Universidade de Brasília – UnB
Faculdade UnB Gama – FGA
Engenharia de Software

Modelagem de fluxos de uma aplicação voltada para a educação jurídica

Autor: Guilherme Rogelin Vial e Lucas Braun Vieira Xavier
Orientador: Prof. Dr. André Luiz Peron Martins Lanna

Brasília, DF
2024

Guilherme Rogelin Vial e Lucas Braun Vieira Xavier

Modelagem de fluxos de uma aplicação voltada para a educação jurídica

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília – UnB

Faculdade UnB Gama – FGA

Orientador: Prof. Dr. André Luiz Peron Martins Lanna

Brasília, DF

2024

Guilherme Rogelin Vial e Lucas Braun Vieira Xavier
Modelagem de fluxos de uma aplicação voltada para a educação jurídica/ Guilherme Rogelin Vial e Lucas Braun Vieira Xavier. – Brasília, DF, 2024-
60 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. André Luiz Peron Martins Lanna

Trabalho de Conclusão de Curso – Universidade de Brasília – UnB
Faculdade UnB Gama – FGA , 2024.

1. Modelagem de Fluxo. 2. Educação Jurídica. I. Prof. Dr. André Luiz Peron Martins Lanna. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Modelagem de fluxos de uma aplicação voltada para a educação jurídica

CDU 02:141:005.6

Guilherme Rogelin Vial e Lucas Braun Vieira Xavier

Modelagem de fluxos de uma aplicação voltada para a educação jurídica

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Trabalho aprovado. Brasília, DF, 20 de Setembro de 2024 – Data da aprovação do trabalho:

Prof. Dr. André Luiz Peron Martins Lanna
Orientador

Prof. Me. Cristiane Soares Ramos
Convidado 1

Prof. Me. Ricardo Ajax Dias Kosloski
Convidado 2

Brasília, DF
2024

Resumo

Este trabalho apresenta o desenvolvimento de uma solução tecnológica destinada a promover a modelagem e distribuição de fluxos de navegação voltados à educação jurídica, utilizando o Google Forms como interface de construção. A proposta permite que profissionais da área jurídica modelem formulários com lógica de navegação condicional, que direciona o usuário a diferentes seções conforme suas respostas. Os fluxos gerados são exportados em formato JSON, tornando-os compatíveis com outras plataformas e sistemas externos. A aplicação tem como objetivo contribuir para a disseminação do conhecimento jurídico, promovendo um maior entendimento dos direitos e deveres da população de maneira acessível e automatizada. Além disso, o uso de serviços em nuvem garante a automação e escalabilidade da solução.

Palavras-chave: Modelagem de fluxo. Educação jurídica. Google Forms. Computação em Nuvem.

Abstract

This work presents the development of a technological solution aimed at promoting the modeling and distribution of navigation flows focused on legal education, using Google Forms as a construction interface. The proposal allows legal professionals to model forms with conditional navigation logic, which directs the user to different sections according to their answers. The generated flows are exported in JSON format, making them compatible with other platforms and external systems. The application aims to contribute to the dissemination of legal knowledge, promoting a greater understanding of citizens' rights and duties in an accessible and automated way. Furthermore, the use of cloud services ensures the automation and scalability of the solution.

Keywords: Flow modeling. Legal education. Google Forms. Cloud Computing.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

TCC	Trabalho de Conclusão de Curso
UnB	Universidade de Brasília
FGA	Faculdade do Gama
USP	Universidade de São Paulo
XP	*Extreme Programming*
JSON	*JavaScript Object Notation*
AWS	*Amazon Web Services*
ECR	*Elastic Container Registry*
API	*Application Programming Interface*
URL	*Uniform Resource Locator*
TBD	*Trunk-Based Development*
REST	*Representational State Transfer*
TI	Tecnologia da Informação
IaaS	*Infrastructure as a Service*
PaaS	*Platform as a Service*
Saas	*Software as a Service*
FaaS	*Function as a Service*
HTTP	*Hypertext Transfer Protocol*

Sumário

1 Introdução

1.1 Contextualização

O sistema jurídico brasileiro é regido pela máxima “Ninguém se escusa de cumprir a lei, alegando que não a conhece”(BRASIL, 1942). Essa premissa, fundamental para o ordenamento jurídico, estabelece que o simples desconhecimento das normas não constitui justificativa válida para a exclusão de pena ou responsabilidade. No entanto, essa noção pressupõe que as pessoas tenham um nível básico de acesso à informação jurídica, o que nem sempre é uma realidade no Brasil (DATASENADO, 2013). Para além do cumprimento das normas legais, é essencial que os cidadãos compreendam profundamente seus direitos e deveres, evitando que vivam alheios a essas responsabilidades como membros da sociedade.

Indivíduos bem informados juridicamente, por outro lado, encontram-se em posição privilegiada para integrar-se de maneira mais ativa e efetiva na vida política e social do país. A informação jurídica lhes confere não apenas o conhecimento necessário para o cumprimento de obrigações legais, mas também o poder de reivindicar seus direitos de forma crítica e reflexiva. Uma população educada em direitos é capaz de identificar e resistir a abusos de poder, bem como combater práticas de exclusão social e econômica perpetuadas por aqueles que detêm o controle político e financeiro do país. Como apontam Silva, Choucino e Machado (2019), o conhecimento jurídico permite uma postura crítica diante das injustiças sociais, capacitando os cidadãos a questionar estruturas que favorecem grupos privilegiados.

No entanto, o cenário atual revela uma realidade preocupante. Segundo pesquisa realizada pelo DataSenado (2013), a maioria dos brasileiros possui apenas um conhecimento mediano da Constituição Federal (50,8%), seguido por um número significativo de pessoas com conhecimento baixo (35,1%), nenhum conhecimento (7,8%) e apenas uma pequena parcela com conhecimento elevado (5,3%). Esses dados reforçam a urgência de iniciativas voltadas à educação jurídica acessível para toda a população, de modo a empoderar cidadãos e promover uma maior equidade no acesso à justiça.

1.2 Problema e Contexto da Solução

A Faculdade de Direito da Universidade de São Paulo (USP) identificou a necessidade de novas abordagens para disseminar o conhecimento jurídico de forma acessível e eficiente. O desafio encontrado foi a falta de ferramentas tecnológicas adaptadas para o ensino jurídico, capazes de facilitar o aprendizado e a compreensão das leis. Para atender

a essa demanda, surgiu uma parceria com o curso de Engenharia de Software da Universidade de Brasília (UnB), visando desenvolver uma solução tecnológica voltada para esse propósito.

A solução proposta visa permitir que profissionais da área criem fluxos de navegação baseados em perguntas e respostas. Esses fluxos serão transformados em um formato apropriado para integração com uma aplicação *mobile*, também desenvolvida pela Universidade de Brasília. Dessa forma, o sistema permite a criação de conteúdos interativos de ensino sem a necessidade de conhecimento técnico avançado, ampliando o acesso ao aprendizado jurídico de forma dinâmica e acessível.

1.3 Impactos da Solução

A criação dessa plataforma tem o potencial de disseminar o conhecimento jurídico de maneira ampla e significativa. Ao integrar tecnologia e educação jurídica de maneira inclusiva, a solução pode ser uma ferramenta valiosa para profissionais da área jurídica e o público em geral, permitindo que o aprendizado seja acessível a um número maior de pessoas, de forma dinâmica e interativa.

A tecnologia tem desempenhado um papel fundamental na melhoria do acesso ao conhecimento jurídico, democratizando o acesso à informação e ampliando as possibilidades de educação jurídica. De acordo com o *Harvard Journal of Law & Technology*, as ferramentas digitais têm permitido que informações jurídicas sejam disponibilizadas de forma mais acessível e compreensível para o público em geral. Plataformas digitais, por exemplo, permitem que cidadãos sem formação jurídica acessem conteúdos simplificados sobre seus direitos e obrigações, facilitando a compreensão de questões legais complexas, contribuindo para uma disseminação mais ampla e eficiente do conhecimento jurídico. Essas inovações não apenas aumentam o alcance do conteúdo jurídico, mas também tornam o processo mais inclusivo e eficiente, especialmente para populações que anteriormente tinham dificuldades em acessar tais informações (CABRAL et al., 2012).

1.4 Objetivos

1.4.1 Objetivo Geral

Desenvolver uma solução tecnológica voltada à criação e disseminação de fluxos de perguntas e respostas para a educação jurídica, visando ampliar o acesso ao conhecimento sobre direitos e deveres dos cidadãos de maneira acessível e eficaz.

1.4.2 Objetivos específicos

- **Desenvolver documentos de arquitetura utilizando FaaS:**
  - Definir a arquitetura da solução com foco em serviços *serverless* (FaaS) para garantir escalabilidade e baixo custo operacional.
- **Desenvolver o código da aplicação:**
  - Implementar autenticação com a API do Google através de credenciais de conta de serviço.
  - Listar os formulários pela API do Google Drive.
  - Coletar os formulários pela API do Google Forms.
  - Adaptar o formulário obtido para o formato necessário.
  - Desenvolver função *handler* para o AWS Lambda que orquestre e execute todas as etapas do processo em resposta a um evento.
  - Criar imagem Docker contendo todas as dependências e configurações necessárias para execução do código.
- **Configurar e utilizar os serviços do AWS:**
  - Criar uma conta AWS.
  - Criar e configurar o AWS Lambda: Utilizar o AWS Lambda para processar os dados do Google Forms convertidos em JSON.
  - Usar URL de função no AWS Lambda: Configurar uma URL dedicada para a função Lambda, permitindo o acesso por reqisições HTTP.
  - Configurar o AWS Parameter Store: Armazenar e gerenciar de forma segura as credenciais necessárias para o uso da API do Google.
  - Configurar o Amazon ECR (Elastic Container Registry): Configurar repositórios para armazenar imagens de contêineres usadas pelo AWS Lambda.
- **Configurar o Google Cloud Console e as APIs necessárias:**
  - Criar um novo projeto no Google Cloud Console.
  - Criar conta de serviço e gerar as credenciais.
  - Ativar as APIs do Google Forms e Google Drive.
  - Configurar a autenticação OAuth 2.0: Implementar o processo de autenticação e autorização para acessar os dados do Google Forms de forma segura.

1.5 Estrutura do Texto

Capítulo 2 - Referencial Teórico: Apresenta os conceitos chave e a fundamentação teórica essencial para a compreensão do projeto.

Capítulo 3 - Suporte Tecnológico: Detalha as escolhas tecnológicas feitas, justificando o uso no desenvolvimento da solução.

Capítulo 4 - Metodologia: Descreve os métodos e processos utilizados para a execução do projeto.

Capítulo 5 - Resultados: Expõe os resultados obtidos, documentando a solução, com exemplos práticos de aplicação.

Capítulo 6 - Considerações Finais: Discute as conclusões e sugere possíveis direções para trabalhos futuros baseadas nos resultados obtidos.

2 Referencial Teórico

2.1 Engenharia de *Software*

A engenharia de *software* é definida como a aplicação de princípios de engenharia para o desenvolvimento de *software* que seja economicamente viável, confiável e eficiente quando executado em máquinas reais (BAUER; BAUER, 1977). Além disso, essa disciplina abrange todas as fases do ciclo de vida de um sistema, desde a engenharia de requisitos até o *design*, a implementação, os testes e a manutenção do *software* (BOEHM, 1976). O objetivo central da engenharia de *software* é garantir que o sistema desenvolvido atenda aos requisitos de qualidade, prazo e custo, maximizando a eficiência e minimizando erros ao longo do processo.

A Engenharia de *software* abrange um conjunto de processos, métodos e ferramentas que são fundamentais para a construção de sistemas complexos baseados em computador, garantindo que esses sistemas sejam desenvolvidos dentro do prazo e com a qualidade necessária. Esses processos são estruturados em cinco atividades principais: comunicação, planejamento, modelagem, construção e emprego (PRESSMAN, 2011).

- Comunicação: Compreender os objetivos das partes interessadas e levantar as necessidades para definir funções e características do *software*.
- Planejamento: Criação de um "mapa" para guiar a equipe ao longo do projeto.
- Modelagem: Criação de "esboços" ou modelos para visualizar a arquitetura do *software* e como suas partes interagem.
- Construção: Envolve a escrita de código.
- Emprego: Entrega do *software* ao cliente, seja como uma entidade completa ou em incrementos parciais.

2.2 Desenvolvimento Ágil de Software

O método tradicional de desenvolvimento de *software* começa com a elicitação e documentação de um conjunto completo de requisitos, seguido pelas fases de design arquitetônico, desenvolvimento e inspeção. No entanto, a partir da década de 1990, muitos profissionais começaram a perceber que essa abordagem era frustrante e, em muitos casos, inviável, devido à rápida evolução da tecnologia e das necessidades do mercado. As mudanças nos requisitos ocorrem em uma velocidade que os métodos tradicionais não

conseguem acompanhar, e os clientes, muitas vezes, não conseguem definir claramente suas necessidades logo no início do projeto. Esse cenário impulsionou a busca por alternativas mais dinâmicas, como os métodos ágeis, desenvolvidos para lidar com a constante mudança e responder melhor às expectativas dos clientes (COHEN; LINDVALL; COSTA, 2003).

Foi nesse contexto de insatisfação com os métodos tradicionais que, em 2001, um grupo de 17 especialistas em metodologias de *software* se reuniu para criar o Manifesto Ágil. Esse documento consolidou um conjunto de princípios que promove uma abordagem mais flexível e adaptável ao desenvolvimento de *software*, refletindo a necessidade de respostas rápidas e eficazes às mudanças frequentes no mercado (BECK et al., 2001).

O desenvolvimento ágil de *software*, baseado nos princípios do Manifesto Ágil, valoriza a flexibilidade, a colaboração e a entrega contínua de valor ao cliente. A metodologia ágil prioriza indivíduos e interações sobre processos e ferramentas, *software* em funcionamento sobre documentação extensa, colaboração com o cliente sobre negociações contratuais, e adaptação a mudanças sobre o cumprimento rígido de um plano. Com essa abordagem, as equipes ágeis conseguem se ajustar rapidamente às necessidades em constante mudança, tornando o processo de desenvolvimento mais eficiente e responsivo ao longo do tempo (BECK et al., 2001).

2.2.1 *Extreme Programming* - XP

O *Extreme Programming* (XP) é uma metodologia ágil de desenvolvimento de *software* introduzida nos anos 1990 por Kent Beck, com o objetivo de melhorar a capacidade das equipes de desenvolvimento de responder rapidamente a mudanças nos requisitos do cliente. Em um cenário onde projetos de *software* frequentemente falhavam devido à rigidez dos métodos tradicionais, como o modelo cascata, o XP propôs um enfoque mais iterativo e colaborativo, priorizando a entrega contínua de valor ao cliente. Um dos elementos centrais do XP é o uso do *backlog*, uma lista priorizada de funcionalidades e tarefas que orienta o desenvolvimento e garante que os itens mais importantes sejam entregues primeiro. Esse *backlog* é continuamente atualizado conforme o *feedback* do cliente é incorporado, o que permite que o XP responda rapidamente às mudanças de requisitos. Ao contrário de metodologias mais rígidas, o XP incentiva a interação constante entre desenvolvedores e clientes, garantindo que o *feedback* seja integrado rapidamente ao processo de desenvolvimento (BECK, 2000).

O XP é guiado por cinco valores fundamentais: comunicação, simplicidade, *feedback*, coragem e respeito. A comunicação é essencial para garantir que todos os membros da equipe e os *stakeholders*estejam alinhados. A simplicidade incentiva o desenvolvimento de soluções que atendam aos requisitos atuais, sem adições desnecessárias de funcionalidades ou complexidade.O *feedback* rápido, tanto dos clientes quanto da equipe de desenvolvi-

mento, permite ajustes constantes, enquanto a coragem e o respeito são incentivados para que as equipes se sintam seguras para inovar e sugerir mudanças quando necessário.

2.2.2 *Trunk-Based Development*

O *Trunk-Based Development* (TBD) é uma abordagem de controle de versão em que todos os desenvolvedores integram seu código em uma única *branch*, ou seja, uma ramificação de desenvolvimento. Essa *branch* principal é geralmente chamada de *trunk* ou *main*, ao contrário de outras metodologias que utilizam várias *branches* de longa duração para o desenvolvimento de diferentes funcionalidades (HAMMANT, 2020).

No *Trunk-Based Development*para times pequenos, cada desenvolvedor, ou uma dupla de programação em par, envia pequenos *commits*. Um *commit* é o registro de uma alteração no código, que, neste contexto, é enviado diretamente para a *branch* principal. Antes de integrar esses *commits*, é realizada uma etapa de pré-integração, onde o sistema verifica se o *build* é bem-sucedido. O termo *build* refere-se ao processo de transformar o código-fonte em um programa executável, o que pode envolver compilar (converter o código de uma linguagem de programação em um formato que o computador possa entender) e rodar, garantindo que o código funcione corretamente (HAMMANT, 2020).

2.3 *Cloud Computing*

A computação em nuvem, ou *cloud computing*, é um modelo de fornecimento de recursos de tecnologia da informação (TI) sob demanda como armazenamento, poder de processamento e plataformas de desenvolvimento, acessados pela *internet*. Em vez de gerenciar e manter servidores físicos e infraestruturas de TI internamente, as organizações podem alocar e pagar apenas pelos recursos computacionais que realmente utilizam, de forma escalável e flexível. Esse modelo oferece diversas vantagens, como a redução de custos com infraestrutura, maior agilidade na implementação de soluções e facilidade de escalabilidade (AWS, 2023b).

Existem três principais modelos de serviço em *cloud computing*: IaaS (*Infrastructure as a Service*), PaaS (*Platform as a Service*) e SaaS (*Software as a Service*). O IaaS fornece os componentes fundamentais da infraestrutura de TI na nuvem, como recursos de rede, servidores (virtuais ou físicos) e armazenamento de dados, oferecendo o maior nível de flexibilidade e controle sobre os recursos, sendo semelhante aos sistemas de TI tradicionais. No PaaS, o foco está na implantação e no gerenciamento de aplicativos, sem a necessidade de gerenciar a infraestrutura subjacente, como *hardware* e sistemas operacionais, aumentando a eficiência ao eliminar tarefas repetitivas, como manutenção e atualizações. Já o SaaS oferece produtos completos diretamente ao usuário final, sem que o usuário precise se preocupar com a infraestrutura ou manutenção, concentrando-se

exclusivamente na utilização do *software* (AWS, 2023b).

2.3.1 *Serverless Computing*

Serverless computing, ou computação sem servidor, é um modelo de execução em nuvem onde o provedor de serviços gerencia a infraestrutura necessária para executar o código, permitindo que os desenvolvedores concentrem-se apenas na lógica da aplicação. Diferente dos modelos tradicionais de hospedagem, onde os desenvolvedores precisam configurar, gerenciar e escalar servidores, na computação *serverless* o provedor de nuvem lida automaticamente com esses aspectos, eliminando a necessidade de gerenciamento direto de servidores (IBM, 2023).

A principal característica do modelo *serverless* é que o código é executado sob demanda, em resposta a eventos, e o usuário só paga pelos recursos computacionais utilizados durante a execução dessas funções, em vez de pagar por capacidade ociosa. Esse modelo é altamente escalável, uma vez que a nuvem ajusta automaticamente os recursos para atender a variações na demanda, sem intervenção manual (IBM, 2023).

Uma variação de *serverless computing* é o FaaS, ou *Function-as-a-Service* (função como serviço), que permite aos desenvolvedores executar trechos de código em resposta a eventos, sem a necessidade de gerenciar a infraestrutura subjacente, como servidores ou sistemas operacionais (IBM, 2024).

2.4 API (*Application Programming Interface*)

API, abreviatura de *Application Programming Interface*, compreende um conjunto de ferramentas, definições e protocolos usados no desenvolvimento de aplicações de *software*. Essas interfaces facilitam a interação entre diferentes soluções e serviços, permitindo a integração sem que haja a necessidade de entender os detalhes internos de como esses sistemas são implementados (HAT, 2023).

Os *endpoints* de uma API representam os locais específicos dentro do sistema de comunicação onde as interações ocorrem. Esses pontos de contato incluem URLs que apontam para servidores ou serviços específicos, e são os locais a partir dos quais as informações são tanto enviadas quanto recebidas entre diferentes sistemas (AWS, 2023a).

2.4.1 API REST

APIs REST, ou *Representational State Transfer*, são atualmente as interfaces de programação de aplicações mais populares e flexíveis encontradas na *web*. Esse modelo de API opera com o cliente enviando solicitações ao servidor na forma de dados. O servidor, por sua vez, processa essas entradas para executar funções internas e, em seguida, retorna

os dados de saída ao cliente. Essa troca de dados é feita utilizando o protocolo HTTP, que é o Protocolo de Transferência de Hipertexto utilizado para enviar e receber dados na *web* (AWS, 2023a).

3 Suporte Tecnológico

Este Capítulo tem como objetivo descrever os aspectos técnicos referentes ao desenvolvimento da solução.

3.1 Ferramentas

3.1.1 Google Forms

Google Forms é uma ferramenta de criação de formulários *online* fornecida pelo Google dentro de seu conjunto de aplicativos. Essa ferramenta permite que os usuários desenvolvam formulários versáteis que suportam diversos tipos de perguntas. Um de seus recursos mais notáveis é a lógica condicional, que ajusta as perguntas exibidas com base nas respostas anteriores dos usuários (GOOGLE, 2023).

Para o uso no projeto, várias ferramentas de criação de formulários foram avaliadas com base em três critérios principais: gratuidade, capacidade de exportação para formato JSON e suporte a fluxos condicionais. Esses fatores foram essenciais na escolha da plataforma mais adequada, garantindo que a ferramenta atendesse às necessidades de criação dinâmica de fluxos de perguntas e respostas, permitindo a integração com outras aplicações e mantendo a viabilidade do projeto sem custos adicionais. A possibilidade de desenvolver uma aplicação *web* própria para a modelagem de fluxogramas também foi considerada. No entanto, essa alternativa foi descartada como desnecessária, uma vez que já existem outras ferramentas disponíveis que satisfazem plenamente essa necessidade.

Ferramenta	**Grátis**	**Exportável para JSON**	**Navegação Condicional**
Google Forms	Sim	Sim	Sim
Microsoft Forms	Sim	Não	Sim
Typeform	Sim (limitado)	Sim	Sim
SurveyMonkey	Sim (limitado)	Sim	Sim
Zoho Forms	Sim (limitado)	Sim	Sim
Form.io	Não	Sim	Sim

Tabela 1 – Funcionalidades das ferramentas de formulários avaliadas

A Tabela 1 compara as ferramentas de criação de formulários avaliadas de acordo com os critérios estabelecidos, destacando que o Google Forms é a única que oferece as funcionalidades desejadas de forma completa e gratuita. Além disso, sua ampla adoção

e reconhecimento tornam a ferramenta familiar para muitos usuários, reduzindo a curva de aprendizado e facilitando a integração e a colaboração entre diferentes equipes. A capacidade de exportar dados de maneira eficiente através de sua API também se mostra um grande diferencial, permitindo uma integração simples com outros sistemas, essencial para a automação dos processos.

3.1.2 Google Drive

O Google Drive é uma plataforma de armazenamento em nuvem que permite aos usuários salvar arquivos de diversos tipos, incluindo formulários do Google Forms. Além de armazenar, o Google Drive oferece a funcionalidade de compartilhamento, possibilitando que os arquivos sejam facilmente acessados e colaborados por outras contas Google (GOOGLE, 2024). Essa funcionalidade é essencial para o projeto, pois permite identificar quais formulários podem ser acessados pelo sistema.

3.1.3 Git e GitHub

O Git é um sistema de controle de versão distribuído, amplamente utilizado para gerenciar o histórico de alterações em arquivos de um projeto à medida que equipes ou colaboradores individuais trabalham coletivamente. Com o Git, cada desenvolvedor tem uma cópia completa do repositório em sua máquina local, o que permite que ele faça mudanças de forma independente e sincronize essas mudanças com outros colaboradores. Uma das principais características do Git é sua capacidade de criar *branches*, ou ramificações, que permitem o desenvolvimento paralelo de novas funcionalidades ou correções sem afetar o código principal. Isso garante flexibilidade e segurança ao desenvolvimento, permitindo a integração de código apenas quando ele está totalmente testado e funcional (GITHUB, 2023).

O GitHub, por sua vez, é uma plataforma *web* que hospeda repositórios Git, visando facilitar a colaboração em projetos (GITHUB, 2023). Além de armazenar e versionar o código, o GitHub oferece uma gama de funcionalidades adicionais que aprimoram o processo de desenvolvimento, como o rastreamento de problemas (*issues*), onde os colaboradores podem relatar e acompanhar *bugs* ou novas funcionalidades. O sistema de *pull requests* também é uma ferramenta essencial do GitHub, permitindo que as alterações feitas em uma *branch* sejam revisadas e discutidas antes de serem integradas ao código principal do projeto (GITHUB, 2024).

3.1.4 Docker

Docker é uma plataforma de código aberto projetada para facilitar a criação, implementação e execução de aplicativos utilizando contêineres. Contêineres são unidades

leves e portáteis que incluem tudo o que é necessário para executar um *software*, como o código, as bibliotecas e as dependências. Eles isolam o aplicativo do ambiente em que estão sendo executados, garantindo que ele funcione de maneira consistente, independentemente de onde esteja sendo executado. Isso resolve um dos maiores desafios no desenvolvimento de *software*: garantir que o código funcione da mesma forma em diferentes ambientes, como na máquina de desenvolvimento, no servidor de testes e em produção (DOCKER, 2020).

Uma das principais vantagens do Docker é sua capacidade de criar ambientes reproduzíveis, o que facilita a colaboração entre equipes de desenvolvimento e operações. O conceito de conteinerização promove o isolamento dos aplicativos, permitindo que múltiplos contêineres sejam executados em uma única máquina sem conflito de dependências. Isso os torna ideais para cenários de integração e entrega contínuas, acelerando o ciclo de desenvolvimento e a implantação de novos serviços (DOCKER, 2020).

Para criar um contêiner, é necessário primeiro definir sua imagem. A imagem do Docker é como uma "fotografia" do ambiente de execução do aplicativo, contendo todas as configurações e dependências necessárias. Essa imagem é construída a partir de um arquivo chamado *Dockerfile*, que define, por meio de uma série de comandos, a configuração do ambiente, a instalação de dependências, a cópia do código-fonte do aplicativo e outras instruções necessárias para que o programa seja executado dentro do contêiner. Após a criação da imagem, ela pode ser usada para gerar múltiplos contêineres idênticos, garantindo consistência na execução do aplicativo (DOCKER, 2020).

No escopo do projeto, o Docker é empregado para assegurar a consistência entre os ambientes de desenvolvimento e produção, facilitando assim o gerenciamento de dependências e promovendo o isolamento. Além disso, o AWS Lambda suporta o uso de imagens Docker, garantindo que o ambiente seja padronizado e que a função Lambda execute com todas as dependências necessárias, independentemente das variações na infraestrutura subjacente.

3.1.5 JSON

JSON (*JavaScript Object Notation* - Notação de Objetos JavaScript) é um formato de intercâmbio de dados leve, amplamente utilizado para representar dados estruturados de maneira legível para humanos e fácil de ser interpretada por máquinas. Embora tenha suas raízes na linguagem JavaScript, o JSON é independente de linguagem, o que significa que pode ser utilizado em praticamente qualquer linguagem de programação moderna, como Python, Java, entre outras (JSON.ORG, 2023).

As duas estruturas que o JSON representa são coleções de pares nome/valor e listas ordenadas de valores (também chamadas de *arrays*). Essas estruturas são universais, o

que facilita a adoção do JSON como formato de intercâmbio de dados (JSON.ORG, 2023). Por exemplo, em uma API REST, as requisições e respostas geralmente são trocadas nesse formato, permitindo que dados sejam facilmente serializados e desserializados em ambas as pontas da comunicação.

A **serialização** é o processo de conversão de dados em um formato que pode ser facilmente armazenado ou transmitido, ideal para a troca de informações entre sistemas distintos. A **desserialização** é o processo inverso, no qual os dados serializados são convertidos de volta para um formato utilizável, reconstruindo a informação original.

No projeto, o formato JSON é utilizado para serializar os dados das seções e itens do formulário Google, convertendo-os em um formato apropriado para envio através da função Lambda.

3.1.6 Google Cloud - Conta de serviço

As contas de serviço são um tipo de identidade especial dentro do Google Cloud - plataforma de serviços de computação em nuvem do Google - projetadas para serem utilizadas por aplicações ou máquinas ao invés de usuários humanos. Elas permitem que programas ou serviços externos autentiquem-se e interajam com as APIs do Google Cloud de forma segura e controlada. As contas de serviço são essenciais em cenários onde aplicações precisam acessar recursos do Google Cloud, como o Google Drive ou o Google Forms, sem a necessidade de autenticação manual por um usuário (CLOUD, 2024).

O Google Cloud é utilizado para a criação e gerenciamento de um projeto relacionado à aplicação que serve como o ambiente de integração com as APIs necessárias. Dentro desse projeto, as principais ações são:

- Criação de uma conta de serviço: A aplicação precisará de uma conta de serviço para realizar chamadas autenticadas às APIs do Google.
- Ativação das APIs do Google Drive e Google Forms: Para que a aplicação possa capturar a lista de formulários existentes, será necessário ativar a API do Google Drive e, para acessar um formulário específico, a API do Google Forms.

3.1.7 Amazon Web Services - AWS

A AWS (Amazon Web Services) é uma plataforma de computação em nuvem amplamente reconhecida por fornecer suporte tecnológico robusto e escalável para empresas de todos os tamanhos. Com uma infraestrutura global e segura, a AWS oferece uma variedade de serviços que permitem a execução de aplicações com alta disponibilidade, desempenho e segurança (AWS, 2024b).

3.1.8 AWS - Systems Manager Parameter Store

O AWS Systems Manager atua como um centro de operações para aplicações e recursos do AWS, permitindo o gerenciamento integrado de ambientes na nuvem (AWS, 2024d). Dentro dele, existe o Parameter Store, que oferece um armazenamento hierárquico seguro para o gerenciamento de dados de configuração e segredos. O Parameter Store permite que os usuários armazenem informações sensíveis, como senhas, strings de conexão com bancos de dados, códigos de licença e outras credenciais como valores de parâmetros. Esses valores podem ser armazenados em formato de texto simples ou como dados criptografados, garantindo segurança e controle de acesso adequado (AWS, 2024a).

O AWS Parameter Store será utilizado no projeto para armazenar de forma segura as credenciais de autenticação necessárias para acessar as APIs do Google. Essas credenciais precisam estar disponíveis para a função Lambda que irá interagir com as APIs, eliminando a necessidade de expor as chaves diretamente no código ou em arquivos de configuração.

3.1.9 AWS - Lambda

O AWS Lambda é um serviço de computação sem servidor (*serverless*) que permite executar código em resposta a eventos sem a necessidade de provisionar ou gerenciar servidores. O Lambda se destaca pela sua capacidade de escalar automaticamente, com cobrança baseada apenas no tempo de execução do código e nos recursos consumidos. Isso torna o Lambda uma solução eficiente tanto em termos de custo quanto de escalabilidade para aplicações que exigem flexibilidade e alta disponibilidade (AWS, 2024e).

No projeto, o AWS Lambda será utilizado como o serviço principal para executar o código da aplicação, utilizando a capacidade de rodar imagens Docker como ambiente de execução. A função Lambda é acionada quando recebe uma requisição HTTP de um cliente em seu *endpoint* de função dedicado, e então executa o código e retorna a resposta com o resultado da operação.

As principais vantagens de utilizar uma função Lambda são:

- Suporte nativo à execução de contêineres Docker, reduzindo problemas de compatibilidade e dependências entre diferentes ambientes.
- Gerencia automaticamente a escalabilidade do sistema, ajustando-se à demanda sem que seja necessária intervenção manual.
- O AWS Lambda opera em um modelo de pagamento baseado no consumo real de recursos. Isso significa que o projeto pagará apenas pelo tempo em que o código estiver sendo executado, eliminando custos com servidores ociosos.

- O AWS Lambda integra-se facilmente com outros serviços do AWS, como o AWS Parameter Store para o gerenciamento seguro de credenciais, por exemplo.

3.1.10 AWS Elastic Container Registry - ECR

O Amazon Elastic Container Registry (ECR) é um serviço de repositório gerenciado que facilita o armazenamento, o gerenciamento e a implantação de imagens de contêineres Docker. O ECR integra-se de maneira transparente com o Lambda e outros serviços do AWS, permitindo que os desenvolvedores implementem suas aplicações de contêiner com facilidade e segurança (AWS, 2024c).

O ECR é utilizado para armazenar e gerenciar a imagens Docker que é executada no AWS Lambda. Após a criação da imagem, ela é enviada para o ECR.

3.2 Linguagem de Programação

3.2.1 Python

Python é uma linguagem de programação de alto nível interpretada e orientada a objetos. É conhecida por sua sintaxe simples que enfatiza a legibilidade, tornando-a ideal para desenvolvimento rápido de aplicações e como linguagem de *script* para unir componentes de *software* existentes. A capacidade de modularidade do Python e sua extensa biblioteca padrão, que é gratuita para todas as principais plataformas, facilitam a reutilização de código. Além disso, permite um ciclo rápido de edição, teste e depuração, onde os erros geram exceções que são facilmente tratadas, melhorando a eficiência dos desenvolvedores (PYTHON, 2024).

Python é a principal linguagem de programação utilizada no projeto para capturar, processar e responder às requisições recebidas no *endpoint* da função Lambda com os dados dos formulários criados no Google Forms. Através de uma série de bibliotecas específicas é realizada a interação com as APIs do Google e com os serviços de nuvem AWS.

Foi escolhido como linguagem de programação para o projeto por possuir integração com as outras ferramentas escolhidas e por ser de amplo conhecimento da equipe. Os seus principais usos no projeto incluem:

- Captura de dados por meio das APIs do Google Drive e Google Forms.
- Serialização e desserialização de dados no formato JSON, facilitando a manipulação e integração com outras aplicações.
- Adaptação dos dados dos formulários para o formato necessário, transformando as informações capturadas em uma estrutura adequada para consumo externo.

- Recuperação segura de credenciais para a API do Google a partir do AWS Parameter Store.

3.2.2 Principais Bliotecas Externas

3.2.2.1 `googleapiclient.discovery`

A biblioteca `googleapiclient.discovery` é utilizada para se conectar às APIs do Google. Especificamente, é feita a integração com a API do Google Drive para listar os arquivos do tipo formulário e com a API do Google Forms para capturar os formulários.

3.2.2.2 `google.oauth2`

A biblioteca `google.oauth2` é utilizada para autenticar o acesso às APIs do Google Cloud, etapa necessária para que seja possível acessá-las.

3.2.2.3 `googleapiclient.errors`

Caso ocorra algum erro na comunicação, a biblioteca `googleapiclient.errors` é utilizada para tratar exceções, garantindo que as falhas sejam gerenciadas de forma apropriada.

3.2.2.4 `boto3`

A biblioteca `boto3` é empregada para interagir com os serviços do AWS, sendo utilizada no projeto principalmente para recuperar as credenciais de acesso da conta de serviço armazenadas no AWS Parameter Store.

4 Metodologia

4.1 Gerência de Projeto

A gestão de projetos desempenhou um papel crucial no planejamento, organização, coordenação e controle do trabalho realizado . Neste projeto foram empregados aspectos de abordagens ágeis, em específico o XP, visando promover a flexibilidade quanto a possíveis mudanças no escopo de seus requisitos.

4.1.1 *Extreme Programming* - XP

Conforme discutido no Capítulo 2, o XP oferece ao projeto uma abordagem ágil e colaborativa, destacando a comunicação efetiva e flexibilidade para se adaptar a mudanças nos requisitos. As práticas do XP usadas são:

- *Pair Programming*: A programação em pares foi usada sempre que possível, garantindo que todo código escrito por um desenvolvedor fosse revisado por outro. A revisão constante do código por duas perspectivas diferentes levou a um código mais limpo e mais eficiente, consequentemente aumentando sua qualidade. A programação em pares também permitiu a transferência de conhecimento entre os membros da equipe, de forma que, durante a interação, um desenvolvedor pudesse aprender boas práticas, truques e técnicas de programação com o outro.

- *Backlog*: Foi utilizado como uma lista organizada de maneira dinâmica e priorizada, contendo as funcionalidades e tarefas essenciais para o projeto.

- *Design* Incremental: O *design* do sistema foi revisado e alterado conforme necessário, sempre que surgiam alterações no escopo do projeto, refletindo a prática do XP de adaptabilidade contínua ao longo do desenvolvimento.

4.2 Gerência de Configuração de Software

4.2.1 Repositório

O código fonte foi hospedado no GitHub, garantindo que todos os desenvolvedores tivessem acesso ao código atualizado para uma colaboração eficaz.

4.2.2 *Trunk Based Development*

Especialmente considerando que a equipe é composta por apenas dois desenvolvedores, a abordagem de *Trunk-Based Development* provou ser extremamente eficaz para o controle de versão. Esta metodologia facilitou uma maior agilidade no desenvolvimento, permitindo que alterações rápidas e incrementais fossem feitas e integradas ao tronco principal do projeto de forma contínua e sem complicações. Tal prática minimizou a complexidade do gerenciamento de múltiplas *branches*, otimizando o fluxo de trabalho e aumentando significativamente a produtividade da equipe.

4.3 *Backlog*

O *backlog* contém os requisitos levantados para o projeto, servindo como base para o desenvolvimento das funcionalidades previstas.

4.3.1 Requisitos Funcionais

- O sistema deve ser capaz de listar o nome e o ID de formulários que foram compartilhados com uma conta de serviço específica no Google Drive.
- O sistema deve ser capaz de recuperar a representação em JSON dos formulários diretamente através da API do Google Forms, utilizando as credenciais autorizadas.
- O sistema deve ser capaz de transformar o JSON recuperado da API do Google Forms em um formato JSON adaptado, adequado às necessidades específicas do projeto.
- O sistema deve ser capaz de enviar o JSON adaptado como resposta às requisições HTTP feitas pelo aplicativo *mobile*.

4.3.2 Requisitos Não Funcionais

- O sistema deve ser executado em uma infraestrutura de *cloud*, garantindo flexibilidade, facilidade de gerenciamento e disponibilidade global.
- O sistema deve ser economicamente viável, priorizando soluções que mantenham os custos baixos.
- O sistema deve ser sempre disponível, minimizando períodos de inatividade.
- O sistema deve manejar as credenciais da conta de serviço do Google de maneira segura.

- O sistema deve ser escalável, permitindo que a capacidade de processamento aumente de acordo com a demanda.

5 Manual de Configuração

Este capítulo apresenta um guia passo a passo para a configuração completa da solução, desde o download do código-fonte até a integração com as plataformas em nuvem necessárias. O objetivo é fornecer instruções detalhadas para que qualquer usuário, mesmo com conhecimento técnico básico, consiga configurar a aplicação de forma correta e eficiente.

5.1 Repositório Git

A primeira etapa do processo envolve o *download* do código-fonte da aplicação a partir do repositório no GitHub. Esse repositório inclui o arquivo *Dockerfile*, o código da função Lambda e as bibliotecas externas necessárias. O repositório está disponível em https://github.com/GRVial/Fluxo-JSON.

As próximas etapas envolvem a configuração dos serviços de *cloud* necessários: Google Cloud e AWS.

5.2 Google Cloud

Primeiro, acesse o Google Cloud com uma conta Google.

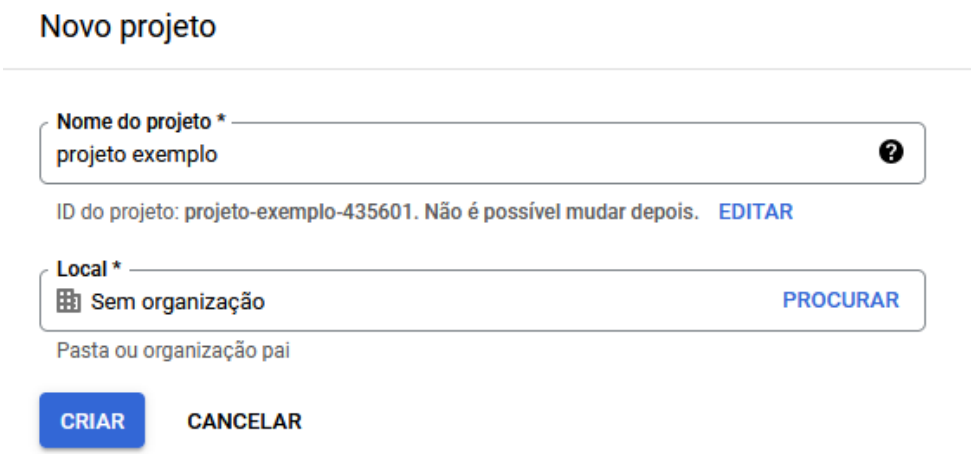

Figura 1 – Criação de novo projeto no Google Cloud

Após acessar o Google Cloud, o primeiro passo é criar um projeto. No console do Google Cloud, clique no ícone de seleção de projeto no topo da página. Dê um nome ao projeto conforme desejado, como ilustrado na Figura 1. Esse projeto será a base para

todas as configurações futuras e deve ser lembrado, pois será referenciado nas próximas etapas.

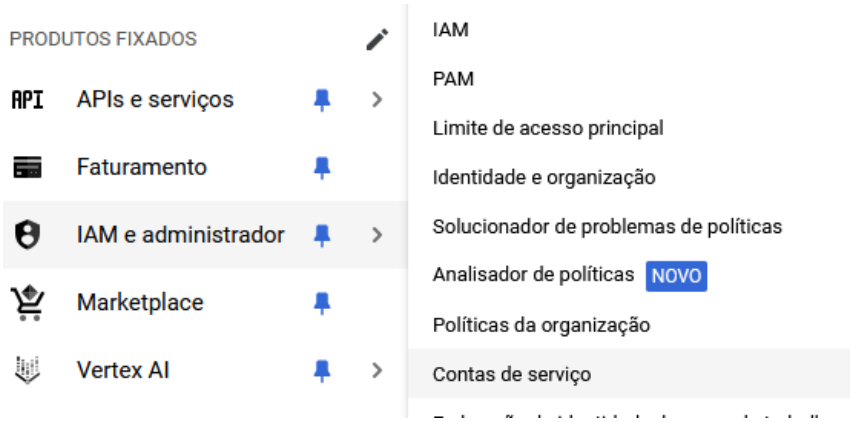

Figura 2 – Seção IAM e administrador do Google Cloud

Com o novo projeto selecionado, navegue até a seção "IAM e administrador"como mostra a Figura 2, no menu lateral esquerdo, e entre em "Contas de Serviço". Aqui, será criada uma conta de serviço que permitirá à aplicação se comunicar de maneira segura com as APIs do Google.

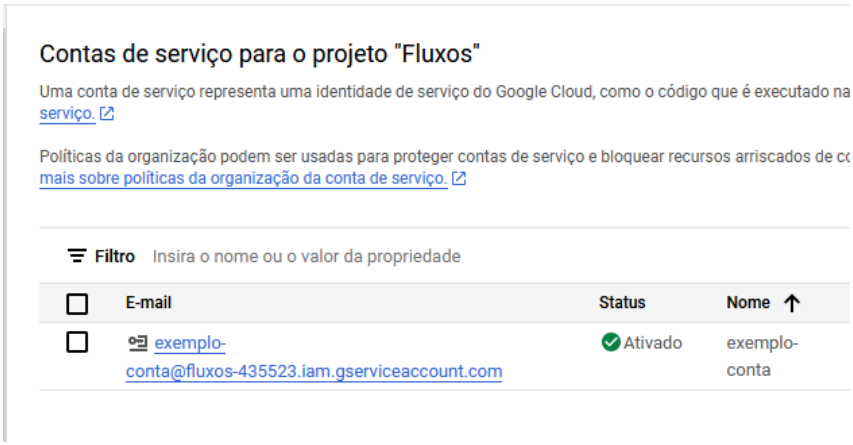

Figura 3 – Conta de Serviço do projeto Google Cloud

Após criar a conta de serviço, na lista de contas do projeto como da Figura 3, clique nela e acesse a aba “Chaves”. Agora ser gerada uma nova chave do tipo JSON. O arquivo JSON será baixado automaticamente para o seu computador. Esse arquivo contém as credenciais que permitirão à sua aplicação acessar as APIs do Google de maneira segura. Guarde-o pois ele será usado em etapas posteriores.

A próxima etapa consiste em ativar as APIs que serão utilizadas pelo projeto. No console do Google Cloud, vá até a Biblioteca de APIs e, dentro do projeto criado, ative as APIs do Google Forms e do Google Drive. Essas APIs permitirão que a conta de serviço

acesse, leia e processe formulários armazenados no Google Drive. Com isso, a configuração do Google Cloud está completa.

5.3 AWS

O próximo passo é configurar a parte da aplicação que roda na AWS. Comece criando uma conta na AWS. Uma vez dentro do console da AWS, poderá configurar os serviços necessários.

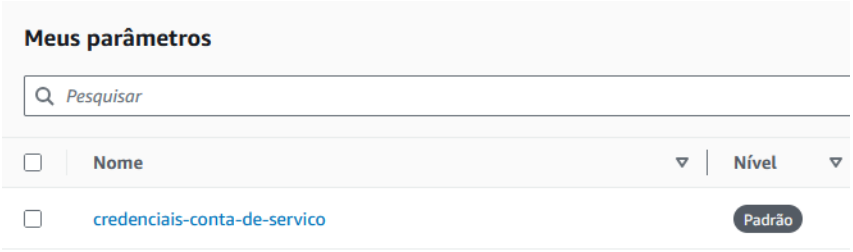

Figura 4 – Lista de parâmetros no AWS Systems Manager

Dentro do AWS Systems Manager, acesse o Parameter Store para criar um novo parâmetro. Esse parâmetro armazenará as credenciais da conta de serviço do Google, obtidas anteriormente. Ao criar o parâmetro, insira o conteúdo do arquivo JSON das credenciais em "Valor". A figura 4 mostra o parâmetro já configurado.

O próximo passo é configurar o Elastic Container Registry (ECR), onde será armazenada a imagem Docker que contém o código da aplicação. Crie um novo repositório de imagens no ECR e siga as instruções fornecidas na seção "Visualizar comandos push" localmente no diretório onde está o código fonte adquirido do GitHub. Isso permitirá o *upload* da imagem Docker para o repositório. Depois de seguir os comandos, a imagem estará disponível para ser usada na função Lambda.

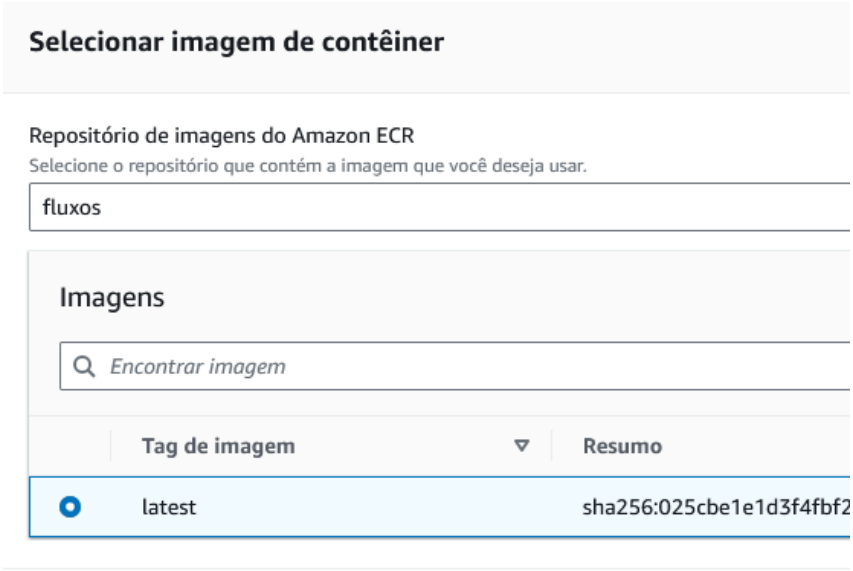

Figura 5 – Seleção de Imagem de contêiner implantado na função Lambda

Agora, crie uma nova função Lambda baseada na imagem de contêiner registrada anteriormente no ECR, como indica a Figura 5. Após criar a função, acesse a seção "Configuração"e aumente o tempo limite da execução para 15 segundos, garantindo que haja tempo suficiente para o processamento das requisições. Acesse a configuração de URL da função e crie uma URL da função com tipo de autenticação “NONE”.

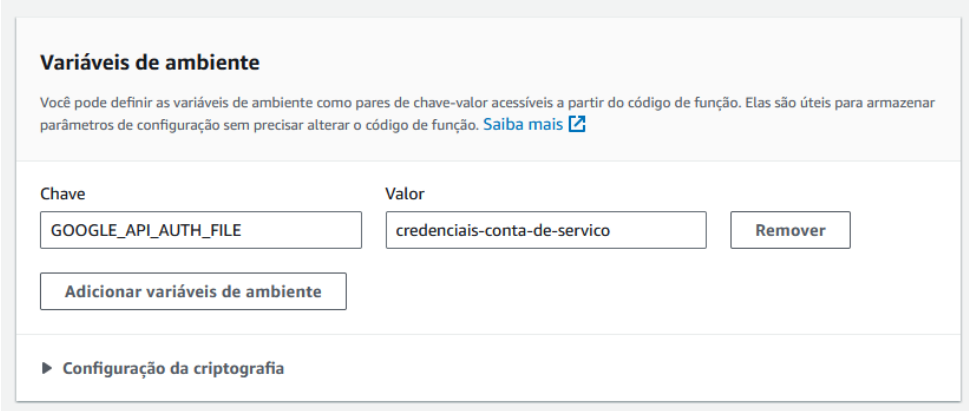

Figura 6 – Configuração da variável de ambiente da função Lambda

Em seguida, acesse as variáveis de ambiente da função e adicione uma nova variável com a chave `GOOGLE_API_AUTH_FILE`, utilizando o nome do parâmetro criado anteriormente no Parameter Store como o valor dessa variável, como exemplificado na Figura 6.

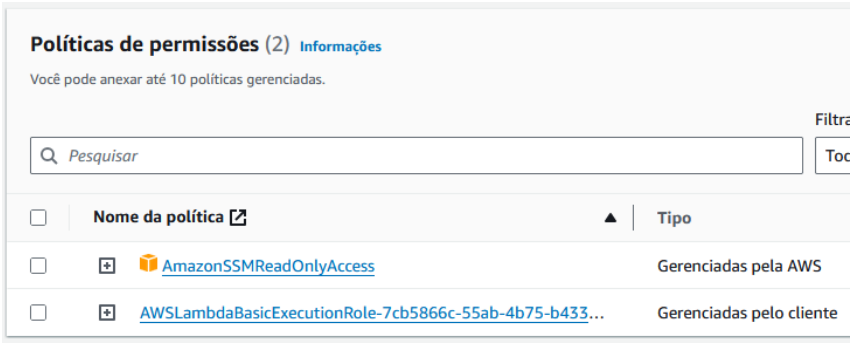

Figura 7 – Políticas de permissões da função Lambda

Por fim, acesse as permissões da função Lambda e adicione a política de permissão `AmazonSSMReadOnlyAccess`, o resultado deve parecer com a Figura 7. Isso garante que a função Lambda tenha as permissões necessárias para ler o parâmetro de credenciais armazenado no Parameter Store.

Com tudo configurado, a aplicação estará pronta para receber requisições em sua URL de função e processar formulários do Google Forms, retornando os dados no formato desejado. As funcionalidades e os padrões de resposta da função são detalhados no próximo capítulo.

6 Resultados

A ferramenta desenvolvida consiste em uma função Lambda que transforma os dados de um formulário do Google em um arquivo JSON, o qual descreve detalhadamente o fluxo de perguntas e respostas definido no formulário, incluindo a navegação condicional entre as seções. Esse processo envolve a extração dos dados estruturados diretamente da API do Google Forms, a conversão desses dados para o formato JSON desejado, respeitando a lógica de navegação estabelecida pelo formulário. O resultado final dessa transformação é disponibilizado por meio de uma API REST, permitindo que outras aplicações consumam o fluxo gerado, de forma automatizada e integrada.

Neste capítulo, serão apresentados os principais resultados obtidos com o desenvolvimento da solução. Serão discutidos os desafios enfrentados ao longo do processo, aspectos específicos da implementação, além de exemplos práticos de uso da ferramenta. Também serão detalhados a arquitetura do sistema e o fluxo de execução da aplicação, oferecendo uma visão geral sobre seu funcionamento e como os elementos se integram para cumprir os objetivos propostos.

6.1 Desafios Encontrados

Um dos principais desafios identificados no desenvolvimento foi o gerenciamento de seções que contêm mais de uma pergunta com lógica de navegação condicional. O comportamento padrão do Google Forms ao lidar com múltiplas perguntas com lógica de navegação condicional em uma única seção pode gerar incoerências na navegação. Por exemplo, se uma seção contém duas perguntas, cada uma com um destino diferente configurado pela lógica de navegação, o Forms tende a direcionar o usuário apenas para um dos fluxos, ignorando a lógica da outra pergunta. Isso pode resultar em um problema de inconsistência de fluxo, fazendo com que se "pule" uma seção que deveria ser acessada por uma das perguntas.

Para evitar esses problemas e garantir a integridade e continuidade dos fluxos de navegação, é sugerido que se limite cada seção a uma única pergunta com lógica de navegação condicional. Essa abordagem assegura um direcionamento mais claro e contínuo, evitando possíveis falhas na navegação.

Um problema menor identificado foi a inclusão das bibliotecas necessárias junto ao ambiente de execução no AWS Lambda. Essa dificuldade foi facilmente resolvida utilizando um contêiner Docker, baseado em uma imagem Python para Lambda disponível na galeria pública de contêineres da Amazon. Essa abordagem garantiu que todas as dependências fossem incluídas de forma eficiente e integrada ao ambiente de execução.

6.2 Arquitetura

O diagrama de arquitetura da aplicação ilustra de forma detalhada a estrutura e a interação entre os diversos componentes e serviços utilizados no sistema. A arquitetura do sistema foi projetada para integrar eficientemente Google Cloud e serviços AWS.

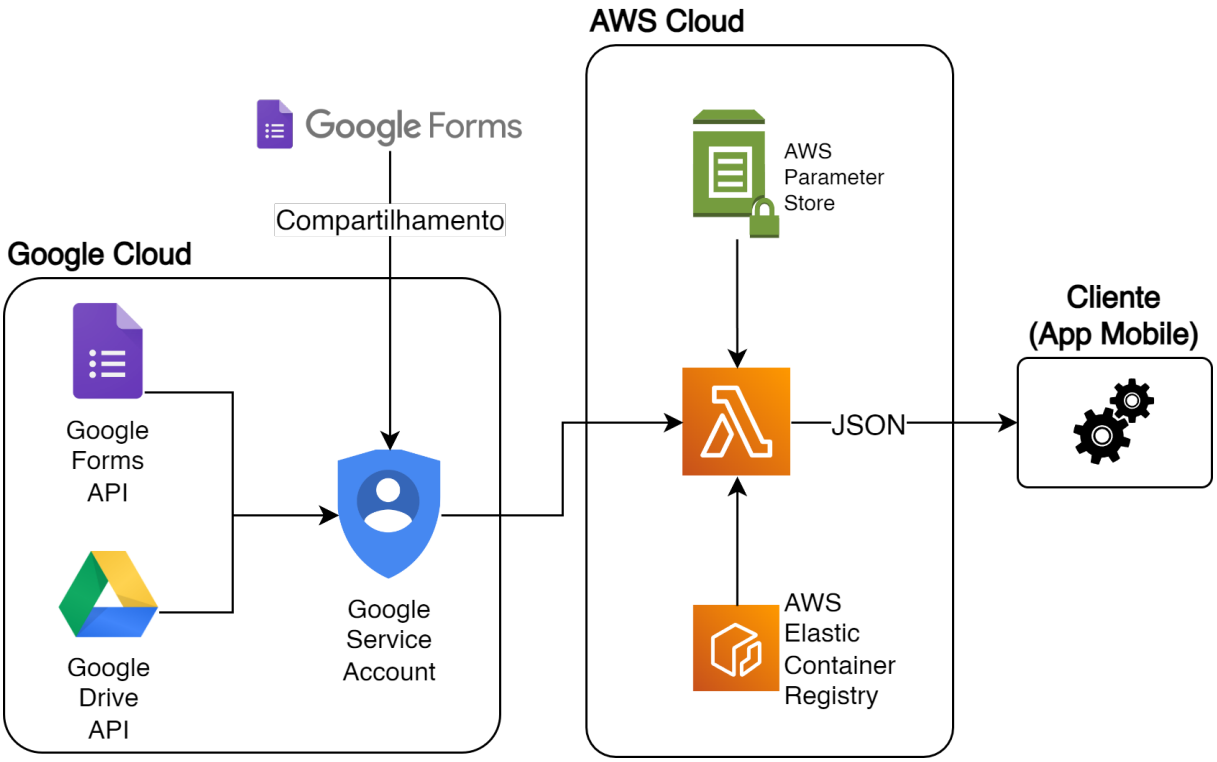

Figura 8 – Diagrama de arquitetura

A Figura 8 apresenta o diagrama de arquitetura da aplicação, que ilustra a estrutura e interação entre os principais componentes do sistema. A aplicação *mobile* aciona o *endpoint* da função Lambda, que, por sua vez, instancia a imagem Docker cadastrada no AWS ECR e recupera as credenciais da conta de serviço do Parameter Store. Utilizando essas credenciais, a função Lambda realiza chamadas à API do Google Forms ou do Google Drive, conforme o contexto da requisição, para obter os dados dos formulários compartilhados com a conta de serviço. Por fim, os dados obtidos são processados e retornados na resposta para o aplicativo.

6.3 Sequência

O diagrama de sequência da aplicação ilustra o fluxo de interação entre os principais componentes do sistema.

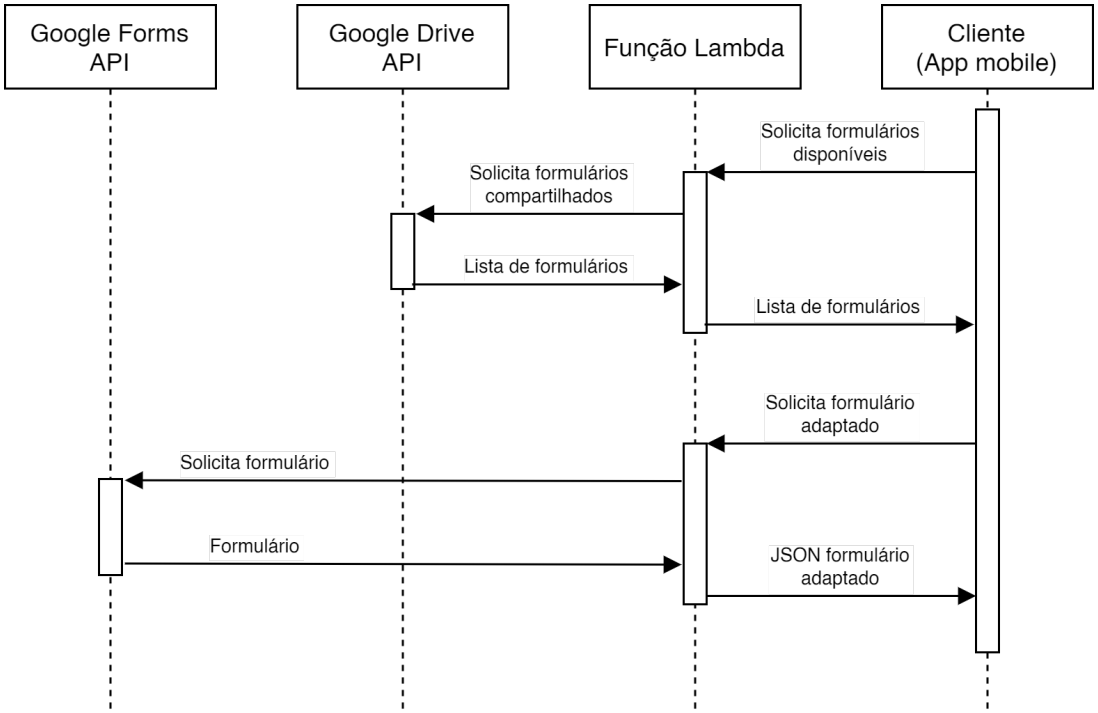

Figura 9 – Diagrama de sequência

A Figura 9 apresenta o diagrama de sequência da aplicação, ilustrando a ordem da interação entre os principais componentes do sistema: Google Forms API, Google Drive API, Função Lambda e o Cliente (aplicativo *mobile*). O fluxo inicia com o aplicativo solicitando à função Lambda a lista de formulários disponíveis. A função Lambda, por sua vez, consulta a API do Google Drive para obter esses dados e os retorna ao aplicativo. De posse da lista de formulários, o cliente seleciona um formulário específico e solicita à função Lambda o seu JSON adaptado. A função Lambda então acessa a API do Google Forms, recupera os dados do formulário, os adapta para o novo formato JSON, e finalmente envia o resultado ao aplicativo.

6.4 Formato do JSON adaptado

A adaptação dos dados JSON provenientes da API do Google Forms para um formato adaptado é fundamental no projeto, devido a algumas inadequações presentes na estrutura original dos dados. Primeiramente, os componentes são organizados de maneira sequenciada nos dados originais, sem ligação direta das seções e seus componentes filhos. Esta organização pode dificultar o acesso direto a informações específicas, aumentando a complexidade do processamento. Adicionalmente, os dados incluem uma quantidade excessiva de metadados que são irrelevantes para o escopo do projeto.

Abaixo, é definida a estrutura do JSON adaptado a partir da estrutura do formulário do Google Forms. Cada componente do formulário é convertido para um formato

JSON, onde suas chaves representam os atributos do formulário, como títulos, perguntas e lógicas de navegação. As informações estão organizadas em tabelas que descrevem os campos presentes no JSON, e contêm as seguintes colunas:

- **Chave**: Nome da chave que representa um atributo no JSON.
- **Tipo**: O tipo de dado associado à chave, como `string`, `boolean` ou `array`.
- **Obrig.**: Indica se o campo é obrigatório ou opcional.
- **Descrição**: Uma breve explicação sobre a função da chave no contexto do formulário.

As tabelas a seguir especificam os quatro principais objetos JSON que compõem a estrutura do formulário: o próprio formulário, `section`, `item` e `option`.

Chave	Tipo	Obrig.	Descrição
`title`	string	Sim	Título do formulário
`sections`	array[section]	Sim	Seções do formulário

Tabela 2 – Descrição do JSON do formulário adaptado

A Tabela 2 define a estrutura base do JSON do formulário. Vale notar que o atributo `sections` possui como tipo *array* de objetos `section`, ou seja, são objetos aninhados, definidos na próxima tabela:

Chave	Tipo	Obrig.	Descrição
`sectionId`	string	Sim	Identificador único da seção
`title`	string	Sim	Título da seção
`description`	string	Não	Descrição da seção
`items`	array[item]	Sim	Itens da seção

Tabela 3 – Descrição do objeto `section`

A Tabela 3 especifica os atributos de objetos `section`, que representam as seções do formulário. Cada `section` agrupa diversos objetos do tipo `item`:

Chave	Tipo	Obrig.	Descrição
`itemId`	string	Sim	Identificador único do item
`text`	string	Não	Texto do item
`description`	string	Não	Descrição do item
`type`	string	Sim	O tipo do item (pergunta ou texto)
`required`	boolean	Sim	Se o item é obrigatório ou não
`image`	string	Não	URL da imagem associada ao item
`options`	array[option]	Não	Opções de resposta, caso seja uma pergunta de múltipla escolha

Tabela 4 – Descrição do objeto `item`

A Tabela 4 detalha a estrutura dos objetos `item`. Objetos `item` podem representar perguntas ou textos dentro de uma seção. Caso exista, o atributo `options` pode conter vários objetos `option`:

Chave	Tipo	Obrig.	Descrição
`value`	string	Sim	Texto ou valor da opção
`goToSectionId`	string	Não	Identificador da seção para a qual a escolha desta opção direcionará

Tabela 5 – Descrição do objeto `option`

Por fim, a Tabela 5 define os objetos `option`. Esses objetos representam as opções de resposta para perguntas de múltipla escolha, e existem apenas em objetos `item` do tipo pergunta.

O exemplo de JSON abaixo representa a estrutura dos componentes mencionados nas tabelas anteriores, evidenciando como cada um dos elementos é organizado.

```
{
    "title": "Título",
    "sections": [
        {
            "sectionId": "ID da seção",
            "title": "Título da seção",
            "description": "Descrição da seção",
            "items": [
                {
                    "type": "RADIO",
```

```
                    "required": true,
                    "image": "URL da imagem",
                    "itemId": "ID do item",
                    "description": "Descrição do item",
                    "text": "Texto da pergunta",
                    "options": [
                        {
                            "value": "Resposta 1",
                            "goToSectionId": "id da seção 2"
                        },
                        {
                            "value": "Resposta 2",
                            "goToSectionId": "id da seção 3"
                        },
                        // Restante das opções
                    ],
                },
                // Restante dos itens
            ]
        },
        // Restante das seções
    ]
}
```

6.5 *Endpoints*

A seguir são documentados os *endpoints* da API desenvolvida no projeto. Cada *endpoint* é descrito com sua respectiva rota, o método HTTP utilizado e, quando aplicável, os parâmetros esperados na requisição. Além disso, também são apresentados exemplos de requisições e suas respostas, demonstrando os formatos de dados utilizados e os códigos de *status* HTTP retornados.

6.5.1 Obter lista de formulários

- **Método:** `GET`
- **Rota:** `/listForms`
- **Descrição:** Retorna a lista de nomes e identificadores de formulários compartilhados com a conta de serviço do Google do projeto.

Exemplo de Requisição: `GET /listForms`

Exemplo de Resposta:

```
[
    {
        "id": "abcdefghijklmnopqrstuvwxyz",
        "name": "Formulário 1"
    },
    {
        "id": "D2OFIGsdoOS37DUIshdufDS2FHasdDSS",
        "name": "Formulário 2"
    }
]
```

Códigos de Resposta:

- `200 OK` – Requisição bem sucedida e formulários disponíveis listados.

6.5.2 Obter dados de um formulário especificado

- **Método:** `GET`
- **Rota:** `/`
- **Parâmetros de *query*:**
  - `formId`: identificador único do formulário.
- **Descrição:** Retorna o JSON adaptado do formulário especificado pelo parâmetro de identificador.

Exemplo de Requisição: `GET /?formId=abc123`

Exemplo de Resposta:

```
{
    "title": "Exemplo",
    "sections": [
      {
        "sectionId": "FIRST_SECTION",
        "title": "Título da primeira seção",
        "items": [
```

```
          {
            "type": "RADIO",
            "required": true,
            "description": "Descrição da pergunta",
            "text": "Texto da pergunta",
            "options": [
              {
                "value": "Resposta 1",
                "goToSectionId": "abcdef"
              },
              {
                "value": "Resposta 2",
                "goToSectionId": "fedcba"
              }
            ],
            "image": "https://uri.de.imagem/",
            "itemId": "1e2061a2"
          }
        ]
      },
      //Continua...
    ]
}
```

Códigos de Resposta:

- `200 OK` - Requisição bem sucedida e dados do formulário retornados.
- `400 BAD REQUEST` - O parâmetro `formId` está ausente.
- `404 NOT FOUND` - O `formId` solicitado não corresponde a um formulário disponível.

6.6 Exemplos de Uso

Nesta seção, serão apresentados exemplos práticos de uso, desde o compartilhamento de um formulário até como os elementos do Google Forms são transformados no formato JSON adaptado pela aplicação. Cada elemento do formulário é processado e estruturado, garantindo que todas as informações sejam corretamente representadas no formato especificado.

Durante a criação de fluxos no Google Forms, é possível utilizar seções, bem como diferentes estruturas de perguntas e textos descritivos dentro dessas seções. Essas estruturas não só fornecem informações aos usuários, mas também os direcionam para a próxima etapa do processo usando a lógica de navegação condicional, guiando-os efetivamente através de suas dúvidas.

6.6.1 Seção com pergunta

Os itens de pergunta são utilizados para estabelecer a lógica de navegação condicional entre as seções.

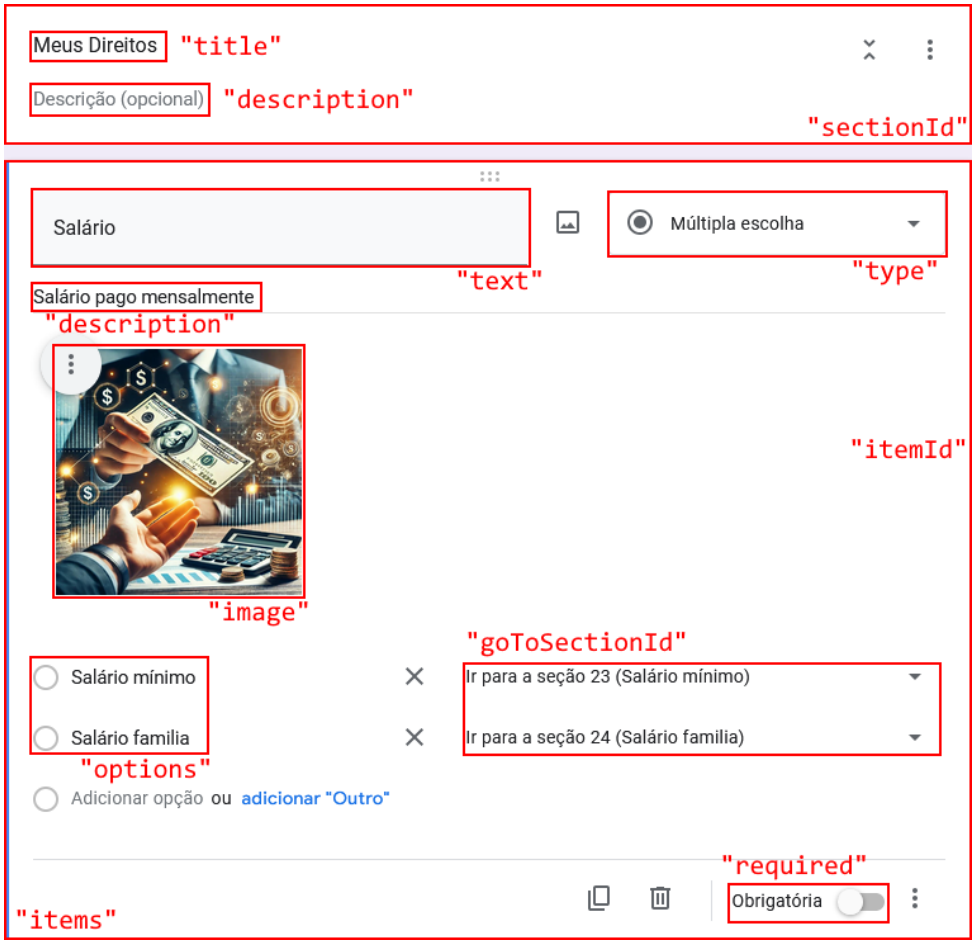

Figura 10 – Seção com pergunta legendado

O exemplo na Figura 10, ilustra a estrutura de uma seção do Google Forms contendo uma pergunta, juntamente com seus atributos correspondentes. Cada um dos componentes destacados na imagem faz parte da estrutura do formulário e será convertido para o formato JSON.

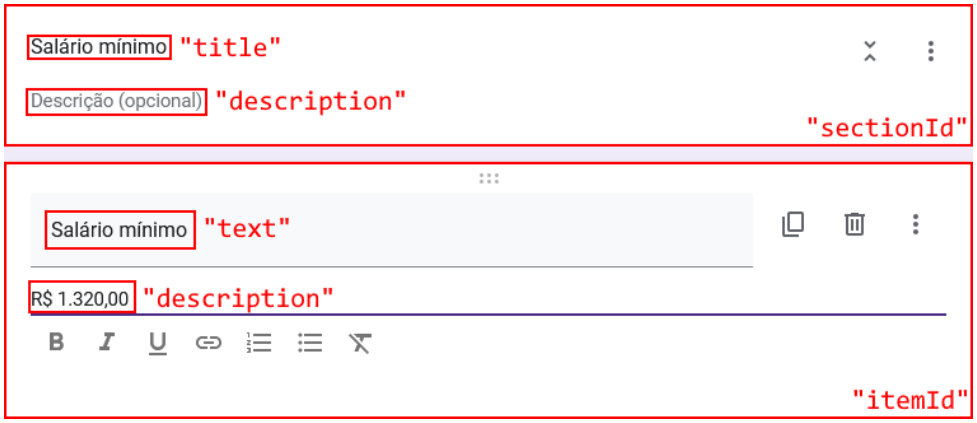

Figura 11 – Seção com texto legendado

6.6.2 Seção de texto

Podem haver seções que têm apenas itens de texto, sem perguntas. Este tipo de seção é utilizado para fornecer informações ou descrições que não exigem interação por parte do usuário.

Seções que contêm apenas texto, como o exemplo da Figura 11, não possuem lógica de navegação condicional, o que sinaliza um fim do fluxo de navegação. No entanto, em casos onde o fluxo deve continuar, é possível combinar o conteúdo textual com perguntas que utilizem lógica de navegação condicional, permitindo que o usuário seja redirecionado para outras seções conforme necessário.

6.6.3 Compartilhamento

Com o formulário pronto, então deve-se adicionar a conta de serviço da ferramenta como colaboradora do formulário.

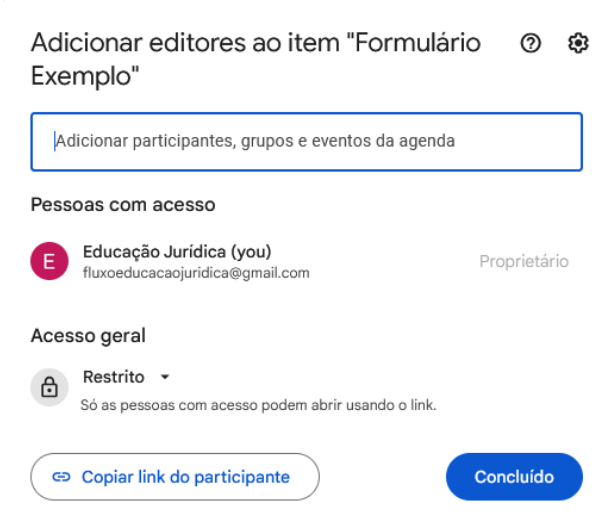

Figura 12 – Menu de adicionar colaborador

A Figura 12 ilustra o *menu* utilizado para realizar o compartilhamento. Para remover o formulário dos resultados do *endpoint*, basta remover o compartilhamento com a conta de serviço da ferramenta.

6.7 Análise de Desempenho e Custo

Nesta seção, serão abordados dois aspectos cruciais da solução: o tempo de resposta das operações e o custo estimado de sua operação em ambiente de produção. Essas métricas são fundamentais para garantir que a solução seja não apenas tecnicamente viável, mas também financeiramente sustentável.

A própria interface do AWS fornece uma visualização dos tempos de execução da função Lambda, permitindo o monitoramento do desempenho. Especificamente, essa métrica registra o tempo que o código da função passa processando um evento recebido. Durante os testes, o tempo médio de execução da função foi de 2.664 milissegundos, com a maior parte desse tempo, cerca de 1.479 milissegundos, sendo atribuída às chamadas para as APIs do Google. O restante do tempo foi utilizado nas etapas de conversão do formulário para JSON e no envio da resposta. É importante destacar que a duração de todas as etapas pode variar de acordo com o tamanho e a complexidade do formulário solicitado.

O AWS também disponibiliza uma ferramenta de estimativa de custos para funções Lambda, que permite calcular o custo de operação com base em parâmetros inseridos, como a quantidade de ativações e o tempo de execução de cada uma. Durante o desenvolvimento, o número de ativações permaneceu baixo. Considerando o tempo de execução médio de 2.664 milissegundos por requisição e o nível gratuito da AWS Lambda, que oferece 1 milhão de solicitações e 400.000 GB-segundos de tempo de computação por mês, foi constatado que, mesmo no cenário máximo de 1 milhão de solicitações mensais (equivalente a 22,4 solicitações por minuto), a ferramenta mantém-se dentro do plano gratuito, sem ultrapassar o limite de tempo de computação oferecido.

Inicialmente, havia a intenção de comparar a solução com plataformas educacionais públicas, para avaliar seu desempenho em cenários de grande escala. No entanto, o site que disponibiliza essas informações estava fora de funcionamento, impossibilitando a realização dessa análise comparativa.

6.8 Planejado x Realizado

Ao comparar a solução implementada com o que foi planejado no *backlog*, verifica-se que todos os requisitos funcionais foram atendidos, com os *endpoints* e as integrações executando corretamente as operações definidas.

A solução foi implementada utilizando uma infraestrutura em nuvem, aproveitando as ferramentas oferecidas pelo AWS para garantir alta disponibilidade e escalabilidade automática, assim como o gerenciamento seguro das credenciais. A função Lambda permite que a aplicação se ajuste de forma dinâmica à demanda, assegurando que os serviços permaneçam sempre disponíveis, mesmo em cenários de carga elevada.

Embora a estimativa mensal de 8,25 dólares para os custos operacionais seja relativamente baixa, há potencial para reduzir ainda mais esse valor. Isso pode ser alcançado por meio da otimização do tempo de execução da função Lambda, o que resultaria em um menor consumo de recursos computacionais.

7 Considerações Finais

A introdução deste trabalho destacou a necessidade de melhorar o acesso ao conhecimento jurídico no Brasil, uma necessidade impulsionada pela complexidade das leis e pela importância de sua compreensão por todos os cidadãos. Para abordar este desafio, foi desenvolvido um sistema que facilita a criação e disseminação de fluxos de perguntas e respostas jurídicas, configurados para transformar esses fluxos em formatos acessíveis através de uma solução tecnológica robusta.

Este sistema forma a base de um projeto colaborativo entre a Faculdade de Direito da Universidade de São Paulo e a Universidade de Brasília, que também inclui o desenvolvimento de uma aplicação *mobile*. Embora o aplicativo *mobile* opere como um componente separado, dentro do mesmo projeto, ambos trabalham em conjunto para promover o acesso interativo ao conhecimento jurídico. O sistema desenvolvido permite não apenas a fácil adaptação dos dados para o formato necessário, mas também sua integração eficiente com o aplicativo *mobile*, facilitando a disseminação do conhecimento de maneira prática.

Os resultados alcançados pelo projeto representam um passo em direção aos objetivos inicialmente propostos, contribuindo para a facilitação do acesso ao direito e à informação legal de forma mais interativa. O uso da tecnologia para melhorar a educação jurídica mostrou-se promissor e oferece um exemplo de como futuras iniciativas podem continuar a reduzir barreiras entre os cidadãos e o entendimento das leis.

A estrutura JSON adotada neste projeto foi projetada para ser flexível e adaptável, permitindo que outras ferramentas de criação de fluxo possam se beneficiar da mesma configuração. Isso facilita a integração com outras aplicações que necessitem de dados estruturados de forma similar, ampliando assim a utilidade do sistema além das funcionalidades atualmente implementadas.

Uma sugestão para futuras melhorias é implementar o armazenamento dos dados recuperados das APIs do Google, uma vez que essa etapa é a mais demorada no processo de execução atual e, consequentemente, contribui para o aumento do custo operacional. Essa mudança reduziria a dependência de conexões frequentes com a API, melhorando a latência do sistema.

Referências

AWS. *What is an API?* 2023. <https://aws.amazon.com/pt/what-is/api/>. Acesso em: 11 set. 2024. Citado 2 vezes nas páginas 24 e 25.

AWS. *What is Cloud Computing?* 2023. Disponível em: <https://aws.amazon.com/pt/what-is-cloud-computing/>. Citado 2 vezes nas páginas 23 e 24.

AWS. *AWS Systems Manager Parameter Store*. 2024. Acesso em: 08 set. 2024. Disponível em: <https://docs.aws.amazon.com/systems-manager/latest/userguide/systems-manager-parameter-store.html>. Citado na página 31.

AWS. *O que é AWS?* 2024. Acesso em: 13 set. 2024. Disponível em: <https://aws.amazon.com/pt/what-is-aws/>. Citado na página 30.

AWS. *O que é o Amazon Elastic Container Registry?* 2024. Acesso em: 08 set. 2024. Disponível em: <https://docs.aws.amazon.com/pt_br/AmazonECR/latest/userguide/what-is-ecr.html>. Citado na página 32.

AWS. *O que é o AWS Systems Manager?* 2024. Acesso em: 10 set. 2024. Disponível em: <https://docs.aws.amazon.com/pt_br/systems-manager/latest/userguide/what-is-systems-manager.html>. Citado na página 31.

AWS. *Recursos do AWS Lambda*. 2024. Acesso em: 08 set. 2024. Disponível em: <https://aws.amazon.com/pt/lambda/features/?pg=ln&sec=hs>. Citado na página 31.

BAUER, F.; BAUER, F. *Software Engineering: An Advanced Course*. Springer Berlin Heidelberg, 1977. (Lecture notes in computer science). ISBN 9780387083643. Disponível em: <https://books.google.com.br/books?id=q7ZQAAAAYAAJ>. Citado na página 21.

BECK, K. *Extreme programming explained: embrace change*. [S.l.]: addison-wesley professional, 2000. Citado na página 22.

BECK, K. et al. *Manifesto for Agile Software Development*. 2001. Disponível em: <https://agilemanifesto.org/>. Citado na página 22.

BOEHM. Software engineering. *IEEE Transactions on Computers*, C-25, n. 12, p. 1226–1241, 1976. Citado na página 21.

BRASIL. *Decreto-lei 4.657, de 4 de setembro de 1942*. 1942. <https://www.planalto.gov.br/ccivil_03/decreto-lei/del4657.htm>. Acesso em: 8 set. 2024. Citado na página 17.

CABRAL, J. E. et al. Using technology to enhance access to justice. *Harvard Journal of Law Technology*, v. 26, p. 241, 2012. Citado na página 18.

CLOUD, G. *Visão geral das contas de serviço*. 2024. Acesso em: 08 set. 2024. Disponível em: <https://cloud.google.com/iam/docs/service-account-overview?authuser=2&hl=pt-br>. Citado na página 30.

COHEN, D.; LINDVALL, M.; COSTA, P. Agile software development. *Dacs Soar Report*, v. 11, p. 2003, 2003. Citado na página 22.

DATASENADO. Datasenado 25 anos da constituição secretaria de transparência coordenação de pesquisa e opinião datasenado. 2013. Disponível em: <https://www12.senado.leg.br/institucional/datasenado/arquivos/brasileiros-reconhecem-importancia-da-constituicao-cidada>. Citado na página 17.

DOCKER. *Docker overview*. 2020. Disponível em: <https://docs.docker.com/get-started/overview/>. Citado na página 29.

GITHUB. 2023. Disponível em: <https://docs.github.com/pt/get-started/using-git/about-git>. Citado na página 28.

GITHUB. *Features - GitHub*. 2024. Acessado em: 11 de setembro de 2023. Disponível em: <https://github.com/features>. Citado na página 28.

GOOGLE. *Gere insights facilmente com o Google Forms*. 2023. Acessado em: 11 de setembro de 2023. Disponível em: <https://www.google.com/forms/about/>. Citado na página 27.

GOOGLE. *Sobre Google Drive*. 2024. Acesso em: 12 set. 2024. Disponível em: <https://www.google.com/intl/pt-br/drive/about.html>. Citado na página 28.

HAMMANT, P. *Trunk Based Development*. 2020. Acesso em: 6 set. 2024. Disponível em: <https://trunkbaseddevelopment.com/>. Citado na página 23.

HAT, R. *O que é API?* 2023. Acesso em: 11 set. 2024. Disponível em: <https://www.redhat.com/pt-br/topics/api/what-are-application-programming-interfaces>. Citado na página 24.

IBM. *What is Serverless?* 2023. Acesso em: 07-09-2024. Disponível em: <https://www.ibm.com/br-pt/topics/serverless>. Citado na página 24.

IBM. *O que é FaaS*. 2024. Acesso em: 07-09-2024. Disponível em: <https://www.ibm.com/br-pt/topics/faas>. Citado na página 24.

JSON.ORG. *JSON*. 2023. Disponível em: <https://www.json.org/json-en.html>. Citado 2 vezes nas páginas 29 e 30.

PRESSMAN, R. S. *Engenharia de Software: uma abordagem profissional*. 7ª. ed. [S.l.]: McGraw Hill, 2011. Citado na página 21.

PYTHON. *What is Python? Executive Summary*. 2024. Acessado em: 11 de setembro de 2024. Disponível em: <https://www.python.org/doc/essays/blurb/>. Citado na página 32.

SILVA, J. R. A. d.; CHOUCINO, C. C.; MACHADO, S. C. D. A falta de conhecimento da população em relação aos seus direitos e a inclusão do direito constitucional nas escolas. *Revista Jurídica da UniFil*, v. 16, n. 16, p. 148–157, Oct 2019. Disponível em: <http://periodicos.unifil.br/index.php/rev-juridica/article/view/1150>. Citado na página 17.