhados, mas não permite respostas diretas através das notificações e nem oferece um resumo dos próximos eventos na semana anterior ao evento.
- **Falta de Integração com Memórias:** Não há suporte para a criação ou compartilhamento de álbuns após os eventos, o que limita a capacidade do usuário de manter registros históricos detalhados e integrar esses registros com serviços de armazenamento na nuvem.

Como Deve Ser? Com Calendarium, o usuário espera uma experiência mais personalizada e eficiente para gerenciamento e compartilhamento de eventos. Aqui estão as principais características desejadas:

- **Controle Granular sobre Compartilhamento:**
  - O usuário deve ser capaz de criar convites específicos para grupos ou indivíduos, onde apenas alguns terão permissão para confirmar presença em determinados eventos.
  - As configurações de compartilhamento devem poder ser aplicadas automaticamente quando o usuário decide compartilhar um evento específico.
- **Configuração Flexível de Acesso:**
  - O sistema deve permitir a definição individual de regras para cada evento, onde certos eventos podem exigir confirmação de presença e outros não.
  - Isso inclui a capacidade de ajustar essas configurações para diferentes grupos ou indivíduos com facilidade.
- **Notificações Eficientes:**
  - O sistema deve enviar lembretes automáticos antes do início dos eventos compartilhados.
  - As notificações devem permitir que o usuário responda diretamente ao convite sem abrir a aplicação ou site de calendário.

- O usuário deseja receber um resumo semanal consolidado de todos os próximos eventos compartilhados.
- Compartilhamento e Manutenção de Memórias:
  - Após cada evento, o usuário deve ser capaz de criar e compartilhar álbuns digitais onde todos os participantes podem contribuir com fotos e vídeos.
  - O Calendarium deve suportar a integração com serviços de armazenamento na nuvem para facilitar a publicação direta dessas memórias.

Como Pode Ser Implementado? Para atender aos requisitos do usuário, o Calendarium pode ser projetado da seguinte forma:

- Sistema de Acesso Personalizado:
  - Desenvolver um sistema onde cada evento pode ter regras individuais para controle de acesso e confirmação de presença.
  - Implementar um mecanismo que aplique automaticamente essas configurações quando o usuário compartilha eventos.
- Configuração Flexível por Evento:
  - Criar um painel onde o usuário pode ajustar as regras de acesso para cada evento individualmente, tornando a gestão do calendário mais personalizada e eficiente.
- Sistema de Notificações Avançadas:
  - Integrar um sistema robusto de notificações que envia lembretes automáticos antes dos eventos.
  - Adicionar funcionalidades para respostas diretas a convites através de notificações e consolidar os próximos eventos em um único resumo semanal.
- Integração com Serviços de Armazenamento na Nuvem:
  - Implementar APIs que permitem a integração com serviços de armazenamento na nuvem para facilitar o compartilhamento e manutenção de memórias dos eventos.
  - Criar ferramentas dentro do Calendarium para gerenciar e colaborar em álbuns digitais após os eventos.

### Conclusão

O desenvolvimento dessas funcionalidades no Calendarium permitirá que o usuário tenha uma experiência mais completa, eficiente e personalizada ao gerir e compartilhar seus compromissos e momentos especiais. Isso incluirá a capacidade de controlar quem pode visualizar e confirmar presença em eventos com precisão, receber notificações personalizadas e manter registros históricos detalhados através da integração com serviços de armazenamento na nuvem. Essa história de usuário pode ser usada pelos engenheiros de software para extrair requisitos específicos do sistema necessários para implementar essas funcionalidades, incluindo aspectos como segurança, interação do usuário e integração com outros sistemas.