UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ (UTFPR)

BRUNO CEZAR VOLPATO LERIA

NEKOCHAT – CHAT PARA ADMINISTRAR SUPORTE AO CLIENTE INTEGRADO AO WHATSAPP

PONTA GROSSA

2023

BRUNO CEZAR VOLPATO LERIA

NEKOCHAT – CHAT PARA ADMINISTRAR SUPORTE AO CLIENTE INTEGRADO AO WHATSAPP

NekoChat - Chat for managing customer support integrated with WhatsApp

Trabalho de Conclusão de Curso de graduação apresentado como requisito para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas da Universidade Tecnológica Federal do Paraná (UTFPR).
Orientador: Prof. Dr. Richard Duarte Ribeiro
Coorientador: Prof. MSc. Vinícius Camargo Andrade

PONTA GROSSA
2023

BRUNO CEZAR VOLPATO LERIA

NEKOCHAT – CHAT PARA ADMINISTRAR SUPORTE AO CLIENTE INTEGRADO AO WHATSAPP

Trabalho de Conclusão de Curso de graduação apresentado como requisito para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas da Universidade Tecnológica Federal do Paraná (UTFPR).

Data de aprovação: 16/outubro/2023

Nome completo e por extenso do Membro 1 (de acordo com o Currículo Lattes)
Título (Especialização, Mestrado, Doutorado)
Nome completo e por extenso da instituição a qual pertence

Nome completo e por extenso do Membro 2 (de acordo com o Currículo Lattes)
Título (Especialização, Mestrado, Doutorado)
Nome completo e por extenso da instituição a qual pertence

Nome completo e por extenso do Membro 3 (de acordo com o Currículo Lattes)
Título (Especialização, Mestrado, Doutorado)
Nome completo e por extenso da instituição a qual pertence

PONTA GROSSA
2023

Dedico este trabalho à minha querida família,
que tem sido a minha âncora durante toda a
jornada da elaboração deste trabalho.

AGRADECIMENTOS

Primeiramente, gostaria de agradecer aos meus orientadores, o Prof. Dr. Richard Ribeiro e o Prof. MSc. Vinícius Camargo Andrade, pela sabedoria com que me guiaram ao longo desta trajetória. Suas orientações e conselhos foram cruciais para o sucesso deste trabalho.

Também quero deixar registrado o meu profundo reconhecimento à minha família. Sem o apoio incondicional de vocês, seria muito difícil vencer os desafios que surgiram no caminho. Mãe, pai, e todos os meus entes queridos, este trabalho é uma pequena homenagem à grandeza do apoio que vocês me proporcionaram. As noites em que abdicaram de meu tempo para me ouvir, as palavras de encorajamento nos momentos de dúvida, e a fé inabalável em meu potencial moldaram este projeto.

Por fim, expresso minha gratidão a todos que, de alguma forma, contribuíram para a realização desta pesquisa. Seja através de discussões inspiradoras, apoio moral ou até mesmo uma palavra amiga nos momentos de dúvida, cada um de vocês teve um impacto significativo.

Esta conquista não é apenas minha, mas de todos nós, e é um testemunho do poder da colaboração e do apoio mútuo. Obrigado por fazerem parte desta jornada.

RESUMO

O atendimento ao cliente evoluiu significativamente, passando de um mero meio para solucionar problemas para se tornar uma estratégia voltada para a construção de confiança e o estabelecimento de um diálogo contínuo com o consumidor. No entanto, a comunicação utilizada nesse atendimento deve ser acessível e fornecer ferramentas que permitam às equipes de suporte manter uma rotina eficaz. O objetivo deste projeto é desenvolver uma solução que integre um método de comunicação amplamente utilizado no país com uma aplicação que ofereça ferramentas para a gestão de conversas. Para isso, foi criado um aplicativo de atendimento que se integra ao WhatsApp, e essa integração oferece uma oportunidade otimizar o atendimento, distribuir a carga de trabalho entre as equipes e fornecer gerenciamento de tarefas relacionadas às conversas. Como resultado, é possível utilizar uma única ferramenta, vinculada a um único número de celular, para facilitar o atendimento ao consumidor de empresas que utilizam, ou desejam utilizar, o WhatsApp como principal meio de comunicação com seus clientes. Isso representa uma abordagem mais eficiente e conveniente para as empresas e uma experiência aprimorada para os consumidores.

Palavras-chave: atendimento ao consumidor; aplicativo web; whatsapp; gereciamento de tarefas.

ABSTRACT

Customer service has evolved significantly, going from just a way to solve problems to becoming a strategy focused on building trust and establishing an ongoing dialogue with the consumer. However, the communication used in this service should be accessible and provide tools that enable support teams to maintain an effective routine. The goal of this project is to develop a solution that integrates a widely used communication method in the country with an application that offers conversation management tools. To achieve this, it was created a customer service app that integrates with WhatsApp, and this integration provides an opportunity to optimize service, distribute the workload among teams, and provide task management related to conversations. As a result, it's possible to use a single tool, linked to a single phone number, to streamline customer service for companies that use or want to use WhatsApp as their primary means of communication with their customers. This represents a more efficient and convenient approach for businesses and an enhanced experience for consumers.
Keywords: customer service; web application; whatsapp; task management.

LISTA DE FIGURAS

LISTAGEM DE CÓDIGOS FONTE

SUMÁRIO

1 INTRODUÇÃO

Atendimento ao cliente, uma prática no mundo dos negócios, serve como uma ponte entre a empresa e seus consumidores. Sheth, Jain e Ambika (2020) explicam que o suporte ao cliente é uma função administrativa que lida com o atendimento à consultas de clientes, reclamações, devoluções de mercadorias e questões relacionadas a pagamentos, como a cobrança de contas em atraso. Surgindo com o intuito primordial de resolver inquietações dos clientes acerca de produtos ou serviços oferecidos, esse canal transcendeu suas funções iniciais. Não se trata apenas de solucionar problemas, mas também de ser um meio onde os consumidores podem expressar suas reclamações, sugestões e *feedbacks*.

Segundo Sheth, Jain e Ambika (2020), o suporte ao cliente desempenha um papel importante, servindo como um canal pelo qual os clientes mantêm uma conexão significativa com uma empresa. Além de fornecer soluções, o atendimento ao cliente busca estabelecer uma relação de confiança, promovendo um diálogo contínuo com o consumidor.

As companhias de tecnologia geralmente dispõem de um setor de atendimento ao cliente para sanar dúvidas ou problemas que os consumidores possam ter em relação aos produtos ou serviços. Kumar et al. (2017) explica que o objetivo principal dessa área é manter uma boa relação com o cliente, pois um cliente fiel não só ajuda as companhias a divulgarem novos produtos, como também os serviços, por indicação. Sharma (2012), por sua vez, constata que diversas pesquisas já demonstraram a existência de uma ligação entre o atendimento ao cliente, a sua satisfação e a fidelidade à marca.

Reconhecendo a imperatividade do atendimento ao cliente, o próximo passo é discernir como esse contato direto pode ser efetivamente conduzido. A questão então passa a ser: *quais são as melhores tecnologias atualmente disponíveis para otimizar esse contato direto?* A resposta à essa pergunta pode variar dependendo do perfil do público-alvo e da natureza do negócio, mas o objetivo permanece o mesmo: estabelecer um canal de comunicação eficiente e confiável com o consumidor.

Segundo pesquisa da Opinion Box citada por Salgado (2022), aplicativos de mensagens, em especial o WhatsApp, são amplamente usados por brasileiros para se comunicar com empresas. Do total de entrevistados, 80% afirmaram usar o WhatsApp para este fim. Em termos de preferências dos clientes, 78% dos entrevistados consideram o aplicativo ideal para esclarecer dúvidas e solicitar informações, 69% para suporte técnico e 53% para receber promoções.

Outra pesquisa realizada pelo IBGE (2022), apresenta que 155,2 milhões de brasileiros acima de 10 anos têm celular para uso pessoal, correspondendo a 84,4% dessa faixa etária em 2021 .

Diante da crescente importância do atendimento ao cliente, uma abordagem adequada pode não só fidelizar um consumidor, mas também estreitar laços, criando uma relação de confiança e lealdade entre empresa e cliente. No Brasil, um país de dimensões continentais e diversidade de hábitos, é essencial identificar os meios de comunicação mais eficazes para

atingir esse objetivo. As pesquisas citadas apontam que o WhatsApp se destaca, atualmente, como a principal ferramenta de comunicação entre os brasileiros, refletindo sua praticidade e imediatismo.

Vendo essa tendência, foi notada uma chance: criar um aplicativo de atendimento ao cliente que se integre ao WhatsApp. Esta solução não apenas capitalizaria na popularidade do aplicativo, mas também incluiria recursos adicionais para aprimorar a interação.

Sheth, Jain e Ambika (2020) comentam que um aplicativo de suporte pode auxiliar os consumidores a obterem a assistência de que necessitam com mais facilidade e celeridade. Isso conduz a um aumento dos níveis de satisfação com o produto ou serviço e auxilia na fidelização dos clientes. Eles também concordam com Muller (1991) que afirma que uma assistência bem estruturada pode tornar uma organização diferente das outras do mesmo segmento e proporcionar uma vantagem competitiva. Por fim, Chen e Popovich (2003) mencionam o auxílio na coleta de dados, onde um aplicativo de suporte pode recolher informações das dúvidas, problemas e opiniões dos clientes. Estas informações podem ser aplicadas para o aperfeiçoamento do produto ou serviço, identificação de tendências e padrões, e decisões tomadas com base em dados.

Resumindo, tal aplicativo tem a capacidade de promover uma melhor experiência ao cliente, otimizar processos, entregar atendimento sob medida e proporcionar uma vantagem no mercado.

A crescente popularidade do WhatsApp entre os brasileiros ressalta sua importância estratégica para negócios.

Com a evolução constante da internet e das ferramentas de desenvolvimento, é oportuno criar uma aplicação web que auxilie os profissionais a oferecer suporte via WhatsApp, tendo em vista os benefícios já mencionados.

1.1 Objetivos

A seguir, serão demonstrados os objetivos deste projeto:

1.1.1 Objetivo geral

Desenvolver um aplicativo de atendimento ao consumidor que utiliza troca de mensagens, recursos de administração e distribuição das conversas com os clientes entre os membros da equipe de suporte.

1.1.2 Objetivos específicos

Para atingir o objetivo geral são necessários os seguintes objetivos específicos:

- Levantar requisitos funcionais e não funcionais do aplicativo.
- Modelar o software.
- Implementar o software utilizando Node.js e Vue.js.
- Validar aplicação utilizando o método teste unitário.

2 REFERENCIAL TEÓRICO

No presente capítulo, serão apresentados e discutidos os conceitos, tecnologias e ferramentas que fundamentam este trabalho.

2.1 Atendimento ao consumidor

Bodet (2008) destaca que a fidelidade dos clientes é essencial em estudos de marketing e estratégias de gestão, principalmente devido à acirrada concorrência nas indústrias de serviços. O foco central é a relação cliente-empresa, que é vital na estratégia de marketing de relacionamento com o consumidor. Já Posselt e Gerstner (2005) argumentam que melhorar a satisfação do cliente potencializa sua fidelidade, impactando positivamente os lucros das empresas.

Uma maneira de avaliar a satisfação de um cliente com um serviço é pedir para ele compará-lo com um padrão ou expectativa que tinha antes de usufruí-lo (OLIVER, 1977; OLIVER, 1993; POSSELT; GERSTNER, 2005). O surgimento da Internet auxiliou estabelecimentos de venda on-line a automatizarem diversos serviços, o que dispensou a necessidade de uma comunicação pessoal entre as pessoas, uma vez que passou a ser possível fazer tudo por um computador (POSSELT; GERSTNER, 2005).

Estudos anteriores em ciência do comportamento indicam que o lapso de tempo após a prestação de um serviço pode interferir na opinião de um cliente a respeito dele. Quando é feito uma análise a satisfação geral com o serviço, os clientes podem notar uma maior relevância para os serviços pós-venda do que para os pré-venda devido a um “efeito de finalização do serviço” ou “efeito de recência”. Um efeito de recência é quando um indivíduo dá à informação vista depois um peso maior do que a vista antes (POSSELT; GERSTNER, 2005).

Os clientes terão a oportunidade de experimentar dois tipos diferentes de serviços: um que ocorre pré-venda e outro pós-venda. Uma maneira prática de diferenciá-los é examinar as dimensões do serviço antes de finalizar a aquisição e após receber o produto (POSSELT; GERSTNER, 2005).

Foi estimado que a satisfação pós-venda do cliente com a intenção de voltar a comprar é dez vezes mais relevante do que a satisfação pré-venda com a intenção de voltar a comprar, e que a satisfação pós-venda, em geral, é quinze vezes mais relevante do que a pré-venda. Diante desses efeitos de recência, foi recomendado que os estabelecimentos de venda on-line aumentem os recursos disponíveis no processo pós-venda para aumentar a hipótese do cliente realizar uma nova compra e aumentar as avaliações positivas gerais em relação ao produto (POSSELT; GERSTNER, 2005).

Os itens mais relevantes no processo pós-venda para concretizar uma nova compra são (POSSELT; GERSTNER, 2005):

1. O produto satisfez as expectativas;
2. A qualidade do atendimento ao consumidor;
3. A entrega do produto no tempo esperado;
4. A disponibilidade do produto desejado.

Em relação à avaliação do serviço em geral, os itens são os mesmos, mas a sua relevância é diferente (POSSELT; GERSTNER, 2005):

1. A entrega do produto no tempo esperado;
2. O produto satisfez as expectativas;
3. Atendimento ao consumidor;
4. A disponibilidade do produto desejado.

O atendimento ao consumidor é formado pelas tarefas de entrega, instalação, manutenção, financiamento, solução de dúvidas e tratamento de queixas que os clientes podem ter depois de uma aquisição e durante o uso de um produto ou serviço (SHETH; JAIN; AMBIKA, 2020).

Um dos princípios de gestão de qualidade, o de foco no cliente, considera a comunicação com os clientes uma das ações possíveis e apresenta uma série de benefícios relacionados à mesma. Como é possível observar, há quatro ações possíveis que são dedicadas ou incluem o atendimento ao cliente, sendo elas (ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS, 2015):

- Ter ciência das necessidades e expectativas atuais e futuras dos clientes.
- Planejamento, projeto, desenvolvimento, produção, entrega e suporte a produtos e serviços para atender às necessidades e expectativas dos clientes.
- Acompanhar e supervisionar a satisfação do cliente e tomar as ações adequadas;
- Gerir as relações com os clientes ativamente forma para alcançar o sucesso.

Os benefícios mostrados confirmam o que as companhias suspeitavam sobre a melhoria da satisfação do cliente, sendo um aumento do valor para o consumidor, da sua satisfação, da fidelidade, e da hipótese de repetição de negócios. Além disso, as empresas têm uma melhoria na reputação, uma ampliação da base de clientes e um aumento da receita e participação no mercado (ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS, 2015).

2.2 Aplicativos de mensagens instantâneas

Aplicações de conversa, também chamados de aplicativos de mensagens instantâneas. São sistemas que permitem a comunicação em tempo real entre os usuários. Eles permitem a troca de mensagens, geralmente em texto, imagens, áudios ou vídeos, com um ou mais usuários (PCMAG, 2023b).

Podem ser utilizados no trabalho ou na vida pessoal, além de poderem ser usados em uma grande variedade de dispositivos, como celulares, tablets, notebooks, e computadores desktops (META, 2021). Alguns deles têm recursos extras, como chamadas de voz e de vídeo, conversas em grupo, opção de compartilhamento de arquivos e conexão com outros servidores (MAIZE, 2020).

São divididos em dois grupos principais: aplicações de conversação independentes ou integradas com outro sistema. As independentes foram criadas apenas para trocar mensagens e podem oferecer as outras funcionalidades mencionadas acima. As integradas são construídas a partir de outro serviço, como plataformas de mídias sociais ou sistemas de e-mail (MAIZE, 2020).

Eles são uma forma conveniente e eficiente de se comunicar com outros em tempo real, tanto em suas vidas pessoais quanto profissionais (SALGADO, 2022).

A história dos aplicativos de conversa começa nos anos setenta, quando o primeiro sistema de conversação chamado Talkomatic foi criado no computador PLATO na Universidade de Illinois. Era uma sala de conversação que possibilitava a cinco pessoas conversarem entre si em tempo real. Talkomatic foi inovador para a sua época por ser a primeira vez que as pessoas puderam se comunicar entre si em tempo real via uma rede de computadores (WOOLLEY, 1994).

Em 1988, o Internet Relay Chat (IRC) foi criado, permitindo que as pessoas conversassem em tempo real através de uma conexão de servidores (REID, 1991). Os usuários podiam criar salas e convidar outros para participarem (SIMPSON, 2000). Ele se popularizou nos primeiros dias da internet e, ao longo dos anos, construiu o caminho para os aplicativos de conversas modernos, inovando com diversas funcionalidades.

Desenvolvido em 1996, ICQ permitiu aos seus usuários enviar mensagens entre eles, e em 1997 atingiu a marca de 1 milhão de usuários (TECHTUDO, 2019). Após isso, surgiram outros competidores no mercado, que marcam a história da Internet e a popularização desse tipo de comunicação. Alguns deles são o AOL Instant Messenger (AIM), o MSN Messenger, e o Yahoo Messenger.

Nos primeiros anos do novo milênio, as redes sociais começaram a surgir e a incorporar a tecnologia de troca de mensagens em seus novos sistemas. O Facebook lançou em 2011 o Facebook Messenger (MESSEGER, 2018) e o Twitter introduziu as mensagens diretas no ano de 2013 (PEREZ, 2015). Isso possibilitou que os usuários se comunicassem diretamente nas

redes sociais. Em anos recentes, aplicativos como WhatsApp[1], WeChat[2] e Line [3] conseguiram manter a marca de centenas de milhares de usuários no mundo todo, tornando-os uma parte importante de como as pessoas se comunicam diariamente (BAROT; OREN, 2015).

2.2.1 WhatsApp

Como comentado anteriormente, o WhatsApp tem domínio sobre o mercado mundial e nacional, tendo mais de cinco bilhões de downloads, somente na Google Play Store (GOOGLE PLAY STORE, 2023). O aplicativo está diariamente recebendo aproximadamente 100 bilhões de mensagens (SINGH, 2020), levantando a questão sobre quais tecnologias, designs do sistema, ou arquitetura dos servidores conseguem suportar tamanha demanda de troca de dados.

No que diz respeito à interface dos usuários, presente em aparelhos iOS, Android, desktop e *web*, as seguintes tecnologias são usadas em cada uma dessas plataformas (CRESSLER, 2021):

- Android: Java;
- iOS: Swift;
- Aplicativo Web: JavaScript/HTML/CSS;
- Aplicativo Mac Desktop: Swift/Objective-C;
- Aplicativo PC Desktop: C/C#/Java;

Outro fator relevante a ser considerado, além das linguagens de programação, é o banco de dados. O WhatsApp utiliza o SQLite, um banco de dados independente, autônomo e relacional que pode ser usado pelo aplicativo sem ser instalado no dispositivo. Para evitar custos adicionais, sejam financeiros ou computacionais, ao invés de baixar as conversas da nuvem toda vez que o serviço é iniciado, o WhatsApp utiliza esse banco para guardar as mensagens localmente (CRESSLER, 2021).

No lado do servidor, considerando-se o conhecimento que está disponível para o público, o design do sistema é como descrito a seguir (CRESSLER, 2021):

- Erlang é a linguagem de programação principal;
- FreeBSD é o sistema operacional;
- Ejabberd é a aplicação servidor;
- BEAM é a máquina virtual criada em Erlang;

[1] WhatsApp: https://www.whatsapp.com
[2] WeChat: https://www.wechat.com
[3] Line: https://line.me/en/

- Mnesia é o banco de dados criado em Erlang;
- YAWS é o servidor web de multimídia deles.

O aplicativo usa um protocolo diferente para se comunicar com os clientes, uma versão modificada XMPP. O protocolo abre um socket SSL nos clientes para os servidores do WhatsApp, e todas as mensagens enviadas para ele são colocadas em uma fila nos servidores até que o cliente as abra ou reconecte nesse socket para recebê-las. Após o cliente adquirir a mensagem com êxito, uma confirmação de sucesso é enviada novamente para o servidor. O servidor envia isso para o remetente original, informando que aquela mensagem foi recebida e adicionando as marcas de seleção ao lado da mensagem enviada (CRESSLER, 2021).

2.3 Processo de Desenvolvimento

Existem diversas maneiras de criar um aplicativo para atender às demandas ou necessidades de um determinado problema. Em geral, cada uma dessas maneiras segue os mesmos passos, a principal diferença é a quantidade e ordem de vezes que eles são executados ao longo do processo de construção (BUDGEN, 2003).

Alguns exemplos são o processo de desenvolvimento linear, incremental, reativo e orientado a reúso (BUDGEN, 2003; SOMMERVILLE, 2011). A utilização desses formatos não é restrita a um só, e muitas vezes, eles são usados em conjunto, especialmente para a criação de sistemas complexos (SOMMERVILLE, 2011).

O linear é mais indicado para projetos bem estruturados e com um longo prazo de entrega (BUDGEN, 2003). Este formato contempla as atividades básicas do ciclo de especificação, desenvolvimento, validação e evolução, representando cada uma como etapas diferentes, como: especificação de requisitos, projeto de software, implementação, testes, entre outros (SOMMERVILLE, 2011).

O formato incremental é usado em projetos que ainda precisam demonstrar a sua viabilidade, estabelecer uma posição no mercado de forma rápida, ou explorar um mercado já existente. Os projetos que usam esse formato seriam aqueles concebidos como um pacote de aplicações, como software de escritório, jogos, e sistemas operacionais. Além também daqueles que precisam de uma interação próxima dos usuários finais durante o seu desenvolvimento, sendo isso o principal motivo pela escolha deste formato para o projeto a ser construído neste trabalho (BUDGEN, 2003).

O reativo, que a evolução do sistema consiste principalmente na resposta do que o desenvolvedor considera como necessária. Muitos projetos de código-livre e sites são construídos por esse método (BUDGEN, 2003).

Por fim, tem-se o orientado a reúso. Esta abordagem é fundamentada na existência de um considerável número de componentes que podem ser reaproveitados. O processo de

formação do sistema se concentra na união desses elementos em um sistema existente, ao invés de partir do zero (SOMMERVILLE, 2011).

Como dito anteriormente, o formato de processo que será utilizado neste projeto será o incremental. A seguir, o processo será descrito com mais clareza para elucidar a sua utilização e detalhes.

2.3.1 Desenvolvimento Incremental

Uma das principais deficiências do processo de desenvolvimento linear é a necessidade de identificar e resolver logo no começo do projeto os itens necessários para o futuro sistema. Isso é inaplicável em diversos casos, e, apesar de técnicas sofisticadas para determinar esses requisitos, existem muitas dúvidas e possíveis lacunas (BUDGEN, 2003).

Outras áreas da engenharia resolvem isso criando um protótipo para analisar o problema e possíveis soluções. Contudo, quando se pensa em um aplicativo, esse conceito é mais difícil de ser definido com clareza. Para outras áreas da engenharia de software, um protótipo é a sua primeira versão, ou o formato de um modelo que será usado para fins de escala (BUDGEN, 2003).

2.3.1.1 História

A metodologia foi inicialmente desenvolvida por Walter Shewhart em 1930, o pioneiro do ciclo “Planejar, Fazer, Verificar, Agir“. Foi aplicada em projetos como o jato X-15 nos anos 1950 e o Project Mercury da NASA nos anos 1960 (LARMAN; BASILI, 2003). Larman e Basili encontraram a primeira menção do método em um relatório de 1968 de Brian Randell e F.W. Zurcher na IBM. No ano seguinte, M.M. Lehman divulgou essa metodologia em um relatório interno para a gerência da IBM. (LARMAN; BASILI, 2003).

2.3.1.2 Aplicação na engenharia de software

No que diz respeito a um aplicativo, onde a criação do produto final é apenas uma questão de fazer cópias, essas analogias são inválidas. Na produção deles, é provável que o protótipo seja o produto para o futuro, e não tem um conceito semelhante ao do modelo para a escala (BUDGEN, 2003).

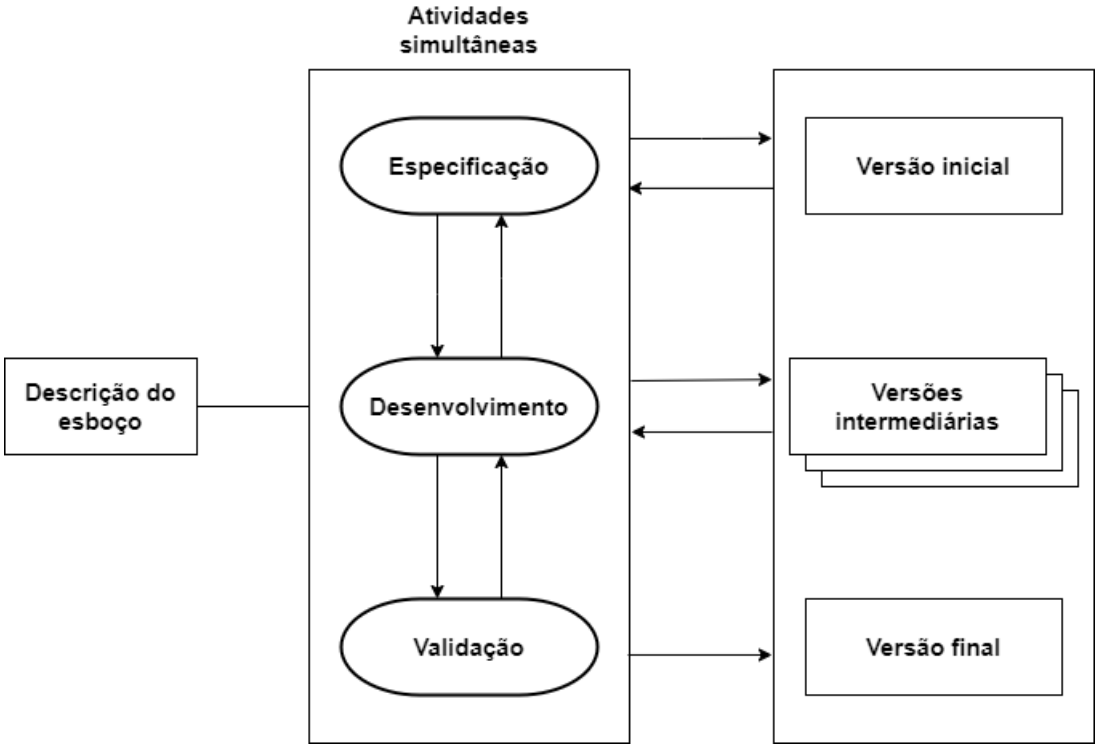
Apesar de a prototipagem diferir na construção de um sistema, a razão para a construção de um permanece a mesma: protótipos são criados para explorar uma ideia de forma mais integral do que outras maneiras (BUDGEN, 2003).

O aplicativo é aperfeiçoado gradualmente, e os requisitos são alterados com o tempo, ficando mais nítidos com o uso. Nesse sentido, modificou-se o sistema para se adequar a eles.

Dessa forma, concebeu-se o protótipo e, aos poucos, moldou-se ele até se chegar à versão final do produto (BUDGEN, 2003).

O desenvolvimento incremental é fundamentado no conceito de se elaborar uma versão inicial, submetê-la à avaliação dos usuários e prosseguir com a criação de diversas outras até se chegar a um sistema satisfatório. As tarefas de especificação, desenvolvimento e validação são executadas em rotatividade, sempre com um rápido retorno entre elas. Ao criar um programa de forma gradual, é mais econômico e prático fazer alterações no programa enquanto ele está sendo criado (SOMMERVILLE, 2011). A Figura 1 ilustra o ciclo do desenvolvimento incremental:

Figura 1 – Ilustração do desenvolvimento incremental



Fonte: Adaptado de (SOMMERVILLE, 2011).

Cada nova versão do sistema tem uma funcionalidade importante para o cliente. As primeiras versões, muitas vezes, apresentam a principal funcionalidade ou aquela que é mais urgente. Ou seja, o cliente pode experimentar o sistema numa etapa preliminar do processo de criação para verificar se oferece o que foi pedido. Em caso de discordância, apenas a funcionalidade desenvolvida terá que ser modificada naquele instante e, eventualmente, uma nova característica precisará ser estabelecida para os incrementos seguintes (SOMMERVILLE, 2011; SHERMAN, 2015).

É possível criar um sistema gradativamente e mostrá-lo para os clientes, sem precisar efetivamente instalá-lo e colocá-lo em funcionamento no ambiente do cliente. A entrega e a implantação gradativas significa que o software é utilizado em tarefas rotineiras. Isso nem sempre é viável, pois as experiências com o aplicativo podem interferir nos procedimentos de trabalho normais (SOMMERVILLE, 2011).

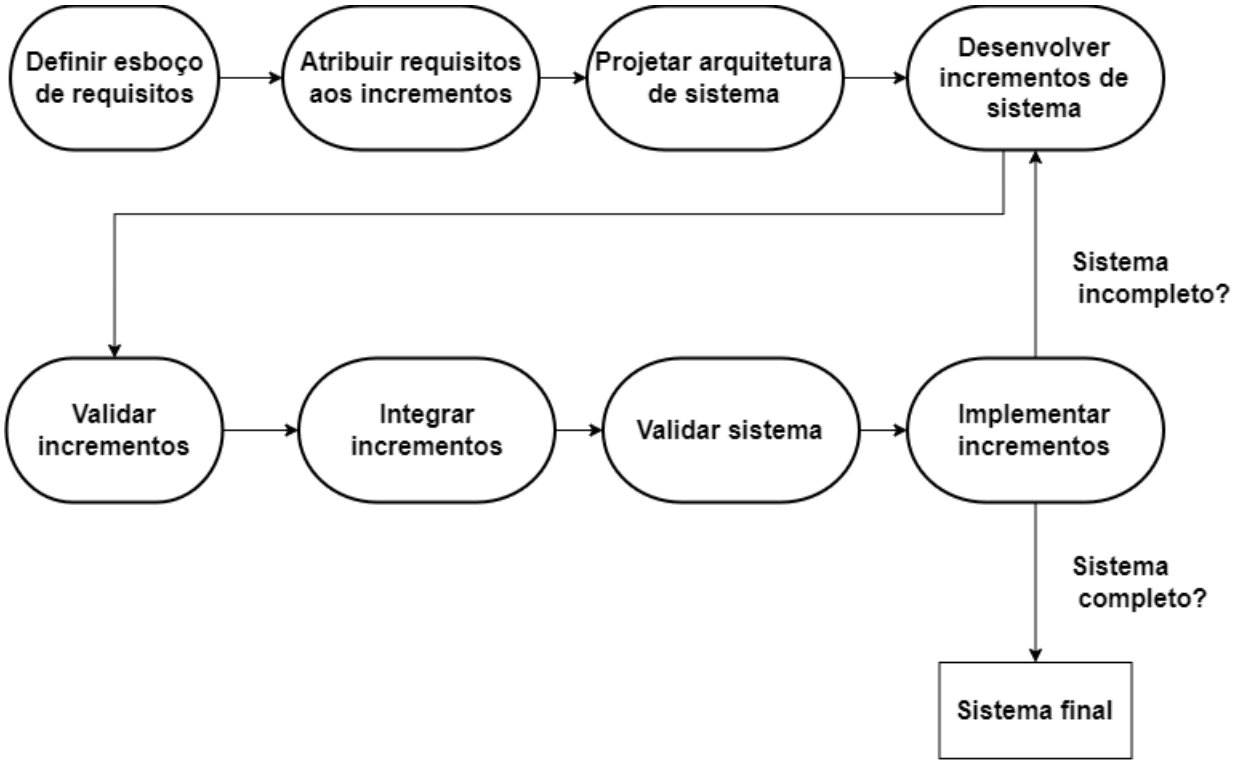
2.3.1.3 Entrega Incremental

Entrega incremental é uma forma de desenvolvimento de aplicações onde os usuários podem utilizar, identificar e classificar a importância das funcionalidades de uma nova aplicação em um ambiente operacional. A atribuição de serviços às novas funcionalidades depende da ordem de prioridade dos serviços (SOMMERVILLE, 2011).

Primeiramente, identifica-se quais módulos têm maior relevância para o uso. Após estabelecer as prioridades, é possível elaborar uma série de versões do sistema que contenham um subconjunto das funcionalidades. Dessa forma, a primeira versão já concordará com o que foi estabelecido. Nesse período, podem surgir algumas análises, adições ou mudanças de acréscimos futuros (SOMMERVILLE, 2011).

Ao finalizar esse desenvolvimento, o cliente inicia o uso das funcionalidades e, caso seja uma modificação posterior à primeira, verifica a sua compatibilidade com o resto da aplicação. Os clientes podem testar o aplicativo, o que os ajuda a compreender suas necessidades para futuras melhorias (SOMMERVILLE, 2011). A Figura 2 mostra o ciclo da metodologia:

Figura 2 – Ilustração da entrega incremental



Fonte: (SOMMERVILLE, 2011).

A entrega incremental apresenta diversas vantagens (SOMMERVILLE, 2011):

- Os clientes podem usar os primeiros aprimoramentos como protótipos e obter experiência, indicando seus requisitos para os aperfeiçoamentos futuros do sistema.

- Os clientes não precisam esperar até que todo o produto esteja finalizado para obter benefícios dele, pois, a partir da primeira versão, já podem utilizar as funções mais críticas do sistema.
- Se mantêm os benefícios do desenvolvimento incremental, o que deve facilitar a agregação das alterações no sistema.
- Quanto mais os serviços forem adicionados e, logo após, mais integrados, mais as funções críticas serão testadas. Isso significa que é menos provável que os clientes encontrem defeitos no software das partes essenciais do sistema.

Contudo, suas desvantagens são (SOMMERVILLE, 2011):

- A maioria dos sistemas necessita de um grupo de funcionalidades básicas, usadas por diferentes sessões do programa. Como os requisitos não são explicitados até que uma versão possa ser construída, pode ser difícil identificar recursos em comum, necessários para todas as melhorias.
- O desenvolvimento iterativo também encontra aversão quando é construído com intenção de substituir outro. Os usuários desejam todas as funcionalidades do sistema anterior e, em muitos casos, não estão dispostos a testar um novo sistema que está incompleto. Sendo assim, é difícil obter feedbacks relevantes dos mesmos.
- O processo iterativo é caracterizado pelo desenvolvimento da especificação em conjunto com o software. Todavia, isso gera problemas com o modelo de aquisições de diversas companhias, no qual a descrição completa do sistema é parte do acordo de criação do aplicativo.
- Na abordagem incremental, o aplicativo não é totalmente especificado até sua versão final. Isso pode exigir que clientes, como agências governamentais, façam novos acordos para se adaptar às mudanças.

Para resolver esses problemas e obter alguns dos benefícios do desenvolvimento incremental, pode-se usar um processo no qual um protótipo de sistema é criado de forma iterativa e usado como uma plataforma para experimentos com os requisitos e projeto do sistema. Com a experiência obtida com o protótipo, pode-se, então, chegar a um acordo em relação aos requisitos finais (SOMMERVILLE, 2011).

2.3.2 Engenharia de requisitos

Sommerville (2011) explica que requisitos de sistema especificam as funções, serviços e limitações de um sistema para atender às necessidades do cliente. “Esse processo de

identificação, análise, documentação e verificação é conhecido como *engenharia de requisitos*." (SOMMERVILLE, 2011).

Larman (2001) informa que os requisitos são as funções e condições que o sistema e o projeto como um todo devem atender. O desafio é identificar, comunicar e documentar essas necessidades de forma clara para o cliente e a equipe de desenvolvimento.

Ambos autores afirmam que é comum a separação dos requisitos em duas definições generalizadas:

- Requisitos funcionais: Sommerville (2011) diz que eles descrevem os serviços que o sistema deve oferecer, sua resposta a entradas específicas e seu comportamento em situações particulares. Eles também podem definir o que o sistema não deve fazer. Larman (2001) explica que eles são identificados e documentados no Modelo de Casos de Uso e na lista de características do sistema, que faz parte do artefato Visão.

- Requisitos não-funcionais: Sommerville (2011) afirma que eles são limitações que afetam o sistema inteiro, como tempo de resposta, regras de desenvolvimento e normas a seguir. Eles diferem dos requisitos funcionais, que geralmente se aplicam a funções ou serviços específicos do sistema. Larman (2001) expressa que eles podem ser documentados nos casos de uso relacionados ou no artefato de Especificações Suplementares.

Sommerville (2011) reconhece que a distinção entre diferentes tipos de requisitos é incerta, ao contrário do que sugerem essas definições. Ele exemplifica com um requisito sobre proteção, como limitar o acesso a usuários autorizados, que pode inicialmente parecer não funcional. Porém, ao explorá-lo mais, é possível considera-lo como um requisito funcional, como a implementação de recursos de autenticação.

2.3.2.1 Elicitação e análise de requisitos

Após os esclarecimentos sobre as definições de requisitos e suas classificações gerais, é necessário ver como levantar e analisar os requisitos para o sistema. Sommerville (2011) explica que nessa atividade, os engenheiros de software colaboram com clientes e usuários para coletar informações sobre o escopo da aplicação, serviços necessários, desempenho e limitações de hardware.

Sommerville (2011) acrescenta que esta fase do desenvolvimento pode incluir diversos tipos de pessoas. Ele apresenta o conceito de um *stakeholder* ou parte interessada, que é "quem tem alguma influência direta ou indireta sobre os requisitos o sistema." (SOMMERVILLE, 2011, p.70). Sommerville (2011) também diz que as partes interessadas pode envolver usuários finais, membros da organização afetados pelo sistema, engenheiros de sistemas relacionados, gerentes de negócios, especialistas de domínio e representantes sindicais.

Sommerville (2011) informa algumas dificuldades que é possível encontrar nesta fase do desenvolvimento, que são elas:

- Partes interessadas geralmente não têm clareza sobre o que querem do sistema e podem fazer pedidos inviáveis.
- Engenheiros podem ter dificuldade em entender requisitos expressos em termos específicos do domínio do cliente.
- Diferentes partes interessadas têm requisitos distintos, e pode haver conflitos e semelhanças entre eles.
- Fatores políticos, como a influência dos gerentes, podem afetar os requisitos do sistema.
- O cenário econômico e empresarial é dinâmico, levando a mudanças e novos requisitos ao longo do processo de análise.

2.3.2.2 Descoberta de requisitos

Esta é fase do desenvolvimento que é levantado o que as partes interessadas acreditam ser necessário para o sistema funcionar. Ela pode decorrer de diversas maneiras, como entrevistas, criações de cenários, observação do comportamento das partes interessadas com outros sistemas que serão integrados ou substituídos pelo novo. Sommerville (2011) explica essas diferentes formas, que são:

- Entrevistas: a equipe questiona as partes interessadas sobre o sistema atual e o que será criado. Essas conversas revelam os requisitos necessários. Existem duas abordagens que são as entrevistas fechadas, com perguntas predefinidas ou as entrevistas abertas, onde a equipe explora variados tópicos para entender melhor as necessidades das partes interessadas.
- Cenários: As pessoas costumam entender melhor exemplos práticos do que conceitos abstratos. Elas podem avaliar cenários de interação com um sistema, fornecendo ideias para os engenheiros de requisitos definirem os requisitos do sistema. Cenários ajudam a detalhar requisitos gerais através de exemplos de interações específicas. Cada cenário foca em um conjunto limitado de interações, fornecendo informações variadas e detalhadas sobre o sistema.
- Casos de uso: Um caso de uso nomeia a interação e identifica os atores envolvidos, fornecendo detalhes adicionais sobre a interação com o sistema. Larman (2001) acrescenta que eles ajudam a identificar e documentar requisitos funcionais através de narrativas que mostram como o sistema atende aos objetivos das partes interessadas.

- Etnografia: é uma técnica onde um analista observa o ambiente de trabalho para entender os processos e identificar requisitos do sistema. Ele foca nas tarefas reais e rotinas diárias, permitindo descobrir requisitos implícitos que podem não ser capturados por processos formais.

Diferentes fontes de requisitos podem ser vistas como perspectivas distintas do sistema, cada uma destacando um conjunto específico de requisitos. Embora essas perspectivas tenham particularidades, elas frequentemente se sobrepõem e trazem requisitos comuns. Essas perspectivas podem ser usadas para organizar a coleta e documentação dos requisitos.

2.3.2.3 Validação de requisitos

A *validação de requisitos* confirma se o sistema atende às reais necessidades do cliente e identifica problemas antecipadamente para evitar custos elevados de correção mais tarde. Sommerville afirma que ela "é importante porque erros em um documento de requisitos podem gerar altos custos de retrabalho quando descobertos durante o desenvolvimento ou após o sistema já estar em serviço." (SOMMERVILLE, 2011, p.76). Erros em requisitos são especialmente caros de corrigir, pois usualmente exigem mudanças no projeto e na implementação do sistema, além de retestes.

Sommerville (2011) descreve alguns diferentes tipos de verificações para o processo de validação de requisitos, que são:

- Validade: Analise se as funções necessárias estão realmente refletidas nos requisitos, considerando que diferentes partes interessadas podem ter diferentes necessidades.
- Consistência: Certifique-se de que os requisitos no documento não se contradizem.
- Completude: Verifique se o documento inclui todas as funções e restrições desejadas pelo usuário.
- Realismo: Com base na tecnologia disponível, no orçamento e no cronograma, confirme se os requisitos são viáveis.
- Verificabilidade: Os requisitos devem ser claros o suficiente para permitir testes que confirmem se o sistema atende às especificações.

É demonstrado algumas técnicas aplicáveis para certificar que os requisitos listados são validos para sistema que está sendo criado, que incluem:

- Revisões: Uma equipe examina os requisitos em busca de erros e inconsistências.
- Prototipação: Um modelo funcional do sistema é apresentado aos usuários e clientes para avaliar se ele atende às necessidades.

- Casos de Teste: Criar testes com base nos requisitos pode revelar problemas. Se for difícil projetar um teste, provavelmente o requisito também será difícil de implementar e deve ser revisto.

2.3.2.4 Gerenciamento de requisitos

Requisitos de software em sistemas grandes estão sempre em fluxo devido à natureza indefinida dos problemas que tentam resolver. “Uma razão para isso é que esses sistemas geralmente são desenvolvidos para enfrentar os problemas ’maus’— problemas que não podem ser completamente definidos.“ (SOMMERVILLE, 2011, p.77). À medida que as partes interessadas ganham novas perspectivas, os requisitos precisam ser atualizados.

Uma vez que o sistema está em uso, novos requisitos surgem com base na experiência do usuário e nas mudanças no ambiente técnico e de negócios. Isso inclui novos hardwares, regulamentos e mudanças nas prioridades empresariais.

Normalmente, afirma Sommerville (2011), os pagadores e os usuários finais do sistema têm requisitos diferentes, frequentemente conflitantes. “Clientes do sistema impõem requisitos devido a restrições orçamentárias e organizacionais, os quais podem entrar em conflito com os requisitos do usuário fina“ (SOMMERVILLE, 2011, p.78). Com o tempo, o sistema pode precisar de ajustes para equilibrar essas necessidades.

“O gerenciamento de requisitos é o processo de compreensão e controle das mudanças nos requisitos do sistema.“ (SOMMERVILLE, 2011, p.78). Isso envolve um processo formal que começa durante a fase de elicitação e continua através do ciclo de vida do projeto.

2.3.3 Casos de uso

“Escrever casos de uso — histórias sobre a utilização de um sistema — é uma excelente técnica para entender e descrever requisitos.“ (LARMAN, 2001, p.45). Casos de uso são um roteiro de como cada requisito levantado para o sistema ocorrerá, incluindo os seus atores, seu fluxo básico, suas extensões. Existem diferentes maneiras de representar um caso, e cada uma delas irá possuir um nível de detalhamento.

Segundo Larman (2001), eles são um mecanismo para ajudar a manter tudo simples e compreensível para todas as partes interessadas. Informalmente, eles são histórias de como utilizar um sistema para atingir objetivos. Fowler (2003) também da um descrição parecida sobre os casos de uso, ele diz que eles funcionam descrevendo as interações típicas entre os usuários de um sistema e o próprio sistema, fornecendo uma narrativa de como um sistema é utilizado. Sommerville (2011) diz que um caso de uso esboça as expectativas do usuário em relação a um sistema. Cockburn (2001) explica que um caso de uso é um acordo sobre como um sistema deve agir, envolvendo todas as partes interessadas.

> Um caso de uso captura um contrato entre as partes interessadas de um sistema sobre seu comportamento. O caso de uso descreve o comportamento do sistema sob várias condições enquanto responde a uma solicitação de uma das partes interessadas, chamada de ator principal. O ator principal inicia uma interação com o sistema para alcançar algum objetivo. O sistema responde, protegendo os interesses de todas as partes interessadas. Diferentes sequências de comportamento, ou cenários, podem se desenrolar, dependendo das solicitações específicas feitas e das condições que cercam essas solicitações. O caso de uso reúne esses diferentes cenários. (COCKBURN, 2001, p. 15).

Larman (2001) explica que os casos de uso são requisitos; principalmente, são requisitos funcionais que indicam o que o sistema fará. É como uma promessa de que o sistema, seja ele um site, aplicativo ou algo diferente, fará essas tarefas específicas. Fowler (2003) concorda afirmando que os casos de uso são uma técnica para capturar os requisitos funcionais de um sistema.

2.3.3.1 História

Larman (2001) conta que foi Ivar Jacobson que introduziu o conceito de casos de uso para descrever requisitos funcionais em 1986 e teve grande impacto devido à simplicidade e utilidade da ideia, e conclui que Alistair Cockburn aprofundou o tema, fornecendo um guia completo sobre como escrever casos de uso eficazes, iniciando seu trabalho em 1992.

Fowler (2003) concorda com Larman (2001), dizendo que Ivar Jacobson originalmente popularizou os casos de uso, porém o livro de Cockburn se tornou o livro padrão sobre o assunto.

Por fim Cockburn (2001) traz uma visão de quem presenciou a criação, informando que Ivar Jacobson inventou casos de uso no final dos anos 1960 enquanto trabalhava em sistemas de telefonia na Ericsson. Duas décadas depois, Jacobson os apresentou à comunidade de programação orientada a objetos, onde foram reconhecidos por preencher uma lacuna significativa no processo de desenvolvimento.

2.3.3.2 Partes do corpo de um caso de uso

Existem diversos componentes que é possível acrescentar em um caso de uso, sendo que alguns tem prioridade de definição maior do que os outros. Larman (2001) reformula as partes que foram apresentadas por Cockburn (2001) na seguinte forma;

- Atores: Podem ser qualquer entidade com comportamento, incluindo o próprio sistema em discussão, quando ele interage com outros sistemas. Atores têm objetivos e utili-

zam aplicativos para alcançá-los. Eles podem ser pessoas, organizações, softwares ou máquinas.

- Ator principal: A pessoa ou sistema que inicia a interação para atingir um objetivo.
- Lista de Partes Interessadas e Interesses: Um rol de quem se importa com o comportamento do sistema e o que eles querem dele.
- Cenário de Sucesso Principal: O fluxo típico que atende às necessidades das partes interessadas, geralmente sem desvios condicionais.
- Extensões: Outros cenários possíveis, incluindo sucesso e falha.
- Pré-condições: Condições que já devem ser verdadeiras antes de começar o cenário.
- Garantia de Sucesso: O que precisa ser verdadeiro quando o caso de uso é completado com sucesso, atendendo a todos os interessados.
- Requisitos Especiais: Qualquer requisito não funcional ou restrição específica a um caso de uso, como desempenho ou confiabilidade.
- Frequência de ocorrência: Intervalo de tempo em que o processo será repetido.
- Lista de Variações de Tecnologia e Dados: Notas sobre diferentes maneiras técnicas de fazer algo, mas sem mudar o objetivo.

2.3.3.3 Formatos de casos de uso

"Casos de uso são escritos em diferentes formatos, dependendo da necessidade." (LARMAN, 2001). Larman (2001) e Cockburn (2001) explicam que a escolha de como os casos de usos serão escritos depende de qual é o objetivo deles em relação a apresentação do projeto. Larman conclui que o essencial é escrever os detalhes do cenário de sucesso principal e suas extensões, de alguma forma. Aqui estão as formas descritas por Cockburn (2001) e por Larman (2001):

- Formato completamente vestido: é uma forma detalhada de apresentar casos de uso. Ele utiliza uma única coluna de texto, sem tabelas, e lista os passos em uma sequência numerada. Esse formato evita o uso de condicionais como "se" ("if statements"). Além disso, na seção de extensões, utiliza uma convenção de numeração que combina dígitos e letras para identificar sub passos ou alternativas (por exemplo, 2a, 2a1, 2a2, e assim por diante).
- Forma casual: formato de parágrafo informal. O primeiro parágrafo foca no cenário de sucesso principal e os próximos parágrafos abordam extensões.

- Tabela de uma coluna: é o formato completamente vestido, porém inserida em uma tabela.
- Tabela de duas colunas: Rebecca Wirfs-Brock criou o conceito de “conversação“, que se destaca visualmente pelo uso de duas colunas. As ações do ator principal ficam na coluna da esquerda e as do sistema na da direita. Esse formato é usado para preparar o design da interface do usuário, podendo conter mais detalhes sobre os movimentos do usuário.
- Estilo RUP: O Processo Unificado Racional utiliza um modelo bastante similar ao formato completamente vestido. Numerar as etapas é opcional. As extensões recebem suas próprias seções com títulos e são chamadas de fluxos alternativos.
- O diagrama de caso de uso UML: O diagrama de caso de uso, composto por elipses, setas e figuras de pauzinho, não é uma notação para descrever casos de uso. As elipses e setas mostram a organização e decomposição dos casos de uso, e não o seu conteúdo. Segundo Cockburn (2001), é possível decompor os casos de uso utilizando o diagrama, porém o diagrama de elipse está faltando informações essenciais, como qual ator está realizando cada etapa e notas sobre a ordem das etapas.

Apesar de não recomendarem nenhum dos formatos e afirmarem que a escolha de um deles depende do nível de detalhamento desejado, ambos autores demonstraram uma preferência em relação ao formato de completamente vestido. Cockburn (2001) informa que a maioria dos exemplos que ele descreveu em seu livro são utilizando este formato.

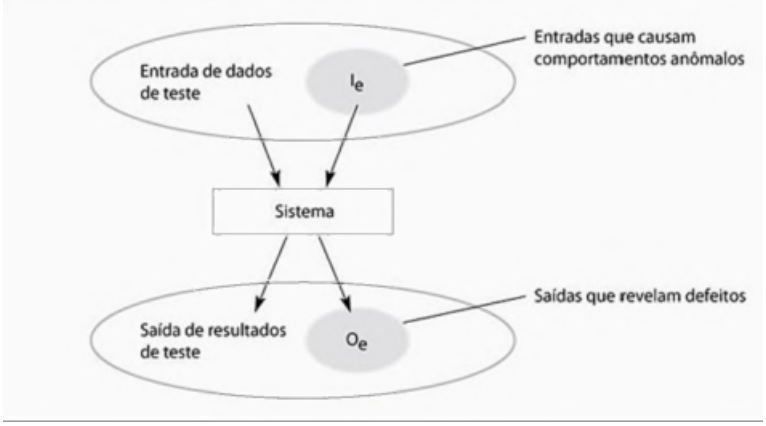
> Essa é minha prática, não uma recomendação. Por alguns anos, eu usei o formato de duas colunas devido à sua clara separação visual na conversa. No entanto, eu voltei para um estilo de uma coluna, pois é mais compacto e fácil de formatar, e o pequeno valor da conversa visualmente separada não compensa esses benefícios para mim. Acho que ainda é simples identificar visualmente as diferentes partes na conversa (Cliente, Sistema, ...) se cada parte e as respostas do Sistema geralmente são alocadas em seus próprios passos. (LARMAN, 2001, p. 54)

2.3.4 Testes

A etapa de testes no processo de desenvolvimento de um sistema consiste em validar que as funcionalidades implementadas mantenham os resultados esperados ao receberem diversos tipos de entradas. Sommerville (2011) diz que o teste serve para confirmar que um programa faz o que deve e identificar problemas antes de usá-lo. Nele, o software é executado com dados fictícios, e os resultados são examinados em busca de erros, anomalias ou questões relacionadas ao funcionamento do programa.

Sommerville (2011) apresenta a seguinte situação sobre a Figura 3, em um cenário hipotético, sreá considerado o sistema sob teste como uma caixa-preta. Este sistema é submetido a entradas de um conjunto predefinido e, em resposta, gera saídas correspondentes. Algumas dessas saídas podem ser incorretas e são classificadas no conjunto denominado Oe, resultante das entradas do conjunto Ie. A ênfase primordial nos testes de detecção de falhas reside na identificação das entradas presentes no conjunto Ie, uma vez que essas evidenciam problemas inerentes ao sistema. Em contrapartida, os testes de validação compreendem a utilização de entradas adequadas que não pertencem ao conjunto Ie, com o intuito de estimular o sistema a produzir resultados corretos.

Figura 3 – Ilustração de entrada-saída de teste de programa



Fonte: (SOMMERVILLE, 2011).

Sommerville (2011) diz que os dois objetivos do processo de teste são:

1. Demonstrar tanto ao desenvolvedor quanto ao cliente que o software está em conformidade com os requisitos estabelecidos. No caso de softwares personalizados, isso implica na realização de, pelo menos, um teste para cada requisito mencionado no documento de requisitos. No contexto de softwares genéricos, a abordagem implica na realização de testes abrangendo todas as características do sistema, incluindo suas diversas combinações, que serão integradas na versão final do produto.

2. Concentrar-se na identificação de situações em que o software apresenta comportamentos incorretos, indesejáveis ou discrepantes em relação às especificações. Tais comportamentos decorrem de defeitos no software. Os testes voltados à detecção de defeitos têm como principal foco a eliminação de comportamentos indesejáveis no sistema, como falhas críticas, interações não desejadas com outros sistemas, processamento inadequado e corrupção de dados.

Sommerville (2011) comenta que o primeiro objetivo orienta a realização de testes de validação, nos quais se espera que o sistema funcione corretamente quando submetido a um conjunto específico de casos de teste que representam o uso típico previsto para o sistema.

Já o segundo objetivo direciona os esforços para os testes de defeitos, nos quais os casos de teste são meticulosamente concebidos para expor eventuais defeitos no software. Segundo Sommerville (2011) , é importante ressaltar que não há uma fronteira rígida entre essas duas abordagens de teste. Durante os testes de validação, é possível identificar defeitos no sistema, enquanto durante os testes de defeitos, alguns casos de teste podem evidenciar que o programa está em conformidade com os requisitos estabelecidos.

Sommerville (2011) declara que os testes não garantem que o software seja perfeito ou que funcione exatamente como esperado em todas as situações. Sempre existe a possibilidade de que um teste omitido possa revelar problemas adicionais no sistema. Ele também cita Dijktra *et al.* que declara que “Os testes podem mostrar apenas a presença de erros, e não sua ausência.“ (1972 apud SOMMERVILLE, 2011, p. 145).

Sommerville (2011) categoriza os tipos de testes por períodos que um sistema de aplicativos comercial passa por, os três deles são:

- Testes durante o desenvolvimento, nos quais o sistema é avaliado enquanto está sendo construído, com o propósito de identificar erros e defeitos. Nessa fase, designers de sistemas e programadores podem estar envolvidos no processo de teste.

- Testes de liberação, nos quais uma equipe de teste independente avalia uma versão completa do sistema antes de ser disponibilizada aos usuários. O objetivo dos testes de liberação é garantir que o sistema esteja em conformidade com os requisitos estabelecidos pelos interessados no sistema.

- Testes de usuário, nos quais os usuários finais ou potenciais testam o sistema em seus próprios ambientes. Em produtos de software, o “usuário“ pode ser um grupo de marketing interno, responsável por decidir se o software pode ser comercializado, lançado e vendido. Os testes de aceitação são uma forma específica de teste de usuário, no qual o cliente avalia formalmente o sistema para decidir se ele deve ser aceito pelo fornecedor do sistema ou se requer desenvolvimentos adicionais.

Sommerville (2011) consta que na prática, o processo de teste geralmente envolve uma combinação de testes manuais e automatizados. Nos testes manuais, um testador executa o programa com um conjunto de dados de teste e compara os resultados com as expectativas, registrando e reportando quaisquer discrepâncias aos desenvolvedores do programa. Já nos testes automatizados, os testes são codificados em um programa que é executado sempre que o sistema em desenvolvimento é testado. Essa abordagem tende a ser mais eficiente do que os testes manuais, especialmente em casos de testes de regressão, nos quais testes anteriores são reexecutados para verificar se as alterações no programa não introduziram novos defeitos.

Sobre os testes de desenvolvimento Sommerville (2011) diz que eles englobam todas as atividades de teste conduzidas pela equipe responsável pelo desenvolvimento do sistema.

Geralmente, o testador de software é o próprio programador que criou o software, embora essa não seja uma regra inflexível.

Sommerville (2011) conta que durante o processo de desenvolvimento, os testes podem ser realizados em três níveis de granularidade distintos:

- Teste unitário: Nesse nível, as unidades individuais do programa, como classes de objetos ou métodos, são avaliadas separadamente. Os testes unitários têm como foco principal verificar a funcionalidade das unidades em questão.
- Teste de componentes: Nesse estágio, diversas unidades individuais são integradas para formar componentes compostos. Os testes de componentes concentram-se na avaliação das interfaces entre esses componentes.
- Teste de sistema: No nível de teste de sistema, ocorre a integração de algumas ou todas as partes de um sistema, e o sistema é avaliado como um todo. O objetivo principal do teste de sistema é verificar as interações entre os diversos componentes do sistema.

2.3.4.1 Testes unitários

Sommerville (2011) explica sobre que testes unitários é o processo de testar componentes do sistema individualmente, onde as funções e métodos são os componentes mais simples do programa. Explica ele também que “os testes devem ser chamadas para essas rotinas com parâmetros de entrada diferentes“ (SOMMERVILLE, 2011).

Sommerville (2011) consta que ao realizar testes em classes de objetos, é fundamental projetar os casos de teste de modo a abranger todas as características do objeto. Segundo ele isso implica nas seguintes ações:

- Testar todas as operações associadas ao objeto: Certifique-se de que todas as operações ou métodos vinculados ao objeto sejam testados.
- Definir e verificar o valor de todos os atributos associados ao objeto: Verifique se todos os atributos do objeto são definidos e verificados adequadamente nos casos de teste.
- Colocar o objeto em todos os estados possíveis: Isso requer a simulação de todos os eventos que podem causar mudanças de estado no objeto.

Sommerville (2011) declara que é importante definir casos de teste para cada um dos métodos associados ao objeto. O ideal é testar os métodos de forma isolada, embora em determinadas situações possa ser necessário criar sequências de teste específicas. Sommerville (2011) diz que não é suficiente testar uma operação apenas na classe onde ela é definida e presumir que funcionará corretamente nas subclasses que herdam a operação. A operação

herdada pode assumir premissas em relação a outras operações e atributos que podem não ser válidos em algumas subclasses que herdam a operação.

Segundo Sommerville (2011) em testes unitários automatizados, é comum utilizar um framework de automação de testes para escrever e executar testes em seu programa. Esses frameworks fornecem classes de teste genéricas que podem ser estendidas para criar casos de teste específicos. Eles são capazes de executar todos os testes implementados e reportar os resultados, frequentemente por meio de uma interface gráfica.

Sommerville (2011) diz que um teste automatizado é composto por três partes:

1. Parte de configuração: Nesta etapa, o sistema é inicializado com o caso de teste, ou seja, são definidas as entradas e as saídas esperadas.

2. Parte de chamada: Aqui, o objeto ou método a ser testado é invocado.

3. Parte de afirmação: Nesta fase, o resultado da chamada é comparado com o resultado esperado. Se a afirmação for verdadeira, o teste é considerado bem-sucedido; caso contrário, ele é considerado falho.

Em algumas situações, o objeto sob teste pode depender de outros objetos que ainda não foram implementados ou que podem atrasar o processo de teste, como no caso de chamadas a um banco de dados que envolvem processos demorados. Nestas circunstâncias, é possível optar por utilizar um "*mock objects*." Sommerville (2011) explica que *mock objects* são objetos que possuem a mesma interface que os objetos externos, mas são usados para simular seu funcionamento. Por exemplo, um *mock objects* que simula um banco de dados pode conter apenas alguns dados organizados em uma estrutura simples, permitindo acesso rápido, sem a sobrecarga de acessar um banco de dados real. Além disso, *mock objects* podem ser usados para simular situações anormais ou eventos raros.

Sommerville (2011) explica que os testes são um processo custoso e demorado, por isso é crucial selecionar cuidadosamente os casos de teste unitário. Segundo ele a efetividade nesse contexto implica em duas considerações:

1. Os casos de teste devem demonstrar que, quando o componente é utilizado de acordo com as expectativas, ele desempenha suas funções conforme especificado.

2. Caso haja defeitos no componente, esses problemas devem ser identificados pelos casos de teste.

Segundo Sommerville (2011) é importante criar dois tipos de casos de teste. O primeiro tipo reflete o comportamento normal do programa e tem como objetivo verificar o funcionamento adequado do componente. O segundo tipo de caso de teste baseia-se em cenários de teste que abordam os problemas mais comuns que surgem na experiência prática. Esses casos de

teste utilizam entradas anormais para verificar se o componente lida corretamente com essas situações e não falha.

Sommerville (2011) demonstra duas estratégias que podem ser eficazes na seleção de casos de teste:

- Teste de Partição: Nessa abordagem, os grupos de entradas que compartilham características semelhantes e devem ser tratados de maneira idêntica são identificados. A escolha dos testes é feita dentro de cada um desses grupos.
- Testes Baseados em Diretrizes: Essa estratégia se baseia em diretrizes de teste que refletem experiências anteriores sobre os tipos comuns de erros cometidos pelos desenvolvedores de componentes durante o desenvolvimento.

Frequentemente, os dados de entrada e os resultados de saída de um software podem ser classificados em diferentes classes com características em comum, como números positivos, números negativos ou seleções em um menu. Sommerville (2011) explica que programas tendem a se comportar de maneira consistente para todos os membros de uma mesma classe. Portanto, ao testar um programa que envolve cálculos e requer dois números positivos, diz Sommerville (2011) que é razoável esperar que o programa funcione de forma similar para todos os números positivos.

2.4 Ferramentas de Desenvolvimento Web

Há diversas opções de linguagens de programação e *frameworks* para serem escolhidos quando se pensa em criar um servidor para uma aplicação de conversas que usa uma API para comunicação (SHUERMANS; VOSKOGLOU, 2019; OVERFLOW, 2022). Algumas das opções de linguagens disponíveis são:

- **Node.js**. É um executor de código de JavaScript usado na criação de servidores. Ele provê diversas bibliotecas e ferramentas externas para a construção de uma API, como Express e Socket.IO. Node.js é especialmente adequado para aplicações em tempo real, como aplicativos de conversação, devido à sua arquitetura baseada em eventos.
- **Python**. É uma linguagem flexível que pode ser usada para diversos fins. Existem diversos *frameworks web* em Python que permitem a criação de APIs, como *Flask* e *Django*. O Python também tem várias bibliotecas para trabalhar com APIs, como *requests* e *aiohttp*.
- **Ruby on Rails**. É um *framework web* usado para a criação de aplicações *web*. Ele oferece diversos recursos úteis para a criação de APIs, como o *ActionCable* e o *ActiveModelSerializers*. Ele é eficiente para o desenvolvimento de aplicações em pouco tempo.

- **Java**. É uma linguagem de programação popular que é usada para a criação de aplicativos corporativos. Existem diversos *frameworks* que podem construir APIs, como *Spring* e *Jersey*. Ela é adequada para criar aplicações escaláveis e robustas.

- **Go**. É uma linguagem de programação recente e projetada para a criação de aplicativos de alto desempenho. Suporta a concorrência de forma nativa, o que a torna adequada para a construção de aplicações em tempo real, como um aplicativo de conversas. Além disso, há diversos *frameworks* que podem ser usados para criar uma API, como o *Gin* e o *Echo*.

Além disso, existem diversas opções para escolher como front-end. Alguns exemplos de escolhas populares são:

- **React**. Ele é um *framework* JavaScript usado para criar interfaces para usuários, sendo conhecido pelo seu desempenho e flexibilidade. O React dispõe de muitas bibliotecas e ferramentas para facilitar a criação de interfaces de usuário complexas e interativas, incluindo um DOM virtual, componentes, e ganchos. Ele é também leve, que significa que tem *uma pequena pegada* nos aplicativos e pode ser integrado facilmente com projetos já existentes. Pode ser usado com qualquer plataforma de servidor para criar uma aplicação de conversas que se comunica por API.

- **Angular**. É um *framework* JavaScript usado para criar aplicativos *web*. Ele fornece ferramentas úteis para a criação de interfaces complexas, como o vínculo de dados, a injeção de dependências e a programação reativa. Além disso, o *framework* também tem suporte para comunicação por API nativo. O *framework* é conhecido como completo, o que significa que ele tem tudo o que é preciso para criar uma aplicação inteira, como roteamento, formulários e testes. Ele foi criado e vem sendo mantido pelo Google, e possui uma grande base de programadores que o ajudam a evoluir.

- **Vue.js**. É um *framework* JavaScript usado para criar interfaces, sendo desenvolvido para ser simples e fácil de usar, permitindo que seja usado até por desenvolvedores novos na área de interfaces. Vue.js, tal como os dois anteriores, tem várias características que facilitam a elaboração de interfaces, como reatividade, diretrizes e componentes. Assim como o React, ele é leve, que significa que possui *uma pequena pegada* nos aplicativos e pode ser facilmente integrado a projetos já existentes.

- **Flutter**, É um *framework* para a criação de aplicativos para dispositivos móveis. A linguagem de programação *Dart* é utilizada e fornece uma série de *widgets* que permitem a criação de interfaces bonitas e responsivas. O *framework* também oferece suporte à comunicação por API nativo.

- **Ionic**. É um *framework* para criação de aplicativos móveis híbridos. Ele usa tecnologias *web*, como HTML, CSS e JavaScript para criar aplicações que funcionam tanto em iOS

quanto Android. O Ionic oferece diversas ferramentas e componentes úteis para criar uma aplicação de conversas, como componentes de UI e *plugins*.

Um conceito utilizado nas descrições anteriores de dois *frameworks* e precisa de mais explicações é o de *uma pequena pegada*. Quando um framework tem essa característica, significa que tem uma pequena quantidade de código e precisa de menos recursos para rodar do que o padrão. Isso traz alguns benefícios, como a rapidez e a eficiência, devidos à menor quantidade de código a ser executado, além de ser mais fácil de usar devido à menor quantidade de funcionalidades para aprender. Em suma, *uma pegada reduzida* significa uma diminuição do quanto de memória e poder de processamento é necessário para rodar o *framework*, o que pode ser relevante em ambientes que requerem uma restrição desses recursos (PCMAG, 2023a).

Resumidamente, cada tipo de linguagem ou *framework* tem as suas qualidades e fraquezas. A escolha ideal de um deles, tanto para um servidor quanto para a interface do usuário, para um aplicativo de conversas, dependerá dos requisitos específicos, habilidades e preferências do usuário.

Para este trabalho, foi escolhido o Node.js para a criação do servidor, e o Vue.js para a interface do usuário. A seleção foi baseada na familiaridade já existente com ambas as tecnologias, com a linguagem de programação JavaScript, e na facilidade de adição de recursos externos que o gerenciador de pacotes NPM, do Node.js, oferece.

2.4.1 Node.js(v18.16.0) e Express(v4.17.3)

JavaScript é uma linguagem de programação que é leve, interpretada ou compilada no momento (MDN, 2023b). Ela tem funções de primeira-classe, o que significa que elas são tratadas da mesma forma que qualquer outra variável (MDN, 2023c). O HTML é usado para armazenar o conteúdo e o formato de uma página da Internet. O CSS define a formatação e a aparência. O JavaScript é usado para tornar uma página *web* mais dinâmica e criar aplicações *web* mais complexas (MDN, 2023d).

Embora seja mais conhecida pelo seu uso em páginas da Internet, com o auxílio de ferramentas como o Node.js, Apache CouchDB e Adobe Acrobat, ela pode ser usada fora do ambiente de um navegador (MDN, 2023b).

Como executor de código de JavaScript assíncrono e direcionado a eventos, Node.js foi criado no V8 JavaScript *engine* (NODE.JS, 2023a), para criar aplicações que podem ser escaláveis na rede (NODE.JS, 2023b).

Node.js tem se tornado cada vez mais popular nos últimos anos como a escolha preferida para a criação de aplicações *web* em tempo-real e APIs, devido ao seu foco em eventos e ao seu modelo de não bloquear as entradas e saídas (MDN, 2023b). Isso possibilita que os desenvolvedores possam administrar diversos acessos de clientes simultaneamente, sem dificultar a execução do código.

Listagem 1 – Exemplo de função *middleware*

```
1 app.use((err, req, res, next) => {
2   console.error(err.stack)
3   res.status(500).send('Something broke!')
4 })
```

Fonte: Autoria própria (2023).

Ele apresenta uma grande variedade de bibliotecas construídas e módulos que podem ser facilmente integrados em uma aplicação *web* para lidar com diferentes tarefas, como um sistema de entrada e saída de arquivos, gerenciamento de rede, criptografia, e transmissão de dados. Além disso, disponibiliza um gerenciador de pacote chamado NPM (*Node Package Manager*), que possibilita aos programadores instalar, administrar e compartilhar com facilidade bibliotecas e módulos de outros criadores.

Ele é uma opção primorosa para a elaboração de microsserviços e aplicações *serverless*. O programador tem a opção de criar e implantar componentes pequenos, modulares e independentes que possam ser mantidos e escalados.

A tecnologia oferece muitas vantagens para um desenvolvedor de back-end. Sua capacidade de lidar com aplicações em tempo real e de grande escala, aliada à sua flexibilidade, escalabilidade e facilidade de integração com outras tecnologias, tornam-no uma opção popular na construção de aplicativos *web* modernos.

Como comentado anteriormente, no conjunto de ferramentas do Node, existe o NPM que permite a fácil integração de módulos externos na aplicação.

Express provê uma série de características robustas para aplicações *web* e móveis (EXPRESS.JS, 2023b). Ele permite que sejam criados administradores para requisições HTTP com diferentes tipos e rotas. Ele está integrado a motores de renderização para gerar respostas ao inserir dados em modelos (MDN, 2023a).

Além disso, é possível adicionar configurações comuns nos aplicativos *web*, como a porta usada para se conectar e a localização dos modelos usados para renderizar as respostas. Adicionalmente é possível adicionar *middleware* extra de processamento de requisição em qualquer ponto na *pipeline* da requisição (MDN, 2023a).

Funções de *middleware* são as que estão entre os objetos das requisições e respostas em uma aplicação que usa o protocolo HTTP para se comunicar. Podem ser empregadas para diferentes finalidades, tais como elaboração de relatórios, autenticação, tratamento de erros, dentre outras (EXPRESS.JS, 2023a). Na Listagem 1, tem-se um exemplo de como uma função *middleware* pode ser utilizada para tratamento de erros.

Outra vantagem do Express é a sua flexibilidade, já que ele não impõe nenhuma regra ou convenção, permitindo aos desenvolvedores estruturar suas aplicações conforme as suas necessidades. Essa característica também torna a integração com outras tecnologias e serviços mais simples, como bancos de dados, sistemas de armazenamento temporário e filas de mensagens.

Listagem 2 – Exemplo de código JS no Vue.js

```
import { createApp } from 'vue'

createApp({
  data() {
    return {
      count: 0
    }
  }
}).mount('#app')
```

Fonte: (VUE.JS GUIDE, 2023a).

Listagem 3 – Exemplo de código HTML no Vue.js

```
<div id=``app``>
  <button @click=``count++``>
    Count is: {{ count }}
  </button>
</div>
```

Fonte: (VUE.JS GUIDE, 2023a).

2.4.2 Vue.js(3.2.25)

Vue é um *framework* JavaScript para construir interfaces de usuários. Com base nos padrões HTML, CSS e JavaScript, ele cria um modelo de programação declarativo e baseado em componente que ajuda a desenvolver as interfaces de maneira eficiente, sejam elas simples ou complexas (VUE.JS GUIDE, 2023a).

Em seguida, as duas funcionalidades fundamentais do Vue (VUE.JS GUIDE, 2023a):

- Renderização declarativa: Vue estende o HTML padrão com uma sintaxe modelo que permite descrever de forma explícita a saída do HTML conforme o estado do JavaScript (VUE.JS GUIDE, 2023a).
- Reatividade: Ele localiza automaticamente as alterações no estado do JavaScript e atualiza o DOM de forma eficiente quando elas ocorrem (VUE.JS GUIDE, 2023a).

Esse é um *framework* e um sistema que contém a maioria das funcionalidades necessárias para um desenvolvimento de front-ends. Contudo, a Internet é bastante diversa, e as coisas criadas para ela podem variar muito em termos de tamanho e forma. Pensando nisso, Vue foi desenvolvido para ser flexível e incrementalmente adaptável (VUE.JS GUIDE, 2023a). Dependendo do caso de uso, Vue pode ser usado de diversas maneiras:

- Melhoramento do HTML estático sem uma etapa de construção.
- Incorporação de componentes em qualquer página;

Listagem 4 – Exemplo de um SFC no Vue.js

```
<script>
export default {
  data() {
    return {
      count: 0
    }
  }
}
</script>

<template>
  <button @click=``count++``>Count is: {{ count }}</button>
</template>

<style scoped>
button {
  font-weight: bold;
}
</style>
```

Fonte: (VUE.JS GUIDE, 2023a).

- Criação de aplicação de página única;
- Renderização no lado do servidor;
- Geração de sites estáticos;
- Geraçao de aplicativos para computadores *desktop*, móveis, WebGL, e até terminais.

Apesar da flexibilidade, todos esses casos de uso compartilham o principal conhecimento de como o Vue funciona. Mesmo para um iniciante, o conhecimento adquirido ao longo do caminho continuará a ser útil em objetivos futuros mais ambiciosos.

Se o desenvolvedor for um profissional experiente, pode escolher a forma mais adequada de aproveitar o Vue, conforme os problemas que está tentando resolver, enquanto mantendo a mesma produtividade. Por isso, é chamado de *framework progressivo*, pois ele é um *framework* que pode crescer com o desenvolvedor e adequar-se às suas necessidades (VUE.JS GUIDE, 2023a).

Na maioria dos projetos Vue, são construídos componentes utilizando-se um formato de arquivo similar ao HTML, chamado de *Componente de arquivo único* (*Single-File Component – SFC*). Como o nome sugere, ele encapsula a lógica do componente (JavaScript), o formato (HTML), e os estilos (CSS) em um único arquivo. SFC é uma das funcionalidades que definem o Vue e é a maneira recomendada para criar componente Vue (se for possível garantir uma configuração de construção) (VUE.JS GUIDE, 2023a). A Listagem 4 apresenta um exemplo de um arquivo escrito no formato de SFC:

Listagem 5 – Exemplo da Options API

```
<script>
export default {
  // Propriedades retornadas do data() se tornam reativas
  // e sao expostas pelo `this`.
  data() {
    return {
      count: 0
    }
  },

  // Methods sao funcoes que mutam os estados e ativam atualizacoes.
  // Eles pode ser vinculados a listeners no formato.
  methods: {
    increment() {
      this.count++
    }
  },

  // Ganchos do tempo de vida sao chamados em
  // diferentes estagios de um componente.
  // Essa funcao eh chamada quando um componente eh montado.
  mounted() {
    console.log(`The initial count is ${this.count}.`)
  }
}
</script>

<template>
  <button @click="increment">Count is: {{ count }}</button>
</template>
```

Fonte: (VUE.JS GUIDE, 2023a).

Os componentes podem ser criados usando dois estilos de APIs diferentes: a API Options e a API Composition (VUE.JS GUIDE, 2023a).

- Com a API Options, estabele-se a lógica do componente usando um objeto de configurações como “data“, “methods“ e “mounted“. Propriedades definidas por opções são apresentadas pelo “this“ nas funções, apontando para a instância do componente (VUE.JS GUIDE, 2023a).

- A API Composition define a lógica do componente usando funções da API importadas. Em SFCs, esse estilo é típico quando se usa o atributo <*script setup*>. O atributo *setup* que indica ao Vue realizar transformações em tempo de compilação que permitem utilizar o estilo com menos código padronizado. As importações, variáveis e funções declaradas no <*script setup*> são utilizáveis diretamente no formato (VUE.JS GUIDE, 2023a).

Listagem 6 – Exemplo da Composition API

```
<script setup>
import { ref, onMounted } from 'vue'

// estado reativo
const count = ref(0)

// funcoes que mutam os estados e ativam atualizacoes.
function increment() {
  count.value++
}

// Ganchos de ciclo de vida do componente
onMounted(() => {
  console.log(`The initial count is ${count.value}.`)
})
</script>

<template>
  <button @click="increment">Count is: {{ count }}</button>
</template>
```

Fonte: (VUE.JS GUIDE, 2023a).

Ambas as APIs, Options e Composition, são baseadas no mesmo sistema e compartilham seus princípios básicos. A API Options é, na verdade, construída sobre a API Composition. Isso as torna capazes de lidar com casos comuns (VUE.JS GUIDE, 2023a).

A API Options está centrada no conceito de uma instância do componente, o que, geralmente, se adéqua melhor a um modelo mental de classes para usuários que vem do modelo de linguagem de programação orientada a objetos. Além disso, ela é mais fácil de ser compreendida por iniciantes, ao evitar os detalhes de reatividade e dificulta a divisão do código em grupos de opções (VUE.JS GUIDE, 2023a). A Listagem 5 demonstra um exemplo da API Options.

A API Composition é composta pela declaração de variáveis que possuem um estado reativo em um corpo de uma função e um estado composto de múltiplas funções que, juntas, cuidam da complexidade. É um formato mais aberto e requer um entendimento de como a reatividade funciona no Vue para ser usado eficientemente. A flexibilidade do componente, por sua vez, permite criar padrões mais poderosos para organizar e reutilizar a lógica do componente (VUE.JS GUIDE, 2023a). A Listagem 6 demonstra um exemplo da API Composition.

Algumas recomendações para se escolher entre elas:

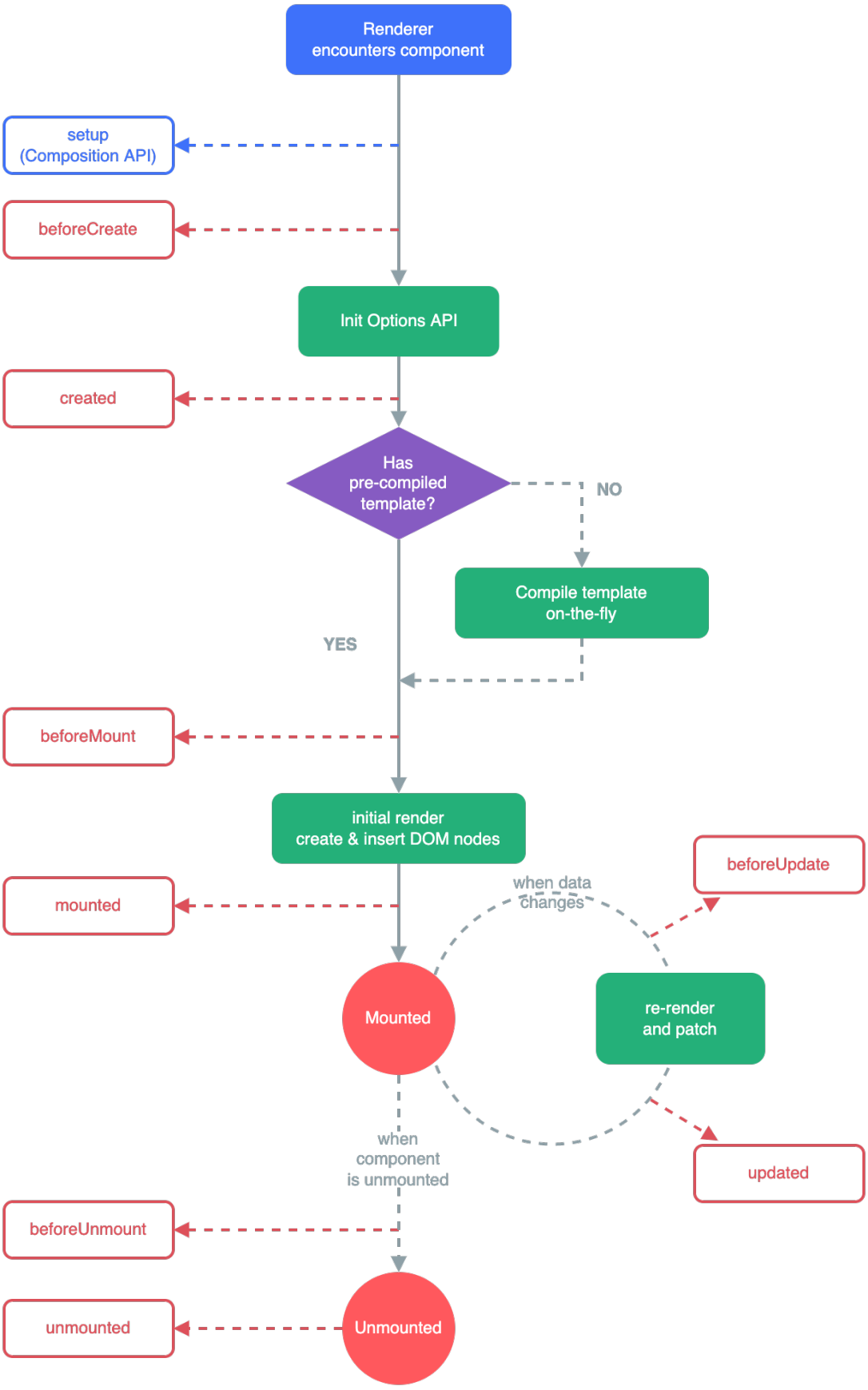
- Para fins de aprendizado, deve-se escolher o estilo que parece mais fácil de compreender. A maioria dos conceitos principais é, novamente, compartilhada entre os dois estilos, sendo assim, sempre é possível escolher outro estilo depois (VUE.JS GUIDE, 2023a).

- Para uso em produção

- Se não são usadas ferramentas de construção ou pretende-se usar o Vue somente em ambientes de pouca complexidade, deve-se escolher a API Options (VUE.JS GUIDE, 2023a).
- Se pretende-se criar aplicativos inteiros usando Vue, deve-se utilizar a API Composition com o SFC (VUE.JS GUIDE, 2023a).

Outro conceito relevante do Vue é o *ciclo de vida dos componentes*. Cada um deles segue uma série de etapas para iniciar quando são criados, como, por exemplo, *configurar um observador de dados*, *compilar o formato*, *montar a instância para o DOM*, ou *atualizar o DOM quando os dados mudam*. Durante essas etapas, ele também executa funções chamadas de *ganchos de ciclo de vida*, que permitem aos usuários adicionar seus próprios códigos em determinados estágios (VUE.JS GUIDE, 2023b). Na Figura 4 pode-se ver um ciclo de vida.

Figura 4 – Ilustração do ciclo de vida de um componente



Fonte: (VUE.JS GUIDE, 2023b).

3 DESENVOLVIMENTO

Neste capítulo, será detalhado o processo de desenvolvimento do aplicativo. Embora tenha sido adotado o modelo de desenvolvimento e entrega incremental, optou-se por apresentar o processo como se tivesse sido realizado no modelo cascata. Essa escolha se dá pelo fato de que a estrutura linear e sequencial do modelo cascata pode proporcionar uma compreensão mais clara e organizada do passo a passo envolvido na criação do sistema.

3.1 Levantamento de Requisitos

A fase de levantamento de requisitos tem como função compreender e documentar as necessidades e restrições das partes interessadas, compostas pelo time de suporte e o tech-lead. O objetivo é garantir a elaboração de um sistema que esteja alinhado e atenda a essas especificações.

3.1.1 Identificação das Partes Interessadas e Metodologia

Identificado-se que as partes interessadas seriam os times de suporte ao cliente. O método de chuva de ideias, que consiste em coletar ideias de um grupo direcionado, foi usado para explorar os potenciais requisitos.

As sessões foram conduzidas por uma equipe composta pelo desenvolvedor, o tech-lead, o gerente do time de suporte, e o analista de suporte com mais tempo de experiência no time. A equipe realizou diversas reuniões para conduzir uma chuva de ideias focada em melhorar o processo de atendimento aos clientes pelo WhatsApp. Durante essas sessões, o grupo foi incentivado a gerar ideias criativas, mantendo-se alinhado com o objetivo principal do projeto. Foram estabelecidos critérios claros para a seleção de requisitos, garantindo que qualquer ideia que ampliasse excessivamente o escopo ou se desviasse do princípio escolhido fosse desconsiderada, mantendo o foco no objetivo principal.

3.1.2 Lista de Requisitos

Será apresentada a primeira iteração da lista desenvolvida com as ideias coletadas e uma pequena descrição dela:

- **Sistema de Atendimentos**: Atribuir, rastrear e gerenciar conversas de clientes garantindo que nenhuma seja ignorada e que sejam atendidas prontamente.
- **Análise de Desempenho**: Gerar relatórios sobre membros individuais da equipe referentes ao tempo de resposta, tempo de resolução de atendimentos e *feedback* do

cliente. Analisar os períodos de pico das conversas dos clientes para otimizar a alocação de pessoal.

- **Auto-Atribuição**: Distribuir automaticamente as conversas recebidas para os membros da equipe disponíveis com base em sua carga de trabalho atual, especialização ou desempenho anterior.
- **Integração com Base de Conhecimento**: Equipar os membros da equipe com uma base de conhecimento constantemente atualizada para responder rapidamente a conversas comuns. Permitir que os membros da equipe contribuam e aprimorem esta base ao longo do tempo.
- **Módulos de Treinamento**: Implementar módulos de treinamento no software para integração de novos membros da equipe. Fornecer oportunidades contínuas de aprendizado para os membros existentes se manterem atualizados com mudanças no produto/serviço.
- **Chat Interno**: Facilitar a comunicação em tempo real entre os membros da equipe para que possam buscar assistência ou colaborar em conversas desafiadoras.
- **Pesquisas de *feedback* do Cliente**: Pesquisas pós-interação para coletar *feedback* dos clientes. Analisar *feedback* para identificar áreas de melhoria e elogiar serviços exemplares.
- **Automação de Tarefas**: Automatizar tarefas repetitivas (como enviar mensagens de reconhecimento) para reduzir a carga de trabalho manual e potenciais erros.
- **Caminhos de Escalonamento**: Definir procedimentos claros de escalonamento, garantindo que questões complexas ou sensíveis sejam encaminhadas aos indivíduos ou departamentos adequados.
- **Garantia de Qualidade**: Implementar ferramentas para que supervisores monitorem e avaliem a qualidade das interações de suporte. Permitir amostragem aleatória de interações para verificações de qualidade.
- **Gerenciamento de Escalas e Turnos**: Ferramentas para gerenciar os horários da equipe, garantindo cobertura ótima em períodos de pico. Fornecer alertas para horas extras ou não conformidade com leis trabalhistas.
- **Motivação & Reconhecimento**: Quadros de líderes para reconhecer membros da equipe de alto desempenho. Implementar estratégias de gamificação para motivar os membros da equipe a alcançar marcos de desempenho.

- **Integração com Outras Plataformas**: Integração com CRM, Gerenciamento de Inventário ou outras plataformas relevantes para fornecer aos representantes de suporte todas as informações necessárias ao alcance de um clique.
- ***feedback* Loop**: Permitir que membros da equipe forneçam *feedback* sobre o próprio sistema, sugerindo melhorias ou relatando problemas.
- **Recursos de Segurança**: Proteger os dados e conversas dos clientes. Garantir conformidade com regulamentos de privacidade.

3.1.3 Requisitos Rejeitados

Alguns dos requisitos listados foram desconsiderados devido à sua complexidade, sendo eles:

- **Integração com a base de conhecimento**: A implementação desta funcionalidade exigiria a criação de uma Wiki ou de um recurso similar. Isso envolveria um esforço significativo na configuração, manutenção e integração com o sistema existente. A complexidade e os recursos necessários para essa integração excedem o escopo e os objetivos do projeto atual.
- **Módulos de Treinamento**: Esse requisito compartilha desafios semelhantes com a integração da base de conhecimento. Seriam necessários recursos adicionais para desenvolver módulos de treinamento interativos e sua integração no sistema atual. A análise determinou que o benefício não justificaria o esforço adicional nesse momento.
- **Automação de tarefas**: A incerteza em relação à conformidade com as políticas do WhatsApp para automatizações de terceiros torna esse requisito arriscado. Sem clareza regulatória, a implementação poderia resultar em problemas legais ou técnicos, impactando negativamente o funcionamento geral do sistema ou até mesmo invalidar este trabalho de conclusão de curso.
- **Caminhos de escalamento**: A definição clara de procedimentos de escalonamento exigiria uma pesquisa mais profunda e específica, envolvendo diversas áreas e possíveis cenários. Isso prolongaria o desenvolvimento e poderia atrasar outras áreas críticas do projeto.
- **Gerenciamento de escala de turnos**: Essa funcionalidade representa um sistema separado e não se alinha com a intenção original do projeto, que se concentra mais no suporte ao cliente do que na gestão de recursos humanos. Portanto, foi decidido não prosseguir com essa integração.

- **Integração com outras plataformas**: Integrar com outras plataformas, como CRM ou Gerenciamento de Inventário, exigiria uma grande quantidade de trabalho de desenvolvimento e testes, além de possíveis acordos com provedores de terceiros. Essa complexidade vai além do escopo deste projeto e, portanto, essa integração foi excluída.
- **Recursos de segurança**: Embora a segurança seja uma consideração vital, a decisão foi criar apenas o mínimo necessário para garantir a segurança e a privacidade dos clientes. A implementação de recursos avançados de segurança exigiria uma análise de risco mais aprofundada, treinamento adicional e possivelmente hardware ou software adicional. Esses fatores seriam desproporcionais ao tamanho e ao escopo do projeto atual.

3.1.4 Categorização dos Requisitos

A seguir são categorizados os requisitos restantes em duas classes distintas. A primeira categoria, 'Necessário ter', inclui os requisitos indispensáveis para o funcionamento básico do sistema. Estes devem ser cumpridos sem exceção. A segunda categoria, 'Gostaríamos de ter', engloba funcionalidades que, embora não essenciais, enriqueceriam o sistema, proporcionando uma experiência de usuário mais robusta e completa.

1. Necessário ter:
    - Sistema de Atendimentos
    - Auto-Atribuição
    - Garantia de Qualidade
    - *feedback* Loop
2. Gostaríamos de ter:
    - Análise de Desempenho
    - Chat Interno
    - Pesquisas de *feedback* do Cliente
    - Motivação & Reconhecimento

O levantamento de requisitos proporcionou uma compreensão clara e concisa das necessidades e expectativas das partes interessadas. Este fundamento sólido nos permite avançar para a fase de modelagem, onde a estruturação de classes, tabelas de banco de dados e diagramas será vital para transformar estas necessidades em uma solução funcional.

3.2 Modelagem

A fase de modelagem marca o momento em que o projeto começa a emergir do conceitual para o concreto. Utilizando a fundação sólida estabelecida no levantamento de requisitos, a estruturação de classes, do banco de dados e dos diagramas desempenha um papel crucial na construção de uma solução funcional. É aqui que as ideias começam a tomar forma, guiadas pelas metas claramente definidas durante o levantamento de requisitos.

3.2.1 Casos de uso

O método escolhido para demonstrar os casos de uso será o completamente vestido, ou *fully-dressed*.

3.2.1.1 Caso de Uso 1: Sistema de Atendimentos

Ator Principal: Analista de suporte
Partes Interessadas e Interesses:

- Analista de suporte: Quer entrada de novas conversas com clientes rápido, e a possibilidade de rastrear e gerenciar-las.
- Cliente: Quer atendimento rápido com esforço mínimo. Quer solução esperada para o problema relatado na conversa.
- Administradores: Querem gerenciar e monitorar as conversas de suporte.
- Empresa: Quer registrar com precisão os atendimentos e satisfazer os interesses do cliente. Quer atualização automática e rápida do status da conversa com o cliente.

Pré-condições: O analista de suporte é identificado e autenticado.
Garantia de Sucesso (Pós-condições): Atendimento é concluído com sucesso. O cliente recebeu a solução adequada ao problema relatado. A conversa é registrada. A administração é informada sobre o status do atendimento encerrado.
Cenário de Sucesso Principal (ou Fluxo Básico):

1. O cliente envia uma mensagem para o atendimento ao cliente utilizando o WhatsApp.
2. O sistema registra a nova mensagem do cliente.
3. O sistema registra tenta cadastrar o cliente, caso ele não tenha um cadastro já feito.
4. O sistema mostra aos analistas de suporte a nova conversa
5. O analista recebe a mensagem do cliente e inicia uma nova tarefa.

6. O analista pede ao cliente as informações necessárias para iniciar um novo protocolo.
7. O cliente responde ao analista com todas as informações necessárias.
8. O analista registra no sistema as novas informações sobre o cliente na tarefa.
9. Os analistas perguntam ao cliente sobre o problema que ele está tendo com o produto.
10. O cliente fornece as informações necessárias para entender o problema.

    O analista e o cliente repetem os passos 9 e 10 até que o analista entenda o problema.
11. O sistema registra uma nova tarefa.
12. O analista relata ao cliente todas as informações necessárias para resolver o problema.
13. O analista pergunta ao cliente se há algo mais com o qual possa ajudar.
14. O cliente informa que não precisa de mais nada.
15. O analista informa ao cliente que está encerrando a conversa.
16. O analista informa ao sistema que a conversa e o problema foram resolvidos, e o que foi feito para resolver.
17. O sistema registra que a tarefa foi resolvida.

Extensões (ou Fluxos Alternativos):

*a. A qualquer momento, se a conexão com o WhatsApp encerrar:

1. O sistema guardará todas as conversas ativas.
2. O sistema marcará com uma bandeira, indicando que essas conversas têm prioridade.
3. O sistema informará aos atendentes quais são essas conversas prioritárias, para que eles possam retornar o chamado quando restabelecer a conexão.

3-14a. O cliente decide cancelar o atendimento:

1. Cliente avisa o analista para cancelar o chamado.
2. O analista pergunta o motivo.
3. Cliente informa ou não o motivo.
4. Caso o cliente informe um motivo, adicioná-lo nos detalhes do cancelamento para administração.
5. O sistema registra que a tarefa foi resolvida e informa o motivo (se houver).

3-14b. O analista decide suspender o atendimento:

1. O analista suspende o chamado.
2. O analista deve informar o motivo da suspensão do chamado.
3. O sistema registra que a tarefa foi resolvida e informa o motivo (se houver).

3-14c. O analista demora para responder o cliente:

Caso haja uma demora de resposta do atendente ou se ele deixar de encerrar o chamado, o sistema irá informar o atendente em 2 momentos:

1. O primeiro momento é na marca de 5 minutos, onde o sistema irá notificar o atendente, utilizando um elemento gráfico amarelo.
2. O segundo momento é na marca de 10 minutos, onde o sistema irá notificar o atendente, utilizando um elemento gráfico vermelho.

9-10. O analista não consegue entender o problema:

1. O analista iniciará o processo pedido de transferência para outro analista.
2. O analista informará a administração o motivo.
3. a. A administração transferirá o atendimento para quem eles acharem mais adequado. a1. O sistema apresentará ao novo analista a conversa com o cliente. b. A administração informará ao analista que o pedido foi negado.

Requerimentos especiais:

- Sem requerimentos especiais.

Lista de Variações de Dados e Tecnologia:

- Informações sobre o cliente.
- Informações sobre o problema do cliente.

Frequência de Ocorrência: Pode ser quase contínuo

Questões em Aberto: Qual será o processo para atualizar o sistema ou lidar com falhas técnicas? Existem leis ou regulamentos que precisam ser cumpridos?

3.2.1.2 Caso de Uso 2: Auto-Atribuição de Conversas

Ator Principal: Cliente

Partes Interessadas e Interesses:

- Analista de Suporte: Quer receber conversas que correspondam à sua especialização, carga de trabalho atual, e desempenho anterior.
- Equipe de Suporte: Quer distribuição justa e eficiente das conversas entre os membros.
- Administradores: Querem supervisar a distribuição de conversas, garantindo eficácia e eficiência.
- Empresa: Quer rápida resolução das conversas, satisfazendo os clientes e mantendo um registro preciso da distribuição.

Pré-condições: Conversas são recebidas pelo sistema. Membros da equipe estão disponíveis.

Garantia de Sucesso (Pós-condições): As conversas são distribuídas aos membros da equipe de acordo com a especialização, carga de trabalho atual e desempenho anterior. A distribuição é registrada.

Cenário de Sucesso Principal (ou Fluxo Básico):

1. O cliente inicia uma nova conversa.
2. O sistema recebe a conversa.
3. O sistema verifica os membros da equipe disponíveis.
4. O sistema avalia a especialização e a carga de trabalho atual.
5. O sistema atribui a conversa ao membro da equipe mais adequado.
6. O sistema registra a atribuição.
7. O membro da equipe recebe a conversa e começa a interação com o cliente.

Extensões (ou Fluxos Alternativos):

*a. A qualquer momento, se não houver membros da equipe disponíveis:

1. O sistema enfileira a conversa.
2. O sistema notifica os administradores sobre a falta de disponibilidade.

2a. Membro da equipe indisponível após a atribuição:

1. O sistema redistribui a conversa a outro membro da equipe.

Requisitos Especiais:

- O sistema deve ser capaz de avaliar a especialização, a carga de trabalho e o desempenho anterior dos membros da equipe.
- A distribuição deve ser realizada em tempo real ou próximo ao tempo real para garantir uma resposta rápida aos clientes.

Lista de Variações de Dados e Tecnologia: Não há nenhuma variação de dados ou tecnologia.

Frequência de Ocorrência: Pode ser contínua, dependendo do volume de conversas.

Questões em Aberto:

- Como o sistema avalia a especialização e o desempenho anterior?
- Quais são os critérios para a distribuição justa das conversas?
- Existe a necessidade de intervenção manual na distribuição?
- Como o sistema lida com a mudança de disponibilidade dos membros da equipe em tempo real?

3.2.1.3 Caso de Uso 3: Garantia de Qualidade

Ator Principal: Supervisor de Qualidade

Partes Interessadas e Interesses:

- Supervisor de Qualidade: Deseja ferramentas para monitorar e avaliar a qualidade do atendimento ao cliente.
- Analista de Suporte: Quer *feedback* sobre seu desempenho para melhorar futuras interações com o cliente.
- Administradores: Querem métricas confiáveis sobre a qualidade do suporte para tomar decisões informadas.
- Empresa: Quer garantir a máxima qualidade no atendimento ao cliente para manter ou aumentar a satisfação do cliente.

Pré-condições: Supervisor de Qualidade é identificado e autenticado.

Garantia de Sucesso (Pós-condições): Interações são revisadas e avaliadas. Relatório de garantia de qualidade é gerado e compartilhado com os interessados.

Cenário de Sucesso Principal (ou Fluxo Básico):

1. O supervisor acessa o sistema.

2. O sistema apresenta uma lista de conversas de todos dos os usuários cadastrados.
3. O supervisor seleciona uma interação para revisão.
4. O sistema carrega e apresenta os detalhes da interação selecionada.
5. O supervisor avalia a interação usando critérios predefinidos (tempo de resposta, qualidade da solução, etc.).
6. O supervisor opta por manter a conversa com o analista atual ou designar a conversa para outro.

Extensões (ou Fluxos Alternativos):
2a. O sistema não possui nenhuma interação guardada

1. O sistema alerta o Supervisor da falta de interações entre os analistas e os clientes.

Requerimentos especiais:

- Sem requerimentos especiais.

Lista de Variações de Dados e Tecnologia:

- Critérios de avaliação podem ser personalizados pelo Supervisor de Qualidade ou pré-definidos pela empresa.

Frequência de Ocorrência: A revisão e avaliação podem ser eventos diários, especialmente em grandes centros de suporte ao cliente.

Questões em Aberto:

- Como lidar com disputas entre o Supervisor de Qualidade e os Analistas de Suporte em relação às avaliações?
- Quais são os critérios específicos de avaliação?
- Há alguma forma de automação para tarefas repetitivas no processo de avaliação de qualidade?

3.2.1.4 Caso de Uso 4: *feedback* Loop

Ator Principal: Membro da Equipe
Partes Interessadas e Interesses:

- Membro da Equipe: Quer uma forma eficaz de comunicar problemas ou sugestões de melhorias no sistema.

- Administradores: Querem coletar *feedback* para aprimorar a eficiência do sistema e resolver problemas rapidamente.
- Desenvolvedores: Querem entender como suas alterações no sistema estão sendo recebidas e que ajustes precisam ser feitos.
- Empresa: Quer garantir que o sistema atenda às necessidades da equipe, para maximizar a eficiência.

Pré-condições: Membro da equipe é identificado e autenticado no sistema.

Garantia de Sucesso (Pós-condições): O *feedback* é registrado e notificado aos administradores e desenvolvedores, de acordo com a categoria designada.

Cenário de Sucesso Principal (ou Fluxo Básico):

1. O membro da equipe acessa a área de registro de "*feedback*" no sistema.
2. O sistema apresenta um formulário de cadastro para o membro da equipe preencher.
3. O membro da equipe preenche o formulário, informando todos os detalhes que acreditar ser necessário, e clica em "Salvar".
4. O sistema valida os dados inseridos.
5. O sistema registra o *feedback*.
6. O sistema notifica os desenvolvedores sobre a criação de um novo "*feedback*".
7. Os desenvolvedores decidem se irão ou não trabalhar no "*feedback*" registrado.

Extensões (ou Fluxos Alternativos):

6a. Dados inválidos ou incompletos:

1. O sistema notifica o membro da equipe sobre o erro.
2. O membro da equipe corrige os dados e reenvia.

Requisitos Especiais:

- O sistema deve ser capaz de categorizar e priorizar os *feedbacks*.

Lista de Variações de Dados e Tecnologia:

- Tipo de *feedback* (Problema ou Melhoria).
- Nível de prioridade do *feedback*.

Frequência de Ocorrência: Esporádico, dependendo das necessidades e experiências da equipe.

Questões em Aberto:

- Como os administradores e desenvolvedores tratarão esse *feedback* de forma eficaz?
- Existe um processo de revisão ou aprovação para as alterações propostas?

Embora ainda existam outros três requisitos que são desejáveis no sistema, foi decidido por omiti-los para evitar um aumento na complexidade da elaboração deste documento e, conforme citado anteriormente com base em Larman, os caso de uso tem como objetivo ser uma ferramenta para estabelecer promessas específicas sobre o que o sistema será capaz de fazer (LARMAN, 2001).

3.2.2 Diagramas de entidade-relacionamento

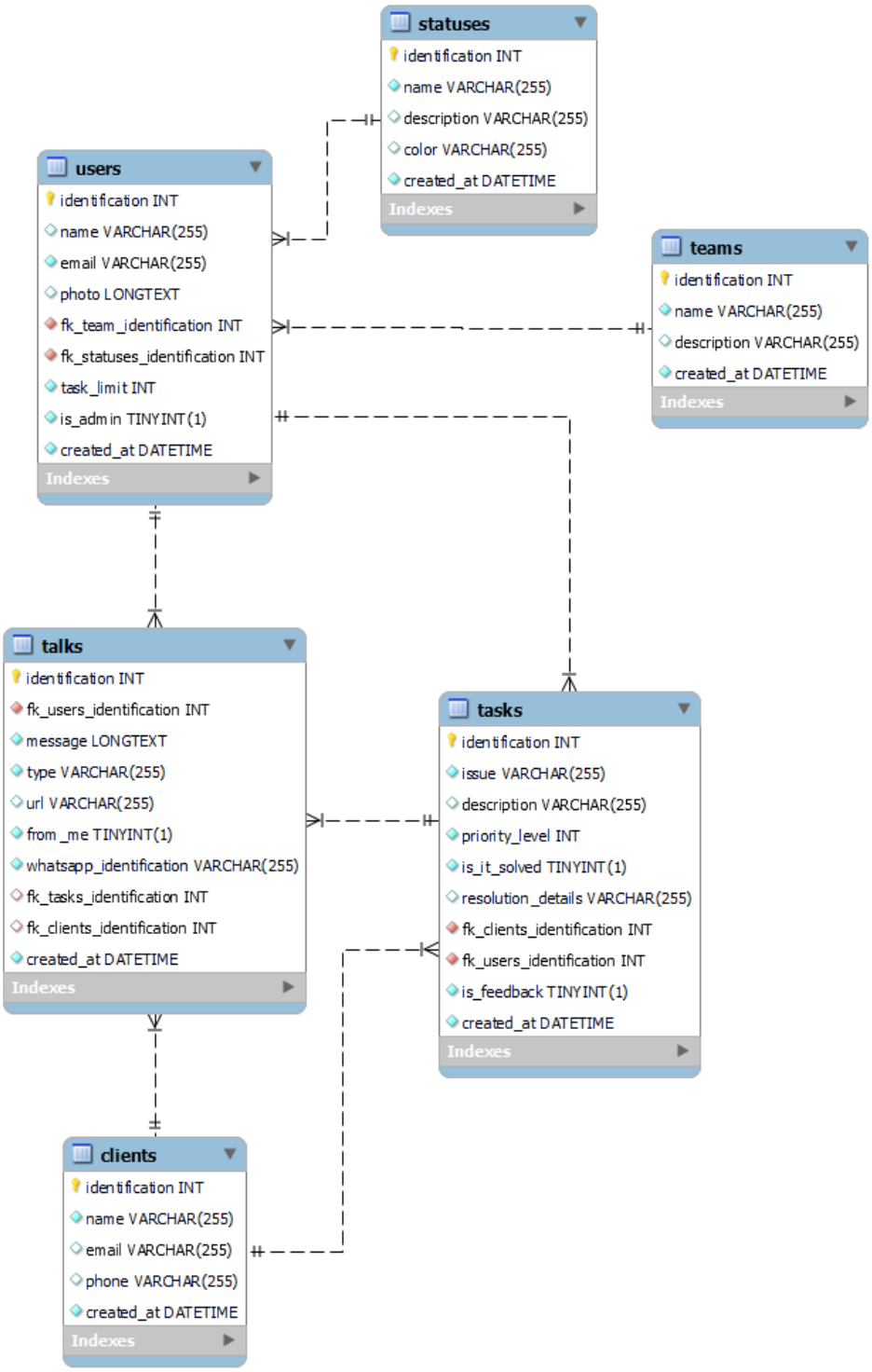
Após a fase de análise de requisitos, é iniciada a etapa de modelagem de dados. Conforme destacado por Hernandez, essa fase emprega métodos como diagramas de entidade-relacionamento (DER), modelagem de objeto semântico, modelagem de papel-objeto e UML para estruturar o banco de dados (HERNANDEZ, 2021).

Esta seção é dedicada à fase de modelagem do banco de dados, que será implementado em MySQL. O foco aqui é na definição dos campos e sua associação com as tabelas correspondentes. Cada tabela terá seus elementos essenciais, como chaves primárias, diferentes níveis de integridade de dados e respectivos relacionamentos.

O diagrama, demostrado na Figura 5 na página 55, é composto por seis tabelas: Usuário, Times, Status, Clientes, Conversas e Tarefas.

- Usuário: Cada usuário deve possuir um único status e deve pertencer a um único time.
- Status: Um status pode estar associado a múltiplos usuários, mas cada usuário só pode ter um status.
- Times: Semelhante ao status, um time pode conter vários usuários, mas cada usuário pertence a apenas um time.
- Conversas: Estas devem estar ligadas a um usuário e podem opcionalmente estar ligadas a um cliente ou a uma tarefa.
- Tarefas: Cada tarefa deve pertencer a um usuário e a um cliente obrigatoriamente.
- Clientes: Podem estar associados a conversas e tarefas, mas essa associação é opcional para conversas e obrigatória para tarefas

Figura 5 – Ilustração do Diagrama de Entidade-Relacionamento do Banco de Dados



Fonte: Autoria própria (2023).

3.3 Desenvolvimento

Um projeto de grande porte pode ser decomposto em diversas partes menores, cada uma com seu próprio ciclo de planejamento, design, construção e teste. Essas partes menores são conhecidas como iterações e têm uma duração fixa, por exemplo, quatro semanas. Ao final de cada iteração, uma parte funcional do sistema é entregue. Esse processo é semelhante à construção de uma casa, em que, em vez de construir tudo de uma vez, os cômodos são construídos um de cada vez, com a certeza de que cada parte está correta antes de avançar para a próxima. Esse método é conhecido como desenvolvimento iterativo (LARMAN, 2001).

3.3.1 Autenticação

Na seção de modelagem, ao abordar os casos de uso, observou-se que todos eles exigem que o usuário esteja autenticado no sistema. Portanto, a seção de desenvolvimento começa com essa sequência fundamental de ações no aplicativo. Para simplificar o processo de autenticação, utilizou-se a ferramenta *Firebase Authentication*, desenvolvida pelo Google.

3.3.1.1 Login

Na Figura 6 da página 57, é possível visualizar a tela de Login, criada usando os recursos gráficos do *Tailwind*. Aqui, o usuário é solicitado a inserir seu e-mail e senha previamente cadastrados.

No caso em que o usuário insira um e-mail não registrado ou digite uma senha incorreta, o sistema exibirá uma mensagem acima do campo de e-mail, em vermelho: “Usuário não encontrado. Por favor, contate o suporte em caso de problemas com o servidor.“

Além disso, o sistema oferece dois *hyperlinks*, um para o registro no sistema e outro para a recuperação de senha.

Figura 6 – Ilustração da tela de Login

Fonte: Autoria própria (2023).

3.3.1.2 Register

A tela de registro solicita três informações básicas para a autenticação inicial: endereço de e-mail, senha e confirmação de senha. Além disso, incorporou-se o *plugin* chamado “zxcvbn“ para avaliar a qualidade da senha. Para que uma senha seja considerada válida, ela deve conter no mínimo 4 caracteres, incluindo pelo menos uma letra maiúscula, uma letra minúscula e um número. A Figura 7 da página 58, ilustra essa tela de registro:

Erros possíveis durante o preenchimento e envio do formulário nesta página são indicados em vermelho e incluem:

- Senha não atende aos requisitos mínimos: “A senha deve conter pelo menos 4 caracteres, 1 letra maiúscula, 1 letra minúscula e 1 número.“;
- Senha na caixa “Confirmar senha“ não coincide com a senha inserida no campo “Senha“: “As senhas não coincidem.“;
- E-mail já cadastrado inserido no campo “E-mail“: “Este e-mail já está em uso.“;
- E-mail inválido no campo “E-mail“: “Este e-mail não é válido.“;
- Erro de permissão ao cadastrar um novo usuário no *Firebase Authentication*: “Não é possível criar usuários.“;

Figura 7 – Ilustração da tela de Registro

Fonte: Autoria própria (2023).

- Senha enviada para o *Firebase Authentication* é considerada fraca: “A senha é muito fraca.“

3.3.1.3 ForgotPassword

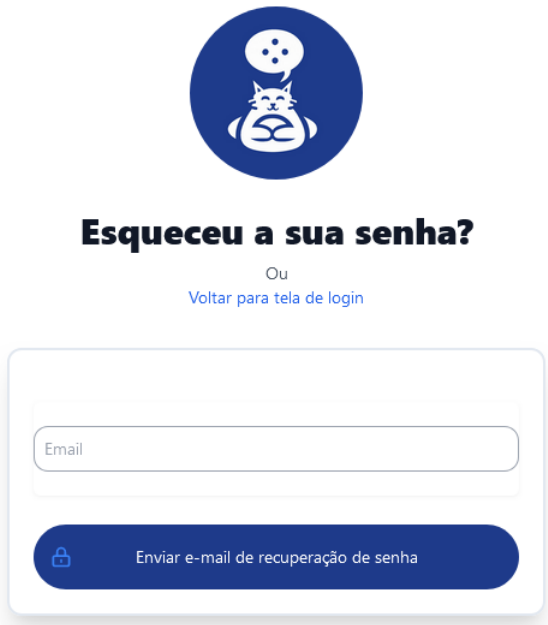
A tela de recuperação de senha possui apenas um campo para inserção do e-mail. A Figura 8 da página 59, seguinte ilustra essa tela:

O processo de recuperação de senha é administrado pelo *Firebase Authentication*. A ferramenta envia um e-mail para o endereço informado, caso ele esteja cadastrado, contendo um link de redirecionamento para a tela de configuração de senha, mostrada na Figura 9 na página 59. Nessa nova tela, o usuário define sua nova senha, e a ferramenta realiza a substituição das senhas.

Se o e-mail for enviado com sucesso, o sistema mostrará a seguinte mensagem em verde: “E-mail enviado com êxito. Por favor, verifique sua caixa de entrada. Redirecionando para a tela de login em 5 segundos.”

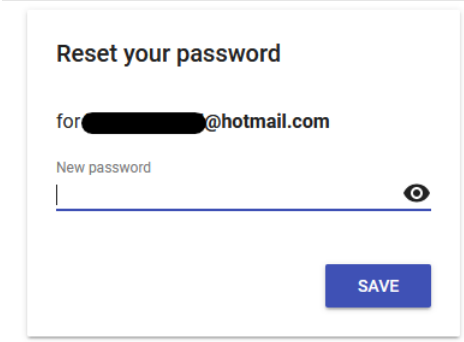
Possíveis erros durante o preenchimento e envio do formulário nesta página são destacados em vermelho e incluem:

Figura 8 – Ilustração da tela de Recuperação de senha



Fonte: Autoria própria (2023).

Figura 9 – Ilustração da tela de definição da nova senha



Fonte: Autoria própria (2023).

- E-mail inválido no campo “E-mail“: “Este e-mail não é válido.“;
- E-mail informado não cadastrado: “Este e-mail não está cadastrado.“;
- Tentativa de envio do formulário sem preencher o campo “E-mail“: “Por favor, informe seu e-mail.“;
- Erro do *Firebase Authentication* ao tentar enviar o e-mail: “Ocorreu um erro ao enviar o e-mail. Entre em contato com o suporte sobre o erro: “

3.3.1.4 Sistema de atendimentos

A primeira função desse sistema é a chamada “createTalkWebHook“, demonstrada na Listagem 7 na página 61, que é responsável por lidar com uma solicitação de webhook de forma assíncrona. Todas as funções que devem possuir um corpo como dos tipos POST e PATCH, possuem uma verificação se o corpo da solicitação está vazio ou inválido e, se for o caso, retorna um código de status 400 com uma mensagem de erro.

Em seguida, a função extrai informações relevantes do corpo da solicitação, como nome de quem enviou, destinatário, tipo de mensagem e conteúdo da mensagem. Ela usa essas informações para pegar detalhes da conversa, legenda e URL da mensagem. A função também chama outras três funções, “checkForEspecialist”, “checkForClientInfo” e “checkForUserTaskActiveNumber”, para obter os valores de identificação associados ao remetente e ao destinatário. Ambas as funções “checkForEspecialist“ e “checkForUserTaskActiveNumber“ serão explicadas com detalhes no texto da próxima funcionalidade, Auto-Atribuição de Conversas.

A função chamada “checkForClientInfo“, também é assíncrona que recebe dois parâmetros: “whatsappIdentification“ e “name“. Sua função principal é encontrar um cliente no banco de dados usando o método “findOne“ e retornar o cliente, caso exista. Se o cliente não existir e um parâmetro “name“ for fornecido, a função cria um novo cliente no banco de dados usando o método “create“ e retorna o cliente recém-criado.

Para realizar isso, a função utiliza a opção “where“ para especificar a condição da consulta e a opção “limit“ para limitar o resultado a uma única linha. Se o cliente não for encontrado e um parâmetro “name“ for fornecido, a função criará um novo cliente no banco de dados usando o método “create“. Esse método é usado para inserir uma nova linha na tabela de clientes, com os atributos “name“ e “phone“ definidos com os valores fornecidos.

Além disso, a função utiliza a função “checkForActiveTasks“ para obter informações relacionadas à tarefa ativa associada ao destinatário.

Por fim, a função “createTalkWebHook“ cria um novo registro de conversa no banco de dados usando o método “create“ e retorna um código de status 201 se a criação for bem-sucedida. Se ocorrer algum erro, ela retorna um código de status 500 com uma mensagem de erro. Esta função também emite um evento através do socket.io para atualizar as conversas.

Listagem 7 – Entrada para novas conversas vindas do WhatsApp

```
export async function createTalkWebHook (req, res) {
  try {
    // Validate request
    if (!req.body || Object.keys(req.body).length === 0) {
      return res.status(400).send({
        message: exceptions.emptyBody()
      });
    }

    if (!req.body.data) {
      return res.status(400).send({
        message: exceptions.emptyData()
      });
    }

    const {
      pushName,
      key: { remoteJid },
      messageType,
      msgContent
    } = req.body.data;

    const { conversation, caption, url } =
      msgContent || msgContent[messageType];

    const message =
      messageType === 'conversation' ? conversation : caption || '';
    const finalUrl = messageType === 'conversation' ? null : url;

    let userIdResult = await checkForEspecialist(remoteJid);

    const clientIdResult = await checkForClientInfo(remoteJid, pushName);

    if (!userIdResult) {
      userIdResult = await checkForUserTaskActiveNumber();
    }

    const taskIdentification = await checkForActiveTasks(remoteJid);

    // Save Talk in the database
    const talk = await talks.create({
      fk_users_identification: userIdResult || 1,
      fk_clients_identification: clientIdResult || null,
      message,
      fk_tasks_identification: taskIdentification,
      whatsapp_identification: remoteJid,
      url: finalUrl,
      type: messageType,
      from_me: false
    });

    io.emit('talks', 'Atualizar conversas');
    return res.status(201).send(talk);
  } catch (err) {
    return res.status(500).send({
      message: exceptions.createError('Conversa', err.message)
    });
  }
}
```

Fonte: Autoria própria (2023).

Listagem 8 – Checagem de tarefas ativas relacionadas ao número de WhatsApp

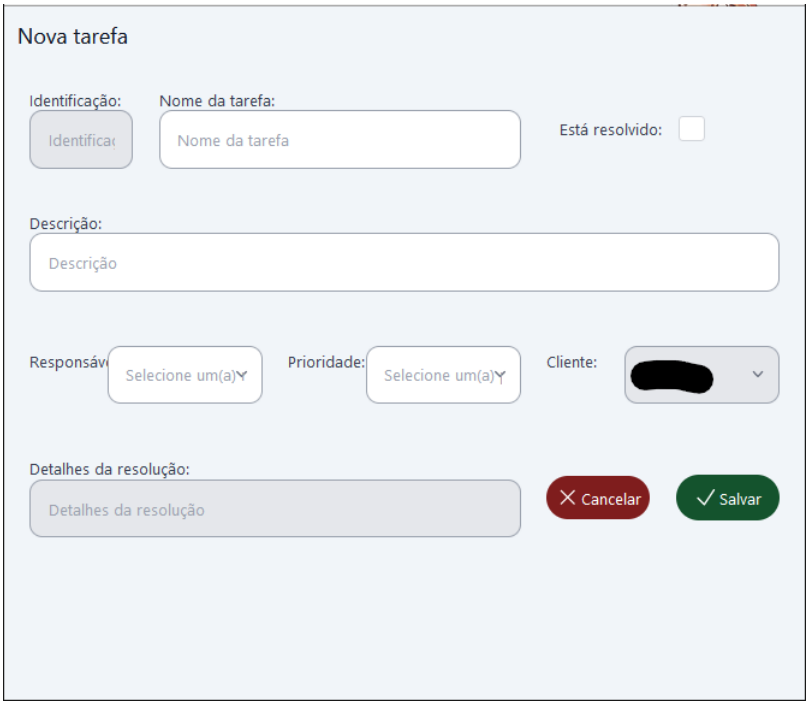
```
async function checkForActiveTasks (phone) {
  const previousTalk = await talks.findOne({
    where: {
      whatsapp_identification: phone,
      fk_tasks_identification: {
        [Op.not]: null
      }
    },
    order: [['created_at', 'DESC']]
  });

  if (previousTalk) {
    const task = await tasks.findByPk(previousTalk.fk_tasks_identification);
    if (!task.is_it_solved) {
      return task.identification;
    }
  }

  return null;
}
```

Fonte: Autoria própria (2023).

Após essa etapa inicial de cadastro do cliente, cadastro da conversa e verificação de quem é o analista mais adequado para ser responsável pelo atendimento, quem foi designado deve iniciar uma nova tarefa, selecionando qual mensagem deve ser considerada o começo da tarefa dando um clique em cima da mensagem, isso fará que a mensagem fique destacada com uma borda verde.

Em seguida, o analista deve utilizar um botão presente na tela de conversa com um cliente, que abrirá o formulário de criação de tarefas demonstrado na Figura 10.

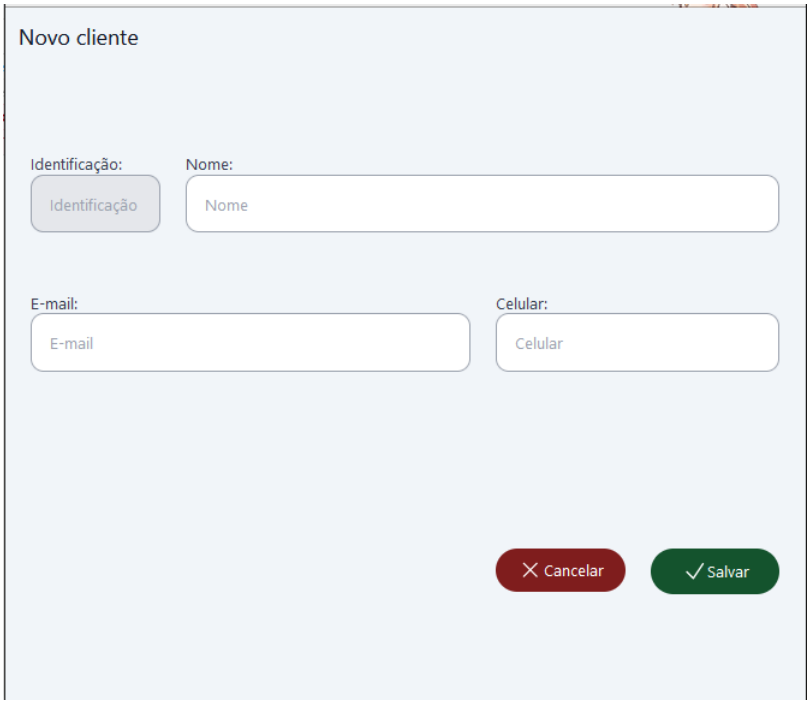
Figura 10 – Ilustração do formulário de criação de tarefa



Fonte: Autoria própria (2023).

Caso não seja possível para o sistema cadastrar o cliente utilizando a função “checkForClientInfo“, esse mesmo botão será substituído por um botão similar que levará ao assistente para o formulário de criação de clientes, demonstrada na Figura 11 na página 64.

Figura 11 – Ilustração do formulário de criação de cliente



Fonte: Autoria própria (2023).

Após terminar o cadastrado de uma nova tarefa o sistema utilizará a função “checkForActiveTasks“, toda vez que receber ou enviar uma nova mensagem.

A função assíncrona chamada “checkForActiveTasks“, demonstrada na Listagem 8 na página 62, recebe um parâmetro “phone“. A função tem a finalidade de encontrar a conversa mais recente associada ao telefone fornecido e a uma identificação de tarefa (fk_tasks_identification) não nula, usando o método “findOne“.

Se a conversa existir e a tarefa associada não estiver resolvida, a função retorna o atributo “identification“ da tarefa. Se a conversa não existir ou a tarefa associada estiver resolvida, a função retorna “null“.

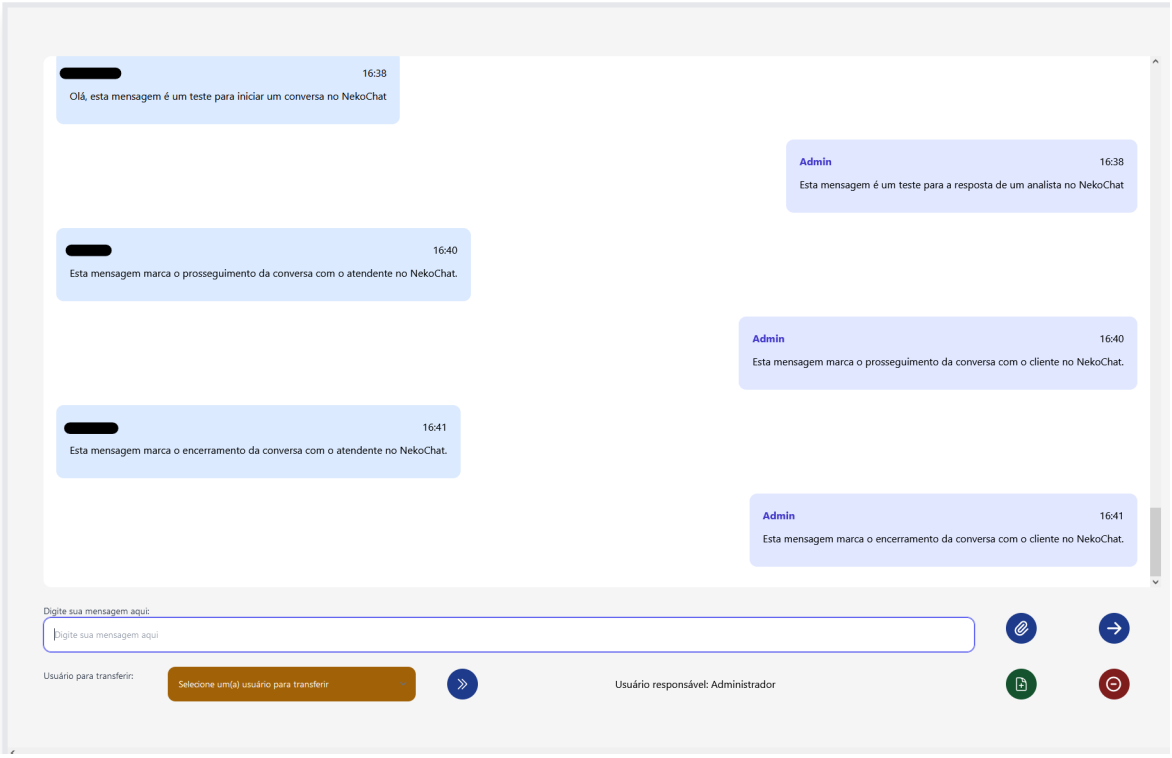
Para fazer isso, a função utiliza a opção “where“ especificando a condição da consulta, a opção “order“ classificando o resultado com base no atributo “created_at“ em ordem decrescente, e a opção “include“ incluindo a tarefa associada no resultado. A função utiliza o operador “Op.not“ que especifica que o atributo “fk_tasks_identification“ não deve ser nulo. Isso garante que todas as novas mensagens sejam relacionadas com essa nova função.

Após a realização do atendimento, o analista de suporte pode encerrar a conversa fazendo a mesma seleção, só que agora para indicar onde a tarefa terminou, e clicar no botão de encerrar atendimento. Isso fará que o sistema atualize a tarefa ativa como resolvida, e todas as mensagens posteriores a escolhida, terão o campo “fk_tasks_identification“ igual a “null“.

O ideal seria que o analista após o encerramento da tarefa, utilizasse a tela de edição da tarefa para informar os detalhes da solução encontrada para o problema. Mas o sistema não obriga o analista a informar esses detalhes.

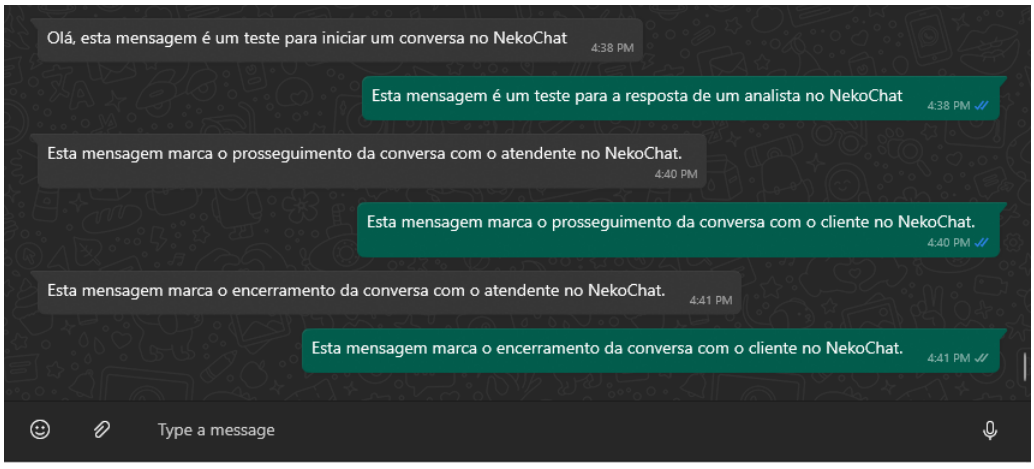
Aqui está uma demostração na Figura 12 de como as mensagens recebidas pelo WhatsApp é apresentada no aplicativo, e o equivalente no WhatsApp Desktop na Figura 13:

Figura 12 – Ilustração da conversa no aplicativo



Fonte: Autoria própria (2023).

Figura 13 – Ilustração da conversa no WhatsApp Desktop



Fonte: Autoria própria (2023).

Listagem 9 – Checagem usuários responsáveis anteriormente pela conversa

```
async function checkForEspecialist (whatsappIdentification) {
  const result = await talks.findOne({
    where: {
      whatsapp_identification: whatsappIdentification
    },
    attributes: ['fk_users_identification'],
    order: [['created_at', 'DESC']],
    limit: 1
  });

  return result !== null
    ? JSON.parse(JSON.stringify(result)).fk_users_identification
    : result;
}
```

Fonte: Autoria própria (2023).

3.3.1.5 Auto-Atribuição de Conversas

Como foi informado na última sessão, aqui serão explicadas as funções responsáveis por fazer o balanceamento inicial das conversas entre os analistas de suporte, “checkForEspecialist“ e “checkForUserTaskActiveNumber“. Sobre a “checkForEspecialist“ descrita na Listagem 9, é uma função assíncrona que recebe o parâmetro “whatsappIdentification“.

Ela tem a finalidade de encontrar a conversa mais recente associada à identificação do WhatsApp fornecida, utilizando o método “findOne“, e retorna o atributo “fk_users_identification“ dessa conversa, se existir. Se a conversa não existir, a função retorna “null“. A ideia é que se a conversa possuir anteriormente um usuário que foi responsável por ela, o sistema considera esse usuário como um especialista sobre aquele cliente.

Para fazer isso, a função utiliza a opção “where“ para especificar a condição da consulta, a opção “attributes“ para determinar quais atributos incluir no resultado, a opção “order“ para classificar o resultado com base no atributo “created_at“ em ordem decrescente, e a opção “limit“ para limitar o resultado a uma única linha.

Caso o sistema não encontre um usuário considerado como “especilista“, ele executa a função “checkForUserTaskActiveNumber“, descrita na Listagem 10 na página 67, que é assíncrona e tem como objetivo encontrar a identificação de um usuário com tarefas não resolvidas abaixo do limite permitido.

Primeiramente, ela procura todos os usuários com um status de “ativo“ (representado pelo valor 1) usando o método “findAll“. Em seguida, ela obtém a identificação e o limite de tarefas (task_limit) de cada usuário e classifica o resultado com base na data de criação, em ordem decrescente. A ideia é sempre passar primeiro por usuário senior. Se nenhum usuário for encontrado, a função retorna “null.“

Listagem 10 – Checagem usuários disponíveis para receber a nova conversa

```
async function checkForUserTaskActiveNumber () {
  const usersData = await users.findAll({
    where: {
      fk_statuses_identification: 1
    },
    attributes: ['identification', 'task_limit'],
    order: [['created_at', 'DESC']]
  });

  if (usersData === null) {
    return null;
  }

  for (const user of usersData) {
    const result = await tasks.findAndCountAll({
      where: {
        fk_users_identification: user.identification,
        is_it_solved: false
      }
    });

    if (result.count < user.task_limit) {
      return user.identification;
    }
  }

  return null;
}
```

Fonte: Autoria própria (2023).

Para cada usuário encontrado, a função verifica o número de tarefas não resolvidas associadas a esse usuário usando o método “findAndCountAll.“ Ela obtém a identificação do usuário (fk_users_identification) e o estado de resolução da tarefa (is_it_solved).

Se o número de tarefas não resolvidas associadas a esse usuário for menor do que o limite de tarefas permitido para o usuário (task_limit), a função retorna a identificação desse usuário.

Se nenhum usuário for encontrado com tarefas não resolvidas abaixo do limite permitido, a função também retorna “null.“

Essa função tem como objetivo encontrar um usuário ativo que tenha capacidade para receber uma nova conversa, considerando que suas tarefas não resolvidas estejam dentro do limite permitido. Caso nenhum dessas funções encontre um usuário disponível para receber o atendimento, a função “createTalkWebHook“ irá atribuir a conversa para o administrador (representado pelo usuário com número de identificação 1), que ficará responsável por transferir a conversa posteriormente para um dos analistas.

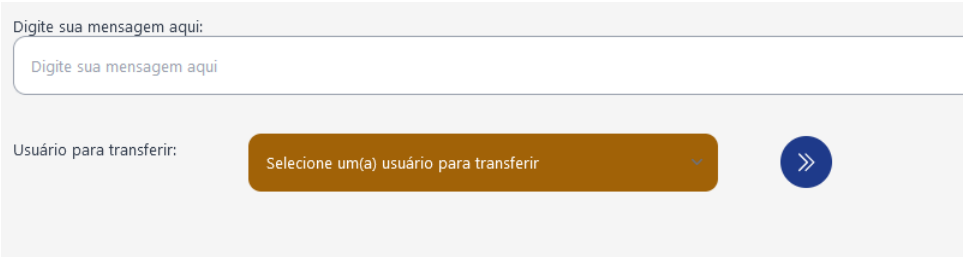
3.3.1.6 Garantia de Qualidade

O sistema de garantia de qualidade tem poucos passos para iniciar. Quando um usuário que possui a flag is_admin marcada, o sistema irá optar para realizar a chamada da função “findAllTalks“, demostrada na Listagem 11 na página 69, que realiza uma chamada da API para o servidor buscar todas as mensagens cadastradas no servidor. E caso a flag esteja desabilitada, o sistema irá chamar a função findAllTalksByUser que limita a busca pelo identificador do usuário. Essa verificação é feita pela função “fetchTalks“, demostrada na Listagem 12 na página 69,.

A função “findAllTalks“ é uma função assíncrona que obtém uma lista de conversas de um servidor remoto e preenche o objeto de conversas (talks) com os dados recuperados. O que ela faz é enviar uma solicitação GET para o servidor em uma URL específica usando a API de busca (fetch), com o cabeçalho Content-Type configurado como application/json. Se a solicitação for bem-sucedida, a resposta é analisada usando o método response.json(), e os dados resultantes são usados para preencher o objeto de conversas (talks). Esse objeto “talks“ é uma estrutura que associa números de identificação do WhatsApp a objetos de conversa. Se a solicitação falhar, qualquer erro é registrado no console e o usuário é redirecionado para a página de erro 404 usando a função router.push.

Isso permite que o administrador visualize e interaja com qualquer conversa, mesmo se ela estiver atribuída a outro usuário. O administrador também possuí algumas outras opções que usuário normais não tem, a mais relevante é a atribuição manual da conversa para outro usuário. Essa opção permite ao administrador transferir a visualização da conversa atual para outro, e assim retirar do usuário anterior o poder de visualizar ou interagir com a conversa. Na Figura 14 é possível ver a área de interação disponível para um administrador em uma conversa.

Figura 14 – Ilustração da tela de conversa



Fonte: Autoria própria (2023).

Listagem 11 – Verificação se o usuário é ou não administrador

```
async function findAllTalks() {
        const url = GCurl + ``talks``;
        try {
            const response = await fetch(url, {
                method: ``GET``,
                headers: {
                    ``Content-Type``: ``application/json``
                }
            });
            const data = await response.json();
            if (data) {
                talks.value = {};
                for (const talk of data) {
                    talks.value[talk.whatsapp_identification] = talk;
                }
            }
            return;
        } catch (error) {
            console.error(error);
            router.push({
                name: ``404Resource``,
                params: { resource: ``chamada encontrar conversa`` }
            });
        }
    }
```

Fonte: Autoria própria (2023).

Listagem 12 – Verificação se o usuário é ou não administrador

```
function fetchTalks() {
        userStore.user.is_admin ? findAllTalks() : findAllTalksByUser();
}
```

Fonte: Autoria própria (2023).

3.3.1.7 *Feedback* Loop

A funcionalidade de *feedback* foi projetada para se integrar ao sistema de gerenciamento de tarefas já existente. Em termos de interface, ela usa as mesmas telas que o sistema de tarefas convencional demonstrada na Figura 10 na página 63, com a principal diferença sendo a localização do botão para criar uma nova entrada, presente no menu lateral da tela principal. Neste caso, o botão serve para adicionar uma "tarefa de *feedback*".

No sistema, as tarefas de *feedback* são automaticamente enviadas para um time de desenvolvimento específico. Este time é responsável por avaliar e decidir quais delas são relevantes e que alterações sugeridas deverão ser implementadas. Para diferenciar tarefas comuns de tarefas de *feedback*, uma bandeira é usada na tabela de tarefas. Isso foi especificado no diagrama de entidade de relacionamento do projeto, demonstrado na Figura 5 na página 55. Essa bandeira é responsável por diferenciar para o sistema entre os dois tipos de tarefas.

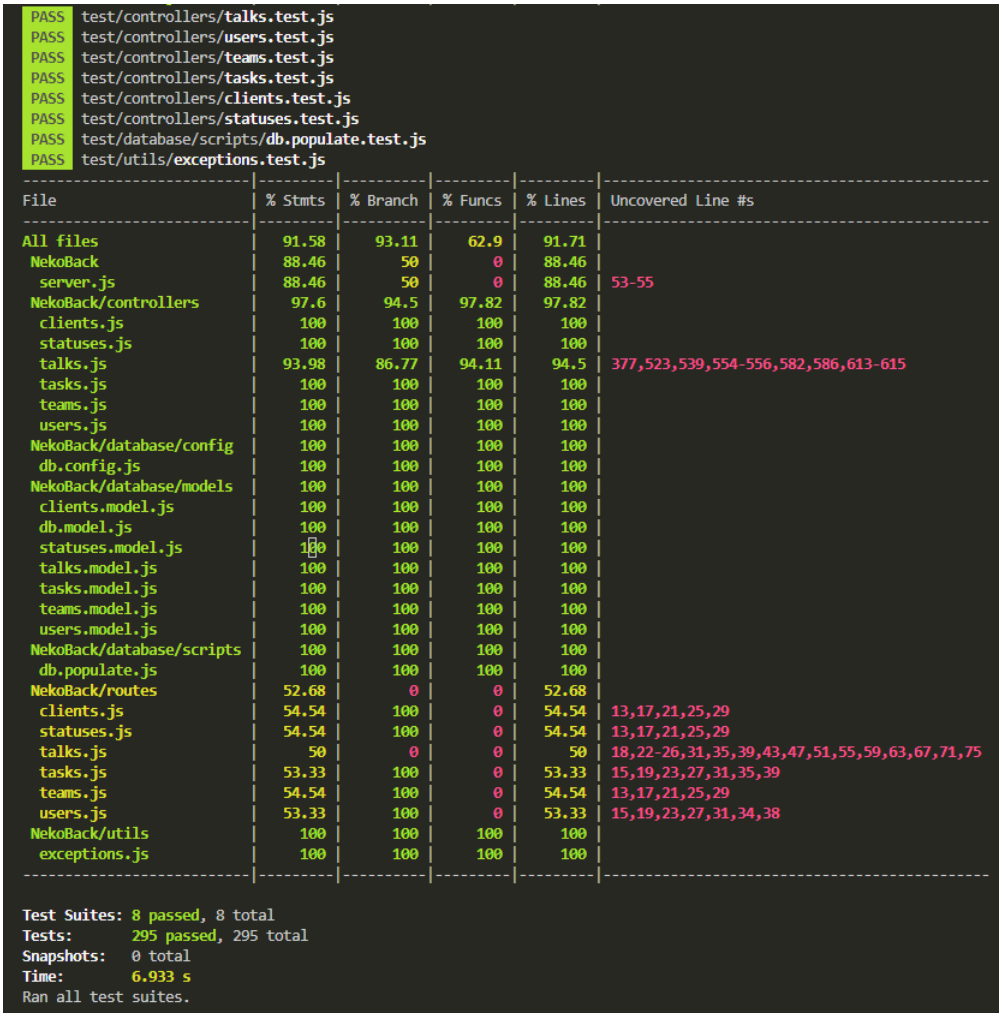
Outro detalhe é que, ao criar uma tarefa de feedback, a opção de escolher um time diferente para alocá-la está desabilitada. O time de desenvolvimento responsável já é predefinido pelo sistema.

3.4 Testes

Esta seção é utilizada para relatar os testes realizados para confirmar que as implementações realizada no sistema desempenhe o papel necessário para as funcionalidades descritas anteriormente. O método utilizado para realizar os testes foi o de Testes Unitários, com a ferramenta Jest (v29.6.4). Como explicado anteriormente no Capítulo 2, Sommerville (2011) diz os testes unitários tem o foco em testar componentes do sistema individualmente, onde as funções e métodos são os componentes mais simples do programa. E também foi comentado no Capítulo 2, que Sommerville (2011) recomenda a utilização de *frameworks* de automação para escrever e executar testes, pois eles são capazes de executar todos os testes implementados e repotar resultados.

Após a implementação e execução dos testes pelo Jest, é possível visualizar algumas estatísticas relevantes sobre a cobertura oferecida pelos testes em relação a diferentes arquivos e ao código como um todo, demonstrado na Figura 15.

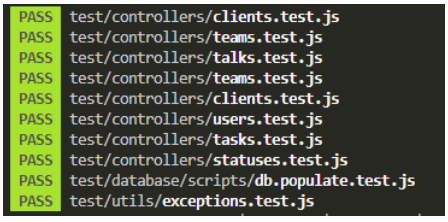
Figura 15 – Ilustração da cobertura de testes unitários



Fonte: Autoria própria (2023).

É possível separar a imagem em três sessão menores, que demonstra diferentes dados. Começando uma analise de cima para baixo, na Figura 16 é possível ver quais arquivos foram adicionados para realizar os testes. A separação utilizada na criação dos arquivos é semelhante aos que eles estão testando, onde os arquivos dos controladores, dos *scripts* do banco de dados e o arquivo utilitário estão separados por pastas. Também é informado em verde que todos os testes implementados nesse arquivos resultaram no era esperado deles.

Figura 16 – Ilustração dos arquivos cobertos pelos testes unitários



Fonte: Autoria própria (2023).

Nos registros dos controladores, foram conduzidos testes para verificar as operações de cadastro, busca, atualização e exclusão dos dados relacionados a cada modelo de entidade no banco de dados. Os principais testes elaborados para esses cenários foram consolidados na lista a seguir. Para facilitar a legibilidade, a lista foi organizada de forma a destacar os testes que se repetiram em diversos modelos de entidade:

- Cadastro:
  - Deve criar uma entidade.
  - Deve retornar a entidade criada.
  - Deve retornar a entidade criada com identificação.
  - Deve enviar uma mensagem de mídia.
  - Deve retornar a entidade criada com a identificação de entidade estrangeira.
  - Deve retornar um erro se você tentar criar uma nova entidade com os mesmos dados.
  - Deve retornar um erro se você tentar criar uma nova entidade com o corpo vazio.
  - Deve retornar um erro se você tentar criar uma nova entidade com o nome vazio.
  - Deve retornar um erro se você tentar criar uma nova entidade com o e-mail vazio.
  - Deve retornar um erro se você tentar criar uma nova entidade com a mensagem vazia.
  - Deve retornar um erro se você tentar criar uma nova entidade com a identificação de WhatsApp vazia.
  - Deve retornar um erro se você tentar criar uma nova entidade com o tipo vazio.
  - Deve retornar um erro se você tentar criar uma nova entidade com a descrição vazia.
  - Deve retornar um erro 500 usando um *mock*.
- Busca:
  - Deve encontrar uma entidade.
  - Deve retornar a entidade encontrada.
  - Deve encontrar todas as entidades.
  - Deve retornar as entidades encontradas.
  - Deve conter a entidade criada.

  - Deve encontrar todas as entidades pela chave estrangeira da entidade.
  - Deve retornar um erro se você tentar encontrar uma entidade com um ID vazio.
  - Deve retornar um erro se você tentar encontrar uma entidade com um ID inválido.
  - Deve retornar um erro se você tentar encontrar todas as entidades e a lista estiver vazia.
  - Deve retornar um erro se você tentar encontrar uma entidade com o email vazio.
  - Deve retornar um erro se você tentar encontrar uma entidade com um email inválido.
  - Deve retornar um erro se você tentar encontrar todas as entidades com um ID vazio.
  - Deve retornar um erro se você tentar encontrar uma entidade com um ID de entidade estrangeira inválido.
  - Deve retornar um erro 500 usando um *mock*.

- Atualização:

  - Deve atualizar a entidade criada.
  - Deve atualizar uma entidade.
  - Deve receber uma mensagem confirmando que a entidade foi atualizada.
  - Deve retornar a entidade atualizada.
  - Deve retornar um erro se você tentar atualizar uma entidade estrangeira que não existe vinculada a uma entidade.
  - Deve retornar um erro se você tentar atualizar uma entidade com os mesmos campos pelos quais já está registrada.
  - Deve retornar um erro se você tentar atualizar uma entidade com o corpo vazio.
  - Deve retornar um erro se você tentar atualizar uma entidade com o ID vazio.
  - Deve retornar um erro se você tentar atualizar uma entidade com um ID inválido.
  - Deve retornar um erro se você tentar atualizar uma entidade com um campo vazio dentro do corpo.
  - Deve retornar um erro 500 usando um *mock*.

- Exclusão:
  - Deve excluir uma entidade.
  - Deve retornar a entidade excluída.
  - Deve excluir a mensagem de mídia.
  - Deve retornar um erro se você tentar excluir uma entidade duas vezes.
  - Deve retornar um erro se você tentar excluir uma entidade com um ID vazio.
  - Deve retornar um erro se você tentar excluir uma entidade com um ID inválido.

Essa cobertura de testes foi criada com objetivo de averiguar que todas as funções dos controladores retornem os resultados esperados mesmo se forem propositalmente sabotados por uma falha de outras funções ou pelo envio de entradas erradas, alinhando com o que Sommerville (2011) explica sobre os objetivos dos testes unitários.

Os testes unitários criados para o *script* responsável por popular o banco de dados com algumas entidades padrões foram consolidados na lista a seguir. Diferente dos controladores os testes necessários para o *script* não foram tão complexos:

- Deve popular o banco de dados com entidades padrão.
- Não deve popular o banco de dados com entidades padrão.
- Deve retornar um erro ao tentar popular o banco de dados com entidades padrão.

E, por último, o arquivo de exceções foi submetido a apenas dois tipos de testes. O primeiro verifica se a função recebeu os parâmetros, enquanto o segundo verifica se a função não recebeu parâmetros.

Não foram elaborados testes adicionais devido à explicação fornecida por Sommerville (2011), que argumenta ao testar uma função que envolve cálculos e requer dois números positivos, é razoável esperar que o programa funcione de maneira semelhante para todos os números positivos.

Essa mesma explicação foi utilizada na decisão de não construir testes para os arquivos de rotas, pois eles fariam o mesmo percurso de testes que os controladores dado que a função deles é disponibilizar os controladores para API.

Ao prosseguir com a análise da Figura 15 na página 71, vem a seção estatística da cobertura de código. Nessa seção, demonstrada pela Figura 17 na página 76, é possível visualizar a porcentagem de cobertura tanto para todos os arquivos quanto para cada um deles individualmente. As porcentagens indicam os seguintes elementos:

- Ramos (representados como ’Branches’) são declarações condicionais cujas condições foram satisfeitas pelo menos uma vez durante os testes unitários.

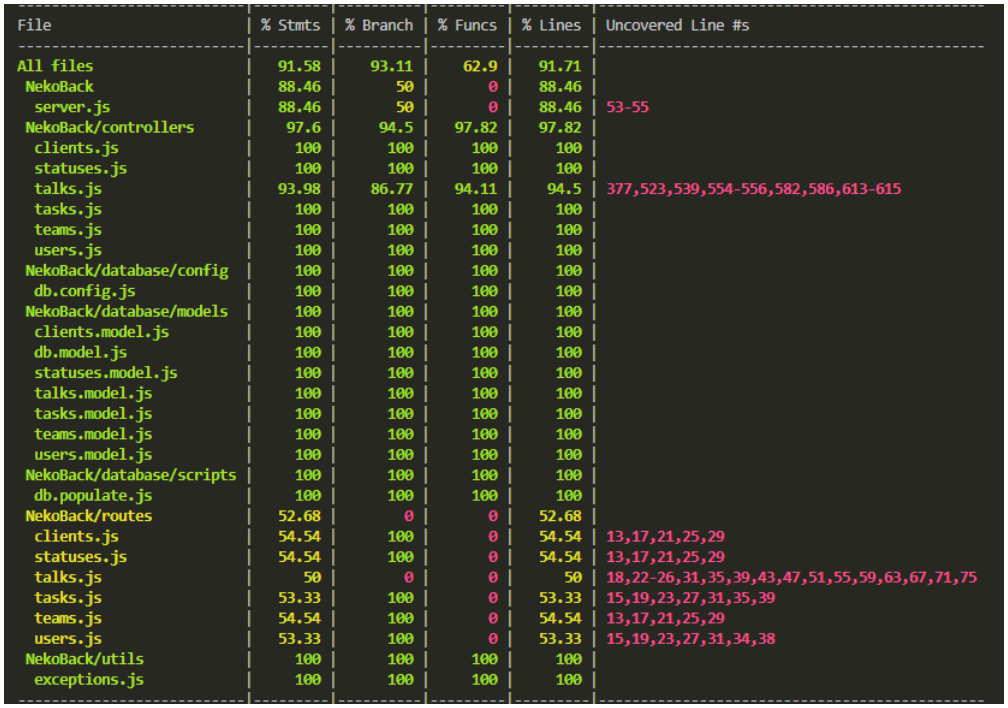
- Funções (representadas como ’Funcs’) são funções que foram chamadas pelo menos uma vez durante os testes unitários.
- Linhas (representadas como ’Lines’) são linhas de código que foram executadas pelo menos uma vez durante os testes unitários.
- Declarações (representadas como ’Stmts’) são instruções que foram executadas pelo menos uma vez durante os testes unitários. Por exemplo, uma única linha pode conter duas declarações.

E por último, as Linhas Não-Cobertas (identificadas como ’Uncovered Line #s’), que se referem às linhas de código em que nenhum teste foi executado. É notável que no arquivo “controllers/talks.js“, há várias linhas que não foram submetidas a testes. Isso ocorre devido ao fato de estarem contidas em funções que não são exportadas pelo arquivo, uma vez que são destinadas ao uso interno por outras funções.

Mesmo que sejam criados casos de teste para as funções que as chamam, a cobertura dessas linhas seria desafiadora, pois se destinam a exceções muito específicas.

Outro ponto de observação é que todos os arquivos, com exceção das rotas, do controlador das conversas e do iniciador do servidor, apresentam uma cobertura completa. Isso indica que foram desenvolvidos testes para todas as possibilidades de execução de cada linha de código.

Figura 17 – Ilustração das estatística da cobertura dos testes unitários

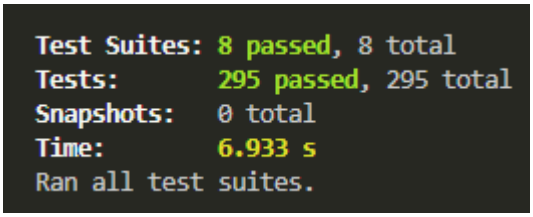


File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	91.58	93.11	62.9	91.71	
NekoBack	88.46	50	0	88.46	
server.js	88.46	50	0	88.46	53-55
NekoBack/controllers	97.6	94.5	97.82	97.82	
clients.js	100	100	100	100	
statuses.js	100	100	100	100	
talks.js	93.98	86.77	94.11	94.5	377,523,539,554-556,582,586,613-615
tasks.js	100	100	100	100	
teams.js	100	100	100	100	
users.js	100	100	100	100	
NekoBack/database/config	100	100	100	100	
db.config.js	100	100	100	100	
NekoBack/database/models	100	100	100	100	
clients.model.js	100	100	100	100	
db.model.js	100	100	100	100	
statuses.model.js	100	100	100	100	
talks.model.js	100	100	100	100	
tasks.model.js	100	100	100	100	
teams.model.js	100	100	100	100	
users.model.js	100	100	100	100	
NekoBack/database/scripts	100	100	100	100	
db.populate.js	100	100	100	100	
NekoBack/routes	52.68	0	0	52.68	
clients.js	54.54	100	0	54.54	13,17,21,25,29
statuses.js	54.54	100	0	54.54	13,17,21,25,29
talks.js	50	0	0	50	18,22-26,31,35,39,43,47,51,55,59,63,67,71,75
tasks.js	53.33	100	0	53.33	15,19,23,27,31,35,39
teams.js	54.54	100	0	54.54	13,17,21,25,29
users.js	53.33	100	0	53.33	15,19,23,27,31,34,38
NekoBack/utils	100	100	100	100	
exceptions.js	100	100	100	100	

Fonte: Autoria própria (2023).

Encerrando a análise da Figura 15 na página 71, é possível observar o número total de arquivos de testes, a quantidades de testes realizados e o tempo de execução deles, demonstrados na Figura 18. Foram 8 arquivos de testes, 295 testes criados e executados em um período 6.933 segundos.

Figura 18 – Ilustração das quantidades da cobertura dos testes unitários



Fonte: Autoria própria (2023).

4 CONCLUSÃO

A plataforma digital criada se dedicada à otimização do gerenciamento de conversas em equipe. Esta aplicação incorpora funcionalidades destinadas à alocação, monitoramento e supervisão de diálogos, proporcionando aos grupos de suporte a capacidade de registrar suas interações com os clientes para fins de análise posterior.

Esse recurso capacita tanto os analistas de suporte quanto os administradores a examinar, avaliar, analisar em detalhes, desmembrar e extrair conhecimento dessas interações, visando à adaptação futura. Além disso, concede aos administradores a autoridade de intervenção, caso julguem necessário, durante ou após o processo de atendimento.

Adicionalmente, a plataforma inclui um algoritmo de distribuição automática das interações recebidas, direcionando-as para os membros da equipe com base em critérios como a carga de trabalho atual, especialização na área temática e histórico de desempenho. Estas métricas podem ser personalizadas pelo administrador da plataforma, proporcionando ferramentas para o acompanhamento, avaliação qualitativa e controle das interações de suporte.

O resultado é a implementação de um sistema robusto que simplifica a gestão e a distribuição eficaz das conversas dentro da equipe, utilizando a troca de mensagens como o principal meio de comunicação.

A integração com o aplicativo WhatsApp proporcionou uma maior facilidade na comunicação com os consumidores, uma vez que este se revelou um meio amplamente utilizado pelos brasileiros em sua rotina diária, como indicado em uma pesquisa conduzida pela Opinion Box e referenciada por Salgado (2022).

A seleção das tecnologias adotadas possibilitou a utilização de uma linguagem de programação unificada tanto para o servidor quanto para a aplicação cliente, o que resultou na redução da curva de aprendizado durante o desenvolvimento deste projeto.

Além disso, a vibrante comunidade em torno do JavaScript viabilizou a integração de ferramentas que desempenharam papéis cruciais no sistema, incluindo a modelagem de dados, a comunicação com o banco de dados, a otimização do processo de autenticação, a avaliação da robustez das senhas cadastradas no sistema e a simplificação do desenvolvimento de testes unitários.

Embora o JavaScript é geralmente considerado uma linguagem de fácil aprendizado para iniciantes, porém, é importante destacar que a natureza assíncrona e o gerenciamento de eventos no Node.js podem apresentar desafios adicionais para desenvolvedores menos experientes.

Também, a atenção à performance do código é fundamental, uma vez que o Node.js não é a escolha ideal para operações intensivas de CPU, pois isso pode resultar no bloqueio do processamento de outras solicitações. Por último, embora a comunidade JavaScript seja ampla e ofereça muitas soluções, é necessário ter em mente que algumas dessas soluções

podem ser instáveis ou possuir documentação insuficiente, o que requer cautela na escolha e implementação de bibliotecas e *frameworks*.

É interessante notar que a curva de aprendizado inicial do Vue.js é relativamente curta, graças à sua capacidade de interagir diretamente com o conteúdo da página de forma dinâmica. Isso possibilita a criação de interfaces que podem alterar seus estados sem interromper o fluxo de trabalho do usuário.

Por outro lado, o Tailwind CSS requer um período de adaptação devido à sua abordagem única na estruturação de estilos CSS. No entanto, uma vez superada essa fase inicial, a conveniência de organizar o código por meio de classes que representam mudanças na interface simplifica a leitura e análise do código implementado, tornando-o mais compreensível e fácil de manter.

É compreensível que, ao longo deste projeto, tenham sido implementados muitos recursos enquanto outros desejáveis tenham sido deixados de lado, devido à complexidade ou à falta de tempo. No entanto, essa abordagem possibilita melhorias futuras no sistema, já que abre caminho para a adição de módulos e funcionalidades que podem aprimorar significativamente o valor do trabalho realizado.

A inclusão de recursos como a criação de relatórios para a análise estatística do desempenho das interações de suporte, a comunicação interna da equipe por meio do aplicativo, a capacidade de registrar diretamente as opiniões, reclamações e sugestões dos consumidores, bem como recursos para incentivar e reconhecer o alto desempenho dos membros da equipe de suporte, certamente contribuiria para enriquecer ainda mais o sistema.

Uma atividade prospectiva adicional envolveria a submissão deste sistema à homologação em contextos empresariais que albergam uma equipe de atendimento ao cliente, independentemente do emprego do WhatsApp ou de outros meios de comunicação, a fim de corroborar a suficiência do aplicativo em substituir o paradigma de atendimento previamente empregado por tais entidades.

Por fim, este trabalho demonstra a habilidade do aplicativo em estabelecer uma comunicação eficaz com uma plataforma de mensagens amplamente adotada pelos consumidores no país. Além disso, ele oferece recursos essenciais para a gestão e distribuição das interações com os clientes entre os membros da equipe de suporte que fazem uso dele. Isso ressalta a possibilidade do aplicativo otimizar as operações de suporte e melhorar a comunicação com os consumidores.

REFERÊNCIAS

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **ABNT NBR 9000**: Sistema de gestão da qualidade: Fundamentos e vocabulário. Rio de Janeiro, 2015. 3–4 p.

BAROT, T.; OREN, E. Communications and community on internet relay chat. **Gitbooks**, 2015.

BODET, G. Customer satisfaction and loyalty in service: Two concepts, four constructs, several relationships. **Journal of Retailing and Consumer Services**, v. 15, n. 3, p. 156–162, 2008.

BUDGEN, D. **Software Design**. 2. ed. [*S.l.*]: Pearson Education Limited, 2003. ISBN 0201722194.

CHEN, I. J.; POPOVICH, K. Understanding customer relationship management (crm). **Business Process Management Journal**, MCB UP Ltd, v. 9, n. 5, p. 672–688, Out 2003. ISSN 1463-7154. Disponível em: https://doi.org/10.1108/14637150310496758.

COCKBURN, A. **Writing Effective Use Cases**. Boston, MA: Addison-Wesley, 2001. ISBN 978-0-201-70225-5.

CRESSLER, C. **Understanding WhatsApp's Architecture & System Design**. [*S.l.*], 2021.

EXPRESS.JS. **Using middleware**. [*S.l.*], 2023. Disponível em: https://expressjs.com/en/guide/using-middleware.html. Acesso em: 21/03/2023.

EXPRESS.JS. **Web Applications**. [*S.l.*], 2023. Disponível em: https://expressjs.com. Acesso em: 21/03/2023.

FOWLER, M. **UML Distilled: A Brief Guide to the Standard Object Modeling Language**. 3. ed. Boston, MA: Addison-Wesley, 2003. ISBN 978-0-321-19368-1.

GOOGLE PLAY STORE. **WhatsApp Messenger**. 2023. Disponível em: https://play.google.com/store/apps/details?id=com.whatsapp. Acesso em: 28/03/2023.

HERNANDEZ, M. J. **Database Design for Mere Mortals®: 25th Anniversary Edition**. 25th anniversary edition. ed. [*S.l.*]: Addison-Wesley, 2021.

IBGE (Ed.). **Acesso à internet e à televisão e posse de telefone móvel celular para uso pessoal 2021 / IBGE, Coordenação de Pesquisas por Amostra de Domicílios**. Rio de Janeiro: IBGE, 2022. ISBN 9788524045431. Disponível em: https://biblioteca.ibge.gov.br/index.php/biblioteca-catalogo?view=detalhes&id=2101963. Acesso em: 01/04/2023.

KUMAR, A. *et al.* Measuring and improving customer retention at authorised automobile workshops after free services. **Journal of Retailing and Consumer Services**, v. 39, p. 93–102, 2017.

LARMAN, C. **Applying UML and Patterns: An introduction to Object-Oriented Analysis and Design and the unified process**. 2. ed. [*S.l.*]: Prentice Hall, 2001.

LARMAN, C.; BASILI, V. Iterative and incremental developments. a brief history. **Computer**, v. 36, n. 6, p. 47–56, 2003.

MAIZE. **A history of Instant Messaging and Chat**. 2020. Disponível em: https://www.maize.io/news/lizshemaria-historyof-instant-messaging/. Acesso em: 28/03/2023.

MDN. **Express/Node introduction**. [*S.l.*], 2023. Disponível em: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction. Acesso em: 20/03/2023.

MDN. **First-class Function**. [*S.l.*], 2023. Disponível em: https://developer.mozilla.org/en-US/docs/Glossary/First-class_Function. Acesso em: 20/03/2023.

MDN. **JavaScript**. [*S.l.*], 2023. Disponível em: https://developer.mozilla.org/en-US/docs/Web/JavaScript. Acesso em: 20/03/2023.

MDN. **JavaScript technologies overview**. [*S.l.*], 2023. Disponível em: https://developer.mozilla.org/en-US/docs/Web/JavaScript/JavaScript_technologies_overview. Acesso em: 20/03/2023.

MESSEGER. **MESSENGER - Tudo que você precisa saber.** 2018. Disponível em: https://m.facebook.com/messengerfacts. Acesso em: 28/03/2023.

META. **How WhatsApp enables multi-device capability**. 2021. Engineering at Meta. Disponível em: https://engineering.fb.com/2021/07/14/security/whatsapp-multi-device/. Acesso em: 28/03/2023.

MULLER, W. Gaining competitive advantage through customer satisfaction. **European Management Journal**, v. 9, n. 2, p. 201–211, 1991. ISSN 0263-2373. Disponível em: https://www.sciencedirect.com/science/article/pii/0263237391900855.

NODE.JS. **About Node.js**. [*S.l.*], 2023. Disponível em: https://nodejs.org/en/about#about-node.js. Acesso em: 21/03/2023.

NODE.JS. **About this documentation**. [*S.l.*], 2023. Disponível em: https://nodejs.org/dist/latest-v18.x/docs/api/documentation.html. Acesso em: 21/03/2023.

OLIVER, R. L. Effect of expectation and disconfirmation on postexposure product evaluations: An alternative interpretation. **Journal of Applied Psychology**, v. 62, n. 4, p. 480–486, Aug 1977.

OLIVER, R. L. Cognitive, affective, and attribute bases of the satisfaction response. **Journal of Consumer Research**, v. 20, n. 3, p. 418–430, 1993.

OVERFLOW, S. **Most used programming languages among developers worldwide as of 2022**. 19-21 Hatton Gardens, London, EC1N 8BA, 2022.

PCMAG. **Definition of memory footprint**. 2023. Disponível em: https://www.pcmag.com/encyclopedia/term/memory-footprint. Acesso em: 26/03/2023.

PCMAG. **Messaging app**. 2023. Disponível em: https://www.pcmag.com/encyclopedia/term/messaging-app. Acesso em: 28/03/2023.

PEREZ, S. Twitter now lets you opt in to receive direct messages from anyone. **Techcrunch**, April 2015.

POSSELT, T.; GERSTNER, E. Pre-sale vs. post-sale e-satisfaction: Impact on repurchase intention and overall satisfaction. **Journal of Interactive Marketing**, v. 19, n. 4, p. 35–47, 2005.

REID, E. Communications and community on internet relay chat. **Electropolis**, 1991.

SALGADO, D. **WhatsApp no Brasil: pesquisa revela dados sobre o comportamento do brasileiro**. 2022. Disponível em: https://blog.opinionbox.com/pesquisa-whatsapp-no-brasil/. Acesso em: 01/04/2023.

SHARMA, P. Offshore outsourcing of customer services – boon or bane? **Journal of Services Marketing**, v. 26, n. 5, p. 352–364, Jan 2012.

SHERMAN, R. Chapter 18 - project management. *In*: SHERMAN, R. (Ed.). **Business Intelligence Guidebook**. Boston: Morgan Kaufmann, 2015. p. 449–492. ISBN 978-0-12-411461-6. Disponível em: https://www.sciencedirect.com/science/article/pii/B9780124114616000186.

SHETH, J.; JAIN, V.; AMBIKA, A. Repositioning the customer support services: the next frontier of competitive advantage. **European Journal of Marketing**, v. 54, n. 7, p. 1787–1804, Jan 2020.

SHUERMANS, S.; VOSKOGLOU, C. **The Global Developer Population 2019 - How many developers are there?** 19-21 Hatton Gardens, London, EC1N 8BA, 2019.

SIMPSON, C. Internet relay chat. **Teacher Librarian**, v. 28, n. 1, p. 18, 2000.

SINGH, M. Whatsapp is now delivering roughly 100 billion messages a day. **Techcrunch**, October 2020.

SOMMERVILLE, I. **Engenharia de Software**. 9. ed. [*S.l.*]: Pearson Education do Brasil, 2011. ISBN 9788579361081.

TECHTUDO. **ICQ, MSN, SMS e WhatsApp: relembre a evolução de mensagens e apps**. 2019. Techtudo Redes socias. Disponível em: https://www.techtudo.com.br/listas/2019/06/icq-msn-sms-e-whatsapp-relembre-a-evolucao-de-mensagens-e-apps.ghtml. Acesso em: 27/03/2023.

VUE.JS GUIDE. **Introduction**. [*S.l.*], 2023. Disponível em: https://vuejs.org/guide/introduction.html. Acesso em: 26/03/2023.

VUE.JS GUIDE. **Lifecycle Hooks**. [*S.l.*], 2023. Disponível em: https://vuejs.org/guide/essentials/lifecycle.html. Acesso em: 27/03/2023.

WOOLLEY, D. R. Plato: The emergence of online community. **Social Media Archeology and Poetics, MIT Press, Cambridge**, 1994.