Guilherme Rimoldi Kameoka

Documentação do Desenvolvimento de um *Chatbot* como Ferramenta de Apoio ao Desenvolvimento Cognitivo de Crianças com Transtorno do Espectro Autista

2025

Guilherme Rimoldi Kameoka

Documentação do Desenvolvimento de um *Chatbot* como Ferramenta de Apoio ao Desenvolvimento Cognitivo de Crianças com Transtorno do Espectro Autista

Orientador: Prof. Renato Aparecido Pimentel da Silva

2025

Dedico este trabalho aos meus pais, que sempre me apoiaram incondicionalmente, acreditaram em meu potencial e me incentivaram a nunca desistir dos meus sonhos. Sem o apoio, a confiança e o exemplo de vocês, nada disso teria sido possível.

AGRADECIMENTOS

Primeiramente, gostaria de expressar minha mais profunda e eterna gratidão aos meus pais, cujo apoio incondicional e incentivo constante foram fundamentais para que eu nunca desistisse dos meus sonhos. Sua confiança e exemplo de perseverança me deram segurança e motivação ao longo de toda a minha trajetória acadêmica.

Agradeço também à minha irmã, cuja disciplina, dedicação e empenho nos estudos sempre foram fonte de inspiração. Seu exemplo constante me incentivou a buscar excelência em meus próprios esforços e a enfrentar os desafios com confiança e determinação.

Minha sincera gratidão se dirige à minha namorada, por seu apoio constante, paciência e incentivo inabalável ao longo de toda a graduação. Seu encorajamento foi fundamental para minha conquista, me fazendo a acreditar mais em meu potencial e a superar e não desistir dos desafios que encontrei nessa jornada.

Por fim, mas não menos importante, agradeço aos amigos que fiz na FACOM, que tornaram o percurso acadêmico mais leve, agradável e memorável. Suas companhias, conversas e momentos compartilhados contribuíram significativamente para que minha experiência universitária fosse enriquecedora e prazerosa.

RESUMO

Este trabalho tem por objetivo apresentar e documentar o desenvolvimento de uma aplicação web em formato de *chatbot*, concebida como tecnologia assistiva destinada a apoiar o desenvolvimento cognitivo de crianças com Transtorno do Espectro Autista (TEA). O projeto foi conduzido no âmbito de um estágio supervisionado e fundamenta-se em uma arquitetura de microsserviços *serverless*, implementada na infraestrutura da Amazon Web Services (AWS). A solução integra serviços especializados, dentre os quais se destacam o Amazon Lex, para processamento de linguagem natural, o Amazon Rekognition, para análise de imagens, o Amazon Polly, para síntese de voz, bem como a API da OpenAI, empregada para a geração de narrativas. O sistema desenvolvido proporciona interações multimodais, permitindo que a criança expresse emoções e elabore histórias personalizadas a partir do envio de imagens ou da seleção de cartões visuais. Assim, esta monografia contribui com a proposição de um modelo arquitetural de caráter replicável e economicamente viável, ao mesmo tempo em que demonstra a aplicabilidade de tecnologias emergentes na construção de ferramentas educacionais inclusivas.

Palavras-chave: Transtorno do Espectro Autista. Tecnologia Assistiva. *Chatbot*. Computação em Nuvem. Inteligência Artificial.

ABSTRACT

This work presents and documents the development of a web-based *chatbot* application designed as assistive technology to support cognitive development in children with Autism Spectrum Disorder (ASD). The project was conducted as part of a supervised internship and is built upon a *serverless* microservices architecture deployed on Amazon Web Services (AWS) infrastructure. The solution integrates several specialized services, including Amazon Lex for natural language processing, Amazon Rekognition for image analysis, Amazon Polly for text-to-speech synthesis, and the OpenAI API for narrative generation. The developed system enables multimodal interactions, allowing children to express emotions and create personalized stories through image uploads or visual card selection. This monograph contributes by proposing a replicable and cost-effective architectural model while demonstrating the practical application of emerging technologies in developing inclusive educational tools.

Keywords: Autism Spectrum Disorder. Assistive Technology. Chatbot. Cloud Computing. Artificial Intelligence.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API *Application Programming Interface* (Interface de Programação de Aplicações)

ARN *Amazon Resource Name* (Nome de Recurso da Amazon)

ASR *Automatic Speech Recognition* (Reconhecimento Automático de Fala)

AWS *Amazon Web Services*

BCIs *Brain-Computer Interfaces* (Interfaces Cérebro-Computador)

CAA Comunicação Aumentativa e Alternativa

CDC *Centers for Disease Control and Prevention* (Centros de Controle e Prevenção de Doenças)

CLI *Command Line Interface* (Interface de Linha de Comando)

CNN *Convolutional Neural Network* (Rede Neural Convolucional)

DSM-5 *Diagnostic and Statistical Manual of Mental Disorders* (Manual Diagnóstico e Estatístico de Transtornos Mentais)

FAQ *Frequently Asked Questions* (Perguntas Frequentes)

GPT *Generative Pre-trained Transformer* (Transformador Pré-treinado Generativo)

IA Inteligência Artificial

IAM *Identity and Access Management* (Gerenciamento de Identidade e Acesso)

IBGE Instituto Brasileiro de Geografia e Estatística

IDE *Integrated Development Environment* (Ambiente de Desenvolvimento Integrado)

IoT *Internet of Things* (Internet das Coisas)

LLMs *Large Language Models* (Modelos de Linguagem de Grande Escala)

NER *Named Entity Recognition* (Reconhecimento de Entidade Nomeada)

NLU *Natural Language Understanding* (Compreensão de Linguagem Natural)

NLP *Natural Language Processing*

PECS *Picture Exchange Communication System* (Sistema de Comunicação por Troca de Figuras)

PLN Processamento de Linguagem Natural

RNN *Recurrent Neural Network* (Rede Neural Recorrente)

TA Tecnologia Assistiva

TEA Transtorno do Espectro Autista

TTS *Text-to-Speech* (Conversão de Texto em Fala)

UFU Universidade Federal de Uberlândia

SUMÁRIO

1 INTRODUÇÃO

O Transtorno do Espectro Autista (TEA) é uma condição do neurodesenvolvimento caracterizada por dificuldades persistentes na comunicação social, na interação interpessoal e por padrões restritivos e repetitivos de comportamento, interesses ou atividades. Por se tratar de um espectro, o autismo manifesta-se de diferentes formas, com níveis variados de comprometimento e habilidades, exigindo abordagens individualizadas em contextos como educação, saúde e desenvolvimento social (World Health Organization, 2023). O diagnóstico precoce, aliado a um acompanhamento terapêutico adequado, é considerado essencial para o progresso da criança, especialmente durante a primeira infância, fase crítica para o desenvolvimento de competências cognitivas, socioemocionais e de linguagem (Associação Brasileira de Autismo, Comportamento e Intervenção (ABRACI-DF), 2024).

Nos últimos anos, as tecnologias digitais têm se consolidado como ferramentas complementares de apoio pedagógico e terapêutico para indivíduos com TEA. A crescente prevalência do transtorno e a sua complexidade tornam o tema relevante tanto para a sociedade quanto para a comunidade acadêmica. Em âmbito internacional, dados divulgados pelo *Centers for Disease Control and Prevention* (Centros de Controle e Prevenção de Doenças) (CDC) referentes ao ano de 2022 indicam que aproximadamente 1 em cada 31 crianças de 8 anos nos Estados Unidos foi diagnosticada com TEA, o que corresponde a uma prevalência estimada de 3,2% nessa faixa etária (Centers for Disease Control and Prevention, 2023). No Brasil, a dimensão dessa realidade foi, pela primeira vez, mapeada em escala nacional pelo Censo Demográfico de 2022, que identificou 2,4 milhões de pessoas com diagnóstico de autismo, correspondendo a 1,2% da população do país (Instituto Brasileiro de Geografia e Estatística (IBGE), 2024). Esses números reforçam a urgência de políticas públicas e práticas educacionais e terapêuticas que sejam inclusivas e eficazes.

Nesse sentido, a análise dos dados nacionais evidencia um aspecto relevante para a formulação de estratégias de intervenção. A prevalência do TEA mostra-se significativamente elevada durante a infância, atingindo 2,6% na faixa etária de 5 a 9 anos (Instituto Brasileiro de Geografia e Estatística (IBGE), 2024). Essa concentração etária configura uma janela crítica de desenvolvimento, na qual intervenções bem estruturadas podem potencializar a aquisição de habilidades e promover melhorias na qualidade de vida das crianças diagnosticadas. Nesse cenário, observa-se um crescimento no uso de tecnologias digitais como instrumentos de apoio pedagógico e terapêutico, com aplicativos interativos, sistemas de ensino personalizados e plataformas acessíveis sendo utilizados como recursos complementares no processo educacional, promovendo o desenvolvimento de competências sociais, comunicativas e de autonomia, de forma alinhada às especificidades de cada

indivíduo.

Nesse contexto, diversas pesquisas realizadas no meio acadêmico nacional têm explorado o uso de tecnologias como ferramenta de apoio ao desenvolvimento de crianças com TEA. (DANTAS, 2022), por exemplo, propôs uma abordagem computacional que combina redes neurais convolucionais e modelagem 3D em jogos sérios para aprimorar a habilidade de reconhecimento facial. Já (DINIZ, 2023) investigou o uso da robótica socialmente assistiva no ensino de habilidades emocionais, desenvolvendo um robô interativo capaz de se comunicar por meio de expressões e atividades lúdicas. (RUFINO, 2020), por sua vez, elaborou um jogo digital fundamentado na abordagem de Reuven Feuerstein, com ênfase na mediação pedagógica como estratégia para promover o desenvolvimento cognitivo infantil. Complementando essas iniciativas, o trabalho de (VALENTIM, 2023) analisou o uso da Inteligência Artificial (IA) como recurso para estimular a atenção compartilhada em crianças na primeira infância, por meio de uma abordagem adaptativa e personalizada baseada no desempenho de cada usuário. Conjuntamente, esses estudos demonstram o potencial das tecnologias digitais como aliadas no suporte terapêutico e educacional de crianças com TEA, beneficiando também suas redes de apoio, incluindo familiares, educadores e profissionais da saúde.

Apesar dos avanços, ainda existem lacunas em termos de personalização, usabilidade e adoção de abordagens mais lúdicas e centradas na criança no desenvolvimento de tecnologias voltadas ao público com TEA. Considerando o cenário demográfico brasileiro, que evidencia uma alta concentração de diagnósticos na primeira e segunda infância, a necessidade de ferramentas acessíveis e eficazes torna-se ainda mais evidente. Nesse contexto, este trabalho tem como objetivo apresentar, contextualizar e documentar o desenvolvimento de um aplicativo web realizado no âmbito de um programa de bolsas em parceria com a Universidade Federal de Uberlândia (UFU). Em atendimento às normas institucionais, não foi permitido divulgar os nomes da empresa e da instituição parceira atendida, mas o projeto foi conduzido em conformidade com as diretrizes e metodologias estabelecidas nesse contexto. O sistema foi concebido para auxiliar o desenvolvimento de crianças com autismo, oferecendo funcionalidades que promovem a aprendizagem, a comunicação e o estímulo de habilidades cognitivas e sociais, com uma abordagem acessível, interativa e centrada no público infantil. Além disso, o trabalho discute os recursos de personalização e acompanhamento voltados a pais, educadores e profissionais da saúde, contribuindo para reflexões sobre o papel das tecnologias digitais na educação inclusiva e nas práticas de intervenção terapêutica.

1.1 Objetivo Geral

Apresentar, contextualizar e descrever de forma sistemática o desenvolvimento de um aplicativo web voltado ao apoio de crianças com TEA, destacando suas funcionalidades interativas, acessíveis e personalizáveis, com ênfase na promoção da aprendizagem, da comunicação e no estímulo a habilidades cognitivas e sociais. O trabalho propõe documentar a criação de uma solução fundamentada nos princípios da arquitetura de microserviços e da computação *serverless*, utilizando serviços da *Amazon Web Services* (AWS), com o objetivo de assegurar baixo custo operacional, flexibilidade, escalabilidade e facilidade de manutenção.

1.2 Objetivos Específicos

1. Discutir o potencial do aplicativo enquanto tecnologia assistiva complementar em contextos educacionais e terapêuticos, contribuindo para a promoção da inclusão, da autonomia e da qualidade de vida de crianças com TEA.

2. Relatar de forma detalhada o processo de concepção, metodologia adotada para o desenvolvimento do aplicativo web proposto, dando ênfase nas tecnologias e ferramentas utilizadas ao longo do projeto.

3. Evidenciar as funcionalidades implementadas no sistema, com destaque para aquelas voltadas ao estímulo de competências socioemocionais, comunicacionais e cognitivas, por meio de estratégias lúdicas e interativas adaptadas ao público infantil.

4. Apresentar a arquitetura técnica da aplicação desenvolvida na nuvem AWS, justificando as decisões de projeto com base nos princípios da computação *serverless* e da arquitetura de microserviços, visando garantir escalabilidade, flexibilidade e baixo custo operacional.

1.3 Motivação

Com o aumento no número de diagnósticos de TEA, sobretudo em crianças em idade escolar, torna-se evidente a urgência de iniciativas que promovam a inclusão, o desenvolvimento e o bem-estar desses indivíduos. Segundo dados recentes do Instituto Brasileiro de Geografia e Estatística (IBGE), mais de 2,4 milhões de pessoas no Brasil já foram diagnosticadas com TEA, com a maior incidência na faixa etária de 5 a 9 anos (Instituto Brasileiro de Geografia e Estatística (IBGE), 2024). A importância dessa etapa do desenvolvimento é ressaltada pela Teoria do Desenvolvimento Cognitivo, proposta por Jean Piaget, que a descreve como um período crucial em que a criança passa a desenvolver

formas de pensamento mais lógicas e concretas, superando o raciocínio predominantemente fantasioso e intuitivo das fases anteriores (PAPALIA DIANE E.; FELDMAN, 2013). Nesse estágio, o pensamento torna-se mais estruturado e o egocentrismo infantil é gradualmente superado, o que contribui para uma maior receptividade a intervenções pedagógicas e terapêuticas (CARNEIRO, s.d.). Diante desse cenário, investir em ferramentas tecnológicas que complementem as práticas já existentes revela-se essencial.

Diante desse contexto, a escolha por desenvolver um aplicativo voltado a esse público específico foi motivada tanto pela relevância social do tema quanto pela constatação de que muitas soluções digitais ainda carecem de personalização e adequação às necessidades reais dos usuários. A proposta de um sistema interativo, acessível e lúdico inspira-se nos princípios piagetianos, segundo os quais a aprendizagem é um processo ativo que se constrói por meio da interação com o meio (CARNEIRO, s.d.), buscando, assim, suprir essas lacunas. A atividade lúdica e a representação simbólica, fundamentais durante a infância, são exploradas como instrumentos de mediação para o aprendizado, oferecendo uma solução prática e adaptável que favoreça o desenvolvimento cognitivo e social das crianças com TEA, ao mesmo tempo em que oferece apoio às famílias, educadores e profissionais da saúde envolvidos nesse processo.

A realização deste trabalho, no âmbito do estágio obrigatório supervisionado, permitiu a aplicação prática dos conhecimentos adquiridos ao longo da formação acadêmica, abrangendo áreas como computação em nuvem, desenvolvimento web, usabilidade e integração de microserviços. Assim, esta monografia busca não apenas apresentar uma solução tecnicamente viável, mas também oferecer uma contribuição prática e socialmente significativa, ao mesmo tempo em que propõe uma análise crítica e fundamentada das tecnologias e conceitos contemporâneos no campo de Sistemas de Informação.

1.4 Organização do Trabalho

A estrutura deste trabalho está organizada nos capítulos descritos a seguir:

- **Capítulo 3:** apresenta a fundamentação teórica, abordando os conceitos sobre o TEA, Tecnologias Assistivas, *chatbots* e a arquitetura de microsserviços na AWS.
- **Capítulo 4:** descreve a metodologia de desenvolvimento, incluindo a abordagem ágil, as ferramentas utilizadas e a arquitetura do projeto.
- **Capítulo 5:** detalha a implementação técnica de cada módulo do sistema, como o roteador de intenções e os fluxos de geração de histórias.
- **Capítulo 6:** apresenta os pré-requisitos e o passo a passo para a configuração do ambiente de desenvolvimento.

- **Capítulo 7:** fornece um guia detalhado para a implantação completa da solução, desde a infraestrutura na nuvem até a integração com as plataformas externas.
- **Capítulo 8:** demonstra os resultados funcionais da aplicação, validando as funcionalidades desenvolvidas.
- **Capítulo 9:** apresenta a conclusão do trabalho, discutindo as contribuições, as limitações do estudo e as sugestões para trabalhos futuros.
- **Apêndice:** oferece um guia prático com o passo a passo para a obtenção das credenciais de acesso (chaves de API) dos serviços externos utilizados.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Transtorno do Espectro Autista (TEA)

O TEA é definido como uma condição de neurodesenvolvimento de início precoce, caracterizada por déficits qualitativos persistentes que afetam a comunicação e interação social e se manifestam pela presença de padrões de comportamento, interesses ou atividades restritos e repetitivos. Esta definição, consolidada pelo *Diagnostic and Statistical Manual of Mental Disorders* (Manual Diagnóstico e Estatístico de Transtornos Mentais) (DSM-5) da Associação Americana de Psiquiatria (American Psychiatric Association, 2013), unificou quadros anteriormente diagnosticados de forma separada, como o Autismo Infantil, a Síndrome de Asperger e o Transtorno Desintegrativo da Infância, sob um único espectro. A abordagem espectral reconhece a vasta heterogeneidade clínica do transtorno, cujas manifestações variam imensamente em tipo e severidade.

As características do TEA se manifestam de maneira diversa, compondo um espectro amplo de apresentações. No campo da comunicação social, os déficits podem ir desde dificuldades em iniciar e manter uma conversação recíproca até a ausência completa de linguagem verbal. Também são comuns desafios na interpretação e no uso de sinais não verbais, como contato visual, expressões faciais e gestos. Quanto aos padrões de comportamento restritos e repetitivos, podem incluir movimentos motores estereotipados (como balançar as mãos), forte apego a rotinas, interesses intensamente focados e inflexíveis, além de reações sensoriais atípicas, caracterizadas por hiper ou hipossensibilidade a estímulos ambientais, como sons, luzes e texturas. De acordo com a Organização Mundial da Saúde, estima-se que o TEA afete cerca de uma em cada 100 crianças no mundo, número que reflete, em parte, o aumento da conscientização e das práticas diagnósticas ao redor do globo (World Health Organization, 2023).

2.1.1 HISTÓRICO E EVOLUÇÃO DO CONCEITO

A trajetória histórica do conceito de autismo é marcada por significativas transformações paradigmáticas. Embora Eugen Bleuler tenha introduzido o termo autismo em 1911 para descrever o retraimento social observado em pacientes com esquizofrenia (BLEULER, 1950), foi somente na década de 1940 que a condição começou a ser delineada como um transtorno do desenvolvimento infantil. Em 1943, o psiquiatra Leo Kanner, em seu trabalho seminal *Autistic Disturbances of Affective Contact*, descreveu um grupo de 11 crianças que exibiam uma "incapacidade inata de formar o contato afetivo, normal e biologicamente fornecido com outras pessoas", associada a um "desejo ansiosamente

obsessivo pela manutenção da mesmice" (KANNER, 1943). Quase simultaneamente, na Áustria, o pediatra Hans Asperger, cujo trabalho só foi amplamente reconhecido décadas depois, descreveu um grupo de crianças com dificuldades sociais semelhantes, mas que não apresentavam os mesmos atrasos significativos na linguagem e no desenvolvimento cognitivo, cunhando a expressão "psicopatia autista" (ASPERGER, 1991).

Nas décadas de 1960 e 1970, teorias psicodinâmicas exerceram forte influência sobre a compreensão do autismo, especialmente a ideia da "mãe-geladeira", proposta por (BETTELHEIM, 1967). Segundo essa visão, o transtorno teria origem em fatores emocionais e na dinâmica familiar, em especial no comportamento frio e distante das mães. Com o avanço das pesquisas, essa hipótese foi amplamente desacreditada e abandonada pela comunidade científica. A virada ocorreu entre os anos 1980 e 1990, quando uma robusta base de evidências científicas começou a apontar para uma etiologia neurobiológica e genética (RITVO et al., 1985; BAILEY et al., 1995). Com o avanço da neurociência e da genética, o TEA passou a ser inequivocamente reconhecido como uma condição do cérebro (HAPPÉ; RONALD; PLOMIN, 2006; MUELLER, 2007). A consolidação da noção de "espectro", a partir dos anos 2000 e oficializada no DSM-5 em 2013, representa o auge dessa evolução, refletindo o entendimento de que as diversas manifestações do autismo compartilham uma base etiológica comum, mas se expressam em diferentes níveis de intensidade e funcionalidade (American Psychiatric Association, 2013).

2.1.2 AVANÇOS RECENTES E O PAPEL DAS TECNOLOGIAS ASSISTIVAS

A interseção entre tecnologia e TEA tem se mostrado um campo fértil para a inovação, permitindo o desenvolvimento de ferramentas assistivas que visam minimizar os desafios centrais do transtorno e favorecer o desenvolvimento de habilidades essenciais. Os avanços recentes em tecnologias digitais têm proporcionado soluções cada vez mais sofisticadas e personalizadas, especialmente na área de Comunicação Aumentativa e Alternativa (CAA). Aplicativos como o *Proloquo2Go* e sistemas digitais baseados no *Picture Exchange Communication System* (Sistema de Comunicação por Troca de Figuras) (PECS) vêm transformando a maneira como indivíduos não verbais ou com fala limitada se expressam, possibilitando a construção de sentenças complexas por meio de símbolos e síntese de voz, o que amplia significativamente as oportunidades de interação e participação social (LIGHT; MCNAUGHTON, 2014).

Ao mesmo tempo, essas inovações tecnológicas têm se mostrado valiosas para o desenvolvimento de habilidades sociais em pessoas com TEA, complementando a comunicação assistiva com estratégias que promovem a percepção social e a regulação emocional. Nesse contexto, (VALENTIM, 2023) propõe em sua tese uma abordagem computacional adaptativa que utiliza técnicas de IA para ajustar dinamicamente a sequência e a complexidade dos exercícios, criando um ambiente personalizado que fortalece a atenção

compartilhada em crianças de 4 a 5 anos com dificuldades sócio-comunicativas, oferecendo um espaço seguro e controlado para a prática de interações sociais.

Complementando essa perspectiva, (DANTAS, 2022) desenvolveu uma ferramenta baseada em jogos sérios que combina descritores manuais e *Convolutional Neural Network* (Rede Neural Convolucional)s (CNNs) para detectar e reconhecer expressões faciais. Avaliada em cinco sessões e apoiada por bancos de dados públicos (CK+, FER2013, RAF-DB e MMI), a solução demonstrou eficácia no aprimoramento das habilidades de percepção e reconhecimento de emoções básicas em indivíduos com TEA, contribuindo para o desenvolvimento socioemocional e para a ampliação das capacidades de interação social. Dessa forma, essas abordagens ilustram como a integração de tecnologias digitais, IA e jogos educativos pode potencializar o aprendizado, a comunicação e a socialização de crianças com TEA, fornecendo recursos adaptativos que se ajustem às necessidades individuais e que promovam experiências de aprendizado mais inclusivas e efetivas.

2.2 *Chatbots* e Sistemas Conversacionais

2.2.1 DEFINIÇÕES, ARQUITETURAS E TIPOLOGIAS

Chatbots, ou agentes conversacionais, são softwares projetados para interagir com usuários humanos por meio da linguagem natural, seja escrita ou falada, simulando conversações coerentes, contextualizadas e relevantes. Eles podem atender a diferentes necessidades, desde respostas básicas pré-programadas até respostas complexas geradas por IA. A arquitetura de um *chatbot* é definida pelo mecanismo de formulação das respostas, que constitui o núcleo da interação.

De acordo com a IBM (IBM Brasil, 2025a), é possível classificar os *chatbots* em diferentes categorias, que variam em nível de sofisticação e aplicabilidade. A seguir, apresentam-se os principais tipos, acompanhados de uma breve descrição de suas características:

- ***Chatbots* baseados em menus ou botões**: forma mais simples de interação, em que o usuário seleciona opções predefinidas em menus, seguindo scripts que guiam a conversa até a resposta desejada.
- ***Chatbots* baseados em regras**: operam com fluxos de conversação definidos por lógica condicional "se, então". São eficientes para tarefas repetitivas ou *Frequently Asked Questions* (Perguntas Frequentes)s (FAQs), mas têm limitações ao lidar com variações linguísticas ou perguntas inesperadas.
- ***Chatbots* de IA**: utilizam *Natural Language Understanding* (Compreensão de Linguagem Natural) (NLU) para compreender a intenção do usuário independentemente

da formulação da pergunta. Empregam aprendizado de máquina e podem aprimorar continuamente suas respostas.

- ***Voice chatbots***: projetados para interação por voz, empregando técnicas de *Natural Language Processing* (NLP) para interpretar comandos falados, oferecendo uma experiência mais natural e acessível.
- ***Chatbots* de IA generativa**: representam uma nova geração de sistemas conversacionais capazes de gerar respostas inéditas e contextualizadas, demonstrando empatia e adaptando-se ao estilo de comunicação do usuário. Não dependem de respostas pré-programadas.
- ***Chatbots* híbridos**: combinam abordagens baseadas em regras com aprendizado de máquina, oferecendo flexibilidade e capacidade de adaptação. Automatizam tarefas repetitivas e aprendem com as interações para melhorar continuamente as respostas.

A Tabela 1 sintetiza essa classificação, apresentando para cada tipo de *chatbots* sua descrição geral e exemplos de aplicação prática, o que facilita a comparação entre as diferentes abordagens.

Tipo	**Descrição**	**Exemplo de aplicação**
Menus ou botões	Interação baseada na seleção de opções predefinidas em menus, seguindo scripts específicos.	FAQs simples, sistemas de triagem
Baseados em regras	Fluxos de conversação automatizados usando lógica condicional "se, então".	Atendimento automatizado, respostas de FAQ
IA	Utilizam NLU e aprendizado de máquina para interpretar a intenção do usuário e gerar respostas adaptadas.	Assistentes virtuais inteligentes
Voice Chatbots	Permitem interação por voz, interpretando comandos falados com NLP.	Assistentes de voz, call centers
IA generativa	Utilizam IA generativa para produzir respostas inéditas, adaptadas ao contexto, demonstrando empatia.	*Chatbots* corporativos avançados, suporte personalizado
Híbridos	Combinam regras e aprendizado de máquina para maior flexibilidade e adaptação contínua.	Sistemas de atendimento com aprendizado progressivo

Tabela 1 – Tipos de *chatbots* segundo a IBM. Fonte: Do Autor

2.2.2 EVOLUÇÃO HISTÓRICA DOS *CHATBOTS*

A história dos sistemas conversacionais, segundo Nsaif et al. (2024), remonta à década de 1960, com o desenvolvimento do **ELIZA**, criado por Joseph Weizenbaum em 1966. Este sistema pioneiro simulava uma psicoterapeuta utilizando técnicas de correspondência de padrões (*pattern-matching*) e regras linguísticas simples para reformular a entrada do usuário como uma resposta, representando um passo fundamental na interação homem-máquina.

Em 1972, o software **PARRY** emulou o comportamento de uma pessoa com esquizofrenia paranoide, evidenciando a complexidade de replicar processos cognitivos humanos. No final da década de 1980, **JabberWacky** introduziu uma abordagem de aprendizado baseada em conversas anteriores, combinando conhecimento contextual com técnicas iniciais de aprendizado de máquina. Já em 1995, **A.L.I.C.E.** *(Artificial Linguistic Internet Computer Entity* — Entidade Computacional Linguística Artificial da Internet) utilizou regras baseadas em *templates* para simular diálogos mais naturais, ampliando a capacidade de interação dos *chatbots*.

Contudo, os avanços mais significativos ocorreram com a consolidação do aprendizado de máquina e, mais recentemente, com a popularização do *deep learning*. A partir de 2011, assistentes virtuais como **SIRI** e, posteriormente, **Alexa** e **Cortana** (2014), passaram a utilizar PLN e reconhecimento de voz para executar tarefas e fornecer informações de forma automatizada. A evolução se intensificou com o surgimento dos *Large Language Models* (Modelos de Linguagem de Grande Escala) (LLMs), cujo paradigma foi transformado com o lançamento do **ChatGPT** em 2020. Esses modelos, treinados com vastos volumes de dados textuais, possibilitaram avanços expressivos na geração de respostas contextualmente coerentes, elevando a naturalidade das interações e inaugurando uma nova fase na evolução dos *chatbots*.

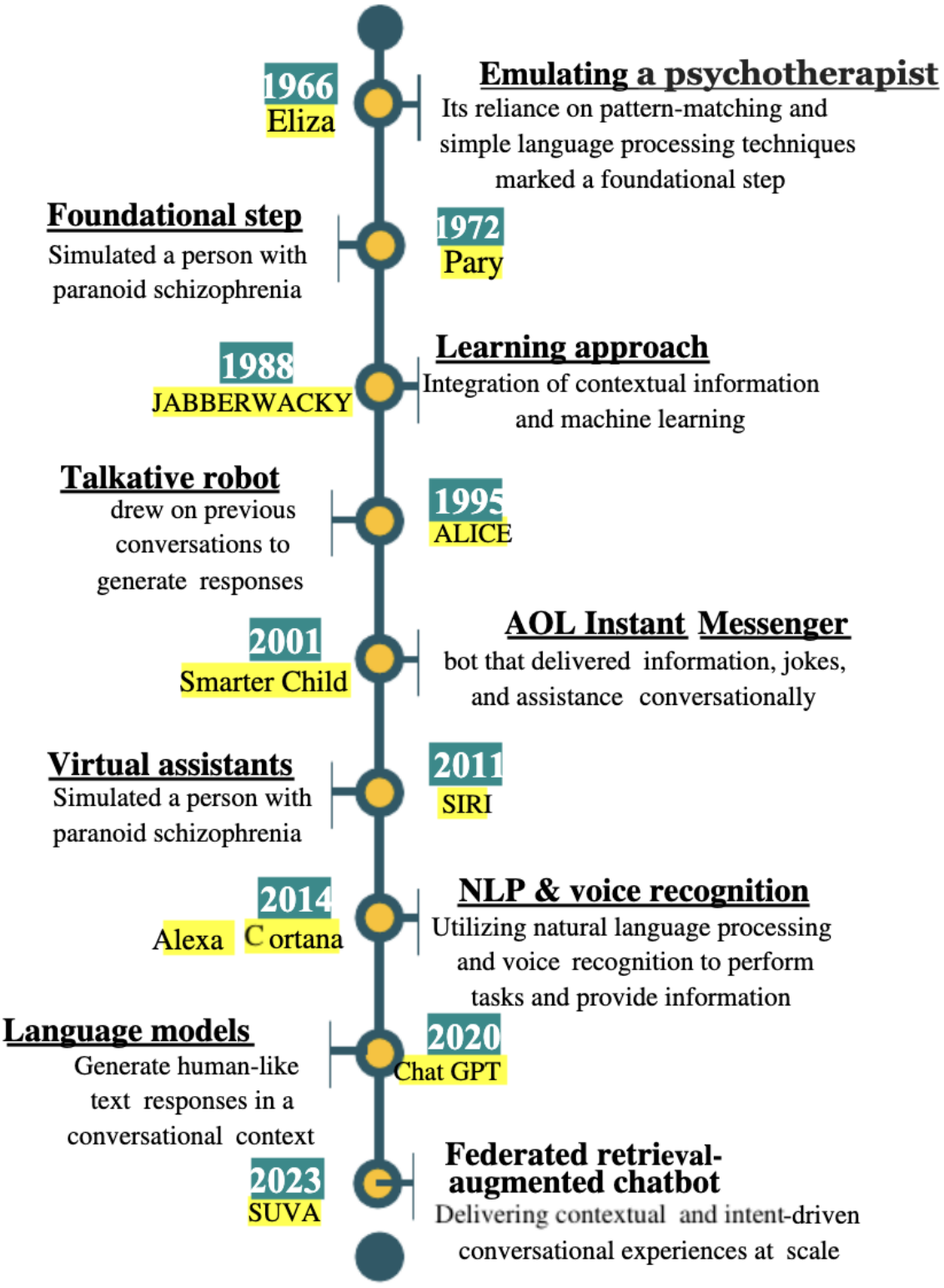

Figura 1 – Evolução Histórica dos *Chatbots* Fonte: (NSAIF et al., 2024)

2.2.3 *CHATBOTS* E TRANSTORNO DO ESPECTRO AUTISTA (TEA)

No contexto do TEA, os *chatbots* emergem como ferramentas tecnológicas com potencial para o apoio à aprendizagem, comunicação e desenvolvimento de habilidades sociais. Crianças diagnosticadas com TEA frequentemente apresentam dificuldades relacionadas à interação social, comunicação verbal e interpretação de contextos sociais complexos (PINTO, 2016). Diante dessa realidade, sistemas conversacionais adequadamente projetados podem constituir ambientes seguros, controlados e previsíveis, propiciando oportunidades estruturadas para a prática e o aperfeiçoamento dessas competências essenciais.

As vantagens pedagógicas dessas ferramentas manifestam-se em múltiplas dimensões. Em primeiro lugar, destaca-se a simplicidade da interface, visto que muitos sistemas permitem que a criança interaja exclusivamente através de botões ou seleções em opções pré-definidas, eliminando a necessidade de formular respostas textuais complexas (IBM Brasil, 2025a). Essa característica reduz a frustração e promove maior autonomia do usuário, sendo complementada pela ampla acessibilidade dos *chatbots*, que podem ser utilizados em diversos dispositivos e contextos, incluindo ambientes domésticos, escolares e terapêuticos.

Além disso, a consistência e a capacidade de repetição desses sistemas favorecem o estabelecimento de rotinas estruturadas e o reforço de comportamentos positivos, ao mesmo tempo em que permitem adaptações personalizadas conforme o progresso individual de cada criança. Essas características são identificadas por estudos recentes como elementos fundamentais em *chatbots* bem projetados para esse público (GU et al., 2025).

Por reunirem essas características, os *chatbots* atuam como facilitadores da comunicação, complementando estratégias pedagógicas tradicionais. Dessa forma, apresentam-se como ferramentas promissoras para o desenvolvimento de soluções tecnológicas inclusivas voltadas a crianças com TEA, integrando acessibilidade, aprendizagem social e suporte ao desenvolvimento cognitivo.

2.3 Processamento de Linguagem Natural (PLN)

O PLN surge como a disciplina que busca transpor a barreira entre a comunicação humana e a interpretação computacional (IBM Brasil, 2025b). Seu objetivo é capacitar máquinas não apenas a ler palavras, mas a compreender contexto, intenção e nuances presentes em textos e falas, de forma análoga à cognição humana. Para isso, integra conhecimentos de linguística computacional com modelos estatísticos e, mais recentemente, de *deep learning*.

A linguagem humana apresenta desafios por sua natureza não estruturada. Para

superá-los, o PLN segue etapas que transformam o texto bruto em dados estruturados e interpretáveis. No pré-processamento, o texto é limpo e padronizado por meio de técnicas como tokenização e lematização. Em seguida, na extração de *features*, o texto é convertido em representações numéricas (vetores), processáveis pelos algoritmos de aprendizado de máquina.

Com a linguagem em formato compreensível para máquinas, o PLN permite executar tarefas complexas, como a identificação de entidades (*Named Entity Recognition* (Reconhecimento de Entidade Nomeada) (NER)) e a análise de sentimentos, extraindo insights de grandes volumes de dados.

A evolução das abordagens reflete a transição do determinístico para o probabilístico. Sistemas iniciais baseavam-se em regras rígidas, pouco flexíveis e difíceis de escalar. O paradigma mudou com modelos estatísticos e, mais impactante, com arquiteturas de *deep learning*, como os *Transformers*, capazes de aprender padrões diretamente de dados brutos. Isso culminou na atual era da IA generativa, exemplificada por modelos como o *Generative Pre-trained Transformer* (Transformador Pré-treinado Generativo) (GPT).

2.4 Visão Computacional e Reconhecimento de Imagens

A visão computacional é um campo da IA que capacita sistemas a extrair e interpretar informações de dados visuais, como imagens e vídeos (IBM Brasil, 2025c). Enquanto a IA busca emular o raciocínio, a visão computacional foca em replicar a percepção visual, permitindo que máquinas observem e compreendam o ambiente de forma análoga à visão humana, porém com escalabilidade e velocidade superiores. Um exemplo prático desse conceito é apresentado na Figura 2.

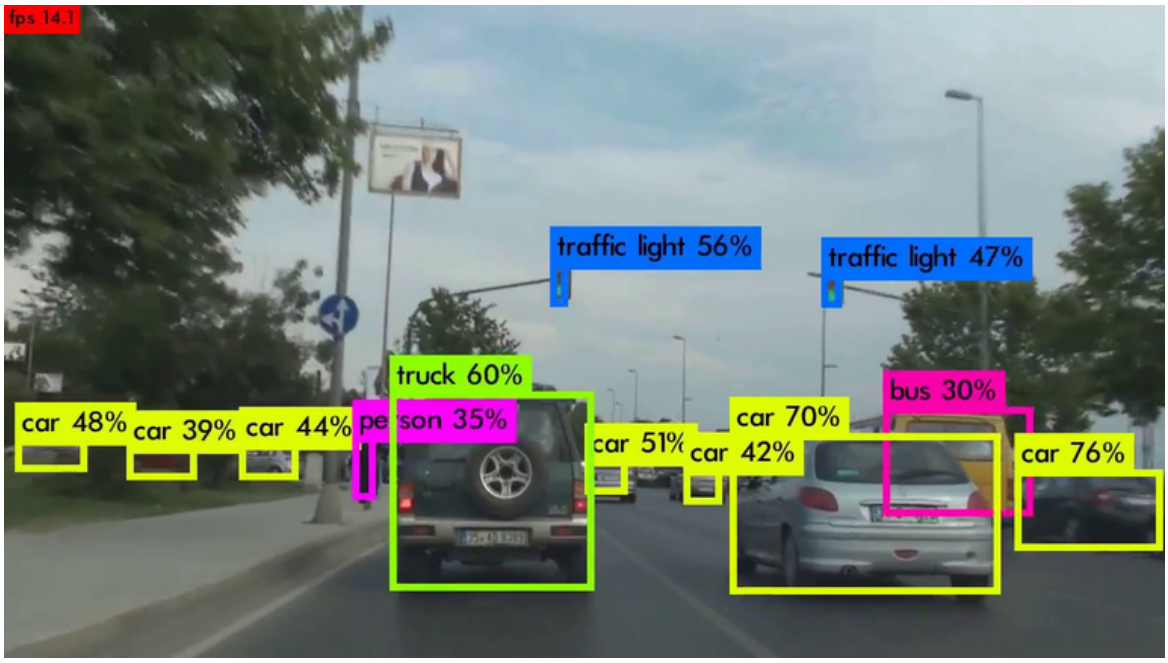

Figura 2 – Representação de uma Rede Neural Convolucional. Fonte: (Serviço Federal de Processamento de Dados (Serpro), 2020-04-03)

O aprendizado da visão computacional baseia-se no treinamento com grandes conjuntos de dados. Por meio da análise iterativa, o sistema desenvolve a capacidade de identificar padrões, extrair características relevantes e classificar objetos de forma autônoma. A base tecnológica para essa tarefa são modelos de *deep learning*, destacando-se as CNNs, que processam informações visuais de forma hierárquica: primeiro identificam características de baixo nível, como bordas e texturas, e depois compõem o reconhecimento de objetos complexos. Para sequências temporais, como vídeos, essa abordagem é frequentemente combinada com *Recurrent Neural Network* (Rede Neural Recorrente)s (RNNs), capazes de interpretar relações contextuais entre os quadros. O processo é ilustrado na Figura 3.

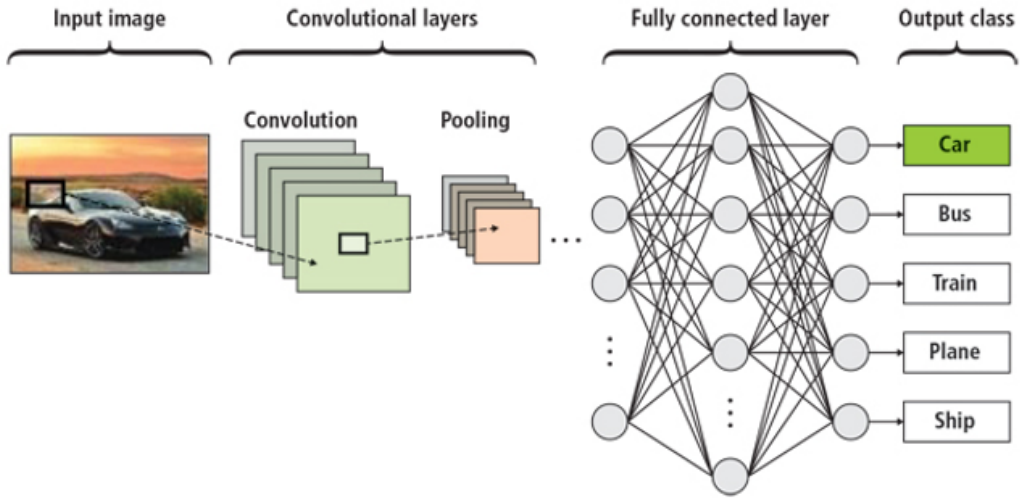

Figura 3 – Representação de uma Rede Neural Convolucional. Fonte: (NVIDIA, 2025)

2.5 Síntese de Voz (TTS)

2.5.1 DEFINIÇÕES E ABORDAGENS TECNOLÓGICAS

A Síntese de Voz, ou *Text-to-Speech* (Conversão de Texto em Fala) (TTS), é a tecnologia responsável pela geração artificial da fala humana a partir de um texto escrito (TAYLOR, 2009). O objetivo é produzir uma fala que seja não apenas inteligível, mas também natural e expressiva. Historicamente, as abordagens para TTS podem ser agrupadas em:

1. **Síntese Concatenativa:** Constrói a fala unindo pequenos segmentos de áudio de uma fala humana pré-gravada. Embora possa soar natural, é limitada pelo banco de dados e pode soar "recortada".

2. **Síntese Paramétrica:** Utiliza um modelo estatístico para gerar os parâmetros acústicos da fala (frequência, duração), os quais são convertidos em áudio. Apesar desse método ser mais flexível, tende a soar mais robótico.

3. **Síntese Baseada em Redes Neurais Profundas:** A abordagem mais moderna, que utiliza redes neurais para mapear diretamente o texto ou representações linguísticas intermediárias em formas de onda de áudio ou espectrogramas.

2.5.2 AVANÇOS RECENTES EM NATURALIDADE E EXPRESSIVIDADE

A qualidade da síntese de voz avançou significativamente com a introdução de modelos de *deep learning*. O **WaveNet**, desenvolvido pela DeepMind (OORD et al., 2016), representou um marco ao gerar a forma de onda do áudio diretamente, proporcionando uma naturalidade sem precedentes. Modelos subsequentes, como **Tacotron** e **FastSpeech**, otimizaram esse processo, permitindo a geração de fala de alta qualidade em tempo real. Esses avanços tornaram a fala sintetizada mais fluida, com entonação e prosódia mais humanas, possibilitando a transmissão de nuances emocionais. As aplicações são especialmente relevantes em tecnologias assistivas, incluindo leitores de tela para pessoas com deficiência visual, sistemas de CAA para indivíduos com TEA ou outras condições que afetam a fala, e ferramentas para o ensino da pronúncia em novos idiomas.

2.6 Tecnologia Assistiva (TA)

2.6.1 DEFINIÇÕES, ESCOPO E CLASSIFICAÇÕES

A TA é um termo abrangente que engloba um vasto campo de conhecimento, de caráter interdisciplinar, que inclui produtos, recursos, metodologias, estratégias, práticas e serviços. O objetivo da TA é promover a funcionalidade e, consequentemente, a autonomia, independência, qualidade de vida e inclusão social de pessoas com deficiência, incapacidades ou mobilidade reduzida (ZALLIO; OHASHI, 2022). A Lei Brasileira de Inclusão (Lei nº 13.146/2015) define TA como "produtos, equipamentos, dispositivos, recursos, metodologias, estratégias, práticas e serviços que objetivem promover a funcionalidade, relacionada à atividade e participação da pessoa com deficiência ou com mobilidade reduzida, visando à sua autonomia, independência, qualidade de vida e inclusão social". As TAs podem ser classificadas de diversas formas, seja pelo seu grau de sofisticação tecnológica (de baixa a alta tecnologia) ou por sua área de aplicação funcional, como comunicação, mobilidade, acesso ao computador, adequação postural, entre outras.

2.6.2 AVANÇOS E APLICAÇÕES ESPECÍFICAS NO CONTEXTO DO TEA

No contexto do TEA, a tecnologia assistiva desempenha um papel transformador, oferecendo suporte frente aos desafios da condição. Avanços recentes potencializaram ainda mais esse impacto. A integração com a *Internet of Things* (Internet das Coisas) (IoT) possibilita a criação de ambientes inteligentes, capazes de se adaptar às sensibilidades sensoriais do indivíduo. A IA viabiliza a personalização em larga escala, permitindo o desenvolvimento de jogos educativos que se ajustam dinamicamente ao nível de habilidade e aos interesses da criança, assim como sistemas de CAA que aprendem o vocabulário e o estilo de comunicação do usuário. As *Brain-Computer Interfaces* (Interfaces Cérebro-

Computador) (BCIs), embora ainda em fase de pesquisa, representam uma fronteira promissora para novas formas de comunicação e controle ambiental. Entre as aplicações concretas para TEA destacam-se sistemas de CAA, aplicativos para organização de rotinas e gerenciamento do tempo (utilizando pranchas de comunicação visual digital), além de uma ampla gama de jogos educativos e terapêuticos projetados para o desenvolvimento de habilidades sociais, cognitivas e de vida diária.

2.7 Arquitetura de Microsserviços na AWS

A arquitetura de microsserviços na AWS viabiliza a construção de aplicações modernas que se caracterizam pela alta escalabilidade, resiliência e flexibilidade. Em contraposição às arquiteturas monolíticas, nas quais as funcionalidades se encontram fortemente acopladas em uma base de código unificada, os microsserviços adotam uma abordagem modular. Essa abordagem segmenta o sistema em componentes menores e independentes, onde cada um é responsável por uma funcionalidade específica. Tal desacoplamento resulta na redução da complexidade do sistema, simplifica os processos de manutenção e impede que falhas pontuais comprometam a aplicação em sua totalidade. Além disso, permite que equipes de desenvolvimento atuem de forma paralela e autônoma, o que acelera o desenvolvimento de software e possibilita ciclos de entrega mais curtos. Outro benefício relevante é a escalabilidade granular, que permite escalar apenas os serviços de maior demanda, otimizando a alocação de recursos e os custos operacionais. A AWS dispõe de um ecossistema completo para suportar essa arquitetura, incluindo soluções de contêineres, sistemas de mensageria e computação *serverless*, ampliando a eficiência operacional e garantindo uma integração ágil com as diversas necessidades de negócio (SERVICES, 2025).

2.7.1 AMAZON LEX

O Amazon Lex é um serviço de IA da AWS destinado à implementação de interfaces conversacionais interativas, tanto por texto quanto por voz. O serviço utiliza as mesmas tecnologias de *Automatic Speech Recognition* (Reconhecimento Automático de Fala) (ASR) e NLU que fundamentam o assistente Amazon Alexa. Com isso, o Lex possibilita o desenvolvimento de *chatbots* e assistentes virtuais capazes de operar em distintos cenários de aplicação e interagir com múltiplos perfis de usuários. Sua integração nativa com outros serviços da AWS permite a construção de fluxos de conversação automatizados, escaláveis e personalizados, aplicáveis em setores como atendimento ao cliente, educação e saúde (SERVICES, 2025).

2.7.2 AWS LAMBDA

O AWS Lambda é um serviço de computação *serverless* que executa código em resposta a eventos, abstraindo a necessidade de provisionamento ou gerenciamento de servidores. Essa abordagem otimiza os custos operacionais, uma vez que a cobrança é estritamente baseada no tempo de execução do código, e eleva a eficiência do processo de desenvolvimento ao permitir que as equipes se concentrem primordialmente na lógica de negócio. O Lambda apresenta integração nativa com uma vasta gama de serviços da AWS, o que o torna um componente central para a implementação de arquiteturas orientadas a eventos, processamento de dados em tempo real e sistemas de alta escalabilidade, ideais para aplicações modernas que demandam elasticidade e disponibilidade contínua (SERVICES, 2025).

2.7.3 AMAZON S3

O Amazon Simple Storage Service (Amazon S3) é um serviço de armazenamento de objetos projetado para oferecer máxima escalabilidade, durabilidade e segurança, atendendo desde aplicações de pequeno porte até soluções corporativas em larga escala. Sua arquitetura distribuída assegura alta disponibilidade e resiliência dos dados. Recursos como versionamento de objetos, controle de acesso granular e integração nativa com serviços de análise de dados e *machine learning* expandem significativamente suas aplicações. O S3 é amplamente empregado em diversos cenários, que incluem *backup* e recuperação de desastres, arquivamento de dados, processamento e análise de *Big Data*, hospedagem de *websites* estáticos, armazenamento de mídias e suporte a *pipelines* de inteligência artificial (SERVICES, 2025).

2.7.4 AMAZON REKOGNITION

O Amazon Rekognition é um serviço de visão computacional que utiliza algoritmos avançados de *deep learning* para a análise de imagens e vídeos. O serviço permite a identificação e rotulagem de objetos, reconhecimento facial, detecção de texto e análise de atividades em tempo real. Suas funcionalidades incluem, ainda, a moderação de conteúdo e a verificação de identidade baseada em biometria facial. Sua integração nativa com o ecossistema AWS o posiciona como uma ferramenta estratégica para aplicações de segurança, análise de mídias, acessibilidade e automação de processos inteligentes (SERVICES, 2025).

2.7.5 AMAZON POLLY

O Amazon Polly é um serviço de conversão de texto em fala (TTS) que emprega técnicas de *deep learning* para sintetizar vozes com sonoridade natural em múltiplos

idiomas e sotaques. Essa tecnologia contribui para a ampliação da acessibilidade em sistemas digitais, viabilizando a criação de soluções como leitores de tela automáticos, assistentes virtuais com maior imersão e aplicações educacionais inclusivas. O Polly disponibiliza ainda recursos para a personalização da prosódia, como entonação e ritmo da fala, o que permite a criação de experiências auditivas mais envolventes e adaptadas a diferentes contextos de uso (SERVICES, 2025).

2.7.6 AMAZON API GATEWAY

O Amazon API Gateway é um serviço gerenciado que funciona como uma camada de abstração para a criação, publicação, manutenção, monitoramento e segurança de APIs em qualquer escala. Ele atua como uma fachada (*façade*) unificada, servindo como a principal porta de entrada para as requisições destinadas a serviços de *backend*, como funções AWS Lambda, aplicações em contêineres ou outros serviços da AWS e *on-premises*. O serviço gerencia tarefas críticas como o controle de tráfego (*throttling*) para prevenir sobrecargas, a autenticação e autorização de requisições por meio de diversos mecanismos, a validação de dados e o monitoramento de métricas em tempo real. Ao centralizar essas funcionalidades, o API Gateway se torna um componente fundamental para a implementação de arquiteturas de microsserviços e *serverless*, simplificando a exposição de serviços de forma segura e escalável (SERVICES, 2025).

3 METODOLOGIA

3.1 Processo de Desenvolvimento

3.1.1 ESTRUTURA DA EQUIPE E ORGANIZAÇÃO

O desenvolvimento do sistema foi realizado no contexto de um programa de bolsas, contando com uma equipe de seis bolsistas responsável pela concepção, implementação e validação da solução. Para assegurar que o produto estivesse alinhado às demandas pedagógicas e às necessidades específicas do público-alvo, estabeleceu-se uma parceria com uma instituição de ensino. A colaboração de uma professora dessa instituição foi um elemento central na estratégia de desenvolvimento participativo adotada pelo projeto, conforme justificado em detalhe na Seção 3.1.2. Por normas institucionais, os nomes da empresa responsável pelo programa e da instituição parceira não podem ser divulgados.

A interação com a professora ocorreu em dois momentos estratégicos. O primeiro contato deu-se no início da *Sprint* 1, ocasião em que foram discutidas as principais expectativas e sugeridas funcionalidades voltadas às demandas de crianças com TEA. O segundo momento aconteceu ao início da *sprint* 2, ocasião em que foi apresentada uma versão parcial do sistema, permitindo que sugestões de melhoria fossem incorporadas ainda durante o ciclo de desenvolvimento. A Figura 4 ilustra a organização temporal do projeto, destacando os dois pontos de *feedback* com a professora parceira e os principais entregáveis de cada *sprint*.

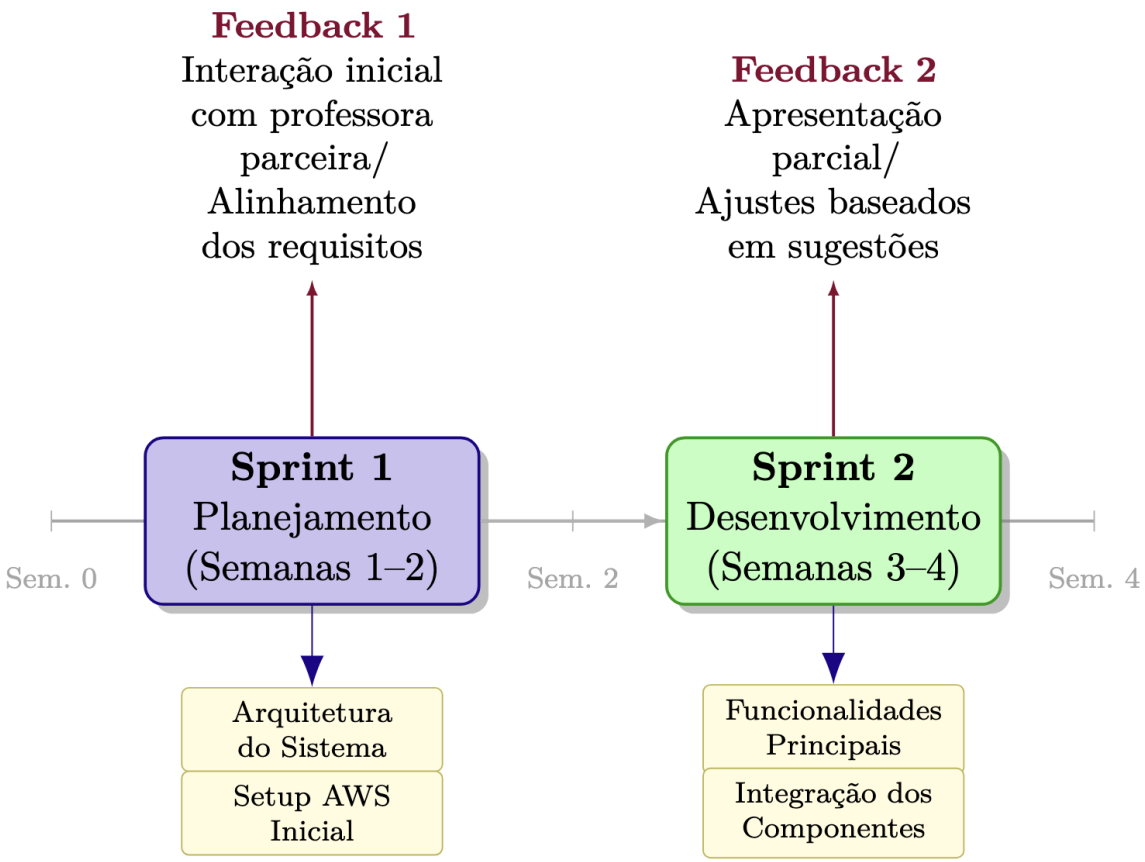

Figura 4 – Cronograma de desenvolvimento com metodologia ágil. Fonte: Do Autor

Devido à limitação temporal do projeto, que contou com apenas quatro semanas, divididas em duas *sprints* de duas semanas cada, não foi viável apresentar o produto finalizado para a instituição parceira. Entretanto, como forma de encerramento do programa, o resultado foi apresentado em um evento interno do programa de bolsas, no qual todos os participantes compartilharam suas soluções desenvolvidas com os demais colaboradores, destacando o processo de desenvolvimento e os principais resultados alcançados.

3.1.2 ABORDAGEM PARTICIPATIVA E COLETA DE FEEDBACK

A concepção de uma ferramenta de tecnologia assistiva, especialmente para crianças com TEA, demanda uma abordagem que vá além dos requisitos puramente técnicos. Por essa razão, adotou-se uma estratégia de desenvolvimento participativo, integrando a perspectiva de um especialista de domínio ao ciclo de desenvolvimento.

A colaboração com a professora da instituição parceira materializou essa estratégia. Desde as primeiras interações, ela apresentou contribuições substanciais que orientaram o desenvolvimento do projeto. Destacou-se, em particular, a necessidade de que o sistema

fosse altamente visual, uma vez que muitas crianças ainda se encontravam em processo de alfabetização e dependiam majoritariamente de recursos gráficos para compreender e interagir com a aplicação.

A partir desse feedback, surgiram duas funcionalidades centrais: a "História por Cards" e a funcionalidade de "Expressar Emoção". A primeira, inspirada na familiaridade das crianças com o uso de pranchetas, funciona como uma porta de entrada para estimular a comunicação e a criatividade, permitindo que a criança participe ativamente da construção narrativa e compare suas produções com as de colegas, refletindo sobre elas a partir de questionamentos do professor. A segunda funcionalidade ampliou a experiência comunicativa ao permitir que a criança não apenas indicasse visualmente seus desejos, mas também os ouvisse por meio do *chatbot*, potencializando compreensão, autonomia e engajamento no processo de interação.

A motivação para incorporar a perspectiva da professora se baseou em três pilares:

- **Validação Pedagógica:** Garantir que as funcionalidades propostas fossem relevantes do ponto de vista pedagógico e adequadas ao contexto real de uso, algo que apenas a experiência prática de um educador pode fornecer.
- **Representação do Usuário:** Diante da limitação de tempo do projeto e das considerações éticas relacionadas ao contato direto com crianças, a professora atuou como representante qualificada do público-alvo. Seu conhecimento sobre dificuldades, interesses e formas de engajamento das crianças com TEA foi fundamental para guiar o design da interação.
- **Alinhamento com a Metodologia Ágil:** O Scrum preconiza ciclos de feedback constantes para validar e refinar o produto. As reuniões com a professora forneceram esse retorno essencial, permitindo ajustes iterativos nos requisitos e na implementação, evitando o risco de desenvolver uma solução inadequada às necessidades dos usuários finais.

Portanto, a participação da professora não foi um evento isolado, mas uma decisão metodológica deliberada, que assegurou a criação de um produto eficaz, útil e centrado no usuário.

3.1.3 METODOLOGIA ÁGIL

A metodologia escolhida para a gestão do trabalho foi o Scrum, adotado como framework ágil de organização (Amazon Web Services, 2025). Embora tenha sido utilizado em um ciclo bastante reduzido, permitiu estruturar entregas incrementais, bem como promover alinhamentos frequentes sobre o progresso. Reuniões diárias de alinhamento

foram realizadas de forma remota por meio do Microsoft Teams, com duração média de quinze minutos. Nessas reuniões, os integrantes da equipe discutiam as atividades concluídas no dia anterior, planejavam o trabalho do dia corrente e reportavam possíveis impedimentos, favorecendo a transparência e a rápida resolução de bloqueios.

Alinhada a essa perspectiva, a monografia foi estruturada para refletir os princípios da metodologia ágil, que privilegia a entrega de software funcional em detrimento de documentação extensa. Em lugar de artefatos tradicionais da engenharia de software, como diagramas formais e descrições detalhadas de requisitos, adotou-se uma documentação prática, elaborada como guia das etapas de implementação, configuração e implantação, permitindo a replicação do projeto e servindo de referência em investigações futuras. A aplicação do Scrum em ciclo reduzido reforçou essa escolha, dispensando formalismos cujo custo superaria o benefício e favorecendo a adaptação contínua e a priorização de entregas incrementais. Dessa forma, o enfoque recai sobre o valor prático e a disseminação do conhecimento gerado no desenvolvimento de uma arquitetura contemporânea alinhada aos princípios ágeis.

3.1.4 FERRAMENTAS DE GESTÃO E COLABORAÇÃO

O controle de versão foi realizado utilizando Git em conjunto com o GitHub, que atuou como repositório centralizado para o armazenamento e gerenciamento do código. A colaboração da equipe foi organizada por meio de *commits* frequentes e *Pull Requests*, acompanhados de revisões obrigatórias, de forma a assegurar a qualidade do código. Diferentemente de projetos de maior duração, não foram utilizadas estratégias de *branching* formais, como o GitFlow, nem ferramentas de automação de integração contínua, como o GitHub Actions, em virtude da restrição temporal do programa.

As atividades de comunicação e acompanhamento de projeto foram conduzidas majoritariamente via Microsoft Teams, que serviu como plataforma central para reuniões, discussões técnicas e alinhamentos de tarefas, garantindo a continuidade da colaboração entre os membros da equipe.

3.1.5 FASES DE DESENVOLVIMENTO

O ciclo de desenvolvimento foi organizado em duas *sprints*, cada uma com duração de duas semanas, totalizando quatro semanas.

Na *Sprint* 1, foram realizadas atividades de planejamento e arquitetura. Nesse período, a equipe se dedicou à definição dos requisitos funcionais e não funcionais, à modelagem da arquitetura inicial do sistema, à seleção das tecnologias e serviços da AWS que seriam empregados e à configuração do ambiente de desenvolvimento. Além disso, ocorreu a primeira reunião com a professora da instituição parceira, que contribuiu com

sugestões a partir de sua experiência pedagógica, permitindo que as funcionalidades fossem pensadas desde o início em consonância com o público-alvo.

Na *Sprint* 2, as atividades concentraram-se no desenvolvimento e integração. Foram implementados os módulos principais do sistema, incluindo as funções Lambda e os *handlers* especializados, além da integração com serviços da AWS como Lex, Rekognition e Polly. Também foram implementadas funções auxiliares necessárias ao funcionamento da aplicação e realizada a integração com a API da OpenAI para a geração de narrativas personalizadas. A infraestrutura *serverless* foi configurada nesse mesmo período, consolidando a base tecnológica do projeto. Ao final da *sprint*, uma nova reunião com a professora da instituição parceira possibilitou a apresentação de uma versão parcial do sistema, na qual foram discutidas percepções iniciais e sugeridos ajustes relevantes.

3.1.6 PRÁTICAS DE QUALIDADE IMPLEMENTADAS

Ainda que o tempo disponível fosse restrito, buscou-se garantir a qualidade do código e da aplicação por meio de boas práticas de desenvolvimento. Foram utilizados *linters* como o *Pylint* para Python, além de ferramentas de formatação automática, como o *Prettier*, que asseguraram consistência no estilo e facilitaram a manutenção do código. A documentação de funções e módulos foi obrigatória, e a nomenclatura adotada seguiu convenções estabelecidas para cada linguagem. A estratégia de testes incluiu diferentes dimensões de avaliação: testes de integração foram realizados para validar a comunicação entre os serviços, testes de desempenho avaliaram os tempos de resposta do sistema, e testes de usabilidade, conduzidos a partir do feedback da professora parceira. Alinhado à abordagem participativa do projeto, ela atuou como uma representante experiente do público-alvo, o que permitiu validar a relevância e a adequação das interações propostas, contribuindo diretamente para a qualidade e adequação do sistema. Por fim, práticas de monitoramento e observabilidade foram aplicadas com o AWS CloudWatch, permitindo acompanhar métricas, identificar falhas e aumentar a confiabilidade da aplicação em operação.

3.2 Escopo Funcional do Sistema

Para representar visualmente as funcionalidades centrais do sistema e a interação do usuário com o *chatbot*, foi desenvolvido um diagrama de casos de uso. Apresentado na Figura 5, o diagrama ilustra os principais objetivos que a criança, como usuária, pode alcançar ao utilizar a aplicação, englobando as três funcionalidades principais: expressão de sentimentos e desejos, criação de histórias por meio de imagens ou desenhos e criação de histórias através da seleção de cartões visuais.

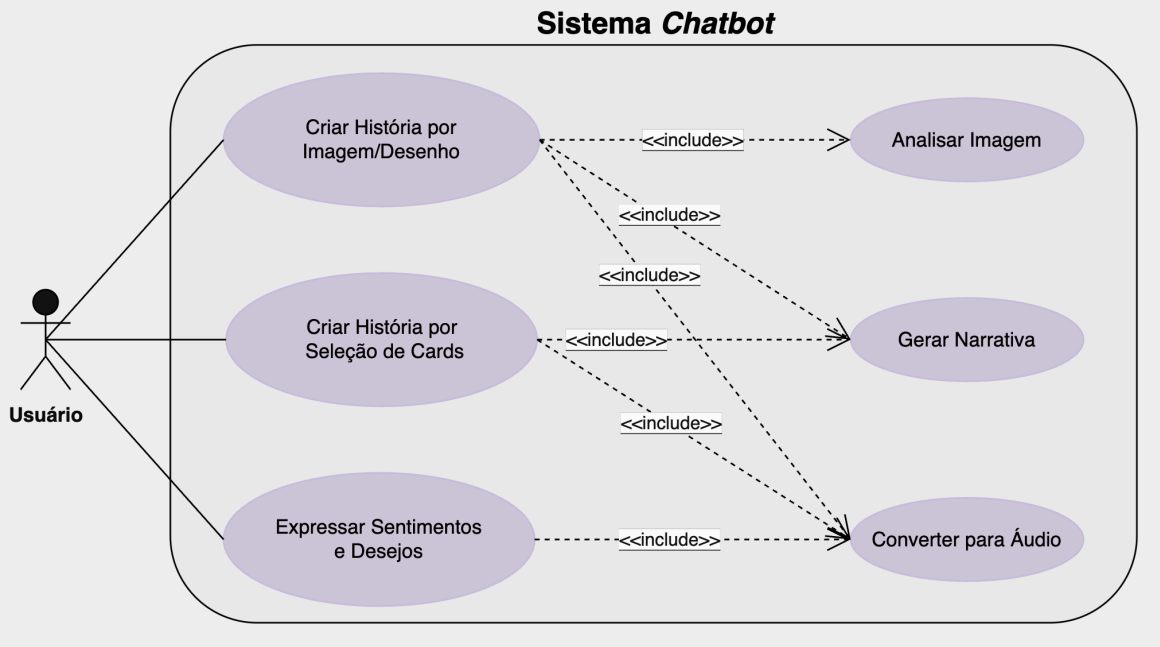

Figura 5 – Diagrama de Casos de Uso do Sistema *Chatbot*. Fonte: Do Autor

3.2.1 DESCRIÇÃO DOS ELEMENTOS DO DIAGRAMA

O diagrama apresenta as interações fundamentais entre o usuário principal e as funcionalidades do sistema *chatbot*, detalhadas a seguir.

3.2.1.1 Usuário

- **Criança**: Representa o usuário principal do sistema, sendo a criança com TEA que interage diretamente com o *chatbot* para realizar as atividades educativas e comunicativas propostas.

3.2.1.2 Casos de Uso Principais

Os casos de uso representam as funcionalidades centrais que o sistema oferece:

- **Criar História por Imagem/Desenho**: Permite que a criança envie uma fotografia ou desenho, que é processado pelo sistema através do Amazon Rekognition para identificação de elementos visuais, seguido pela geração de uma narrativa personalizada utilizando a API da OpenAI.
- **Criar História por Seleção de Cards**: Funcionalidade na qual a criança constrói uma narrativa escolhendo elementos específicos (protagonista, clima, veículo e local) em uma sequência de cartões visuais interativos, resultando na geração de uma história personalizada.

- **Expressar Sentimentos e Desejos**: Permite que a criança comunique emoções, preferências ou necessidades de forma visual e intuitiva, selecionando itens organizados em categorias por meio de cards interativos (emoções, comidas e brincadeiras).

3.2.1.3 Casos de Uso de Apoio

O sistema integra funcionalidades técnicas que suportam os casos principais:

- **Analisar Imagem**: Processamento automático de imagens utilizando Amazon Rekognition para identificação de objetos, pessoas e contexto visual.
- **Gerar Narrativa**: Criação de histórias personalizadas através da API da OpenAI, baseada nos elementos identificados ou selecionados pela criança.
- **Converter para Áudio**: Síntese de voz utilizando Amazon Polly para transformar textos em áudio, garantindo a experiência multimodal do sistema com vozes infantis.

3.2.1.4 Relacionamentos

- **Associação**: As linhas contínuas entre o ator e os casos de uso principais indicam que a criança inicia e participa diretamente dessas interações como usuária primária do sistema.
- **Inclusão (`«include»`)**: Os relacionamentos tracejados demonstram que os casos de uso de apoio são funcionalidades obrigatórias e automáticas, executadas pelo sistema sempre que um caso principal é acionado, garantindo o processamento completo da solicitação e a resposta em formato de áudio.

3.3 Ferramentas e Tecnologias Utilizadas

O desenvolvimento do sistema foi realizado no Visual Studio Code, uma *Integrated Development Environment* (Ambiente de Desenvolvimento Integrado) (IDE) amplamente adotada por oferecer suporte avançado a diversas linguagens e integração com serviços da AWS. Para ampliar a produtividade e garantir padronização no código, foram utilizadas extensões e ferramentas como *AWS Toolkit for Visual Studio Code*, *Python Extension Pack* (com suporte ao *Pylint* para análise estática de código), *GitLens* e *Serverless Framework Snippets*, além do *Prettier* voltado à formatação automática.

A principal linguagem de programação empregada foi o Python, versão 3.12.0, escolhida por sua compatibilidade com os serviços AWS e ampla disponibilidade de bibliotecas para integração com APIs. Python foi utilizado principalmente no desenvolvimento das funções Lambda e na lógica de processamento do sistema, com destaque para as bibliotecas representadas na Figura 6.

bibliotecas principais

```python
import boto3            # SDK AWS para Python
import json             # Manipulação de dados JSON
import requests         # Requisições HTTP para APIs externas
import base64           # Codificação/decodificação de dados
import logging          # Sistema de logs
```

Figura 6 – Principais bibliotecas utilizadas no desenvolvimento em Python. Fonte: Do Autor

Além disso, foi utilizado JavaScript/Node.js, versão 18, aplicado tanto na configuração do *Serverless Framework* quanto na camada de *frontend*. A Figura 7 ilustra as principais dependências e pacotes utilizados, destacando os componentes essenciais para o funcionamento da infraestrutura *serverless*.

dependencias

```
"serverless": "^3.34.0",
"serverless-s3-sync": "^3.1.0",
"serverless-finch": "^4.0.0"
```

Figura 7 – Dependências principais utilizadas no desenvolvimento em Node.js. Fonte: Do Autor

O controle de versão foi realizado com Git, permitindo gerenciamento distribuído do código e organização eficiente das *branches*, seguindo uma estrutura simplificada com *feature*, *develop* e *main*. O repositório foi hospedado no GitHub, que também serviu para armazenamento da documentação do projeto.

A arquitetura do sistema foi construída sobre recursos da AWS, com destaque para Amazon Lex, responsável pelo processamento de linguagem natural; AWS Lambda, para computação *serverless*; Amazon S3, como armazenamento de objetos; Amazon Rekognition e Polly, para análise de imagens e síntese de fala, respectivamente; API Gateway, para gerenciamento de APIs; e CloudWatch, utilizado para monitoramento e geração de logs.

Para complementar as funcionalidades, foram integradas soluções externas, como OpenAI GPT-3.5-turbo, para geração de narrativas personalizadas; Microsoft Translator API, para tradução automática de conteúdos; e a interface web Kommunicate.io, que permite interação conversacional integrada ao *chatbot*.

3.4 Arquitetura de Código e Estrutura do Projeto

O projeto foi estruturado em módulos independentes, estratégia que facilita a manutenção, permite atualizações isoladas e garante maior escalabilidade. Cada módulo possui responsabilidades bem definidas, promovendo um desenvolvimento mais organizado e flexível. A Figura 8 apresenta a organização dos diretórios e arquivos, evidenciando a distribuição do código entre os diferentes módulos.

```
src/
├── bot_function/                    # Módulo principal do chatbot
│   ├── lambda_function.py           # Função de entrada (handler)
│   ├── intent_expressar.py          # Módulo de expressão emocional
│   ├── intent_historia.py           # Módulo de geração de histórias
│   ├── intent_historia_cards.py     # Módulo de histórias com cartas
│   ├── functions.py                 # Funções auxiliares compartilhadas
│   ├── async_message.py             # Processamento assíncrono
│   └── serverless.yml               # Configuração de deployment
├── serverless/                      # Infraestrutura e APIs
│   ├── rekognition.py              # Serviço de análise de imagens
│   ├── polly.py                    # Serviço de síntese de fala
│   ├── create_story.py             # Serviço de criação de histórias
│   └── client/dist/                # Interface web front-end
└── bot/
    └── project_autism-LexJson.zip  # Modelo do Amazon Lex
```

Figura 8 – Estrutura de diretórios do projeto, destacando a distribuição dos módulos e seus arquivos principais. Fonte: Do Autor

A arquitetura distribuída do sistema é ilustrada na Figura 9, evidenciando a interação entre os serviços AWS e os componentes externos. O fluxo de dados inicia-se na interface Kommunicate.io, segue pelo processamento de intenções do Amazon Lex e é distribuído entre serviços especializados, como Rekognition, Polly e OpenAI, por meio de funções Lambda orquestradas pelo API Gateway. Essa representação permite compreender a lógica de integração entre os módulos e a infraestrutura *serverless* adotada, mostrando como o sistema mantém modularidade, escalabilidade e eficiência no processamento das requisições.

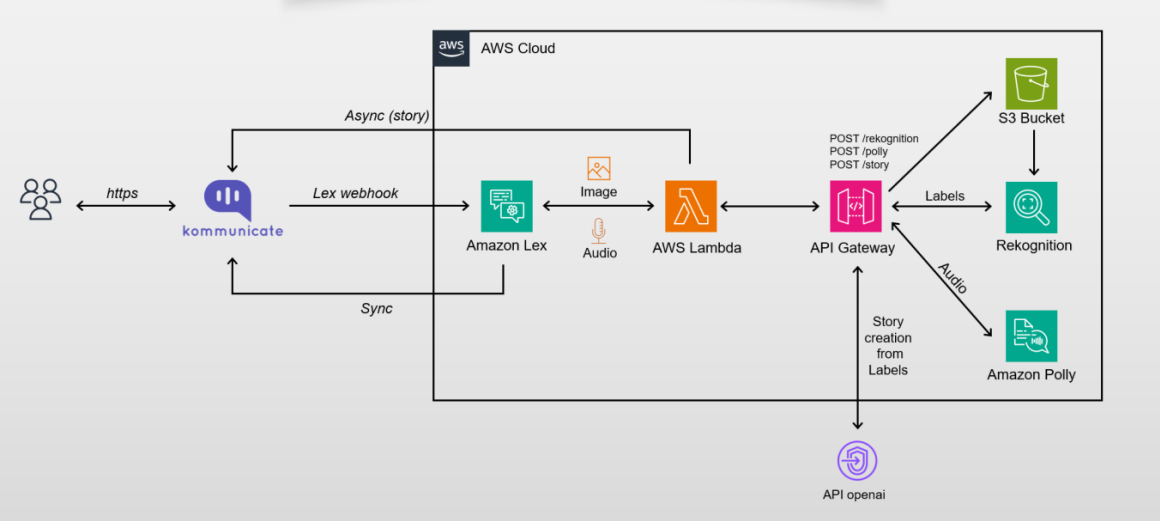

Figura 9 – Arquitetura distribuída do sistema, ilustrando a interação entre serviços AWS e componentes externos. Fonte: Do Autor

O funcionamento do sistema pode ser descrito em cinco etapas principais: a entrada do usuário, capturada pelo Kommunicate.io; o processamento de intenções pelo Amazon Lex; a orquestração de serviços realizada pelas funções Lambda; o processamento especializado por serviços como Rekognition, Polly ou OpenAI; e, finalmente, a resposta consolidada, que é entregue ao usuário em formato multimodal, podendo ser texto, imagem ou áudio.

3.4.1 JUSTIFICATIVA DAS ESCOLHAS TECNOLÓGICAS

As escolhas tecnológicas adotadas neste trabalho foram guiadas por critérios de eficiência operacional, escalabilidade e viabilidade econômica, sempre alinhados aos objetivos da monografia. A seguir, são apresentadas as justificativas para os principais componentes utilizados:

- **Nuvem AWS:** A AWS foi selecionada em razão de sua maturidade no mercado de serviços gerenciados e pela oferta de um modelo gratuito robusto, o que possibilitou a realização do projeto dentro das restrições orçamentárias próprias de um contexto acadêmico.
- **Arquitetura *Serverless* (AWS Lambda):** Optou-se pela computação *serverless* a fim de abstrair a administração de infraestrutura e concentrar esforços na lógica de negócio. O modelo de cobrança do AWS Lambda, baseado no tempo de execução, mostrou-se adequado para aplicações com uso intermitente, contribuindo para a redução dos custos operacionais.

- **Amazon API Gateway:** Esse serviço foi empregado como ponto central de acesso aos serviços de *backend*. Sua função foi essencial para o gerenciamento, a segurança e o monitoramento das APIs responsáveis por acionar as funções Lambda, garantindo maior controle sobre o tráfego e a autorização das requisições.
- **Amazon S3:** O Amazon Simple Storage Service (S3) foi utilizado como repositório de objetos pela combinação de alta durabilidade, escalabilidade e baixo custo. No projeto, teve dupla função: hospedar os arquivos estáticos do *frontend* e armazenar as mídias geradas pela aplicação, como imagens enviadas pelos usuários e áudios produzidos pelo Amazon Polly.
- **Amazon Lex:** Para o núcleo conversacional, o Amazon Lex foi escolhido devido à sua capacidade avançada de PLN e à integração nativa com o AWS Lambda, que facilitou a orquestração dos fluxos de diálogo.
- **Amazon Rekognition e Polly:** As funcionalidades multimodais foram viabilizadas por serviços gerenciados de IA. O Rekognition foi empregado na análise de imagens, eliminando a necessidade de desenvolver modelos próprios de visão computacional. Já o Amazon Polly foi utilizado para a síntese de fala, destacando-se pela qualidade de suas vozes neurais, aspecto relevante para a aceitação pelo público infantil.
- **API da OpenAI (GPT):** A API da OpenAI foi incorporada por representar o estado da arte em geração de linguagem natural. Sua habilidade de produzir narrativas consistentes a partir de instruções mínimas foi fundamental para a implementação da funcionalidade de criação de histórias.
- **Kommunicate.io:** Por fim, a plataforma Kommunicate.io foi adotada como interface de interação com o usuário, acelerando o desenvolvimento do *frontend* por meio de um *widget* de fácil integração e suporte nativo ao Amazon Lex.

A integração dessas tecnologias resultou em um sistema resiliente e funcional, entregue dentro do cronograma proposto. Dessa forma, a arquitetura desenvolvida se confirma como um modelo eficaz para a construção de soluções em tecnologia assistiva.

4 DESENVOLVIMENTO

Este capítulo descreve de forma detalhada o desenvolvimento dos componentes fundamentais do sistema, explicitando responsabilidades, definições claras de comunicação entre módulos e principais algoritmos empregados. Foi adotada uma abordagem modular, com foco em coesão interna e baixo acoplamento entre componentes, favorecendo extensibilidade, testabilidade e manutenção evolutiva. Os módulos foram concebidos com descrição clara do que cada um recebe e retorna, regras de funcionamento e tratamento sistemático de exceções, observando requisitos não funcionais como latência, resiliência e custo operacional. Essa organização permite evoluções incrementais e auditoria técnica do comportamento em produção.

4.1 Módulo de Roteamento (lambda_function.py)

O módulo principal, `lambda_function.py`, é responsável por rotear as solicitações do Amazon Lex para as funções tratadoras de cada intenção. Ele interpreta o nome da intent, aciona a função correspondente e centraliza a orquestração do diálogo. O mapeamento é feito de forma direta, por meio de condicionais, garantindo despacho simples e previsível. Quando a intent não existe, o evento está incompleto ou há inconsistências de tipo, o sistema retorna respostas padronizadas e registra os eventos, facilitando o diagnóstico. A implementação deste módulo é apresentada na Figura 10.

lambda_function.py

```python
def lambda_handler(event, context):

    intent = event['sessionState']['intent']['name']

    response = {}

    if intent == 'ObterHistoria':
        response = ObterHistoria_handler(event)

    if intent == 'ObterHistoriaCards':
        response = ObterHistoriaCards_handler(event)

    if intent == 'ObterExpressar':
        response = ObterExpressar_handler(event)

    return response
```

Figura 10 – Implementação do módulo de roteamento (`lambda_function.py`). Fonte: Do Autor

4.2 Módulo de Expressão Emocional (intent_expressar.py)

O módulo `intent_expressar.py` permite que a criança se comunique por meio de cards que representam diferentes emoções, desejos ou situações, oferecendo uma interação visual e intuitiva. O sistema carrega previamente os dados das opções disponíveis em um arquivo JSON, contendo textos e imagens para cada categoria. Durante a interação, a função *handler* verifica se todos os slots necessários, como a escolha da voz, do desejo e da opção específica, foram preenchidos. Caso algum slot esteja incompleto, o módulo apresenta cards visuais com as opções correspondentes, guiando a criança na seleção. Quando todas as informações estão completas, o módulo utiliza os valores escolhidos para gerar uma narrativa associada à seleção da criança e cria um arquivo de áudio personalizado com a voz selecionada, retornando no formato adequado para o *chatbot*. Ao final, a sessão é

atualizada e um card final é exibido para permitir a continuidade da interação. Dessa forma, o módulo combina validação de entradas, orientação por imagens, geração de áudio através do Amazon Polly e feedback contextualizado, promovendo uma experiência intuitiva.

O fluxo do módulo pode ser dividido em quatro etapas principais: leitura do contexto e validação dos dados de entrada, fase de diálogo para coleta de slots obrigatórios, execução do atendimento com geração de áudio e, por fim, finalização da sessão com adição de card de continuidade. As Figuras 11 a 14 apresentam essas etapas detalhadamente.

intent_expressar.py

```python
def ObterExpressar_handler(event):

    intent = event['sessionState']['intent']['name']
    slots = event['sessionState']['intent']['slots']

    response = {}

    validation = validate_type_card(event, slots)
```

Figura 11 – Início do *handler*: leitura do contexto e validação dos dados de entrada. Nesta etapa, o módulo interpreta o evento do Amazon Lex e verifica se os slots obrigatórios foram preenchidos corretamente. Fonte: Do Autor

Na fase de diálogo, caso haja informações faltantes ou inválidas, a função monta a resposta adequada para o Amazon Lex solicitar o próximo slot. O sistema utiliza cards visuais para guiar a criança na seleção da categoria, da imagem e da opção desejada, garantindo uma interação intuitiva e pedagógica.

intent_expressar.py

```
#checks validation and invalid slots
if event['invocationSource'] == 'DialogCodeHook':

    response = check_validation(validation, intent, slots)
    return response
```

Figura 12 – Fase de diálogo: verificação de slots obrigatórios e construção de respostas adaptativas com cards visuais. Fonte: Do Autor

Quando todos os slots estão preenchidos, o módulo executa o atendimento. Ele coleta a voz e as escolhas do usuário, compõe o texto de saída e seleciona a voz adequada para a síntese de áudio. Em seguida, o áudio é gerado pelo Amazon Polly e empacotado em um payload compatível com o *chatbot*, garantindo a reprodução interativa dentro da interface.

intent_expressar.py

```
#all slots are valid
if event['invocationSource'] == 'FulfillmentCodeHook':

    voz = slots['voz']['value']['originalValue']
    desejo = slots['desejo']['value']['originalValue']
    escolha = slots['escolha']['value']['originalValue']

    story = valores[desejo][escolha]['valor']

    if voz == 'menino':
        voice = 'Thiago'
    else:
        voice = 'Camila'
        story  = story+"!"
```

Figura 13 – Execução do atendimento: coleta das escolhas do usuário, geração do áudio e empacotamento em *payload* para o *chatbot*. Fonte: Do Autor

Por fim, a sessão é finalizada de forma padronizada e um card de continuidade é adicionado à resposta, permitindo que a criança escolha a próxima ação. Esta etapa garante previsibilidade no encerramento das intents, facilita a manutenção e mantém a consistência das interações.

intent_expressar.py

```
        sessionState = close_session(intent, slots)
        response.update(sessionState)

        final_message = end_card(" ")
        final_message["imageResponseCard"]['title'] = "O que você quer fazer?"
        response['messages'].append(final_message)

    return response
```

Figura 14 – Finalização da sessão e inclusão do card de continuidade, permitindo a interação contínua do usuário com o *chatbot*. Fonte: Do Autor

4.3 Módulo de Geração de Histórias

4.3.1 GERAÇÃO DE HISTÓRIAS A PARTIR DE IMAGENS OU DESENHOS

O módulo `intent_historia.py` é responsável por receber a imagem ou desenho feito pela criança através do *chatbot* e processá-la para criar narrativas personalizadas. Inicialmente, a entrada selecionada é analisada pelo Amazon Rekognition, que identifica elementos visuais e retorna labels e informações relevantes sobre o conteúdo. Esses dados são enviados à API da OpenAI, que utiliza as características extraídas para gerar uma história coerente e contextualizada, refletindo os elementos presentes na imagem ou no desenho. O texto gerado é convertido automaticamente em áudio pelo Amazon Polly, permitindo que a criança ouça a narrativa de forma interativa. Todo o fluxo combina validação de entradas, análise visual, geração de conteúdo textual e produção de áudio, proporcionando uma experiência pedagógica, multimídia e intuitiva.

A função `ObterHistoria_handler` organiza o fluxo em etapas que incluem validação dos slots, preparação dos dados de entrada e orquestração do processamento assíncrono. A Figura 15 ilustra o início desse processo, destacando a captura do contexto, a inicialização das variáveis e a validação dos slots recebidos.

intent_historia.py

```
def ObterHistoria_handler(event):

    intent = event['sessionState']['intent']['name']
    slots = event['sessionState']['intent']['slots']

    print(json.dumps(event))

    response = {}

    validation = validate_type_card(event, slots)
```

Figura 15 – Início do *handler*: captura do contexto, inicialização de variáveis e validação dos slots. Fonte: Do Autor

Na fase de diálogo, quando há dados faltantes ou inválidos, a função constrói a resposta para o Lex solicitar o próximo slot e encerrar o ciclo atual. Esse comportamento está representado na Figura 16, que mostra a verificação de informações e a geração de respostas adaptativas.

intent_historia.py

```
    #checks validation and invalid slots
    if event['invocationSource'] == 'DialogCodeHook':

        response = check_validation(validation, intent, slots)
        return response
```

Figura 16 – Fase de diálogo: verificação de slots e geração de respostas adaptativas para coleta de dados do usuário. Fonte: Do Autor

No pré-processamento, a aplicação identifica a origem da imagem, prepara a chamada ao serviço de análise (Rekognition) e recebe o retorno inicial com os rótulos

detectados. Esse fluxo é apresentado na Figura 17.

intent_historia.py

```
#all slots are valid
    if event['invocationSource'] == 'FulfillmentCodeHook':

        slot_historia = slots['historia']['value']['originalValue']

        try:
            imageURL = str(json.loads(event['requestAttributes']['attachments'])[0]['payload']
['url'])

        except:
            imageURL = slot_historia

        http = urllib3.PoolManager()
        url = os.environ['INVOKE_URL']+"/rekognition/v1"


        imageName = str(json.loads(event['requestAttributes']['attachments'])[0]['payload']
['name'])

        Headers = {'Content-Type': 'application/json'}
        payload = {
            "bucket": os.environ['BUCKET_IMAGES'],
            "imageURL": imageURL,
            "imageName": imageName
        }

        encoded_body = json.dumps(payload)
        res = http.request('POST', url, body=encoded_body, headers=Headers)
        respo = ''

        if res.status == 200:
            respo = json.loads(res.data)
```

Figura 17 – Pré-processamento e chamada ao Amazon Rekognition para análise da imagem selecionada. Fonte: Do Autor

Em seguida, ocorre o pós-processamento, no qual os rótulos retornados são traduzidos, normalizados e organizados em frases que servirão como base para a narrativa. A Figura 18 mostra esse processo de transformação dos dados brutos em insumos para a geração da história.

intent_historia.py

```
        labels = []
        for label in respo['labels']:
            labels.append(label['Name'])

        if len(labels) == 0:
            labels = ['Hero']

        translated_labels = ",".join(labels)
        translated_labels = translate(translated_labels)
        translated = translated_labels.split(',')

        print(translated)

        #random index----------------------------------------
        rand_number = lambda: random.randint(0, len(labels)-1)
        #----------------------------------------------------

        label1 = translated[rand_number()]
        label2 = translated[rand_number()]
        label3 = translated[rand_number()]
        label4 = translated[rand_number()]

        id_model = 'gpt-3.5-turbo'
        phrase = f'crie uma história infantil bem curta, máximo de 500 caracteres e não
ultrapasse 80 palavras, que contenha os seguintes elementos: {label1},{label2},{label3},
{label4}'
```

Figura 18 – Pós-processamento dos rótulos: tradução, normalização e montagem da frase para geração da história. Fonte: Do Autor

A orquestração assíncrona, representada na Figura 19, aciona a função Lambda encarregada de gerar a história e, ao mesmo tempo, mantém a interface responsiva, enviando ao usuário uma mensagem de carregamento.

intent_historia.py

```
        #async function ------------------------------------------------------------------------
        client = boto3.client('lambda')
        client.invoke(
            FunctionName=ASYNC_FUNCTION_NAME,
            InvocationType='Event',
            Payload = bytes(json.dumps({'body': event, 'phrase' : phrase, 'id_model' :
id_model}), encoding='utf-8')
        )
        #---------------------------------------------------------------------------------------
```

Figura 19 – Orquestração assíncrona: acionamento da função Lambda para geração da história e retorno de resposta imediata ao usuário. Fonte: Do Autor

Por fim, o sistema envia uma resposta imediata ao usuário, encerrando a intent e exibindo um card de continuidade. Esse comportamento é mostrado na Figura 20, que evidencia a preocupação em manter a interação fluida e engajante.

intent_historia.py

```
        image_kommunicate = {
            "message": "<img src='https://"
            + os.environ["BUCKET_IMAGES"]
            + ".s3.us-east-1.amazonaws.com/loading.gif'><br>Estou pensando em uma história,
talvez demore alguns segundos... vamos fazer outra coisa enquanto isso?",
            "platform": "kommunicate",
            "messageType": "html",
        }

        response = {
            "messages": [
                {
                    "contentType": "CustomPayload",
                    "content": json.dumps(image_kommunicate),
                }
            ]
        }
        sessionState = close_session(intent, slots)
        response.update(sessionState)

        final_message = end_card(" ")
        final_message["imageResponseCard"]["title"] = "O que você quer fazer?"
        response["messages"].append(final_message)

    return response
```

Figura 20 – Resposta imediata ao usuário: mensagem de carregamento e card de continuidade da interação. Fonte: Do Autor

4.3.2 GERAÇÃO DE HISTÓRIAS A PARTIR DE CARDS

O módulo `intent_historia_cards.py` permite que a criança componha uma história selecionando cartões visuais que representam protagonistas, clima, veículo e local. O fluxo valida cada escolha, apresenta cards quando algum slot está ausente e, ao final, aciona a geração assíncrona da narrativa com base nas opções selecionadas. A Figura 21 apresenta a lógica de validação inicial e a construção dos cards de imagem a partir de dados armazenados em JSON.

intent_historia_cards.py

```
def validate_type_card(event, slots):

    valores =
json_bucket_images(json.load(open('json_files/intent_historia_cards_info.json')))

    #card1 "protagonists" check and validation
    isMissing = check_missing_slot(slots, "card1", "PlainText", "Escolha o protagonista da
história, ou escreva um nome!")

    if isMissing:
        for key in valores['protagonists']:
            rescard_title = ' '
            rescard_buttons = [
                {
                    "text": valores['protagonists'][key]['texto'],
                    "value": key
                }
            ]
            imageUrl = valores['protagonists'][key]['link']
            isMissing['messages'].append(build_message("ImageResponseCard", rescard_title,
rescard_buttons, imageUrl))
        return isMissing
```

Figura 21 – Validação e construção de cards: leitura do JSON, verificação de slots e preparação de cards de imagem para o usuário. Fonte: Do Autor

O início do *handler* é mostrado na Figura 22, que ilustra a captura do contexto da intent, a inicialização de variáveis e a verificação dos slots, garantindo que os dados mínimos necessários estejam presentes antes do avanço do fluxo.

intent_historia_cards.py

```
def ObterHistoriaCards_handler(event):

    intent = event['sessionState']['intent']['name']
    slots = event['sessionState']['intent']['slots']

    response = {}

    validation = validate_type_card(event, slots)
```

Figura 22 – Início do *handler*: captura de intent e slots, inicialização da resposta e validação. Fonte: Do Autor

Na fase de diálogo, se algum slot ainda estiver faltando, o sistema gera uma resposta orientando o Amazon Lex a elicitar a próxima informação. Esse comportamento

está representado na Figura 23, que mostra a construção de respostas adaptativas para a coleta dos dados.

intent_historia_cards.py

```
    #checks validation and invalid slots
    if event['invocationSource'] == 'DialogCodeHook':

        response = check_validation(validation, intent, slots)
        return response
```

Figura 23 – Fase de diálogo: verificação de slots e respostas adaptativas para coleta de dados do usuário. Fonte: Do Autor

No cumprimento da intent, todas as escolhas realizadas pela criança são reunidas para formar a base da narrativa. A Figura 24 ilustra essa etapa, onde as informações de cada card são organizadas e transformadas em uma frase inicial que dará origem à história.

intent_historia_cards.py

```
    #all slots are valid
    if event['invocationSource'] == 'FulfillmentCodeHook':

        card1 = slots['card1']['value']['originalValue']
        card2 = slots['card2']['value']['originalValue']
        card3 = slots['card3']['value']['originalValue']
        card4 = slots['card4']['value']['originalValue']

        id_model = 'gpt-3.5-turbo'
        phrase = f'crie uma história infantil bem curta, máximo de 500 caracteres e não
ultrapasse 80 palavras, que contenha os seguintes elementos: {card1},{card2},{card3},{card4}'
```

Figura 24 – Cumprimento da intent: coleta das escolhas dos cards e composição da frase para geração da história. Fonte: Do Autor

A orquestração assíncrona, apresentada na Figura 25, aciona a função Lambda responsável pela geração da história, mantendo a interface responsiva e proporcionando ao usuário um retorno imediato de que o processamento está em andamento.

intent_historia_cards.py

```
        #async function ------------------------------------
        client = boto3.client('lambda')
        client.invoke(
            FunctionName=os.environ['ASYNC_FUNCTION_NAME'],
            InvocationType='Event',
            Payload = bytes(json.dumps({'body': event, 'phrase' : phrase, 'id_model' :
id_model}), encoding='utf-8')
        )
        #-----------------------------------------------------
```

Figura 25 – Orquestração assíncrona: acionamento da função Lambda e manutenção da responsividade da conversa. Fonte: Do Autor

Finalmente, o módulo retorna uma mensagem de carregamento e encerra a intent atual, exibindo um card de continuidade para o usuário. A Figura 26 mostra esse momento, reforçando o aspecto pedagógico e interativo da aplicação.

intent_historia_cards.py

```
        image_kommunicate = {
            "message": "<img src='https://"+os.environ['BUCKET_IMAGES']+".s3.us-east-
1.amazonaws.com/loading.gif'><br>Adorei a sua escolha, vamos fazer outra coisa enquanto eu
penso em uma história?",
            "platform": "kommunicate",
            "messageType": "html"
        }

        response = {
            "messages": [
                {
                    "contentType":"CustomPayload",
                    "content": json.dumps(image_kommunicate)
                }
            ]
        }
        sessionState = close_session(intent, slots)
        response.update(sessionState)

        final_message = end_card(" ")
        final_message["imageResponseCard"]['title'] = "O que você quer fazer?"
        response['messages'].append(final_message)

    return response
```

Figura 26 – Resposta imediata ao usuário: mensagem de carregamento e card para continuidade da interação. Fonte: Do Autor

4.4 Módulo de Funções Auxiliares (functions.py)

O módulo `functions.py` concentra utilitários reutilizáveis que implementam padrões de resposta do Amazon Lex e suportam o gerenciamento do estado conversacional. Ele funciona como uma camada de infraestrutura para os demais módulos, garantindo consistência na comunicação, padronização das respostas e tratamento centralizado de exceções. A utilização deste módulo reduz a duplicação de código entre intents, facilita a manutenção e evoluções futuras, e permite a inclusão de metadados relacionados à acessibilidade e instrumentação.

A função `close_session` é responsável por finalizar uma sessão de diálogo de forma padronizada, sinalizando ao Amazon Lex que a intenção foi atendida. Ela recebe o nome da intent e os slots preenchidos, retornando a estrutura `sessionState` com `dialogAction` do tipo `Close` e estado `Fulfilled`. Essa padronização garante previsibilidade na finalização de intents e simplifica a implementação de *handlers* adicionais. A Figura 27 apresenta a implementação da função.

functions.py

```
def close_session(intent, slots):

    sessionState = {
        "sessionState": {
            "dialogAction": {
                "type": "Close"
            },
            "intent": {
                "name": intent,
                "slots": slots,
                "state": "Fulfilled"
            }
        }
    }
    return sessionState
```

Figura 27 – Exemplo da função `close_session`, mostrando a estrutura de retorno padronizada para finalização de intents no Amazon Lex. Fonte: Do Autor

A função `check_validation` constrói a resposta do Lex com base no resultado da validação. Quando há dados faltantes, a função orienta qual slot deve ser solicitado por meio de `ElicitSlot`; quando a validação é bem-sucedida, delega a continuidade do

diálogo via `Delegate`. Essa função garante que a lógica de coleta de dados seja consistente entre todas as intents do sistema. A Figura 28 mostra a implementação ilustrativa de `check_validation`.

functions.py

```
def check_validation (validation, intent, slots):
    """
    build a response based on validation value, valid or invalid

    Attributes
    ----------
    validation: bool
        valid is True or False
    intent: str
        intent name
    slots: dict
        intent slots of incoming event
    """

    if validation['valid'] == False:
        response = {
            "sessionState": {
                "dialogAction": {
                    "slotToElicit": validation['invalidSlot'],
                    "type": "ElicitSlot"
                },
                "intent": {
                    "name": intent,
                    "slots": slots
                }
            },
            "messages": validation['messages']
        }
    else:
        response = {
            "sessionState": {
                "dialogAction": {
                    "type": "Delegate"
                },
                "intent": {
                    "name": intent,
                    "slots": slots
                }
            }
        }
    return response
```

Figura 28 – Exemplo da função `check_validation`, mostrando como a resposta do Amazon Lex é construída com base na validação dos slots. Fonte: Do Autor

Em síntese, o módulo `functions.py` atua como uma camada de apoio estruturada, fornecendo mecanismos padronizados para finalização de intents e coleta de informações

via slots. Isso garante consistência, reutilização de código e facilidade de manutenção, permitindo que todos os módulos do sistema operem de forma uniforme e assegurem respostas confiáveis no ambiente do Amazon Lex.

4.5 Serviços Auxiliares *Serverless*

O sistema foi desenvolvido com uma arquitetura de microserviços, baseada em funções Lambda especializadas, em que cada serviço possui responsabilidades bem definidas e atua de forma independente. Isso facilita tanto a manutenção quanto a escalabilidade da aplicação. A comunicação entre os serviços ocorre por meio do API Gateway, e sempre que possível, os serviços são implementam para produzir resultados consistentes em execuções repetidas, evitando efeitos colaterais. Além disso, cada serviço conta com tratamento explícito para erros recuperáveis e irreversíveis. Essa abordagem reduz o acoplamento entre os componentes, permitindo que o sistema evolua sem comprometer seu funcionamento.

4.5.1 ANÁLISE DE IMAGENS (REKOGNITION.PY)

O módulo `rekignition.py` é utilizado para realizar a análise de imagens enviadas pelos usuários. O conteúdo visual é processado e submetido à detecção de objetos, cenas e conceitos, retornando rótulos acompanhados de seus respectivos níveis de confiança. O serviço suporta o envio de imagens codificadas em base64, aplica limiares de confiança para filtrar os resultados e organiza a saída em um formato estruturado, adequado ao consumo pela aplicação. Além disso, leva em consideração limitações como o tamanho do payload e o tempo de execução da função, bem como a variação na qualidade das imagens recebidas, que pode influenciar a precisão dos rótulos gerados.A implementação do serviço de análise de imagens pode ser observada na Figura 29.

rekognition.py

```
def event_request(event: dict, detect_type: str, label_field: str) -> dict:

    try:
        ebody = event.get('body')
        if not ebody:
            raise ValueError('Empty body received!! Please try again.')

        body_data = json.loads(ebody)
        bucket = body_data['bucket']
        imageURL = body_data['imageURL']

        imageName = body_data['imageName']

        url = imageURL

        http = urllib3.PoolManager()
        r = http.request('GET', url)

        s3 = boto3.client('s3')

        s3.upload_fileobj(BytesIO(r.data), bucket, imageName)

        s3file = s3.get_object(Bucket=str(bucket), Key=str(imageName))

        objects = rekognition_type(detect_type, bucket, imageName, {'MaxLabels': 10,
'MinConfidence': 60})

    except Exception as e:
        return error_message(str(e))

    body = construct_body(s3file, bucket, imageName)
    body.update({label_field: objects})

    response = {"statusCode": 200, "headers": {"Content-Type": "application/json"}, "body":
json.dumps(body)}

    return response
```

Figura 29 – Implementação do serviço de análise de imagens (`rekognition.py`). Fonte: Do Autor

4.5.2 SÍNTESE DE VOZ (POLLY.PY)

O módulo `polly.py` converte texto em áudio por meio do Amazon Polly, permitindo configuração de voz, velocidade e tom, e disponibiliza o arquivo MP3 no S3. O fluxo principal da função `polly_v1` inclui validação dos parâmetros, seleção da voz, conversão em áudio e retorno da URL resultante, como demonstrado na Figura 30.

polly.py

```
def polly_v1 (event, context):

    # check if phrase keys are present
    try:
        ebody = event['body']
        received_phrase = json.loads(ebody)['phrase']
    except:
        message = {"message":"Wrong parameters"}
        response = {"statusCode": 500, "headers": {"Content-Type": "application/json"},
"body": json.dumps(message)}
        return response

    type_voice = 'Camila'
    try:
        type_voice = json.loads(ebody)['voice']
    except:
        pass

    s3 = boto3.client('s3')
    postId = convert_save_audio(received_phrase, type_voice)

    location = s3.get_bucket_location(Bucket=os.environ['BUCKET_POLLY'])
    region = location['LocationConstraint']
    if region is None:
        url_begining = "https://"+str(os.environ['BUCKET_POLLY'])+".s3.amazonaws.com"
    else:
        url_begining = "https://"+str(os.environ['BUCKET_POLLY'])+".s3-" + str(region) +
".amazonaws.com"

    url = url_begining + "/" + str(postId) + ".mp3"
```

Figura 30 – Fluxo principal da função `polly_v1` para síntese de áudio. Fonte: Do Autor

A função garante tratamento de erros e retorno padronizado, assegurando robustez na comunicação com o *chatbot*, como mostrado na Figura 31.

polly.py

```
    try:
        response = s3.get_object(Bucket = os.environ['BUCKET_POLLY'], Key=postId + ".mp3")

        created_audio = response["LastModified"].strftime("%d-%m-%Y %H:%M:%S")

        body = {"received_phrase": received_phrase,
                "url_to_audio": url,
                "created_audio": created_audio}

        response = {"statusCode": 200,
                    "headers": {"Content-Type": "application/json"},
                    "body": json.dumps(body)}
    except:
        message = {"message":"Internal Error"}
        response = {"statusCode": 500, "headers": {"Content-Type": "application/json"},
"body": json.dumps(message)}

    return response
```

Figura 31 – Tratamento de erros e retorno padronizado da função `polly_v1`. Fonte: Do Autor

Quando a operação é bem-sucedida, os metadados do objeto gerado são recuperados, o corpo de retorno é composto com a frase recebida, a URL do áudio e a data de criação, e a resposta é enviada com código 200. Em caso de falha, um retorno padronizado com código 500 é fornecido, garantindo previsibilidade no comportamento do serviço.

4.5.3 GERAÇÃO DE HISTÓRIAS (CREATE_STORY.PY)

O módulo `create_story.py` concentra a lógica de criação narrativa mediante integração com a API da OpenAI. Ele processa elementos estruturados fornecidos pelo usuário, compõe prompts contextualizados e integra-se a modelos de linguagem GPT para gerar histórias personalizadas para o público infantil. O conteúdo produzido é validado e formatado antes da entrega, atendendo requisitos de clareza, coerência textual e adequação etária. Além disso, mecanismos de validação e mensagens de fallback garantem continuidade da experiência em caso de falhas na geração do texto.

A função principal do módulo, `create_story_v1`, inicia verificando se os parâmetros obrigatórios foram recebidos, construindo a requisição para a API da OpenAI e tratando possíveis falhas de comunicação. O fluxo da função é apresentado na Figura 32.

create_story.py

```python
# check if phrase key is present
try:
    ebody = event['body']
    received_phrase = json.loads(ebody)['phrase']
    KEY_API = json.loads(ebody)['key']
    id_model = json.loads(ebody)['id_model']

    http = urllib3.PoolManager()
    url = 'https://api.openai.com/v1/chat/completions'

    Headers = {'Authorization' : f'Bearer {KEY_API}', 'Content-Type': 'application/json'}
    payload = {'model': id_model, 'messages': [{'role': 'user', 'content': received_phrase}]}

    encoded_body = json.dumps(payload)
    res = http.request('POST', url, body=encoded_body, headers=Headers)

    respo = ''

    if res.status == 200:
        respo = json.loads(res.data)
    else:
        raise Exception ('Was not able to get a proper message')
```

Figura 32 – Trecho inicial da função `create_story_v1`, mostrando a validação de parâmetros e preparação da requisição à API da OpenAI. Fonte: Do Autor

Quando a chamada à API é bem-sucedida, o texto da história é extraído da resposta, encapsulado em um objeto JSON e retornado com código 200. Caso ocorra qualquer falha durante o processo, é enviado um retorno padronizado com código 500 e mensagem de erro, garantindo previsibilidade no comportamento do serviço, como mostrado na Figura 33.

create_story.py

```
story = respo['choices'][0]['message']['content']

story = {"story":story}
response = {"statusCode": 200, "headers": {"Content-Type":
"application/json"}, "body": json.dumps(story)}

except:

    message = {"message":"Internal Error"}
    response = {"statusCode": 500, "headers": {"Content-Type":
"application/json"}, "body": json.dumps(message)}

return response
```

Figura 33 – Trecho final da função `create_story_v1`, demonstrando a extração do texto gerado, construção da resposta e tratamento de erros. Fonte: Do Autor

5 CONFIGURAÇÃO SISTÊMICA

A fase de configuração sistêmica é um pré-requisito essencial para o processo de implantação, garantindo que o ambiente de desenvolvimento esteja devidamente preparado e que todas as dependências, credenciais e ferramentas tecnológicas necessárias estejam corretamente instaladas e validadas. Este capítulo detalha os pré-requisitos de software, as contas de serviço indispensáveis e os procedimentos para preparar o ambiente local, estabelecendo os alicerces operacionais para a implantação da infraestrutura em nuvem.

5.1 Especificações de Pré-requisitos

5.1.1 REQUISITOS DE SOFTWARE E DEPENDÊNCIAS TECNOLÓGICAS

Para configurar o ambiente de desenvolvimento, é necessário instalar e parametrizar um conjunto de softwares fundamentais, capazes de assegurar a interoperabilidade e o funcionamento adequado do sistema. Entre os componentes essenciais, destacam-se:

- **Node.js (versão 18.x ou superior)**, utilizado no gerenciamento de pacotes e como suporte ao *Serverless Framework*, viabilizando a execução de rotinas de implantação e compilação (*build*).
- ***Command Line Interface* (Interface de Linha de Comando) (CLI)**, que permite a comunicação direta com os serviços da AWS, incluindo a criação de recursos e a execução de comandos de *deploy*.
- ***Serverless Framework* (versão 3.x)**, responsável por orquestrar a infraestrutura *serverless*, integrando funções Lambda, *buckets* S3 e *endpoints* do API Gateway.
- **Python (versão 3.12 ou superior)**, utilizado no desenvolvimento das funções Lambda e no processamento interno de dados, garantindo compatibilidade com bibliotecas específicas e com o *runtime* da AWS.
- **Git:** Sistema de controle de versão distribuído, essencial para clonar o repositório do projeto e gerenciar o código-fonte.

A verificação das versões instaladas é uma etapa importante, pois assegura que todos os componentes atendam aos requisitos definidos.

5.1.2 CREDENCIAIS E CONTAS DE SERVIÇO NECESSÁRIAS

Além dos softwares locais, o funcionamento do sistema depende de credenciais válidas associadas a serviços de computação em nuvem e plataformas de terceiros. Entre os principais serviços utilizados, destacam-se:

- **AWS:** exige uma conta com privilégios administrativos para criação e gerenciamento de recursos *serverless*, como funções Lambda, *buckets* S3 e *endpoints* do API Gateway. A configuração correta das credenciais é essencial para evitar falhas de autenticação.
- **Kommunicate.io:** fornece a interface conversacional web, permitindo a interação direta do usuário com o *chatbot* e simplificando a gestão do diálogo.
- **Microsoft Translator API:** acessível via RapidAPI, oferece suporte multilíngue por meio de tradução automática, proporcionando uma experiência globalizada ao usuário.
- **OpenAI API:** utilizada para a geração de narrativas e respostas contextuais, incorporando modelos avançados de linguagem natural ao fluxo de interação.

Para cada serviço externo, recomenda-se validar previamente a criação das contas, obter as chaves de API correspondentes e configurar corretamente as variáveis de ambiente. Essas variáveis, incluindo chaves de acesso e URLs de *endpoints*, devem ser declaradas no arquivo `serverless.yml` do projeto, assegurando acesso seguro e consistente a todos os módulos do sistema, como ilustrado na Figura 34. Um guia detalhado para a obtenção das chaves de API está disponibilizado no último capítulo (Apêndice).

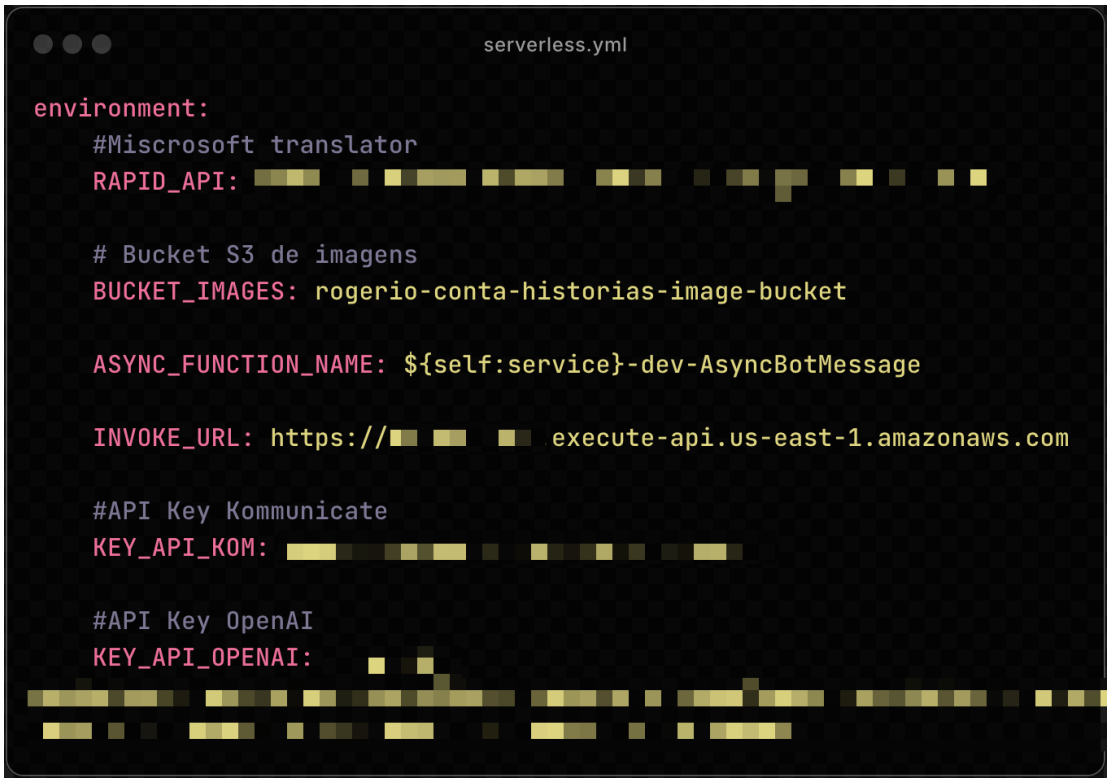

Figura 34 – Exemplo de configuração de variáveis de ambiente no arquivo `serverless.yml`, incluindo chaves de API e URLs de serviços externos. Fonte: Do Autor

5.2 Preparação do Ambiente de Desenvolvimento

A primeira etapa na preparação do ambiente consiste em obter o código-fonte do projeto. O repositório está hospedado no GitHub e pode ser clonado para o ambiente de desenvolvimento local utilizando o seguinte comando no terminal:

```
git clone https://github.com/guilhermekameoka/chatbot.git
```

A execução deste comando criará um diretório chamado `chatbot-main`, contendo todos os arquivos necessários para a configuração e implantação do sistema.

Com o código-fonte disponível localmente, inicia-se a preparação do ambiente de desenvolvimento, cujo objetivo é assegurar a instalação correta e a parametrização das ferramentas essenciais, em versões compatíveis com os requisitos do projeto. Esta fase é fundamental para garantir o funcionamento coeso e consistente dos módulos subsequentes.

O primeiro procedimento nesta etapa envolve a verificação da configuração do CLI. A inspeção das credenciais pode ser realizada por meio do comando `aws configure list`, cuja saída é exemplificada na Figura 35.

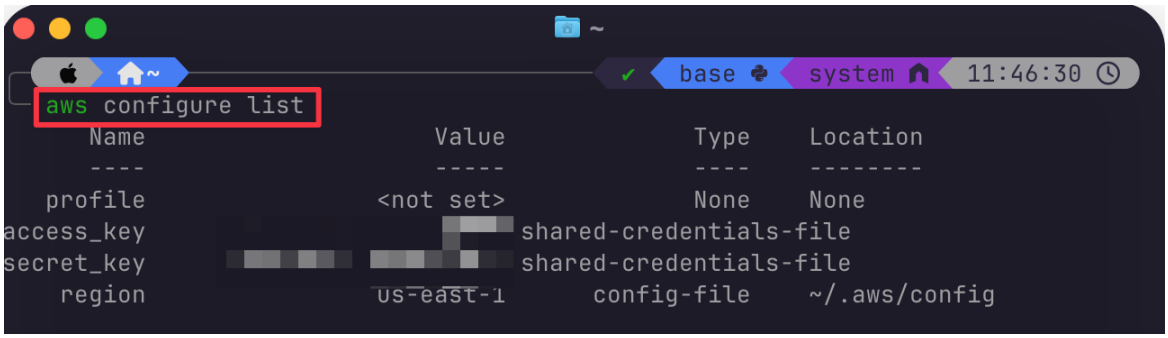

Figura 35 – Exemplo de saída do comando `aws configure list`, mostrando as credenciais e a região configurada na CLI. Fonte: Do Autor

Em seguida, recomenda-se validar a versão do *Serverless Framework* por meio do comando `serverless -version`, sendo recomendada a utilização da versão 3.x ou superior.

A configuração das credenciais da AWS pode ser realizada por dois métodos distintos:

- **Configuração permanente via CLI:** indicada para ambientes de desenvolvimento persistentes. As credenciais são armazenadas localmente nos arquivos `.aws/credentials` e `.aws/config` através do comando `aws configure`, que solicita:
  - `AWS Access Key ID`: identificador da chave de acesso;
  - `AWS Secret Access Key`: chave secreta associada à conta;
  - `Default region name`: região padrão recomendada (`us-east-1`);
  - `Default output format`: formato de saída sugerido (`json`).
- **Configuração temporária via variáveis de ambiente:** adequada para execuções efêmeras ou compartilhadas, em que as credenciais permanecem ativas apenas durante a sessão do terminal:
  - `export AWS_ACCESS_KEY_ID="sua_access_key"`;
  - `export AWS_SECRET_ACCESS_KEY="sua_secret_key"`;
  - `export AWS_DEFAULT_REGION="us-east-1"`.

Ao término desta etapa, o ambiente de desenvolvimento estará totalmente preparado, com todas as ferramentas nas versões corretas e as credenciais devidamente configuradas e validadas.

6 IMPLANTAÇÃO DO SISTEMA

A implantação e configuração do sistema foram estruturadas em etapas progressivas, iniciando-se após a conclusão da configuração sistêmica e avançando até a integração com serviços externos. Este capítulo tem como objetivo apresentar detalhadamente esses procedimentos, evidenciando a justificativa técnica de cada decisão e assegurando que o processo possa ser reproduzido de forma consistente em diferentes ambientes.

6.1 Fase I — Implantação da Infraestrutura Principal

Com o ambiente de desenvolvimento devidamente configurado, o próximo passo consiste em preparar o diretório da aplicação, que concentra os arquivos essenciais para a orquestração da infraestrutura. Entre os principais arquivos, destacam-se:

- `serverless.yml`: define os recursos e serviços da AWS, incluindo funções Lambda, buckets S3 e endpoints do API Gateway.
- `package.json`: gerencia dependências JavaScript necessárias para a execução de *scripts* de *build* e *plugins* do *Serverless Framework*.
- *Scripts* Python (`*.py`): implementam a lógica das funções Lambda e demais processamentos internos do sistema.

Para acessar o diretório responsável pela orquestração da infraestrutura, utiliza-se o comando `cd src/serverless`. Em seguida, a listagem detalhada do conteúdo pode ser obtida por meio de `ls -la`, que exibe todos os arquivos, incluindo os ocultos, juntamente com informações sobre permissões, proprietários e tamanhos. Essa verificação é fundamental para confirmar que todos os arquivos centrais estão presentes e que a estrutura está adequada para a instalação das dependências e para a subsequente implantação da infraestrutura.

Na Figura 36, os números 1 e 2 destacam, respectivamente, a execução dos comandos `cd src/serverless` e `ls -la`. A imagem ilustra a presença dos arquivos essenciais, como `serverless.yml`, `package.json` e os módulos Python necessários para as funções Lambda.

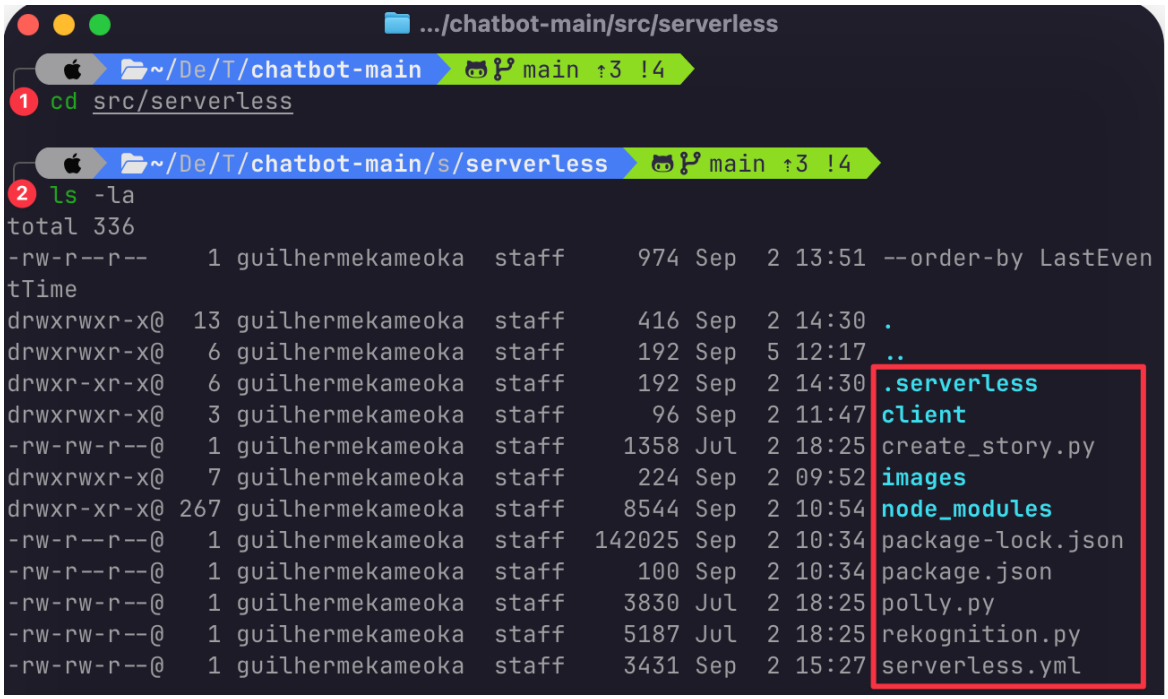

Figura 36 – Conteúdo do diretório `src/serverless`, incluindo `serverless.yml`, `package.json` e scripts Python das funções Lambda. Fonte: Do Autor

Nesta etapa, dois plugins do *Serverless Framework* são fundamentais: o `serverless-s3-sync`, responsável por sincronizar arquivos estáticos com o Amazon S3, e o `serverless-finch`, utilizado para a implantação automatizada da camada de *frontend*. A instalação dos plugins é realizada por meio do comando:

```
npm install serverless-s3-sync serverless-finch -save-dev
```

A execução desse comando pode ser visualizada na Figura 37, que ilustra a instalação dos pacotes no projeto.

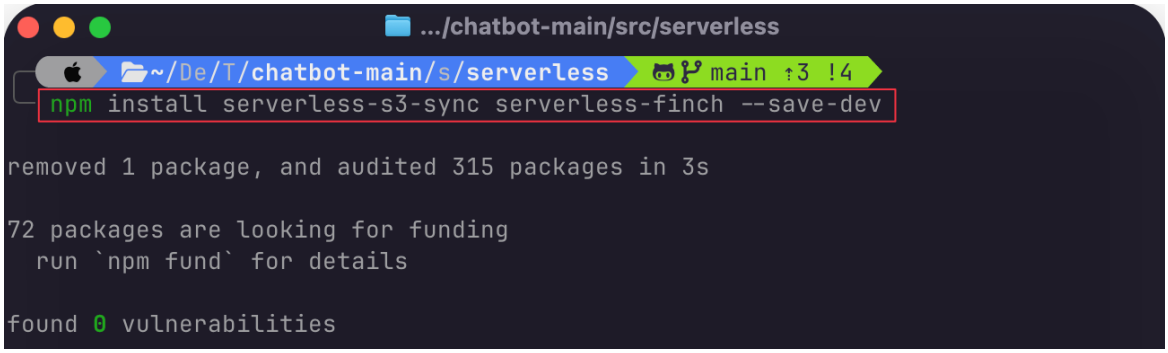

Figura 37 – Instalação dos plugins `serverless-s3-sync` e `serverless-finch` via `npm`. Fonte: Do Autor

Após a instalação, recomenda-se verificar se os pacotes foram corretamente adicionados ao projeto, utilizando o comando `npm list -depth=0`

A Figura 38 exibe a saída do comando, permitindo verificar que todos os plugins necessários foram corretamente instalados. Dessa forma, é possível confirmar que o ambiente está devidamente preparado para a implantação da infraestrutura.

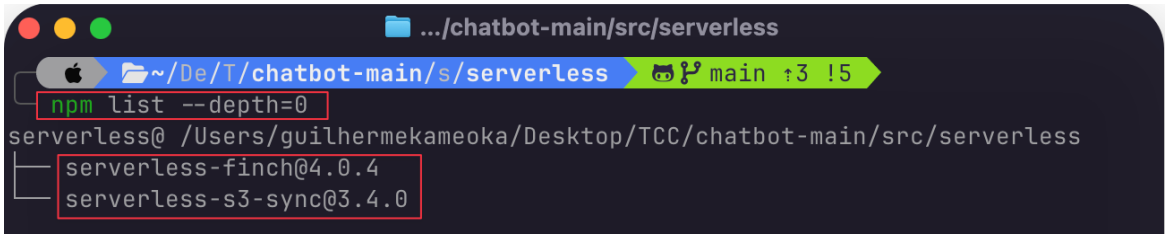

Figura 38 – Verificação das dependências instaladas no projeto utilizando `npm list -depth=0`. Fonte: Do Autor

Outro ponto crítico nesta fase é a configuração de identificadores únicos para os buckets S3, uma vez que o namespace deste serviço é global. Caso dois usuários tentem criar buckets com o mesmo nome, ocorrerá conflito, impedindo a criação correta dos recursos. Para contornar essa restrição, o arquivo `serverless.yml` deve ser editado na seção `environment`, substituindo o valor da variável `BUCKET_IMAGES` por um identificador exclusivo.

Diversas estratégias práticas podem ser adotadas para garantir a unicidade, incluindo a concatenação de datas em formato ISO, a utilização de iniciais do desenvolvedor ou a aplicação de funções hash. Essas abordagens são detalhadas na Tabela 2, que apresenta exemplos e vantagens de cada método. É fundamental substituir o marcador `<SEU_IDENTIFICADOR_UNICO>` para evitar conflitos de nomenclatura durante o processo de implantação.

Método	**Exemplo**
Data ISO	rogerio-conta-historias-polly-bucket-20250902
Iniciais + Timestamp	rogerio-conta-historias-polly-bucket-gk2025
UUID Simplificado	rogerio-conta-historias-polly-bucket-a1b2c3
Hash MD5 Parcial	rogerio-conta-historias-polly-bucket-7f8a9b

Tabela 2 – Estratégias para Geração de Identificadores Únicos. Fonte: Do Autor

Com as dependências instaladas e os identificadores devidamente configurados, a implantação pode ser executada com o comando: `sls deploy`

Esse procedimento dispara uma série de operações automáticas, incluindo a validação da sintaxe do `serverless.yml`, o empacotamento do código das funções Lambda, a

criação de uma *stack* no AWS CloudFormation e a provisão de recursos como buckets S3, endpoints no API Gateway e permissões IAM. Ao término da execução, são exibidos os artefatos gerados, entre eles os endpoints da API, que servirão de base para a configuração posterior do módulo conversacional.

6.2 Fase II — Implantação do Módulo Conversacional

O módulo de processamento conversacional representa o núcleo da lógica do sistema, responsável pelo gerenciamento das intenções do usuário, integração com serviços de IA e orquestração das respostas multimodais. A implantação deste módulo envolve a preparação do ambiente, configuração de variáveis de ambiente para integração com serviços externos e execução do deploy das funções Lambda.

6.2.1 PREPARAÇÃO DO AMBIENTE DE BOT

Para iniciar, é necessário acessar o diretório que contém os arquivos do módulo conversacional. Este diretório inclui:

- `lambda_function.py`: responsável pelo roteamento das intents.
- `serverless.yml`: define a infraestrutura *serverless*.
- Scripts `intent_*.py`: implementam as diferentes intenções.
- `functions.py`: concentra funções auxiliares para interação com Amazon Lex e serviços externos.

A partir da raiz do projeto, a navegação e verificação da estrutura podem ser realizadas com os comandos `cd src/bot_function` seguido por `ls -la`, como ilustrado na Figura 39. Esta etapa é essencial para confirmar que todos os arquivos centrais estão presentes e que a estrutura do diretório está adequada para a instalação das dependências e subsequente implantação.

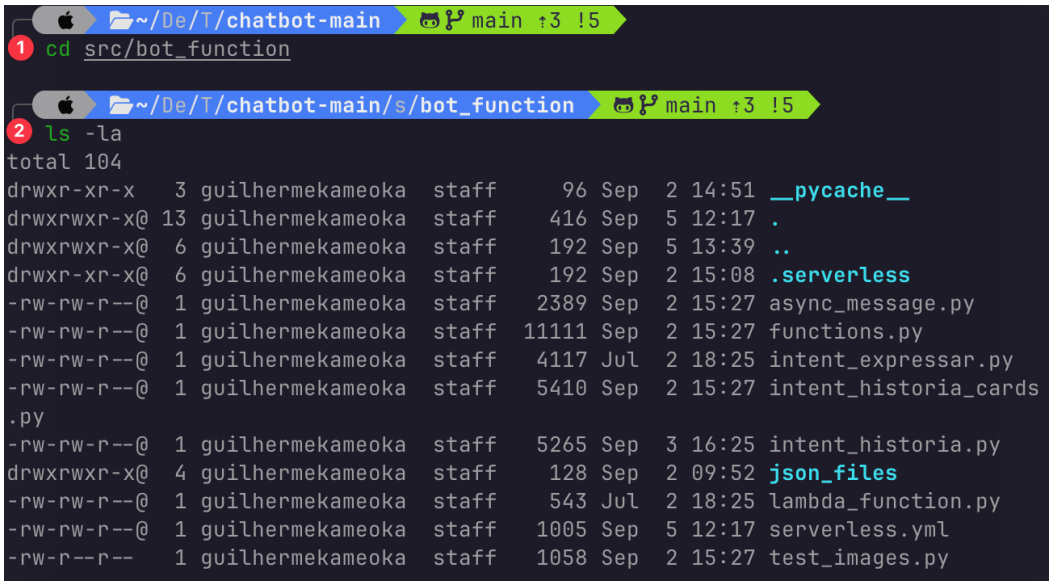

Figura 39 – Estrutura do diretório do módulo conversacional, contendo arquivos centrais para a orquestração das funções Lambda. Fonte: Do Autor

6.2.2 CONFIGURAÇÃO DE VARIÁVEIS DE AMBIENTE E INTEGRAÇÃO COM SERVIÇOS EXTERNOS

Para que o módulo interaja com APIs externas e utilize recursos da nuvem de forma segura, é necessário a configuração de variáveis de ambiente. Estas incluem credenciais para a Microsoft Translator API, Kommunicate.io e OpenAI GPT, além de identificadores de buckets S3 e funções assíncronas. A Figura 40 ilustra um exemplo de configuração dessas variáveis no arquivo `serverless.yml`, demonstrando como os dados sensíveis podem ser integrados de maneira segura à infraestrutura.

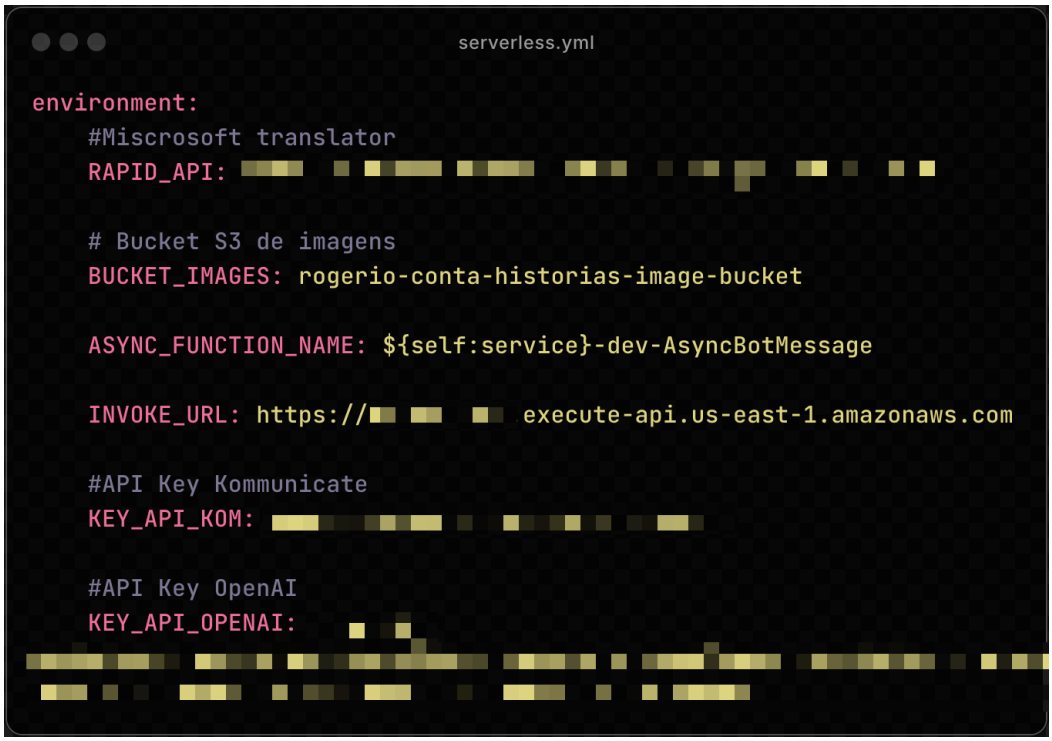

Figura 40 – Exemplo de configuração de variáveis de ambiente no arquivo `serverless.yml`, incluindo credenciais de serviços externos e identificadores de recursos AWS. Fonte: Do Autor

A obtenção das credenciais deve seguir os procedimentos específicos de cada serviço, resumidos a seguir. **Um guia detalhado com o passo a passo para cada plataforma encontra-se no Apêndice A.**

- **Microsoft Translator API (RapidAPI):** criar conta, subscrever ao plano gratuito e copiar a chave `X-RapidAPI-Key`.
- **Kommunicate API:** criar conta, acessar API Keys no *dashboard* e gerar chave com permissões de *bot integration*.
- **OpenAI API:** criar chave secreta na plataforma OpenAI, nomeá-la e armazenar com segurança.

6.2.3 EXECUÇÃO DA IMPLANTAÇÃO DO MÓDULO CONVERSACIONAL

Com o ambiente preparado e as credenciais configuradas, o deploy do módulo é realizado utilizando o *Serverless Framework*. A execução do comando `sls deploy` inicia a criação das funções Lambda, configuração de triggers e provisionamento de permissões IAM. Ao final do processo, os artefatos gerados incluem:

- **botFunction-dev-botFunction:** função principal de processamento conversacional.

- **botFunction-dev-AsyncBotMessage:** função para execução assíncrona de mensagens.
- **Roles IAM:** permissões automaticamente configuradas para acesso seguro aos serviços AWS.

Esta fase garante que o módulo de processamento conversacional esteja operacional, totalmente integrado aos serviços externos e pronto para receber requisições de usuários em tempo real, estabelecendo a base para a integração final com o Amazon Lex.

6.3 Fase III — Configuração e Integração do Amazon Lex

O Amazon Lex V2 é o serviço responsável pelo processamento de linguagem natural, combinando tecnologias de ASR e NLU para interpretar intenções do usuário e orquestrar diálogos complexos. O modelo de conversação é estruturado em componentes principais:

- ***Intents*:** representam ações que o usuário deseja executar.
- ***Slots*:** parâmetros necessários para completar uma *intent*.
- ***Utterances*:** exemplos de como os usuários podem expressar uma intenção.
- ***Fulfillment*:** lógica de negócio responsável por processar requisições.

6.3.1 IMPORTAÇÃO DO MODELO CONVERSACIONAL

A configuração inicia com a importação do modelo pré-treinado do Lex, disponibilizado em arquivo `.zip`. Para tanto, o usuário deve acessar o console do Lex (<https://console.aws.amazon.com/lexv2/>), selecionar a região `us-east-1` e utilizar a função *Import* para enviar o arquivo `src/bot/project_autism-LexJson.zip`. O tempo estimado para processamento pode variar entre um a cinco minutos. A Figura 41 ilustra a interface do console durante a importação do modelo.

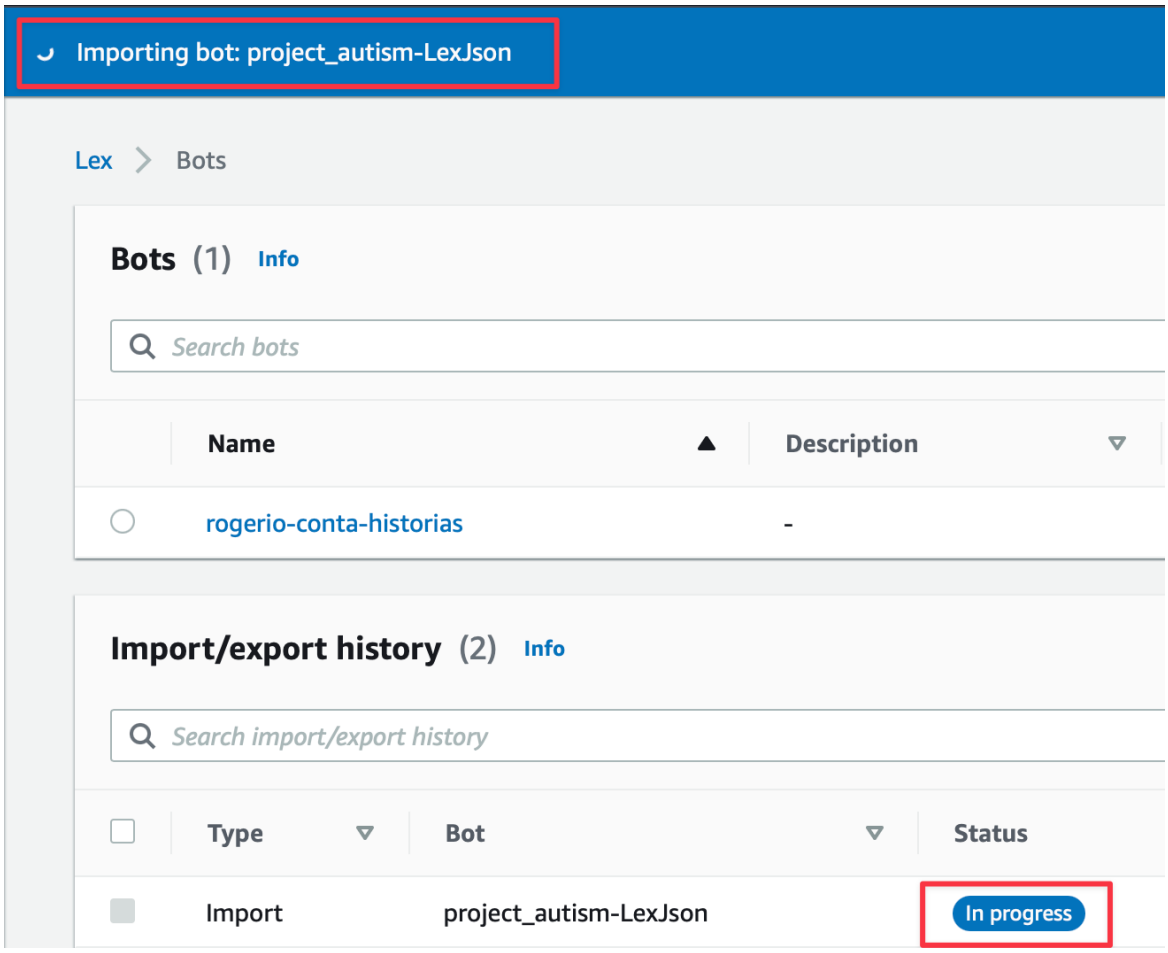

Figura 41 – Interface do Amazon Lex V2 durante a importação do modelo de conversação. Fonte: Do Autor

6.3.2 COMPILAÇÃO E *BUILD* DO MODELO

Após a importação, realiza-se o *build*, etapa que compila intents, slots e utterances em um modelo funcional de aprendizado de máquina, capaz de processar interações em tempo real. Para isso, deve-se acessar a opção **Intents** no menu lateral do Amazon Lex e clicar em ***Build*** no canto superior direito da tela. O monitoramento desta etapa permite acompanhar o status de compilação, garantindo que o modelo esteja pronto para atender às solicitações dos usuários. A Figura 42 apresenta o fluxo de execução do *build* e o monitoramento do status do processo.

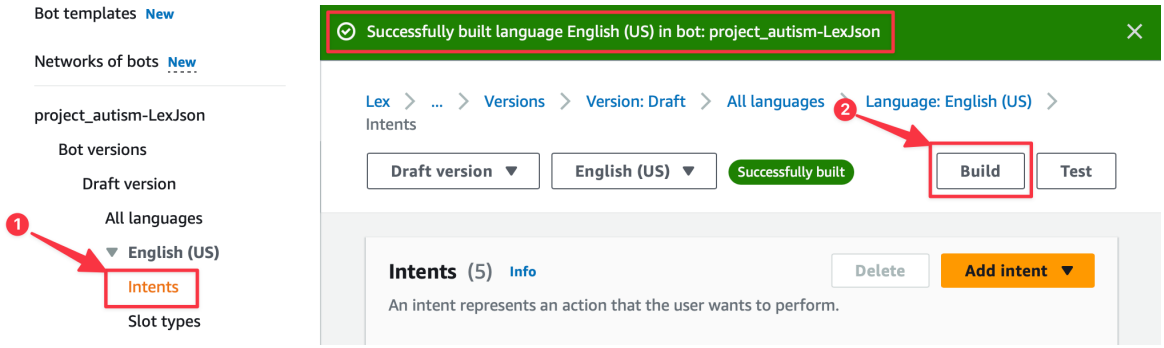

Figura 42 – Fluxo de compilação (*build*) do modelo Amazon Lex, incluindo monitoramento do status do processo. Fonte: Do Autor

6.3.3 INTEGRAÇÃO COM AWS LAMBDA

A integração entre o Amazon Lex e as funções Lambda ocorre por meio de *code hooks*, responsáveis por acionar a lógica de negócio vinculada a cada *intent*. Para que essa comunicação seja possível, é necessário configurar as permissões no IAM, garantindo que o serviço Amazon Lex esteja autorizado a invocar a função Lambda correspondente. Esse processo de configuração é realizado em duas etapas, executadas diretamente no terminal por meio da ferramenta CLI:

1. **Concessão de permissões IAM:** a função Lambda deve permitir explicitamente que o serviço Amazon Lex a invoque. Essa autorização é concedida por meio da ação `lambda:InvokeFunction`, que define a permissão de execução da função. A permissão é concedida diretamente ao serviço Amazon Lex (identificado como `lex.amazonaws.com`). Dessa forma, o Lex passa a ter autorização para executar a função Lambda sempre que um *code hook* for acionado durante o diálogo.

```
# Obter Account ID
ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)

# Conceder permissão para Lex invocar Lambda
aws lambda add-permission \
  --function-name botFunction-dev-botFunction \
  --statement-id lex-invoke-permission-$(date +%s) \
  --action lambda:InvokeFunction \
  --principal lex.amazonaws.com \
  --source-arn "arn:aws:lex:us-east-1:
  $ACCOUNT_ID:bot-alias/$BOT_ID/TSTALIASID" \
  --region us-east-1
```

2. **Configuração do *code hook* no Bot Alias:** cada *intent* do bot pode ser vinculada a uma função Lambda para *fulfillment*. Isso é feito atualizando o alias do bot com a especificação do *code hook*, informando o *Amazon Resource Name* (Nome de Recurso da Amazon) (ARN) da função Lambda e a versão da interface.

```
# Configurar Lambda como fulfillment handler
aws lexv2-models update-bot-alias \
  --bot-id $BOT_ID \
  --bot-alias-id TSTALIASID \
  --bot-alias-name TestBotAlias \
  --bot-version DRAFT \
  --bot-alias-locale-settings '{
    "en_US": {
      "enabled": true,
      "codeHookSpecification": {
        "lambdaCodeHook": {
          "lambdaARN": "arn:aws:lambda:us-east-1:
          '$ACCOUNT_ID':function:botFunction-dev-botFunction",
          "codeHookInterfaceVersion": "1.0"
        }
      }
    }
  }' \
  --region us-east-1
```

Essa integração garante que cada requisição do usuário seja processada de forma consistente e segura, mantendo um fluxo de conversação confiável e escalável. Além disso, o modelo pode ser atualizado incrementalmente, com a adição de novas *intents*, *slots* ou regras de diálogo, sem comprometer a integridade do sistema existente.

6.4 Fase IV — Integração com o Kommunicate

Na etapa final, o sistema é integrado à plataforma Kommunicate.io, que atua como camada de interface entre o Amazon Lex e o usuário, ampliando a acessibilidade da solução ao permitir a incorporação do *widget* conversacional em websites e aplicativos. A plataforma do Kommunicate fornece uma abstração eficiente entre o usuário final e o Amazon Lex,

oferecendo benefícios técnicos como uma interface unificada para incorporação em websites, gerenciamento de sessões que mantém o contexto conversacional, analytics integrados para monitoramento de uso e desempenho do bot, e suporte a múltiplos canais de interação, garantindo maior flexibilidade e alcance da aplicação.

6.4.1 CONFIGURAÇÃO DE USUÁRIO IAM ESPECIALIZADO

Para garantir a segurança e aderência ao princípio de menor privilégio, é criado um usuário IAM dedicado, com permissões restritas apenas às operações necessárias do Amazon Lex.

1. **Criação do usuário IAM:** Para iniciar o processo, acesse o console do IAM (<https://console.aws.amazon.com/iam/>), selecione a opção *Users* e clique em *Create user*. Defina o nome do usuário como `kommunicate-bot`, conforme exemplificado na Figura 43.

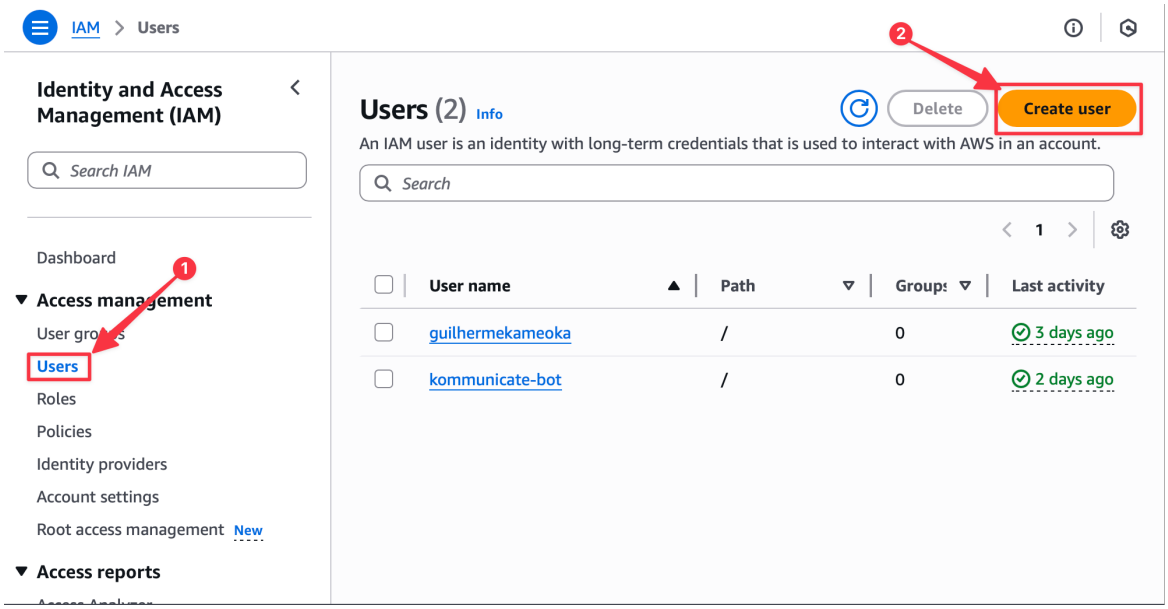

Figura 43 – Criação de um novo usuário no IAM. Fonte: Do Autor.

2. **Configuração de políticas granulares:** Em seguida, é necessário atribuir permissões adequadas ao usuário por meio da associação de políticas IAM, garantindo que apenas os privilégios estritamente necessários sejam concedidos. O fluxo desse procedimento pode ser observado na Figura 44.

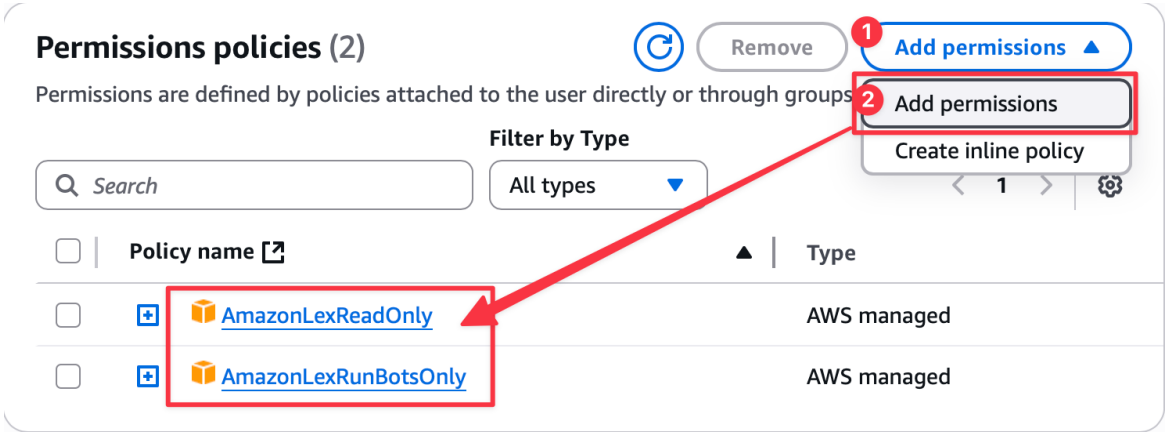

Figura 44 – Configuração de permissões no IAM. Fonte: Do Autor.

3. **Geração e gerenciamento de credenciais:** Para criar uma nova chave de acesso, acesse o serviço *IAM*, selecione *Users* e, em seguida, o usuário *kommunicate-bot*. Na aba *Security Credentials*, localize a seção *Access keys* e clique em *Create access key*, conforme ilustrado na Figura 45.

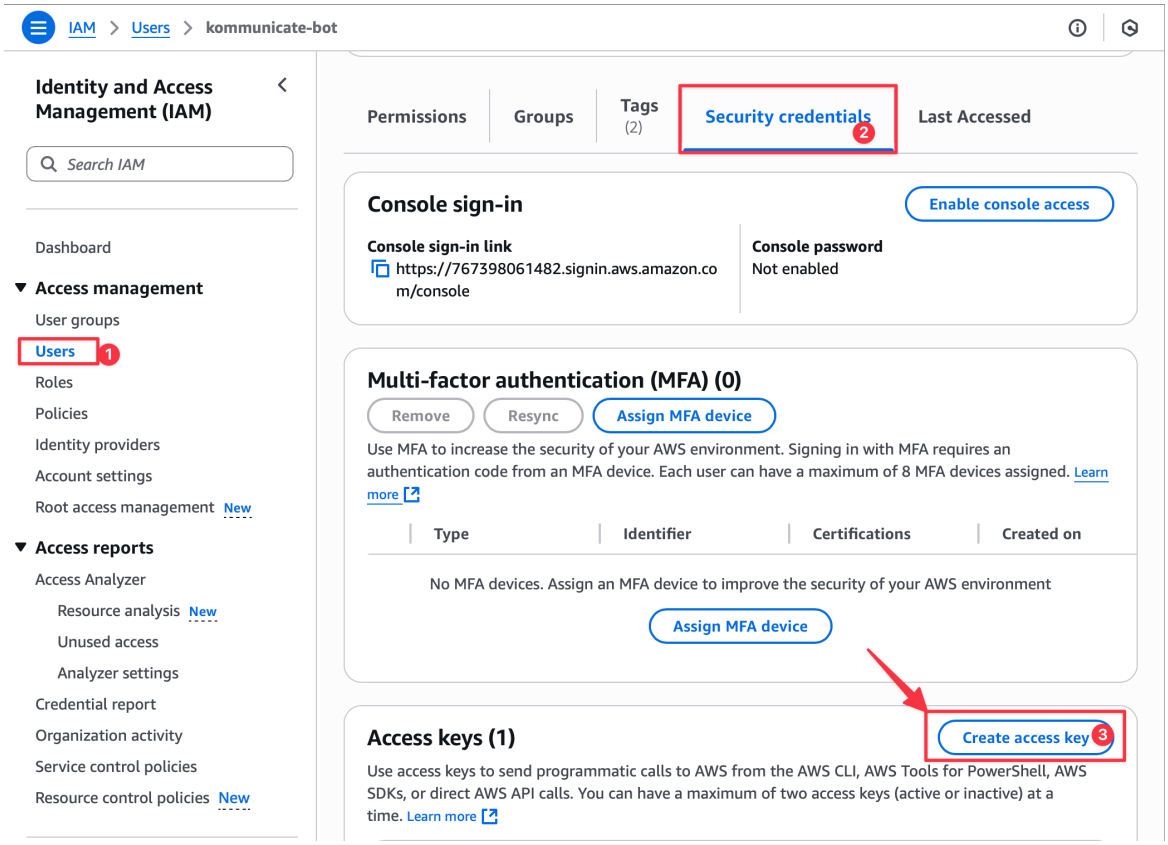

Figura 45 – Processo de geração de chave de acesso no IAM. Fonte: Do Autor.

Criar a *access key* destinada ao uso pelo serviço de terceiros, copiando imediatamente o *Access Key ID* e o *Secret Access Key*, conforme ilustrado na Figura 46. Recomenda-se que essas credenciais sejam armazenadas em um gerenciador seguro de senhas e que nunca sejam versionadas ou expostas em repositórios públicos.

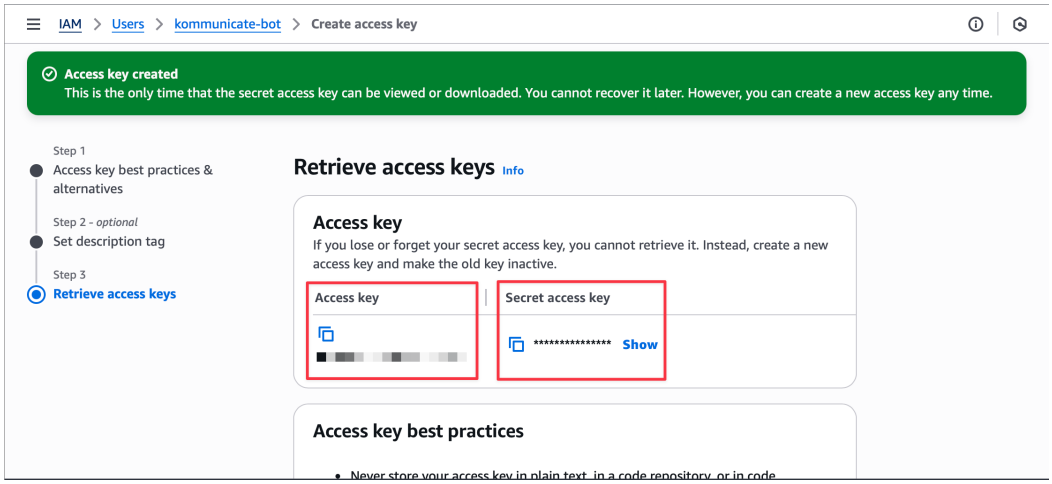

Figura 46 – Geração de credenciais de acesso no IAM. Fonte: Do Autor.

6.4.2 CONFIGURAÇÃO NO *DASHBOARD* DO KOMMUNICATE

Com as credenciais do usuário IAM devidamente geradas, a etapa final consiste em configurar a integração diretamente no painel administrativo da plataforma Kommunicate. O processo envolve o preenchimento de parâmetros específicos, como chaves de acesso, região da AWS e o nome do bot, que estabelecem a conexão entre a interface do usuário e o motor conversacional do Amazon Lex. **Um guia visual com o passo a passo detalhado para esta configuração encontra-se no Apêndice A.3**.

Após a configuração, a conectividade é validada por meio de um comando de teste executado via CLI para confirmar que a comunicação está operacional:

```
aws lexv2-runtime recognize-text \
  --bot-id $BOT_ID \
  --bot-alias-id TSTALIASID \
  --locale-id en_US \
  --session-id test-session-123 \
  --text "Olá" \
  --region us-east-1
```

Essa integração final garante que o bot esteja acessível de forma segura e eficiente em múltiplos canais, mantendo o controle de sessões, segurança e métricas de desempenho.

6.5 Fase V — Implantação do *Frontend Web*

A interface do sistema é disponibilizada por meio do Amazon S3, configurado no modo de hospedagem estática, e o processo consiste em compilar os arquivos do *frontend* e enviá-los para um bucket do S3, que, ao ser configurado como *website*, disponibiliza automaticamente um endpoint público. A implantação é realizada de forma automatizada através do comando `sls client deploy`, que provisiona o bucket, aplica as permissões adequadas, realiza o upload dos arquivos e ativa o modo de hospedagem estática.

6.5.1 PREPARAÇÃO PARA IMPLANTAÇÃO DO *FRONTEND*

Antes da execução do deploy, é necessário garantir que os arquivos compilados do *frontend* estejam presentes no diretório `client/dist/`, incluindo o `index.html`, arquivos de estilização, scripts JavaScript e demais recursos. Além disso, deve-se retornar ao diretório de infraestrutura (`serverless`) para que a ferramenta consiga localizar as configurações adequadas.

6.5.2 EXECUÇÃO DA IMPLANTAÇÃO DO *FRONTEND*

A implantação do *frontend* é realizada de forma automatizada pelo comando `sls client deploy`, que provisiona e configura o bucket do S3 com definições de website, realiza o upload dos arquivos HTML, CSS, JavaScript e demais recursos, e aplica automaticamente as permissões de leitura pública. Ao final, a aplicação web estará hospedada e acessível por meio de uma URL pública, com todas as políticas IAM necessárias para garantir o acesso seguro. Um exemplo de URL final de acesso é:

```
http://<NOME_DO_BUCKET>.s3-website-us-east-1.amazonaws.com
```

7 RESULTADOS

Este capítulo apresenta os resultados obtidos com a implantação e validação do sistema de *chatbot* desenvolvido. O foco desta seção é demonstrar, por meio de evidências funcionais, que os objetivos técnicos e de interação propostos foram alcançados. Os testes a seguir foram realizados utilizando a interface do Kommunicate, integrada ao ecossistema de serviços na nuvem AWS.

7.1 Interface Principal e Interação Inicial

A interface de entrada da aplicação, ilustrada na Figura 47, é composta pela página inicial e pelo *widget* de chat do Kommunicate. Ao iniciar a aplicação, o *chatbot* apresenta uma mensagem de boas-vindas juntamente com as duas funcionalidades principais. A primeira, *"Criar uma história"*, possibilita que o usuário gere uma narrativa a partir de um *Desenho - Foto* ou utilizando *Cartões* interativos. A segunda opção, *"Quero me expressar"*, disponibiliza um conjunto de ícones visuais que facilitam a comunicação de emoções e desejos, tornando a interação mais intuitiva e envolvente para a criança.

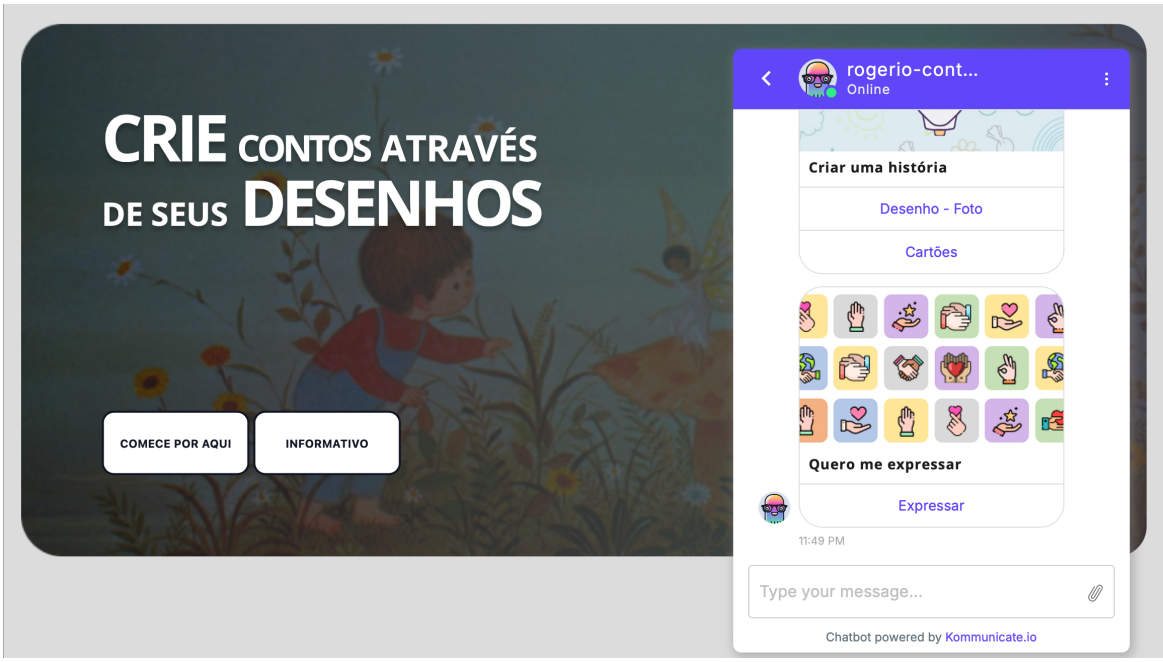

Figura 47 – Tela inicial do *chatbot* com as opções de interação. Fonte: Do Autor

7.2 Validação da Funcionalidade de Expressão Emocional

O módulo de expressão emocional foi desenvolvido para permitir que a criança comunique sentimentos e desejos de maneira visual e intuitiva. Para validar essa funcionalidade, simulou-se um fluxo completo de interação, desde a escolha da intenção de expressão até a geração da resposta multimodal, incluindo texto e áudio.

As Figuras 48 a 51 apresentam todo o fluxo de interação do módulo. O processo inicia-se com a seleção da opção de se expressar, onde o usuário escolhe o gênero da voz por meio de botões visuais claramente identificados, como mostrado na Figura 48. Em seguida, o sistema apresenta opções de expressão divididas em categorias, como *Emoções* e *Comidas*, exibidas em cards interativos, permitindo que a criança selecione de forma intuitiva a categoria desejada (Figura 49).

Ao escolher a categoria *Emoções*, o módulo exibe um conjunto de ícones representando diferentes sentimentos. A criança, então, seleciona a emoção *Feliz*, ilustrando a simplicidade e a clareza do design visual dos cards (Figura 50). Por fim, o sistema gera uma resposta multimodal, apresentando o texto da emoção selecionada no *chatbot* e reproduzindo o áudio correspondente, completando o fluxo de interação (Figura 51).

Dessa forma, o módulo de expressão emocional demonstra como escolhas visuais intuitivas, combinadas com respostas multimodais, podem proporcionar uma experiência interativa envolvente e personalizada, facilitando a comunicação de sentimentos e desejos de maneira natural e acessível para a criança.

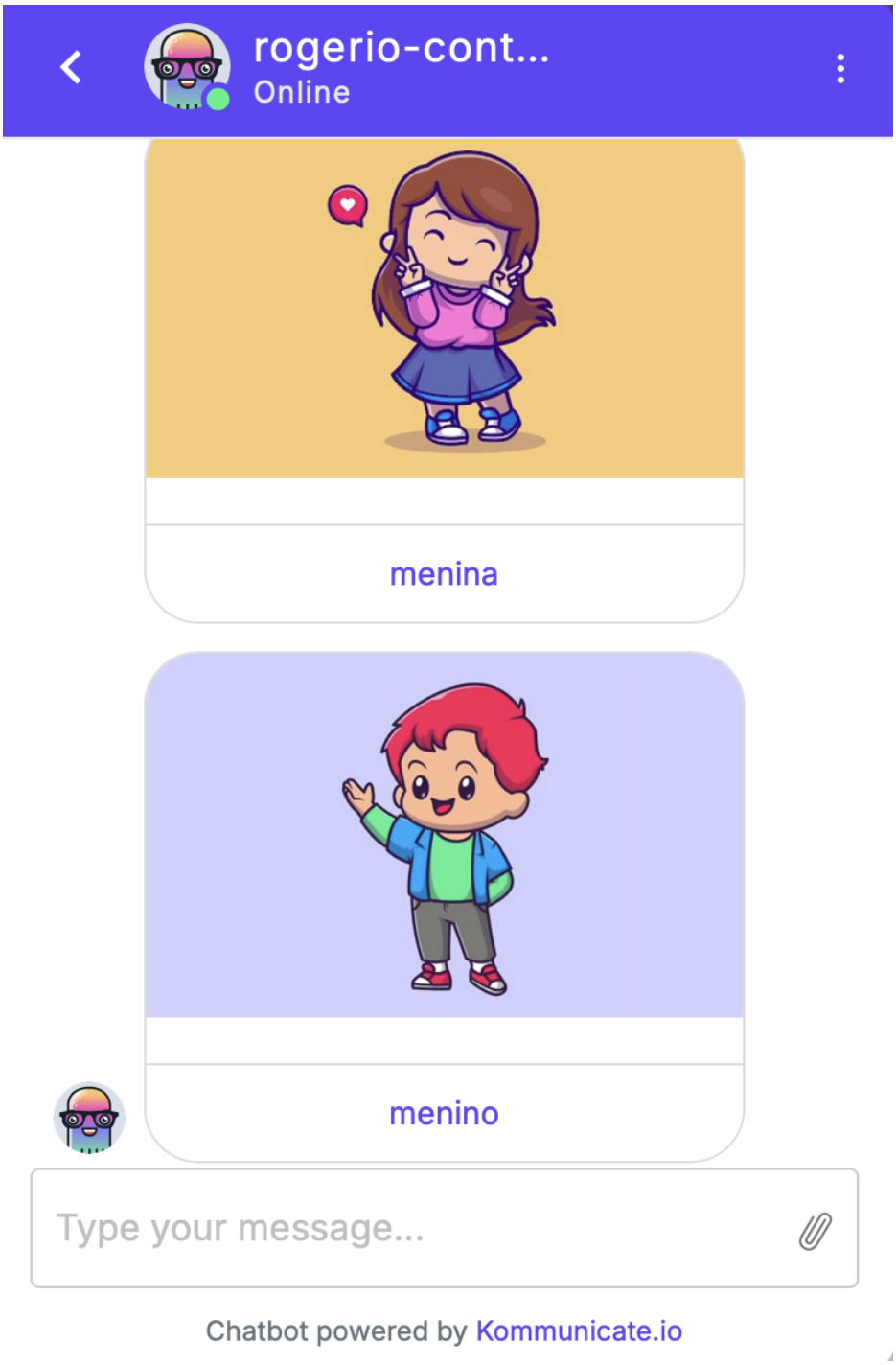

Figura 48 – Seleção da opção de expressar-se e escolha do gênero da voz. Fonte: Do Autor

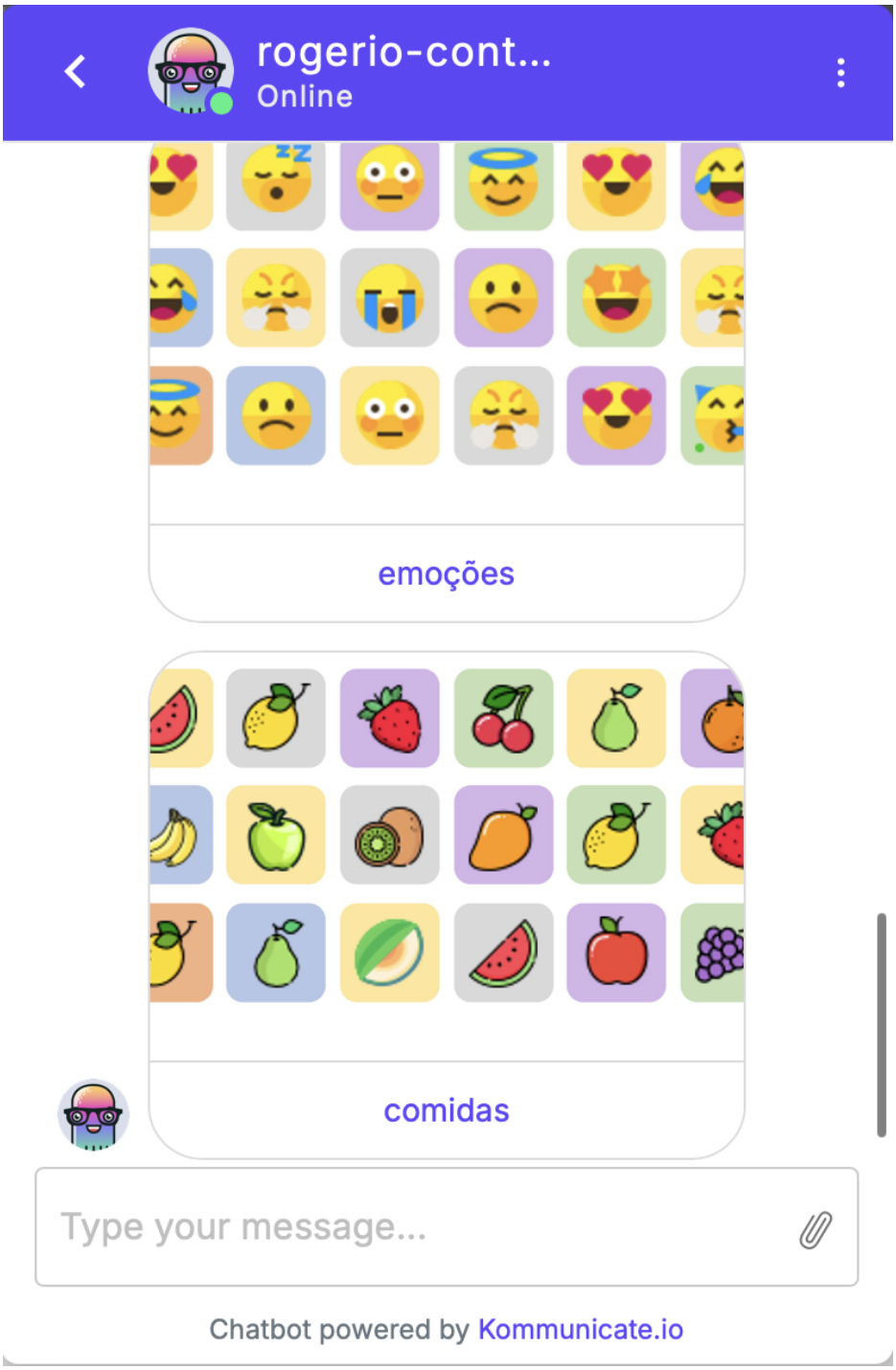

Figura 49 – Escolha do tipo de expressão: emoções ou comidas. Fonte: Do Autor

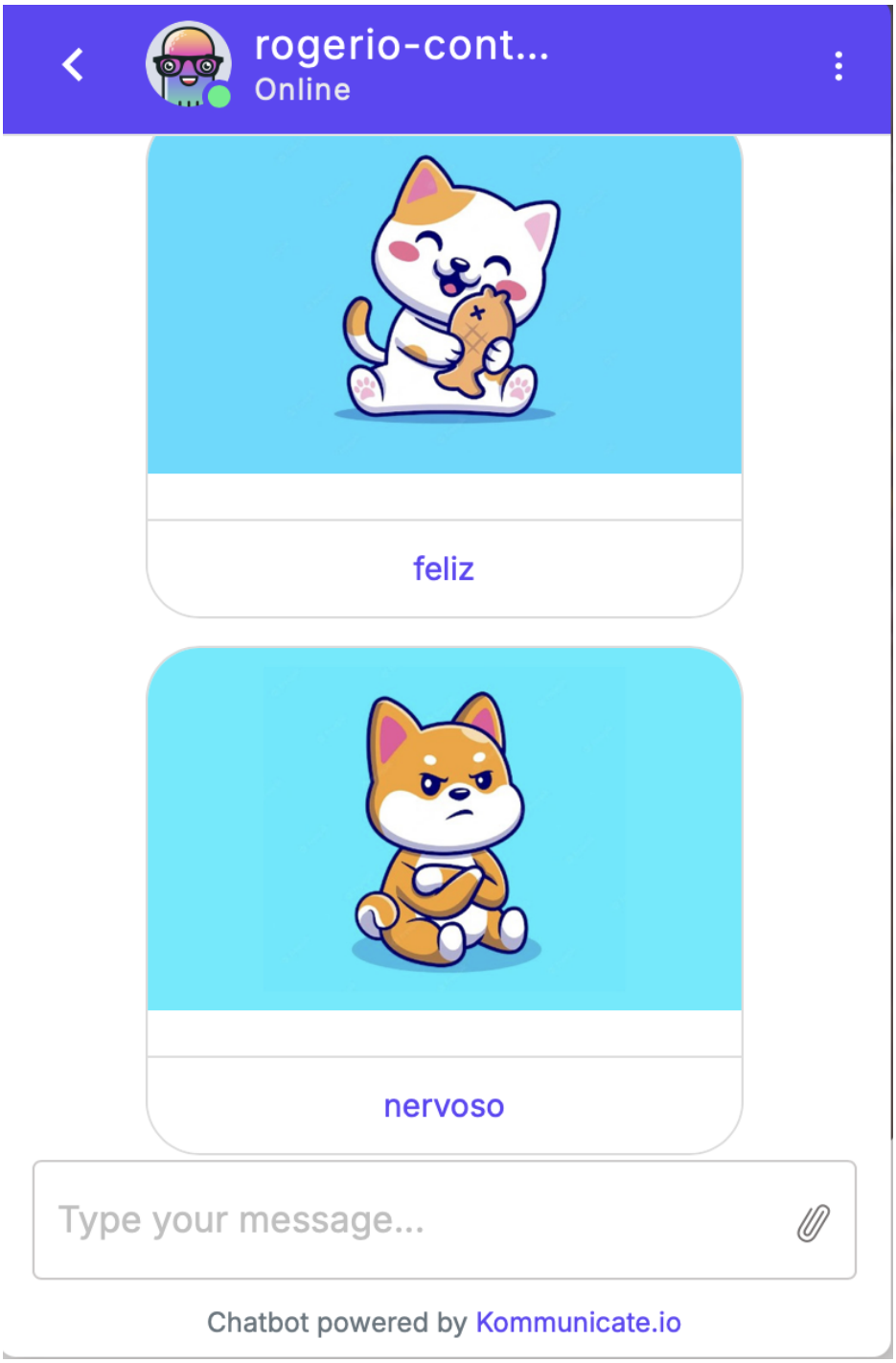

Figura 50 – Seleção da emoção *Feliz* dentre as opções disponíveis. Fonte: Do Autor

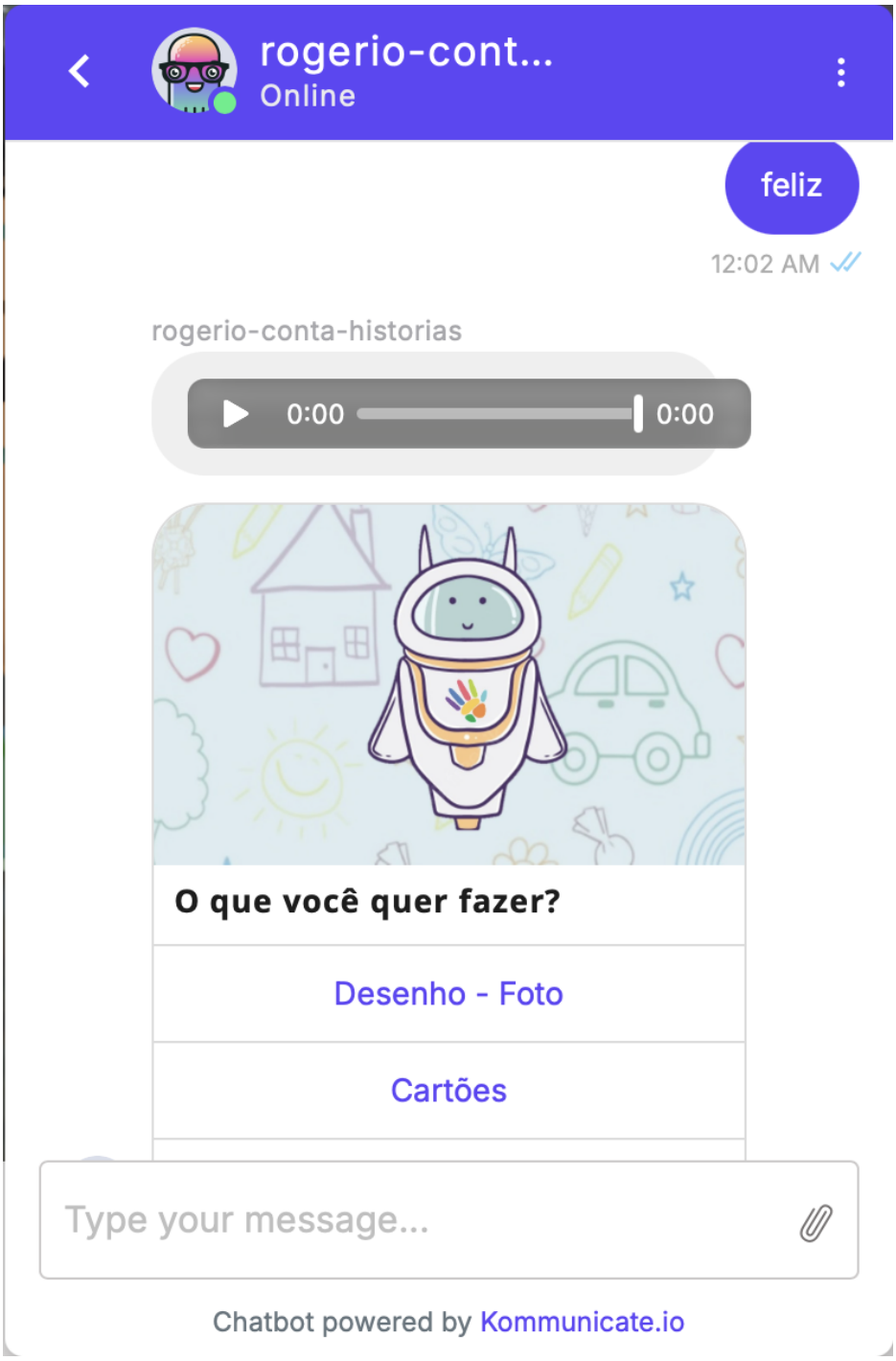

Figura 51 – Geração da resposta em áudio personalizada. Fonte: Do Autor

Dessa forma, o sistema demonstra de maneira clara e intuitiva como a criança pode se expressar, integrando escolhas visuais e respostas multimodais para proporcionar uma experiência interativa envolvente e personalizada.

7.3 Validação da Funcionalidade de Geração de Histórias

A geração de histórias é uma das funcionalidades centrais do sistema, dividida em dois fluxos distintos: a partir de imagens e a partir de cards pré-definidos. Ambos os fluxos foram validados para assegurar a correta integração entre a interface, os serviços da AWS (Rekognition, Polly) e a API da OpenAI.

7.3.1 GERAÇÃO DE HISTÓRIAS A PARTIR DE IMAGENS

Neste teste, uma imagem foi enviada pelo usuário através do chat. O sistema acionou o Amazon Rekognition para identificar os elementos visuais, construiu um prompt para a OpenAI e retornou uma narrativa em texto e áudio. A Figura 52 ilustra o envio da imagem pelo usuário, enquanto a Figura 53 apresenta a história gerada pelo *chatbot* a partir dessa imagem, evidenciando o sucesso da orquestração assíncrona dos serviços.

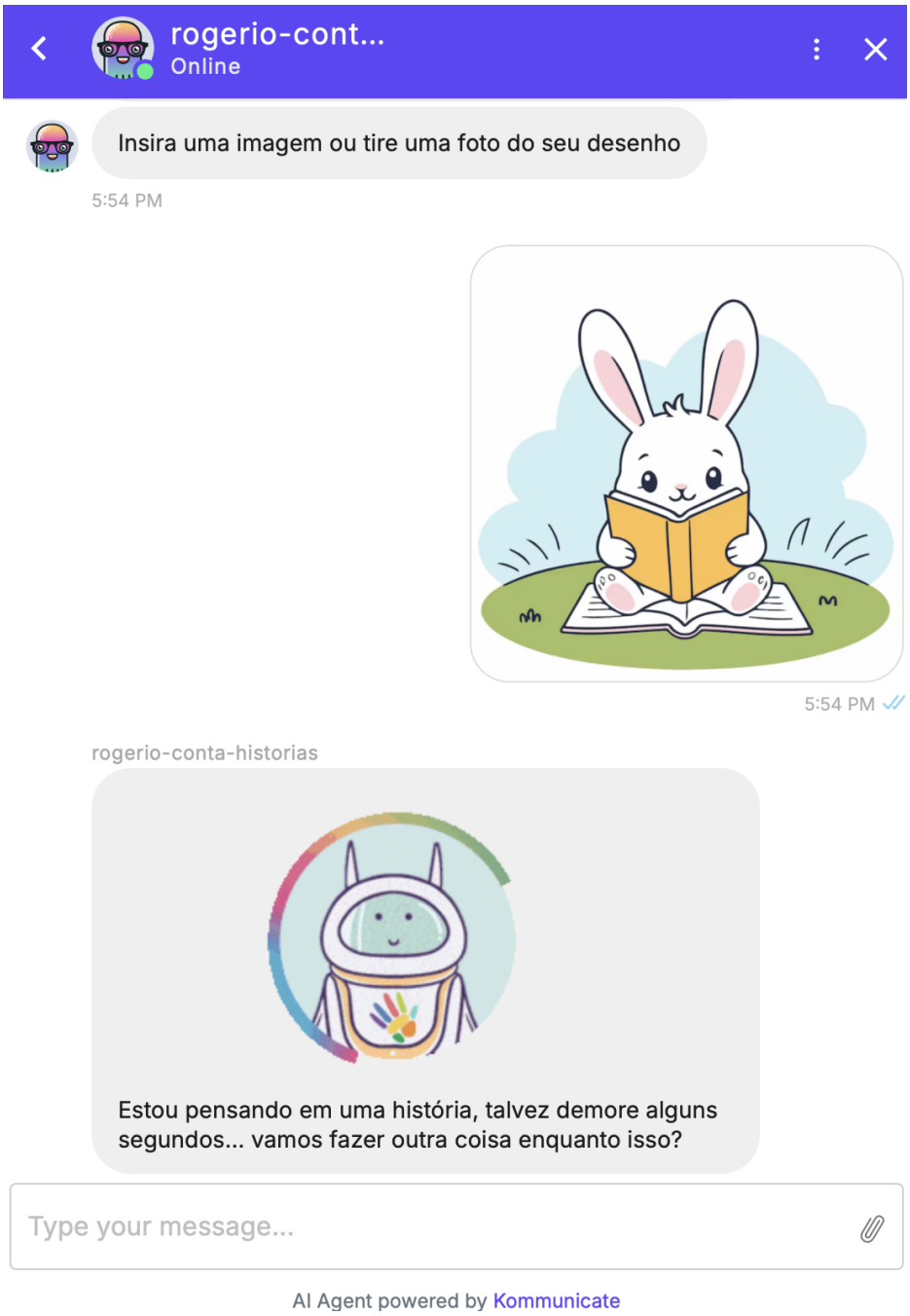

Figura 52 – Envio da imagem pelo usuário no sistema. Fonte: Do Autor

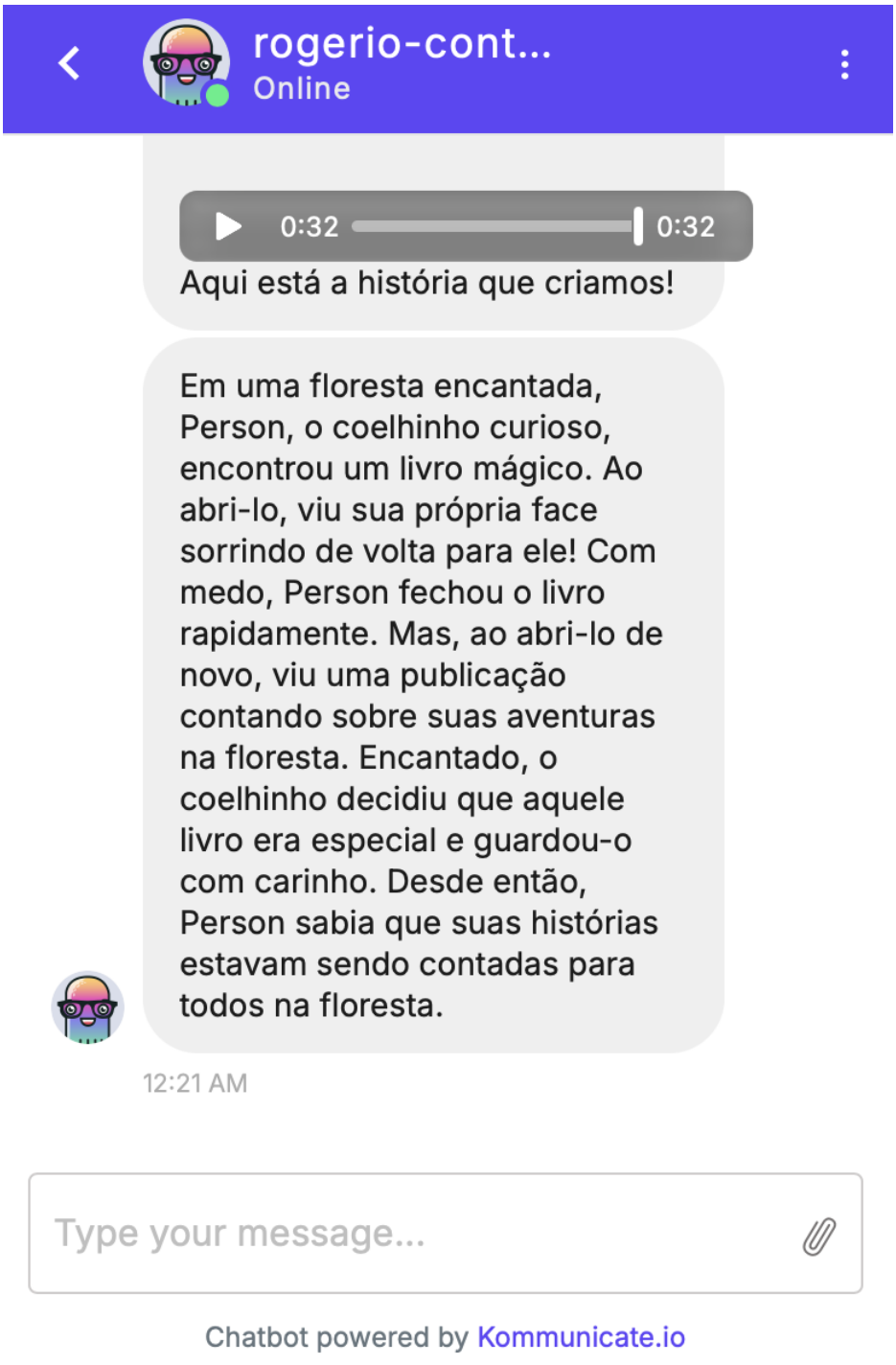

Figura 53 – Exemplo de história gerada a partir de uma imagem enviada pelo usuário. Fonte: Do Autor

7.3.2 GERAÇÃO DE HISTÓRIAS A PARTIR DE CARDS

O segundo fluxo do sistema foi validado por meio da criação de uma história a partir da seleção sequencial de cards: protagonista (gato), clima (chuvoso), meio de locomoção (avião) e local da história (espaço). As Figuras 54 a 58 ilustram cada etapa do processo,

desde a escolha individual de cada card até a narrativa final gerada, evidenciando que o sistema processa corretamente as seleções do usuário e as incorpora à história de forma lúdica, intuitiva e personalizada.

Figura 54 – Seleção do protagonista (gato) por meio de cards visuais. Fonte: Do Autor

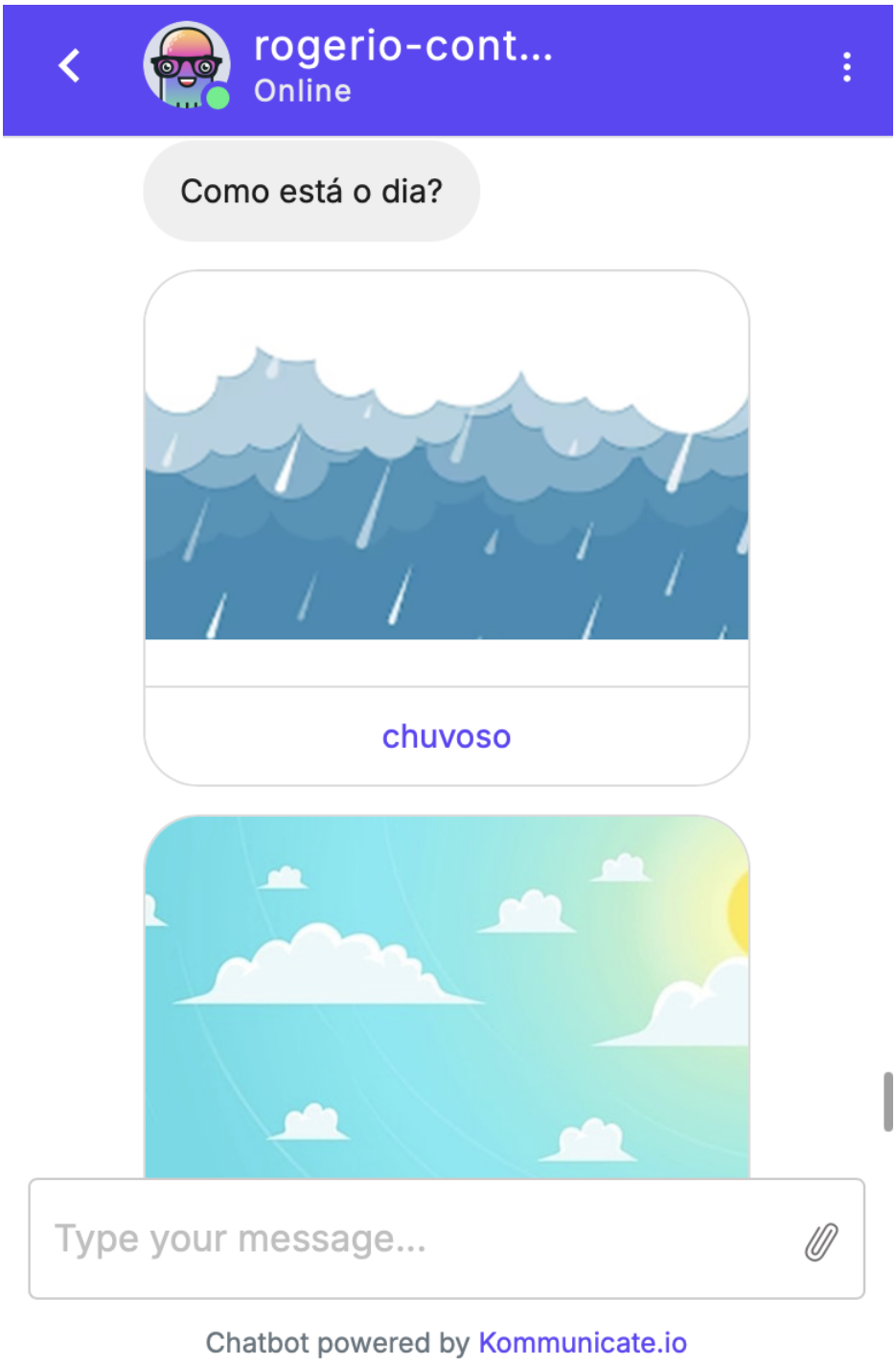

Figura 55 – Seleção do clima (chuvoso) por meio de cards visuais. Fonte: Do Autor

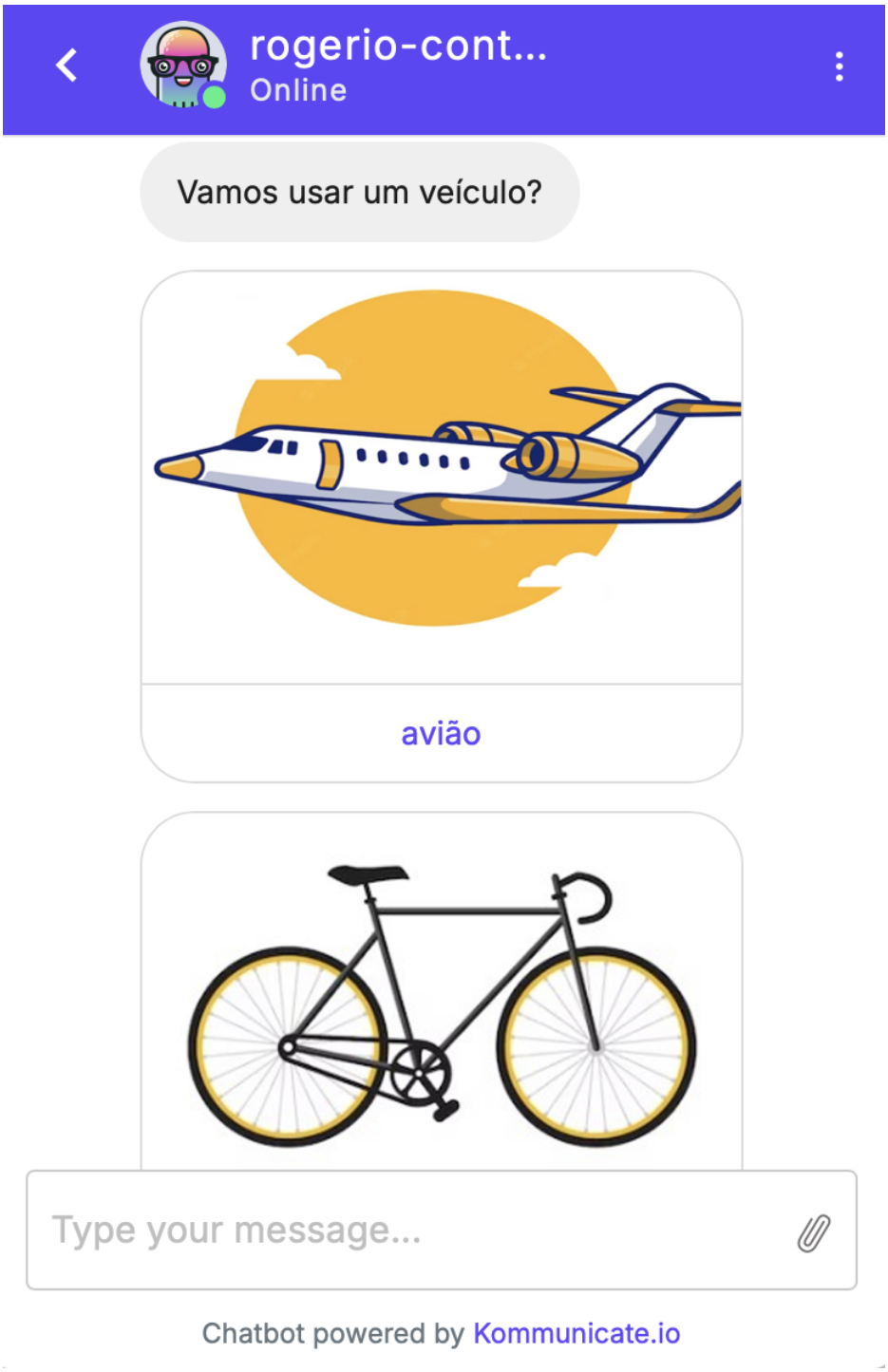

Figura 56 – Escolha do meio de locomoção (avião) por meio de cards visuais. Fonte: Do Autor

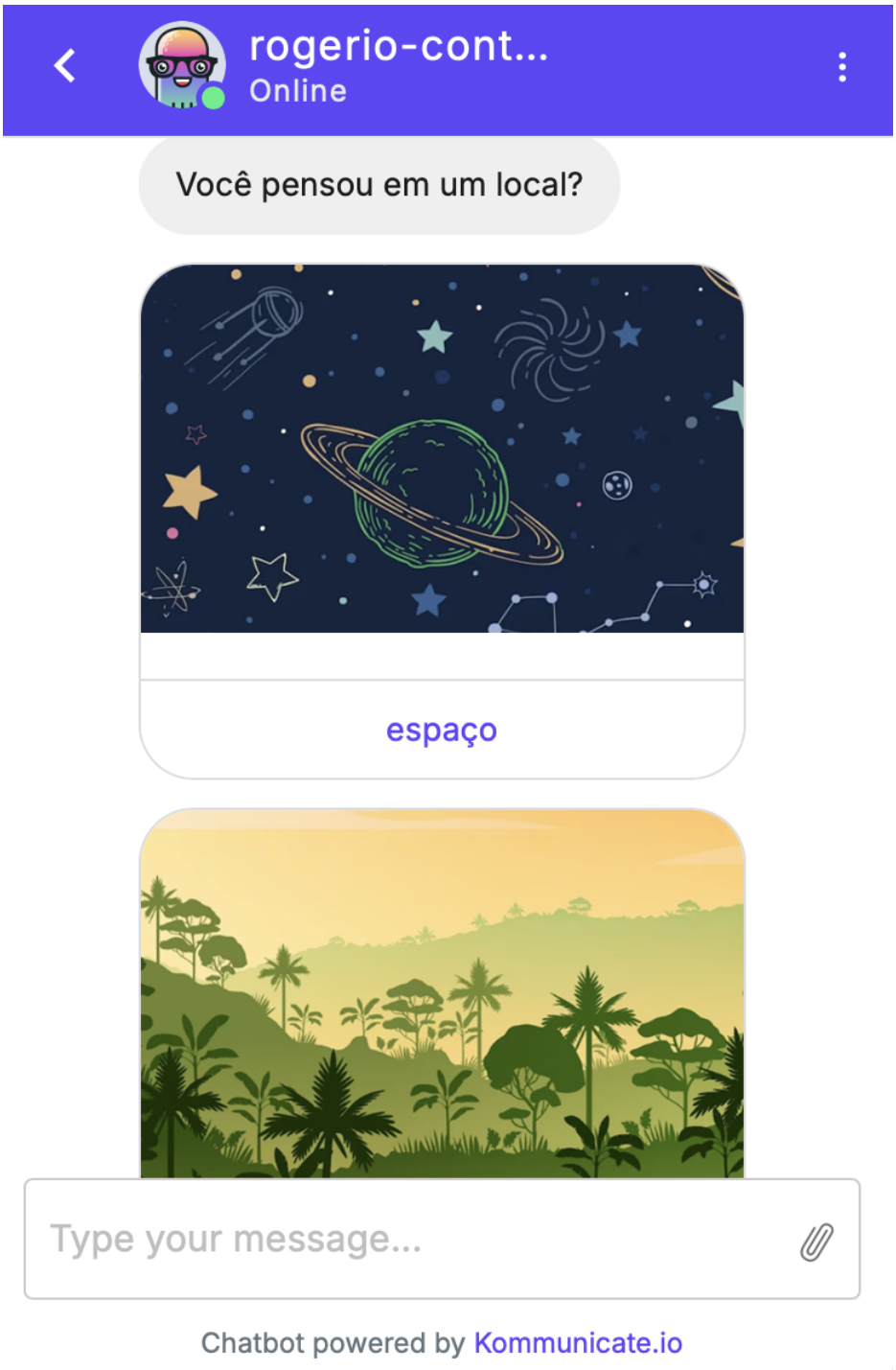

Figura 57 – Seleção do local da história (espaço) por meio de cards visuais. Fonte: Do Autor

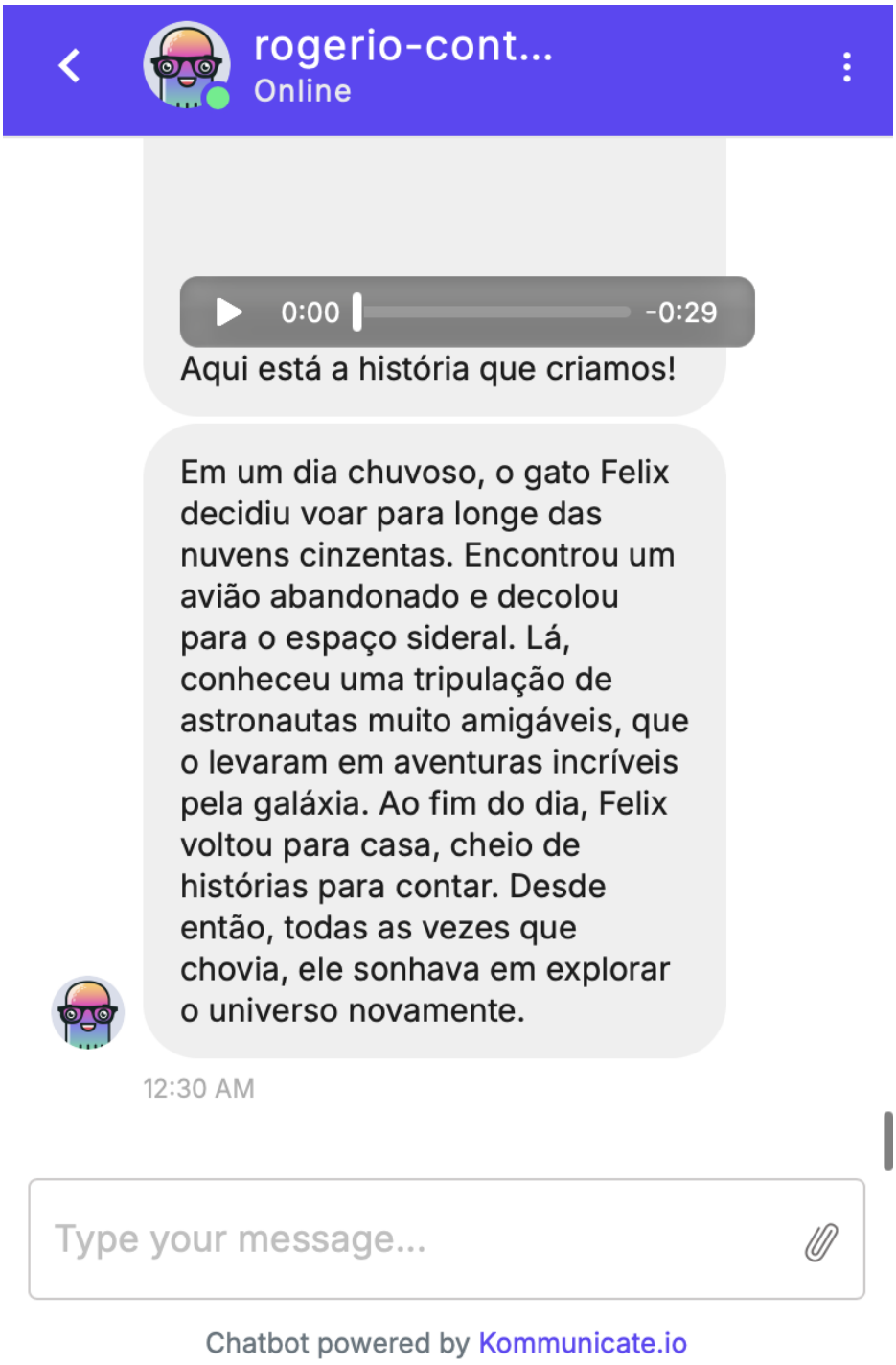

Figura 58 – História final completa gerada pelo sistema, incorporando todas as escolhas do usuário. Fonte: Do Autor

Esse módulo demonstra como a interação por meio de cards visuais, combinada com a geração automática de narrativas, oferece uma experiência lúdica, intuitiva e personalizada, permitindo que a criança construa histórias de maneira natural e envolvente.

7.4 Análise dos Resultados

Os testes funcionais confirmaram que todos os módulos desenvolvidos operam conforme o esperado, demonstrando a robustez da solução implementada. A arquitetura *serverless* na AWS mostrou-se eficaz, permitindo a orquestração resiliente de múltiplos serviços de forma integrada e escalável. A integração com APIs externas, como a da OpenAI, ocorreu de maneira satisfatória, possibilitando a geração de conteúdo dinâmico e personalizado em tempo real. Embora a resposta do sistema dependa da latência de serviços externos, o desempenho permaneceu dentro de limites aceitáveis para uma interação conversacional fluida. Esses resultados validam a prova de conceito proposta, evidenciando a viabilidade de construir tecnologias assistivas inteligentes e escaláveis utilizando a combinação de serviços em nuvem, APIs de IA e arquitetura *serverless*, conforme detalhado nesta monografia.

8 CONCLUSÃO

Este trabalho partiu da lacuna observada na literatura sobre o desenvolvimento de tecnologias assistivas para crianças com TEA, evidenciada pela escassez de estudos e iniciativas relacionadas ao tema. Diante desse cenário, o objetivo deste trabalho foi registrar e documentar o desenvolvimento de um sistema de *chatbot* de apoio, elaborado durante um estágio supervisionado, que, por meio da aplicação de tecnologias modernas como computação em nuvem e IA, se configura como uma ferramenta inclusiva, acessível e eficaz, destinada a apoiar crianças com TEA em seu desenvolvimento e bem-estar.

A principal contribuição técnica do estudo reside na implementação de uma arquitetura *serverless*, que integra de forma robusta múltiplos serviços da AWS e APIs externas. O resultado é um sistema multimodal, capaz de permitir interações por meio de texto, voz e imagens, oferecendo uma experiência segura e alinhada às necessidades do usuário. Do ponto de vista educacional, a ferramenta inclui um recurso que favorece a expressão emocional, estimula a criatividade por meio da construção de narrativas e contribui para o desenvolvimento de habilidades cognitivas de forma lúdica e interativa.

É importante reconhecer as limitações inerentes ao contexto de desenvolvimento deste projeto. O cronograma de quatro semanas, embora intenso e produtivo, não permitiu a realização de testes de usabilidade formais com crianças com TEA, uma etapa fundamental para validar empiricamente o impacto pedagógico da solução. Da mesma forma, o feedback da instituição de ensino parceira, embora valioso, limitou-se a uma versão parcial do sistema. Tais limitações, contudo, não invalidam a prova de conceito e a robustez técnica alcançada, mas sim demarcam um caminho claro para a validação futura.

A partir da base sólida estabelecida, vislumbram-se diversas possibilidades de expansão. Entre elas, destacam-se a integração de análise de sentimentos em tempo real, a implementação de mecânicas de gamificação para ampliar o engajamento e a criação de um painel de monitoramento para pais e terapeutas.

Finalmente, este projeto representou uma oportunidade de aplicar na prática os conhecimentos adquiridos ao longo da graduação em disciplinas como desenvolvimento web, programação, inteligência artificial, computação em nuvem e engenharia de software, consolidando habilidades técnicas e demonstrando como diferentes áreas do conhecimento podem se articular para criar soluções de impacto social. Nesse sentido, o trabalho entrega não apenas uma solução tecnológica funcional e bem arquitetada, mas também um modelo replicável e uma prova de conceito do potencial da tecnologia voltada para a inclusão, oferecendo subsídios para o desenvolvimento de novas ferramentas que ampliem as oportunidades de aprendizado e desenvolvimento para todos.

REFERÊNCIAS

Amazon Web Services. *O que é o Scrum?* 2025. <https://aws.amazon.com/pt/what-is/scrum/>. Acesso em: 22 set. 2025. Citado na página 36.

American Psychiatric Association. *Diagnostic and Statistical Manual of Mental Disorders*. 5th. ed. Arlington, VA: American Psychiatric Publishing, 2013. Citado 2 vezes nas páginas 20 e 21.

ASPERGER, H. 'autistic psychopathy' in childhood. In: FRITH, U. (Ed.). *Autism and Asperger syndrome*. Cambridge University Press, 1991. p. 37–92. Original work published 1944. Disponível em: <https://cpb-us-e1.wpmucdn.com/blogs.uoregon.edu/dist/d/16656/files/2018/11/Asperger-Autistic-Psychopathy-in-Childhood-2h51vw4.pdf>. Citado na página 21.

Associação Brasileira de Autismo, Comportamento e Intervenção (ABRACI-DF). *Tratamento*. 2024. <https://abracidf.com/index.php/autismo/tratamento/>. Acessado em: 04 de julho de 2025. Citado na página 15.

BAILEY, A. et al. Autism as a strongly genetic disorder: evidence from a british twin study. *Psychological Medicine*, v. 25, n. 1, p. 63–77, 1995. Disponível em: <https://doi.org/10.1017/S0033291700028099>, Acesso em: 5 ago. 2025. Citado na página 21.

BETTELHEIM, B. *The Empty Fortress: Infantile Autism and the Birth of the Self*. [S.l.]: Free Press, 1967. Disponível em: <https://archive.org/details/emptyfortressinf0000bett>, Acesso em: 5 ago. 2025. Citado na página 21.

BLEULER, E. *Dementia Praecox or the Group of Schizophrenias*. International Universities Press, 1950. Tradução de 1911 disponível como PDF no Internet Archive; acesso em: 5.ago.2025. Disponível em: <https://archive.org/details/dementiapraecoxo0000bleu>. Citado na página 20.

CARNEIRO, M. A. B. *Jean Piaget e os estudos sobre o desenvolvimento humano*. s.d. <https://www4.pucsp.br/educacao/brinquedoteca/downloads/artigo-jean-piaget-e-os-estudos.pdf>. Artigo acadêmico. Acesso em: 10 jul. 2025. Citado na página 18.

Centers for Disease Control and Prevention. *Data & Statistics on Autism Spectrum Disorder*. 2023. <https://www.cdc.gov/mmwr/volumes/74/ss/ss7402a1.htm>. Acessado em: 04 de julho de 2025. Citado na página 15.

DANTAS, A. C. *Abordagem computacional para aprimoramento das habilidades com as emoções em indivíduos com autismo*. 2022. <https://repositorio.ufu.br/handle/123456789/35070>. Acessado em: 04 de julho de 2025. Citado 2 vezes nas páginas 16 e 22.

DINIZ, C. W. M. *Robótica socialmente assistiva no ensino de habilidades emocionais para crianças com TEA*. 2023. <https://repositorio.ufu.br/handle/123456789/39602>. Acessado em: 04 de julho de 2025. Citado na página 16.

GU, P. et al. Technological affordances and applications of chatbots for conversational skill interventions in autism: A scoping review. *Education and Information Technologies*, v. 30, n. 7, p. 9311–9340, 2025. Acesso em: 26 set. 2025. Disponível em: <https://doi.org/10.1007/s10639-024-13191-z>. Citado na página 26.

HAPPé, F.; RONALD, A.; PLOMIN, R. Time to give up on a single explanation for autism. *Nature Neuroscience*, v. 9, n. 10, p. 1218–1220, 2006. Disponível em: <https://doi.org/10.1038/nn1770>, Acesso em: 5 ago. 2025. Citado na página 21.

IBM Brasil. *Chatbots e agentes conversacionais*. 2025. <https://www.ibm.com/br-pt/think/topics/chatbots>. Acesso em: 22 ago. 2025. Citado 2 vezes nas páginas 22 e 26.

IBM Brasil. *O que é PLN (processamento de linguagem natural)?* 2025. <https://www.ibm.com/br-pt/think/topics/natural-language-processing>. Acesso em: 22 ago. 2025. Citado na página 26.

IBM Brasil. *O que é visão computacional?* 2025. <https://www.ibm.com/br-pt/think/topics/computer-vision>. Acesso em: 22 ago. 2025. Citado na página 27.

Instituto Brasileiro de Geografia e Estatística (IBGE). *Censo 2022 identifica 2,4 milhões de pessoas diagnosticadas com autismo no Brasil*. 2024. Agência de Notícias IBGE. Disponível em: <https://agenciadenoticias.ibge.gov.br/agencia-noticias/2012-agencia-de-noticias/noticias/43464-censo-2022-identifica-2-4-milhoes-de-pessoas-diagnosticadas-com-autismo-no-brasil>. Acessado em: 04 de julho de 2025. Citado 2 vezes nas páginas 15 e 17.

KANNER, L. Autistic disturbances of affective contact. *Nervous Child*, v. 2, p. 217–250, 1943. Versão PDF disponível em Spectrum News / Embryo Project Encyclopedia; acesso em: 5 ago. 2025. Disponível em: <https://www.spectrumnews.org/opinion/viewpoint/leo-kanners-1943-paper-on-autism/?format=pdf>. Citado na página 21.

LIGHT, J.; MCNAUGHTON, D. Communicative competence for individuals who require augmentative and alternative communication: A new definition for a new era of service delivery. *Augmentative and Alternative Communication*, v. 30, n. 1, p. 1–18, 2014. Disponível em: <https://doi.org/10.3109/07434618.2014.885080>. Citado na página 21.

MUELLER, R.-A. Neural integration of perceptual and motor processes in autism: Evidence from fmri and erp. *Progress in Neurobiology*, v. 82, n. 1, p. 1–22, 2007. Disponível em: <https://doi.org/10.1016/j.pneurobio.2007.02.005>, Acesso em: 5 ago. 2025. Citado na página 21.

NSAIF, W. S. et al. Conversational agents: An exploration into chatbot evolution, architecture, and important techniques. *Eurasia Proceedings of Science, Technology, Engineering & Mathematics (EPSTEM)*, v. 27, p. 246–262, 2024. Disponível em: <https://www.researchgate.net/publication/383147827_Conversational_Agents_An_Exploration_into_Chatbot_Evolution_Architecture_and_Important_Techniques>. Citado 3 vezes nas páginas 6, 24 e 25.

NVIDIA. *What Is Computer Vision?* 2025. <https://www.nvidia.com/en-gb/glossary/computer-vision/>. Acessado em: 22 de agosto de 2025. Disponível em: <https://www.nvidia.com/en-gb/glossary/computer-vision/>. Citado 2 vezes nas páginas 6 e 29.

OORD, A. van den et al. WaveNet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016. Acessado em 29 de julho de 2025. Disponível em: <https://arxiv.org/abs/1609.03499>. Citado na página 30.

PAPALIA DIANE E.; FELDMAN, R. D. *Desenvolvimento humano*. 12. ed.. ed. Porto Alegre: AMGH, 2013. Citado na página 18.

PINTO, R. N. M. Autismo infantil: impacto do diagnóstico e repercussões no desenvolvimento. *Revista Gaúcha de Enfermagem*, v. 37, n. 3, p. e57725, 2016. Acesso em: 26 set. 2025. Disponível em: <https://www.scielo.br/j/rgenf/a/Qp39NxcyXWj6N6DfdWWDDrR?utm_source=chatgpt.com>. Citado na página 26.

RITVO, E. R. et al. Concordance for the syndrome of autism in 40 pairs of affected twins. *American Journal of Psychiatry*, v. 142, n. 1, p. 74–77, 1985. Disponível em: <https://doi.org/10.1176/ajp.142.1.74>, Acesso em: 5 ago. 2025. Citado na página 21.

RUFINO, K. A. D. *Contribuições do jogo para a criança com Transtorno do Espectro Autista: perspectiva de Reuven Feuerstein*. 2020. <https://repositorio.ufu.br/handle/123456789/29299>. Acessado em: 04 de julho de 2025. Citado na página 16.

SERVICES, A. W. *AWS Documentation*. 2025. <https://docs.aws.amazon.com/>. Acesso em: 21 ago. 2025. Citado 3 vezes nas páginas 31, 32 e 33.

Serviço Federal de Processamento de Dados (Serpro). *Visão computacional: o que é e como funciona*. 2020–04–03. Disponível em: <https://www.serpro.gov.br/menu/noticias/noticias-2020/visao-computacional-oqueeh-comofunciona>. Citado 2 vezes nas páginas 6 e 28.

TAYLOR, P. *Text-to-Speech Synthesis*. [S.l.]: Cambridge University Press, 2009. Citado na página 29.

VALENTIM, N. A. *A inteligência artificial como recurso tecnológico na atenção compartilhada de crianças com Transtorno do Espectro Autista*. 2023. <https://repositorio.ufu.br/handle/123456789/41982>. Acessado em: 04 de julho de 2025. Citado 2 vezes nas páginas 16 e 21.

World Health Organization. *Autism*. 2023. <https://www.who.int/news-room/fact-sheets/detail/autism-spectrum-disorders>. Acessado em: 04 de julho de 2025. Citado 2 vezes nas páginas 15 e 20.

ZALLIO, M.; OHASHI, T. The evolution of assistive technology: A literature review of technology developments and applications. *arXiv*, 2022. Acesso em: 18 ago. 2025. Disponível em: <https://arxiv.org/abs/2201.07152>. Citado na página 30.

A GUIA PARA OBTENÇÃO DE CREDENCIAIS DE ACESSO

Este apêndice detalha os procedimentos para obter as chaves de acesso (API Keys) necessárias para o funcionamento dos serviços externos integrados ao sistema.

A.1 OpenAI API

A chave da OpenAI é utilizada para acessar os modelos de linguagem generativa (GPT), que são responsáveis pela criação das histórias no sistema. O processo de obtenção da chave segue passos simples e garante que o acesso à API seja seguro e controlado.

O procedimento é descrito a seguir:

1. Acesse a plataforma da OpenAI em `https://platform.openai.com/`.
2. Realize o login ou crie uma nova conta.
3. No menu lateral esquerdo, navegue até a seção **API keys**.
4. Clique no botão **"Create new secret key"**.
5. Dê um nome descritivo à sua chave e clique em **"Create secret key"**.
6. Copie a chave gerada imediatamente e armazene-a em um local seguro. **Atenção:** esta chave não será exibida novamente por motivos de segurança.

O fluxo ilustrativo desse procedimento está apresentado na Figura 59, evidenciando cada etapa desde o acesso à plataforma até a criação da chave de API.

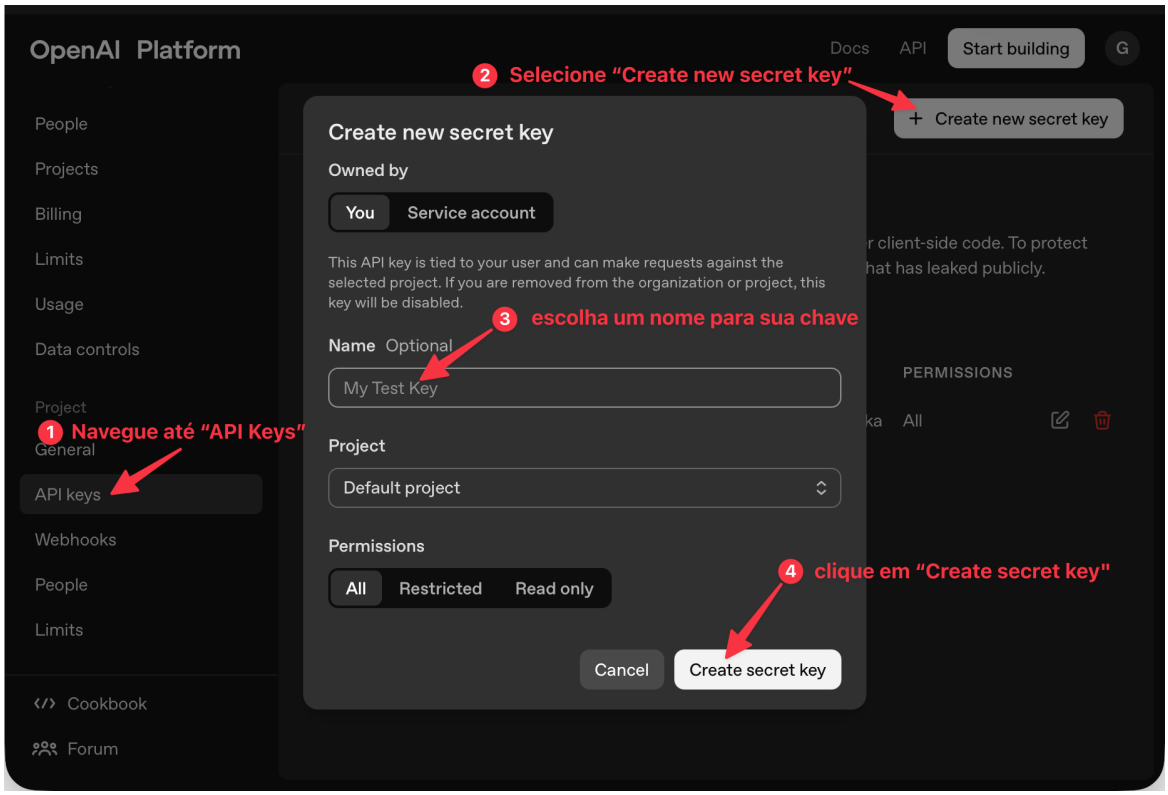

Figura 59 – Fluxo de criação de uma chave de API secreta na plataforma da OpenAI, destacando os passos de login, navegação até a seção de *API keys* e geração da chave. Fonte: Do Autor.

A.2 Microsoft Translator API (via RapidAPI)

O serviço de tradução do Microsoft Translator é consumido através da plataforma RapidAPI, que gerencia o acesso a diversas APIs.

1. Crie uma conta ou faça login na RapidAPI em `https://rapidapi.com/`.
2. Na barra de busca, procure por **"Microsoft Translator"**. O fluxo de busca está ilustrado na Figura 60.

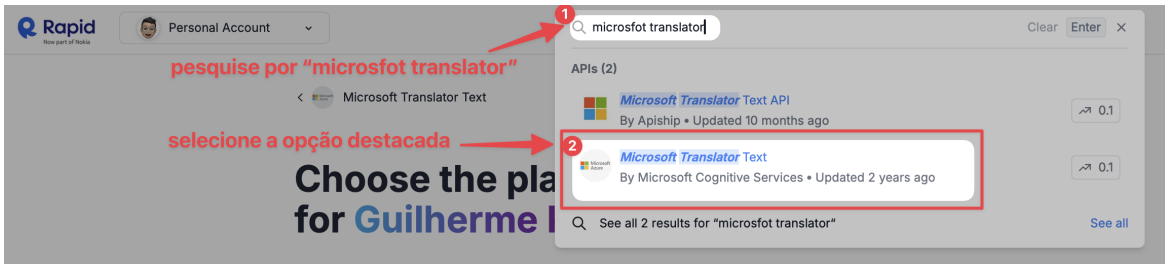

Figura 60 – Fluxo de busca da API Microsoft Translator na plataforma RapidAPI. Fonte: Do Autor.

3. Na página da API, selecione um plano de assinatura. Geralmente, há um plano básico (Basic) gratuito que inclui um número limitado de requisições mensais, suficiente para testes e desenvolvimento.

4. Após subscrever a um plano, navegue para a aba **Endpoints**.

5. No painel à direita, na seção **Code Snippets**, você encontrará sua chave de API no cabeçalho `X-RapidAPI-Key`. Copie e salve este valor em um local seguro. A Figura 61 ilustra onde a chave é exibida.

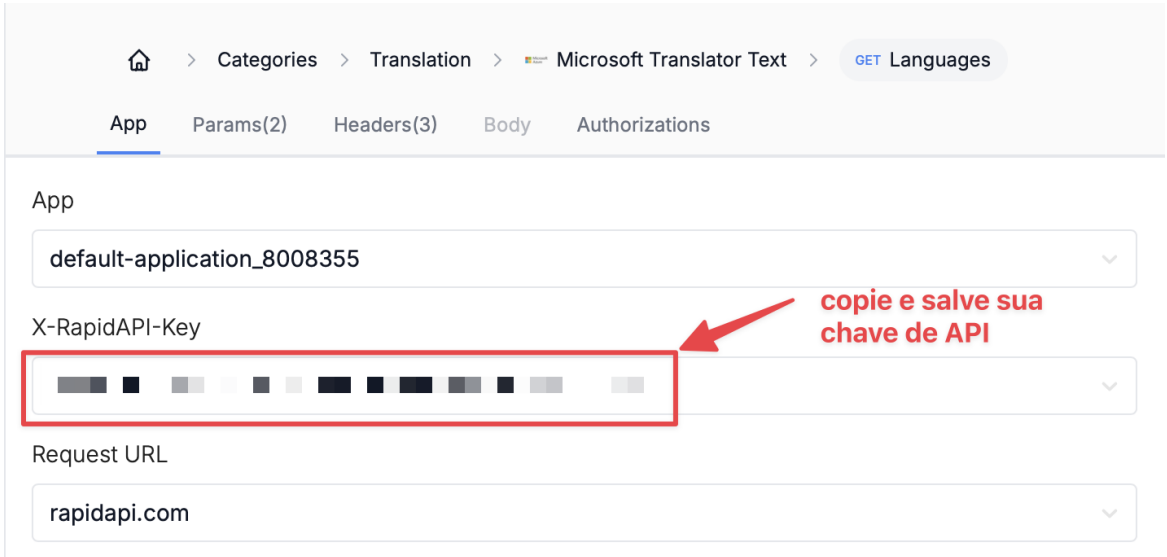

Figura 61 – Localização da chave de API no painel **Code Snippets** da RapidAPI. Fonte: Do Autor.

A.3 Kommunicate.io

A plataforma Kommunicate.io fornece a interface web do *chatbot* e gerencia a integração com o usuário final. As seções a seguir detalham o processo completo de

configuração.

A.3.1 INTEGRAÇÃO COM AMAZON LEX

Esta etapa detalha como conectar a plataforma ao bot Amazon Lex utilizando as credenciais do usuário IAM, criadas conforme descrito na Seção 7.5.1.

1. Acesse o *dashboard* do Kommunicate em <https://dashboard.kommunicate.io/> e realize o login.

2. No menu lateral, navegue até a seção **Bot Integration** e selecione a opção **Amazon Lex** para iniciar a configuração, conforme ilustrado na Figura 62.

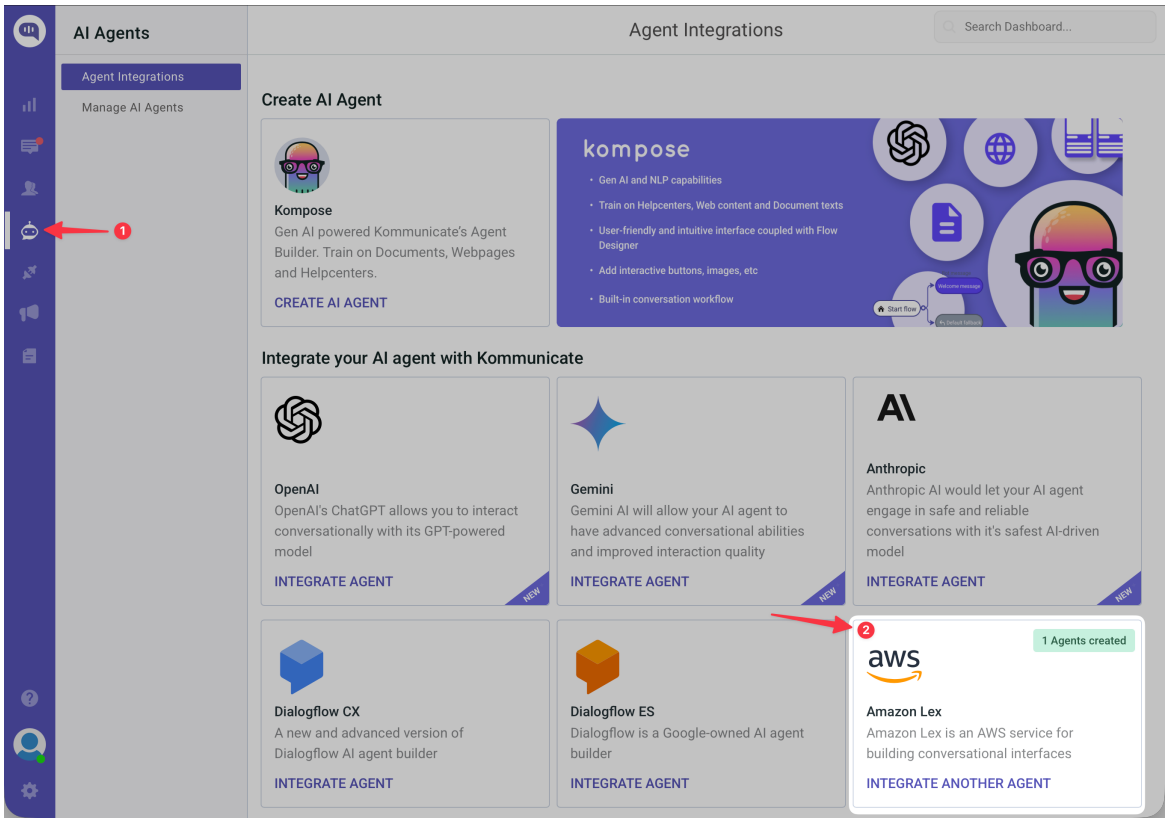

Figura 62 – Configuração de um agente para integração da plataforma Kommunicate com o Amazon Lex. Fonte: Do Autor.

3. Na tela de configuração (Figura 63), preencha os seguintes parâmetros de integração:

   - **Access key ID:** Insira o ID da chave de acesso do usuário IAM dedicado.
   - **Secret access key:** Insira a chave de acesso secreta correspondente.
   - **Region:** Selecione a região `US East (N. Virginia)`.

- **IA agent name in Lex platform:** Insira o nome do seu bot no Amazon Lex.
- **IA agent alias:** Selecione o alias do bot a ser utilizado.
- **Default IA agent language:** Selecione o idioma configurado.

Figura 63 – Tela de configuração dos parâmetros de integração no Kommunicate. Fonte: Do Autor.

4. Após preencher todos os campos, clique em **"Save and proceed"** para finalizar a integração.

A.3.2 INSTALAÇÃO DO *WIDGET* NO WEBSITE

Após a integração com o Lex, o passo final é instalar o *widget* de chat na aplicação web para que os usuários possam interagir com o bot, conforme ilustrado na Figura 64.

1. No *dashboard* do Kommunicate, clique no ícone de engrenagem (**Settings**) no canto inferior do menu lateral.
2. Navegue até a seção **INSTALL** e clique em **Install**.
3. Na tela de instalação, localize o campo com o script JavaScript e copie todo o bloco de código.
4. Abra o arquivo `index.html` do *frontend* da sua aplicação.
5. Cole o script copiado antes do fechamento da tag `</body>`.

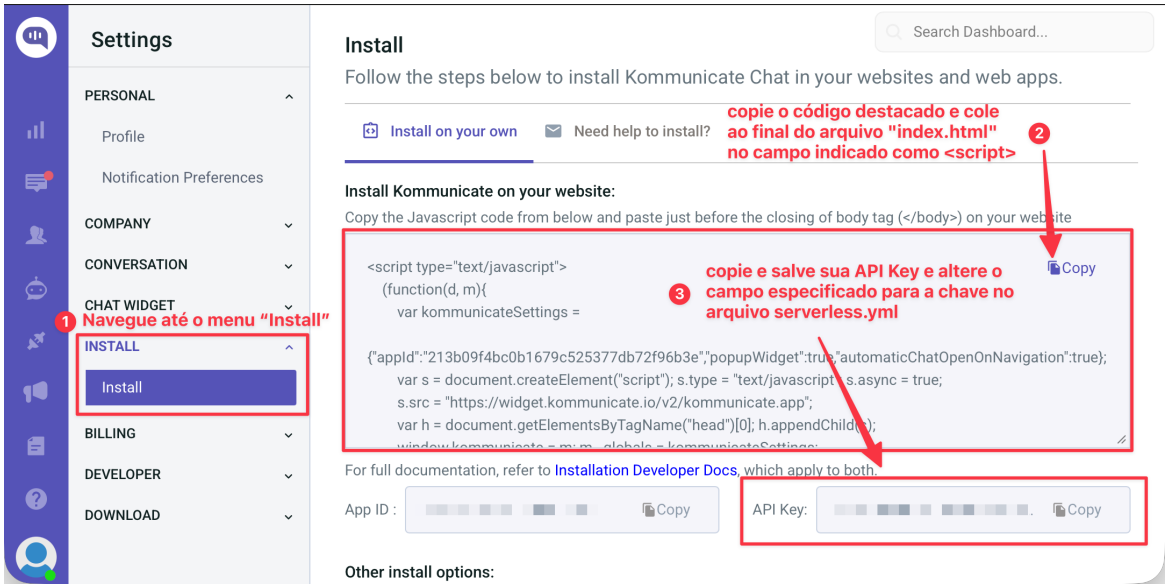

Figura 64 – Processo de instalação do *widget* de chat do Kommunicate. Fonte: Do Autor