UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

ARTHUR LOIFERMAN JAWETZ

Arquitetura Serverless para Sistema de Gerenciamento de Atendimentos Psicológicos

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Engenharia da Computação

Orientador: Prof[a]. Dr[a]. Renata Galante

Porto Alegre
2024

AGRADECIMENTOS

Este trabalho não foi feito sozinho. Gostaria de agradecer a todos que me ajudaram nesta caminhada:

Meus pais Adriana Loiferman e Jose Jawetz; minha família; meus amigos; meus colegas; meus professores e minha orientadora. A todos: meu muito obrigado.

RESUMO

O desenvolvimento de software utilizando a arquitetura *serverless* está em grande crescimento devido à sua capacidade de escalabilidade, flexibilidade e otimização de recursos. Junto a isso, a busca por atendimentos psicológicos vem expandindo significativamente nos últimos anos, necessitando cada vez mais de plataformas acessíveis e eficientes para ajudar os profissionais da área a se organizarem e armazenarem informações sobre seus pacientes. Por esses motivos, é investigada uma solução utilizando uma arquitetura *serverless* voltada a uma aplicação de gerenciamento de atendimentos psicológicos. Ao adotar esta estrutura, espera-se obter uma aplicação performática e econômica. Como avaliação, são apresentados o tempo de resposta da aplicação, bem como uma estimativa de custo mensal para o funcionamento do sistema.

Palavras-chave: Arquitetura *serverless*. AWS. software. psicólogo.

Serverless Architecture for Psychological Management System

ABSTRACT

The development of software using serverless architecture is experiencing significant growth due to its scalability, flexibility, and resource optimization capabilities. Alongside this, the demand for psychological services has been expanding notably over the past few years, requiring accessible and efficient platforms to assist professionals in organizing and storing information about their patients. Therefore, a solution using a serverless architecture for a psychological care management application is investigated. By adopting this structure, it is expected to achieve a more performant and cost-effective application. For this purpose, the application response time will be presented, as well as an estimated monthly cost for operating the system.

Keywords: Serverless architecture, AWS, software, psychologist.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AWS	Amazon Web Services
Saas	Software as a Service
PaaS	Platform as a Service
IaaS	Infrastructure as a Service
FaaS	Function as a Service
BaaS	Backend as a Service
HTTP	Hypertext Transfer Protocol
S3	Simple Storage Service
NoSQL	Non Structured Query Language
API	Application Programming Interface
REST	Representational State Transfer
RDS	Relational Database Service
JSON	JavaScript Object Notation
SES	Simple Email Service
SQS	Simple Queue Service
SNS	Simple Notification Service
RaaS	Robots as a Service
CSS	Cascading Style Sheets
HTML	HyperText Markup Language
IoT	Internet of Things
CFP	Conselho Federal de Psicologia
PK	Partition Key
SK	Sort Key
MAU	Monthly Active Users
CRUD	Create, Read, Update, and Delete

SUMÁRIO

1 INTRODUÇÃO

A psicologia desempenha um papel fundamental na saúde mental e no bem-estar geral da sociedade. Os psicólogos fornecem uma compreensão profunda dos comportamentos humanos, pensamentos e emoções, auxiliando o paciente em suas questões pessoais através do autoconhecimento e diálogo. De acordo com pesquisa realizada pela *American Psychological Association* (ASSOCIATION, 2022), a busca por tratamento de ansiedade e depressão permaneceu alta pelo terceiro ano consecutivo. Além disso, seis em cada dez psicólogos relataram que não têm mais vagas para novos pacientes. Juntamente a isso, conforme o mesmo estudo, 45% dos psicólogos revelaram que se sentiam esgotados. Com esta alta na busca por atendimentos psicológicos, é necessário refletir e investigar como a tecnologia pode auxiliar o dia a dia dos profissionais da saúde.

Uma das principais inovações na relação paciente-psicólogo ficou conhecida como telepsicologia (DIAS et al., 2015). O avanço de estudos na área de comunicação virtual propiciou a possibilidade de atendimentos via Internet, sem a necessidade de um encontro pessoal entre paciente e profissional. A telemedicina é associada a uma melhoria dos serviços de saúde qualificados principalmente em populações desfavorecidas e em áreas remotas (DIAS et al., 2015).

Unido à otimização da comunicação com os pacientes, a tecnologia tem desempenhado um papel fundamental no auxílio aos psicólogos em relação à organização de suas agendas e coordenação de consultas. Aplicativos relacionados ao gerenciamento de atendimentos já estão presentes no mercado, como *Apphealth* (APPHEALTH, 2022) e *Psicomanager* (PSICOMANAGER, 2023), por exemplo.

Com isto em mente, é importante se perguntar quais avanços no desenvolvimento de *software* podem ajudar na implementação de novas ferramentas para o uso do profissional psicólogo. Um paradigma emergente no desenvolvimento de *software* é conhecido como computação *serverless*, que abstrai o gerenciamento de servidores por parte do desenvolvedor, promovendo um *software* mais escalável, flexível e com menos custo (WEN et al., 2022). Para implementar uma solução *serverless* é preciso arquitetar e integrar serviços de um provedor de *cloud* a fim de construir uma aplicação completa.

1.1 Objetivos

Este trabalho busca responder a questão de qual modo é possível arquitetar serviços da AWS - provedor de *cloud* mais utilizado no mercado (LUXNER, 2023) - para a construção de um *software* de gerenciamento de atendimentos para psicólogos. Com o propósito de responder este questionamento, este trabalho tem o objetivo de averiguar e estudar os serviços disponíveis pela AWS e como estes se relacionam entre si. Por conter mais de 300 serviços variados (AMAZON, 2023), são somente apresentados os componentes presentes na aplicação final, demonstrando seus funcionamentos e conexões. Para a construção de um aplicativo viável, é necessário estudar e analisar o tempo de resposta de seus fluxos, tal como o custo de seu funcionamento.

1.2 Organização do texto

O presente trabalho está dividido em seis Capítulos, incluindo a introdução. No próximo Capítulo são apresentados os conceitos e tecnologias utilizadas, assim como os serviços da AWS utilizados no aplicativo. No Capítulo 3, são exibidos trabalhos relacionados ao desenvolvimento de *software* utilizando a arquitetura *serverless* e outros aplicativos relacionados à psicologia, realizando uma comparação dos trabalhos. O objetivo do Capítulo 4 consiste na explicação do desenvolvimento e implementação da arquitetura, descrevendo cada um dos módulos e suas comunicações entre si, bem como a estrutura de classes utilizadas. Após, no Capítulo 5, é realizada uma análise quantitativa do desempenho do sistema e uma estimativa de precificação da arquitetura. Por fim, no Capítulo 6, é apresentada a conclusão da plataforma desenvolvida, bem como do trabalho realizado.

2 CONCEITOS E TECNOLOGIAS UTILIZADAS

Neste capítulo, são apresentadas as principais tecnologias e conceitos utilizados no desenvolvimento de um sistema de gerenciamentos de atendimentos psicológicos empregando uma arquitetura *serverless*.

2.1 Computação em nuvem

A computação em nuvem tem revolucionado a maneira como os serviços digitais são entregues e consumidos (SALIM et al., 2020). Com esta inovação, recursos de computação, tais como servidores e banco de dados, são acessados através da Internet, sem a necessidade de sustentar este poder computacional localmente. Com este novo paradigma, despertou-se um conjunto de oportunidades de desenvolvimento de aplicações, sem a necessidade de investimento em infraestrutura própria, reduzindo custos e aumentando a eficiência operacional.

Esta nova tecnologia criou um mercado de empresas especializadas em oferecer o recurso computacional para o desenvolvedor. Os provedores de serviços em nuvem foram inicialmente classificados pelo seu conjunto de serviços oferecidos, divididos entre SaaS (*Software as a Service*), PaaS (*Platform as a Service*) e IaaS (*Infrastructure as a Service*) (RIMAL et al., 2011).

O SaaS é um modelo de distribuição de *software* no qual a aplicação é hospedada e disponibilizadas aos clientes *out-of-the-box*, ou seja, pronta para o uso. O SaaS permite aos usuários acessarem funcionalidades de *software* sem a necessidade de instalações locais ou manutenção de infraestrutura física. Pode-se citar *softwares* como Salesforce (Salesforce, 2023), Gmail (GOOGLE, 2024) e Moodle (Moodle, 2023), por exemplo.

O PaaS provê aos desenvolvedores um ambiente *online* para desenvolver, testar, implementar e gerenciar aplicações sem a complexidade associada à construção e manutenção da infraestrutura (WEN et al., 2022). Uma das características do PaaS é o alto nível de abstração, incluindo sistema operacional, linguagens de programação, bibliotecas, serviços e ferramentas, permitindo assim, que desenvolvedores se concentrem no código e na lógica de negócios. Dentre os provedores de PaaS se destacam a *Microsoft Azure App Service* (MICROSOFT, 2023) e *Google App Engine* (GOOGLE, 2023).

Já o IaaS consiste em um modelo de provisão de serviços de computação em nuvem, no qual recursos fundamentais de TI, como servidores virtuais, redes, armaze-

namento e ambientes operacionais são oferecidos como um serviço através da Internet. No entanto, no IaaS os desenvolvedores ainda são responsáveis pelo provisionamento de recursos, configuração de tempo de execução, gerenciamento de código e mais (WEN et al., 2022).

2.2 *Serverless*

Computação sem servidor é um modelo de serviço em nuvem no qual os desenvolvedores podem se concentrar exclusivamente no desenvolvimento de código sem se preocuparem com o gerenciamento de servidores. A principal diferença da computação *serverless* para a computação em nuvem tradicional (muitas vezes referenciada como *serverful*) é o fato da infraestrutura e plataforma serem ocultos dos desenvolvedores (SHAFIEI; KHONSARI; MOUSAVI, 2021). Nesse modelo, funções executam em resposta à eventos específicos, sendo escalados automaticamente e cobrados apenas pelo tempo de execução efetivo. O provedor de nuvem é responsável por gerenciar a infraestrutura, como provisionamentos de servidores, escalabilidade e tolerância a falhas, permitindo aos desenvolvedores uma maior agilidade no desenvolvimento de aplicações e uma maior eficiência operacional (WEN et al., 2022).

Atualmente a computação *serverless* vem ganhando tração em diversos domínios, como processamento de vídeo, Internet das Coisas (IoT), processamento de *Big Data* e aprendizado de máquina (WEN; CHEN; LIU, 2022). De acordo com pesquisa realizada em maio de 2019 (RESEARCH, 2019), é previsto que o mercado baseado em *serverless* cresça de três milhões de dólares em 2017 para vinte e um no ano de 2025. Outra pesquisa realizada em 2022 (GARTNER, 2022), prevê-se que a computação *serverless* poderá ser empregada em 50% das empresas globais até 2025. Com isso, empresas como Amazon, Google e Microsoft estão investindo no futuro desta tecnologia, a partir da introdução do *AWS Lambda* em 2014, *Google Cloud Functions* e *Microsoft Azure Functions* em 2016 (SHAFIEI; KHONSARI; MOUSAVI, 2021).

Os aplicativos relacionados à computação sem servidor seguem o estilo de *software* de microsserviço, decompondo o aplicativo em um subconjunto de tarefas independentes (WEN et al., 2022). Essas tarefas são conhecidas como componentes, representando uma unidade de *software* que pode ser substituída e atualizada de forma independente (FOWLER, 2014). Separar uma aplicação em microsserviços promove a escalabilidade, tolerância a falhas e a agilidade no desenvolvimento do *software* (MICROSOFT,

2024).

Uma arquitetura *serverless* é implementada com o uso de dois diferentes componentes: BaaS (*Backend as a Service*) e FaaS (*Function as a Service*). O BaaS consiste em serviços em nuvem personalizados prontos para usar, como serviços de autenticação e notificações (WEN et al., 2022). Já o FaaS permite que os desenvolvedores criem e implementem seus próprios códigos que serão acionados por eventos, como uma atualização no banco de dados ou uma requisição HTTP (solicitação feita por um cliente a um servidor para acessar um recurso) (SHAFIEI; KHONSARI; MOUSAVI, 2021).

2.3 *Amazon Web Services*

A *Amazon Web Services* (AWS) é uma plataforma de serviços em nuvem criada pela Amazon no ano de 2006 (AMAZON, 2023). Ela oferece um amplo conjunto de recursos e soluções tecnológicas para empresas e desenvolvedores. Segundo pesquisa realizada em 2023, a AWS é o provedor de *cloud* mais utilizado no mercado (LUXNER, 2023). Diversos estudos foram realizados sobre a infraestrutura das AWS e sua comparação a outros provedores, como *Azure* e *DigitalCloud* (TASNIM et al., 2022). A seguir, são apresentados e estudados alguns dos serviços disponíveis na AWS utilizados na implementação deste trabalho.

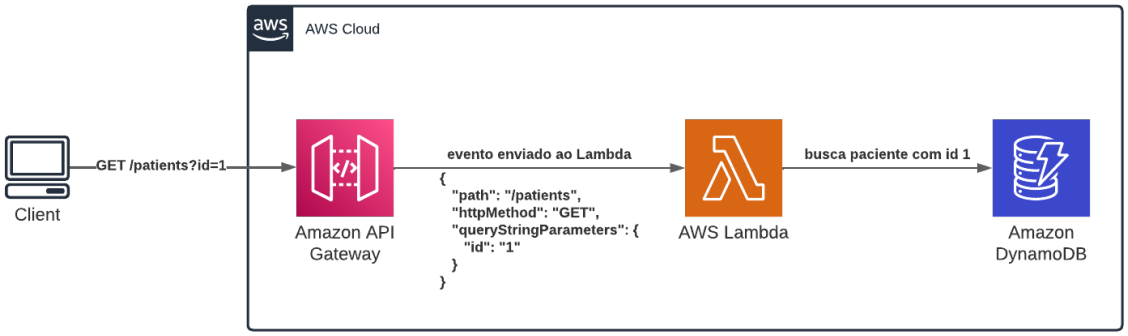
2.3.1 AWS Lambda

O AWS Lambda é um serviço de *Function as a Service* (FaaS) oferecido pela AWS (AMAZON, 2023n). Esta tecnologia permite que desenvolvedores implementem seus próprios códigos em resposta a eventos específicos, sem a necessidade de provisionar ou gerenciar servidores, tornando o processo de desenvolvimento mais eficiente, escalável e econômico. O modelo de cobrança é baseado em uso, pagando somente pelo tempo que o código é executado, tornado altamente eficiente em termos de custos para aplicações que experimentam tráfego variável (AMAZON, 2023n).

O AWS Lambda se destaca pela sua integração com outros serviços da AWS, como o Amazon S3 (AMAZON, 2023h) para o armazenamento de documentos e imagens, o Amazon DynamoDB (AMAZON, 2023e) para a criação de bancos de dados e o Amazon API Gateway (AMAZON, 2023b) para comunicação com outras aplicações. A

Figura 2.1 ilustra o processo do AWS Lambda na resposta a uma requisição feita ao Amazon API Gateway. Esse diagrama demonstra como a função Lambda é conectada a outros serviços da AWS. A informação enviada do Amazon API Gateway é utilizada como *input* da função Lambda. Com isto, o AWS Lambda processa os dados de entrada e emite uma solicitação ao banco de dados Amazon DynamoDB para buscar o dado correto. Uma das vantagens do uso do AWS Lambda, como o representado na Figura 2.1, é o fato da função somente ser acionada quando uma requisição é feita à API, resultando em uma economia de custos.

Figura 2.1 – Diagrama da arquitetura de uma função Lambda conectado ao serviço Amazon API Gateway e ao banco de dados Amazon DynamoDB.



Fonte: Próprio Autor

A estrutura do serviço do AWS Lambda é dividida em duas partes: *Control Plane* e *Data Plane* (AMAZON, 2023h). O *Control Plane* é encarregado pelo gerenciamento da função Lambda, definindo um meio de controlar seu provisionamento, manutenção e distribuição. Por outro lado, o *Data Plane* administra a execução do AWS Lambda, alocando um ambiente para seu código e executando-o. Para cada evento recebido no AWS Lambda é invocado uma nova instância da função, adicionado um tempo extra de começo lento (*Cold Start*) entre a invocação da função Lambda e o início do processamento do dado (AMAZON, 2023g).

O AWS Lambda suporta o uso diversas linguagens de programação, oferecendo flexibilidade para os desenvolvedores. Atualmente, o AWS Lambda permite o desenvolvimento de funções utilizando Node.js, Python, Java, Go, .NET (C#), Ruby, além de fornecer uma opção chamada *Runtime API* que permite o uso de qualquer linguagem adicional (AMAZON, 2023n).

Ao implementar o AWS Lambda em um sistema de gerenciamento de atendimentos psicológicos, é possível criar uma plataforma capaz de adicionar novas funcionalidades rapidamente, escalonando conforme a demanda para lidar com tráfego intenso em

horários de pico. Por exemplo, o sistema pode usar AWS Lambda para acionar funções que enviam lembretes de compromisso, atualizam prontuários de pacientes ou processam pagamentos.

2.3.2 Amazon S3

O Amazon *Simple Storage Service* (S3) consiste em um serviço de armazenamento de objetos, como imagens e documentos em nuvem (AMAZON, 2023h). O Amazon S3 é um sistema escalonável e distribuído, armazenando suas informações de forma confiável em múltiplos data centers da Amazon (AMAZON, 2023h). É oferecida uma ampla gama de recursos de gerenciamento, como políticas de ciclo de vida de objetos, criptografia e controle de versão, possibilitando o armazenamento e a recuperação de grande volume de dados.

O uso do armazenamento Amazon S3 é cobrado de acordo com a quantidade de dados salvos, possibilitando um sistema de baixo custo (AMAZON, 2023i). Como pequenas *startups* geralmente não possuem uma infraestrutura para armazenar uma grande quantidade de dados, normalmente é optado o Amazon S3 para armazenar suas imagens, vídeos e arquivos para minimizar os custos (MURTY, 2008).

Uma das vantagens do uso do Amazon S3 dentro da aplicação final é o fato de ser facilmente integrado a outros serviços oferecido pela Amazon, como, por exemplo, o Amazon RDS (AMAZON, 2023g) (serviço de banco de dados relacional da Amazon) e o AWS Lambda. O Amazon S3 também fornece serviços de hospedagem de sites, possibilitando o *upload* de páginas diretamente ao seu serviço, sendo mapeadas a um domínio público na Internet.

O Amazon S3 é dividido em *buckets*, contêineres que agrupam os arquivos e objetos armazenados (AMAZON, 2023h). Os *buckets* são gerenciados através de uma interface criada pela própria AWS, facilitando a integração do AWS Lambda com este serviço (MURTY, 2008).

2.3.3 Amazon DynamoDB

O Amazon DynamoDB é um serviço de banco de dados NoSQL oferecido pela AWS (AMAZON, 2023e). Um banco de dados NoSQL consiste no armazenamento de

dados de forma não relacional e sem esquema, permitindo que as representações de dados cresçam de forma eficaz e dinâmica (KHAN et al., 2022). O Amazon DynamoDB conta com o recurso de replicação automática e criptografia, possibilitando o acesso e armazenamento de dados de forma segura e rápida (AMAZON, 2023f). O Amazon DynamoDB pode lidar com mais de 10 trilhões de solicitações por dia e suportar picos de mais de 20 milhões de solicitações por segundo. Os dados são armazenados em SSD e replicados automaticamente dentro da região na AWS (AMAZON, 2023p).

O Amazon DynamoDB é um componente *serverless*, ou seja, a própria AWS é encarregada do provisionamento de recursos, backup de dados, criptografia e gerenciamento do *software* utilizado pelo banco de dados (ELHEMALI et al., 2022). Com isso, não são predefinidos limites para a quantidade de dados que cada tabela pode armazenar, auxiliando na escalabilidade da aplicação final.

Uma tabela dentro do Amazon DynamoDB representa uma coleção de itens, no qual cada item é composto por um conjunto de atributos e uma chave primária única. A chave primária é constituída por uma chave de partição (*partition key*, PK), utilizada na pesquisa dos itens, e uma chave opcional de classificação (*sort key*, SK), ordenando os itens na tabela (ELHEMALI et al., 2022).

2.3.4 Amazon API Gateway

O Amazon API Gateway permite que desenvolvedores criem e gerenciem APIs em grande escala (AMAZON, 2023b). Uma API (*Application Programming Interface*) é um conjunto de regras e protocolos que definem como diferentes aplicações devem interagir entre si. Ela funciona como um intermediário, permitindo que dois aplicativos se comuniquem e troquem dados de maneira padronizada, independentemente de como eles foram projetados internamente (WEN et al., 2022).

Com o Amazon API Gateway é possível gerenciar o acesso e monitoramento de APIs. O Amazon API Gateway tem a capacidade de processar centenas de milhares de transações simultâneas, suportando uma grande quantidade de acessos e requisições (AMAZON, 2023b). Além disso, o serviço possibilita a conexão a serviços de monitoramento como Amazon CloudWatch (AMAZON, 2023c) e AWS X-Ray (AMAZON, 2023o) a fim de analisar as requisições feitas pelos usuários. O Amazon API Gateway atua como "porta de entrada"para aplicações hospedadas na AWS (AMAZON, 2023b), permitindo que as mesmas sejam acessadas e consumidas de forma segura pelos usuários

finais.

O Amazon API Gateway permite a criação de uma API REST. O REST (*Representational State Transfer*) é um conjunto de convenções e protocolos para a criação e desenvolvimento de uma API (RIMAL et al., 2011). Seu objetivo é permitir a comunicação e a troca de dados entre diferentes sistemas em um formato que ambos entendem. Uma API REST opera através de métodos como `GET`, `POST`, `DELETE` e `PUT` para ler, criar, excluir e atualizar dados, respectivamente. É amplamente utilizada em serviços da *web* devido à sua simplicidade, confiabilidade e fácil integração com sistemas existentes (RIMAL et al., 2011).

2.3.5 Amazon Cognito

O Amazon Cognito oferece soluções de controle de acesso e gerenciamento de identidade, manuseando todo o processo de *login* e criação de conta de maneira segura (AMAZON, 2023d). Com o Amazon Cognito é possível autenticar e autorizar usuários, armazenando-os na própria AWS, como será implementado neste projeto. O Amazon Cognito é integrado à serviços de *login* como Google e Facebook, permitindo a autenticação de sua aplicação via estes terceiros (AMAZON, 2023a).

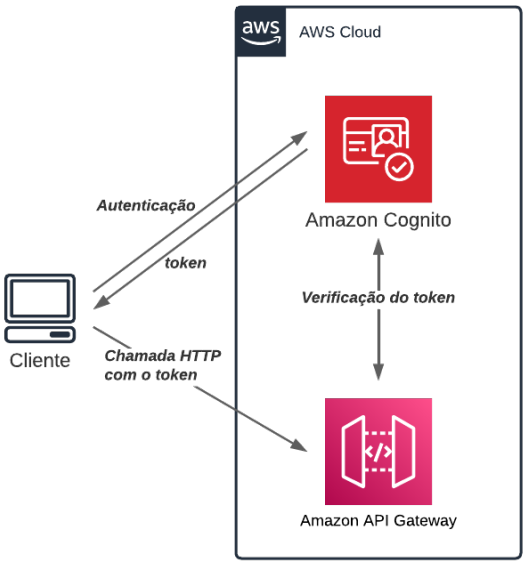
Este componente facilita o fluxo de *login* e registro da aplicação final, disponibilizando uma interface visual de *login* pronta para uso em aplicativos *web* ou *mobile*. Como o Amazon Cognito gerencia todo o fluxo de autenticação, o desenvolvedor não precisa se preocupar com requisitos de segurança e criptografia – ambos manuseados pela AWS –, possibilitando uma implementação mais segura e rápida.

Para compreender o comportamento interno do Amazon Cognito é preciso diferenciar os componentes *user pools* e *identity pools*. O *user pool* disponibiliza opções de *login* e registro para os usuários da aplicação, comportando-se como o banco de dados dos usuários existentes. Com ele, é possível checar por credenciais comprometidas e disponibilizar verificação via e-mail ou telefone (AMAZON, 2023d). O desenvolvedor consegue integrar uma função Lambda ao *user pools* para obter programaticamente todos os usuários registrados de sua aplicação. Já o *identity pool* providencia e maneja os acessos dos usuários da aplicação à serviços da AWS (AMAZON, 2023d). É possível utilizar estes dois componentes em conjuntos, ou separadamente.

O *user pool* pode ser conectado diretamente ao Amazon API Gateway para checar as permissões do usuário que está fazendo a requisição. Para isso é utilizado um *token* de

acesso (*access token*), fornecido ao usuário pelo próprio Amazon Cognito após o *login*. Quando o usuário envia uma requisição ao Amazon API Gateway este *token* é checado automaticamente pelo Amazon Cognito, verificando se o usuário em questão contém as permissões necessárias. A aplicação desenvolvida utiliza este modo de verificação, provendo um ambiente seguro para os usuários finais. A Figura 2.2 apresenta o diagrama de autenticação via o Amazon Cognito. É possível identificar que, primeiro, o usuário realizar a comunicação com o Amazon Cognito a fim de receber um *token* de acesso. Com isso, as próximas requisições realizadas pelo usuário ao serviço Amazon API Gateway apresentarão este mesmo *token*, que será checado internamente pela AWS.

Figura 2.2 – Diagrama de autenticação via Amazon Cognito e Amazon API Gateway.



Fonte: Próprio Autor

2.3.6 AWS Amplify

O componente AWS Amplify agiliza o desenvolvimento de aplicações *full-stack*, disponibilizando ferramentas para criar *back-end* de aplicativos, conectar *front-end* a recursos e hospedar aplicativos (AMAZON, 2023l). Para o desenvolvimento deste trabalho, o AWS Amplify é utilizado para hospedar o código *front-end*. Uma de suas vantagens é a criação de um fluxo de *deploy* automático em conjunto com o aplicativo *Github* (AMAZON, 2023m), facilitando o desenvolvimento e teste do *front-end*. Outra ferramenta disponibilizada pelo AWS Amplify é a implementação de componentes para o *front-end* integrados com serviços da AWS. O componente de *login* e registro conectado com o Amazon Cognito é um exemplo deste componente.

3 TRABALHOS RELACIONADOS

Neste Capítulo são analisados estudos e produtos que auxiliaram no desenvolvimento do presente projeto. Este Capítulo é divido entre trabalhos relacionados à psicologia, e trabalhos relacionados à arquitetura *serverless*. Os artigos relativos ao desenvolvimento *serverless* apresentam aplicações em diversas áreas, desde a saúde até a o envio automático de e-mails, demonstrando a versatilidade e eficácia do uso da arquitetura *serverless*.

3.1 Panorama de aplicativos direcionados a psicólogos

Nas últimas décadas, houve um aumento expressivo na incorporação de tecnologia à prática de psicologia graças aos avanços digitais (DIAS et al., 2015). Em 2012, o CFP (Conselho Federal de Psicologia) aprovou uma resolução que autorizava aos psicólogos realizar atendimento *online* (VERONA, 2012). Com isso, diversos aplicativos foram desenvolvidos e lançados para auxiliar os psicólogos em seus trabalhos. Essas aplicações abrangem uma ampla gama de funcionalidades, desde ferramentas de avaliação e diagnóstico até recursos de terapia *online*. Os novos aplicativos fornecem aos psicólogos mecanismos adicionais para melhorar a eficiência e qualidade de seu trabalho, bem como expandir seu alcance e atender a um número maior de pacientes (MüHLEN, 2020).

3.1.1 *PsicoManager*

Uma dessas aplicações é o *PsicoManager* (PSICOMANAGER, 2023), um sistema de gerenciamento de clínica *online*. Este sistema oferece uma série de recursos para os profissionais, como a gestão de agendamentos, prontuários eletrônicos e anamneses, além de funcionalidades de controle financeiro (PSICOMANAGER, 2023). Desta forma, os psicólogos podem se concentrar mais em seus pacientes e menos em tarefas administrativas, disponibilizando, ao mesmo tempo, um ambiente seguro e organizado para a gestão de suas práticas.

3.1.2 *AppHealth*

Outro exemplo é o *AppHealth* (APPHEALTH, 2022), *software* de gestão de clínicas e consultórios com prontuário eletrônico, agenda *online* e telemedicina. O *AppHealth* permite que psicólogos realizem sessões de terapia remotas, facilitando a continuidade do tratamento mesmo à distância (APPHEALTH, 2022). Além disso, oferece um prontuário eletrônico, no qual o profissional de saúde pode registrar informações do paciente, como sintomas, histórico médico, medicamentos e exames, agilizando o processo de atendimento.

Estes são apenas dois exemplos do amplo espectro de aplicativos que apoiam os psicólogos em seu trabalho profissional. Cada aplicativo apresenta uma série de funcionalidades próprias, com o objetivo de atender diferentes necessidades da prática psicológica, otimizando processos e aprimorando a qualidade do atendimento ao paciente.

3.1.3 Psight: Uma proposta de software de gestão de agendamentos para otimização de atendimentos de profissionais da psicologia

O artigo de Nishihira et al. (NISHIHIRA et al., 2020) descreve o processo de desenvolvimento de um *software* voltado para a gestão de agendamentos e otimização de atendimentos para profissionais da psicologia. O trabalho é separado em duas fases: a primeira envolve o levantamento de requisitos por meio de pesquisas com profissionais da área e a segunda consiste na coleta de dados para a definição dos requisitos do sistema.

O projeto é composto por serviços oferecidos pelo Google, como Google Docs, para o armazenamento de documentos, Google Calendar, para o agendamento de sessões e Google Auth para autenticação e autorização de acesso. O *front-end* é implementado com o uso de *frameworks* como Restify, Adonis.js e Vue.js, juntamente com HTML e CSS. Já o *back-end* é implementado em Node.js, com a linguagem Javascript.

O texto acentua principalmente o processo de desenvolvimento do *software*, passando pelas etapas de pesquisa, especificação de requisitos e classificação de prioridades. O projeto final prioriza o desenvolvimento do *front-end*, ou seja, a interface com o usuário e suas ações. O *back-end* e sua arquitetura não são estudadas profundamente no artigo citado, proporcionando uma complementação do projeto aqui desenvolvido com este trabalho relacionado.

3.2 Uso de arquitetura *serverless* em outras áreas

A arquitetura *serverless* é utilizada como meio de estudo para diversos artigos. O primeiro trabalho explora a aplicação da arquitetura *serverless* na gestão de e-mails em massa, enquanto o segundo apresenta uma implementação para gestão de robôs. Já o terceiro traz uma análise de como a computação em nuvem pode ser aplicada na gestão de sistemas de saúde.

3.2.1 *Serverless Architecture for Bulk Email Management*

No artigo proposto por (CHAND, 2021), é discutido uma arquitetura *serverless* para gerenciamento de e-mails em massa usando serviços da AWS. São destacados desafios de envio e gerenciamento de grandes volumes de e-mails propondo uma solução usando AWS Lambda e AWS SES (*Simple Email Service*). A arquitetura envolve o envio de e-mails por meio de um *endpoint*, processando-os via funções Lambda e rastreando os recibos de devolução. Também são mencionados os serviços de monitoramento de desempenho disponíveis para desenvolvedores de aplicações AWS, como Amazon CloudWatch e AWS X-Ray.

O objetivo da arquitetura é desenvolver um sistema de envio de e-mail em massa para diversos endereços com uma única operação. Para tal, é necessário os possuir endereços de e-mail dos remetentes, o assunto e o corpo do e-mail.

O processo começa com o usuário inserindo os detalhes de e-mail necessários por meio de uma interface *front-end* construída usando o *framework* ReactJS. Os dados são então enviados para o Amazon API Gateway, configurando a solicitação. O Amazon API Gateway aciona uma função Lambda, onde os parâmetros do e-mail são pré-processados e o conteúdo do e-mail é convertido em *SES templates*.

Os e-mails são então agrupados em lotes e incluídos no Amazon SQS (*Simple Queue Service*), serviço de fila em *cloud* da AWS. Outra função Lambda é acionada para enviar e-mails aos destinatários usando os templates SES criados pelo AWS Lambda anterior. Os e-mails devolvidos são enviados ao AWS SNS (*Simple Notification Service*), serviço de notificação da AWS, acionando uma função Lambda para armazenar os e-mails de retorno no Amazon DynamoDB.

Os serviços Amazon CloudWatch e AWS X-Ray são usados para monitoramento de desempenho. Os *logs* do Amazon CloudWatch fornecem informações sobre o tempo

de execução das funções Lambda, enquanto os rastreamentos do AWS X-Ray ajudam a identificar problemas de latência e otimizar seu desempenho.

3.2.2 *Serverless Architecture for Service Robot Management System*

O artigo proposto por (NISHIMIYA; IMAI, 2021) apresenta a construção de um sistema que combina arquitetura *serverless* com Internet das Coisas para o gerenciamento de robôs. O propósito destes robôs é guiar o cliente ao seu destino final em determinados ambientes, como, aeroportos e shoppings. O projeto foi testado em uma instalação real, realizando serviços com sucesso. O documento destaca o potencial de integração de robôs com tecnologias *cloud* e os benefícios de uma arquitetura *serverless*, como a facilidade de adicionar novas funcionalidades, a melhoria da capacidade computacional e redução de esforços operacionais.

A arquitetura *serverless* utilizada permite flexibilidade ao projeto, eliminando a necessidade de novo *hardware* e mantendo os padrões de segurança atualizados. A comunicação do sistema utiliza Honda RaaS (*Robotics as a Service*), plataforma para desenvolvimento e operação de serviços de robótica, que fornece comunicação de dados e serviços de identificação de robôs. O sistema utiliza o Amazon API Gateway para a comunicação do robô com os serviços da AWS. O Amazon API Gateway aciona uma função Lambda encarregada do trabalho de guia e gerenciamento de frota. É utilizado Amazon DynamoDB para banco de dados, Amazon S3 para armazenamento e AWS Step Functions (serviço de criação e gerenciamento de fluxos de trabalho) para gerenciamento de tarefas.

3.2.3 *Serverless Architecture for Healthcare Management Systems*

O artigo proposto por (KUMARI; SAHOO, 2022) propõe soluções de arquitetura *serverless* para a área da saúde em geral, explicando detalhadamente funções da plataforma de computação em nuvem AWS e seus prós e contras na implementação desta inovação para a área da saúde. O artigo menciona diversos serviços e ferramentas que podem ser usados em aplicações de saúde, apresentando vantagens da computação *serverless*, como maior velocidade de entrega, escalabilidade e melhor segurança.

No artigo, é apresentado um exemplo real de uma solução *serverless*. Este caso

de uso consiste em uma solução para gerenciar informações relacionadas à saúde entre várias partes de maneira segura, a fim de permitir o acesso à dados de pacientes em diversas interfaces, como *browsers*, celulares e *desktop* em qualquer lugar do planeta. A aplicação utiliza o Amazon Cognito para fornecer autenticação da identidade do usuário, juntamente com Amazon API Gateway para rotear as solicitações às funções Lambda. É implementado duas funções Lambda, uma para processar as solicitações dos usuários e outra para ler as atualizações e fazer alterações no banco de dados. O Amazon DynamoDB é usado para armazenar os dados de saúde não estruturados e o Amazon S3 para armazenar dados binários de recursos de saúde, como raios-X e gráficos de ECG. O serviço Amazon CloudWatch é usado para registrar as solicitações à API.

3.3 Análise comparativa

A partir da análise dos quatro artigos citados é possível observar semelhanças em relação às tecnologias utilizadas e seus propósitos. A Tabela 3.1 apresenta a linguagem de programação e o *framework* utilizado no *front-end*, se presente. Ambos artigos 3.2.1 e 3.1.3 optaram pela utilização da linguagem Javascript para o desenvolvimento do *front-end*. Já os artigos 3.2.2 e 3.2.3, por não necessitarem de uma interface de usuário na *web*, não utilizam nenhuma linguagem ou *framework*. Os projetos que usam Javascript para o *front-end* selecionaram *frameworks* para facilitar o desenvolvimento dos componentes visuais, o artigo 3.2.1 elegeu ReactJS e o 3.1.3 Vue.js.

Tabela 3.1 – Comparativo entre as linguagens de programação e *frameworks* utilizados no *front-end* dos trabalhos relacionados.

Artigo	**Linguagem de programação**	***Framework***
(CHAND, 2021)	JavaScript	ReactJS
(NISHIMIYA; IMAI, 2021)	Não informado	Não informado
(KUMARI; SAHOO, 2022)	Não informado	Não informado
(NISHIHIRA et al., 2020)	JavaScript	Vue.js

Em relação ao *back-end*, os artigos 3.2.1, 3.2.2 e 3.2.3 apresentam soluções com uma arquitetura *serverless* implementada na AWS. A Tabela 3.2 exibe uma análise comparativa dos serviços da AWS empregados no projeto final. É interessante observar que os três artigos com arquitetura *serverless* apresentam *endpoints* do Amazon API Gateway conectado ao AWS Lambda, que por sua vez, é agregado ao banco de dados NoSQL

Amazon DynamoDB. Ambos artigos 3.2.2 e 3.2.3, por necessitarem do armazenamento de objetos como imagens e documentos utilizam o serviço Amazon S3, demonstrando sua versatilidade. A fim de manter uma observação dos dados da aplicação, os projetos 3.2.1 e 3.2.3 possuem os serviços de AWS X-Ray e Amazon CloudWatch para facilitar o desenvolvimento e análise dos dados.

Tabela 3.2 – Comparativo entre os serviços da AWS utilizados na arquitetura final dos trabalhos relacionados.

Serviço da AWS	**(CHAND, 2021)**	**(NISHIMIYA; IMAI, 2021)**	**(KUMARI; SAHOO, 2022)**
Amazon API Gateway	X	X	X
Amazon Lambda	X	X	X
Amazon DynamoDB	X	X	X
AWS SNS	X		
Amazon CloudWatch	X		X
Amazon SQS	X		
AWS SES	X		
AWS Step Functions		X	
Amazon S3		X	X
AWS IoT Core		X	
Amazon Cognito			X
AWS X-Ray	X		X

4 DESENVOLVIMENTO E IMPLEMENTAÇÃO

Neste Capítulo, é apresentada a construção e o detalhamento da arquitetura *serverless* para uma aplicação de gerenciamento de atendimentos psicológicos. No início do Capítulo, são demonstrados os requisitos do sistema, definindo as características que o sistema deve ter para garantir que ele atenda as suas necessidades e objetivos previstos. Ao compreender o que o sistema deve fazer, é possível estabelecer uma estrutura e componentes necessários.

Dessa forma, são exploradas as classes que definem cada entidade da aplicação e suas relações. Cada classe representa um elemento distinto do sistema com suas próprias responsabilidades e operações.

Por fim, são discutidos os componentes e fluxos do sistema. Os componentes são as partes individuais que compõem o sistema, com os fluxos referindo-se à maneira como as informações e as operações se movimentam através do sistema. Cada componente e fluxo foi projetado para garantir que a aplicação atenda as suas necessidades e requisitos.

4.1 Requisitos

A definição de requisitos funcionais e não funcionais é uma etapa fundamental no processo de desenvolvimento de *software*, fornecendo a base para o *design* e implementação do sistema. Esses requisitos indicam o funcionamento do sistema (requisitos funcionais) e seu comportamento (requisitos não funcionais), sendo essenciais para garantir que o produto atenda às necessidades da organização ou do usuário final (TAVEIRO, 2016).

4.1.1 Requisitos Funcionais

Os requisitos funcionais consistem em descrições detalhadas das atividades da aplicação, como criar conta, adicionar dados ou salvar arquivos. São diretamente ligados às expectativas e necessidades dos usuários, sendo essenciais para compreender e definir o escopo do projeto (TAVEIRO, 2016). Os requisitos funcionais foram adquiridos através de conversas com um psicólogo, compreendendo suas necessidades e as funcionalidades esperadas do sistema. Na Tabela 4.1 é apresentada a lista dos requisitos funcionais do

projeto final.

Tabela 4.1 – Tabela dos requisitos funcionais para o desenvolvimento da plataforma final.

Código	**Requisito Funcional**	**Ator**	**Caso de Uso**
RF001	Efetuar Login	Usuário	Entrar na aplicação com e-mail e senha.
RF002	Gerenciar conta	Usuário	Criar, editar e excluir sua conta do sistema.
RF003	Gerenciar pacientes	Usuário	Criar, editar e excluir pacientes de sua conta.
RF004	Listar pacientes	Usuário	Listar todos seus pacientes.
RF005	Buscar paciente	Usuário	Buscar um paciente específico pelo seu identificador.
RF006	Gerenciar sessões	Usuário	Criar, editar e excluir sessões para cada paciente de sua conta.
RF007	Listar sessões	Usuário	Listar todas as sessões de um paciente de sua conta.
RF008	Buscar sessões	Usuário	Buscar uma sessão pelo seu identificador de um paciente de sua conta.
RF009	Fazer upload de documentos	Usuário	Atribuir documentos a cada paciente, como laudos e anotações.
RF010	Ler documento	Usuário	Ler o documento carregado no sistema.

4.1.2 Requisitos não Funcionais

Os requisitos não funcionais descrevem qualidades e atributos do sistema, como desempenho, segurança e usabilidade. Eles não estão diretamente relacionados com funções ou serviços específicos do sistema, mas sim com o comportamento da aplicação em geral (TAVEIRO, 2016). A Tabela 4.2 enumera os requisitos não funcionais da plataforma.

A importância de definir ambos os requisitos, funcionais e não funcionais, é alinhar a arquitetura do projeto às necessidades do usuário final. Com as definições dos requisitos é possível implementar um arquitetura que atenda às condições e desempenhos esperados, permitindo avaliar a qualidade do sistema e estabelecer critérios de aceitação.

Tabela 4.2 – Tabela dos requisitos não funcionais para o desenvolvimento da plataforma final.

Código	Requisito não Funcional	Categoria
RN001	Dada a natureza sensível das informações em um ambiente psicológico, a segurança deve ser uma preocupação primordial. O aplicativo deve seguir os padrões de segurança para proteger os dados dos pacientes.	Segurança e privacidade dos dados
RN002	O aplicativo deve fornecer tempo de resposta rápido, mesmo se houver um grande número de usuários simultâneos.	Desempenho
RN003	O sistema deve minimizar o uso de recursos computacionais e otimizar o processo de cobrança a fim de garantir um custo operacional baixo.	Eficiência de custo
RN004	A aplicação deve funcionar de maneira confiável, sem falhas ou interrupções de serviço frequentes.	Confiabilidade
RN005	A plataforma precisa ser capaz de suportar mais usuários e maior carga de trabalho à medida que o número de psicólogos e pacientes aumenta.	Escalabilidade

4.2 Classes

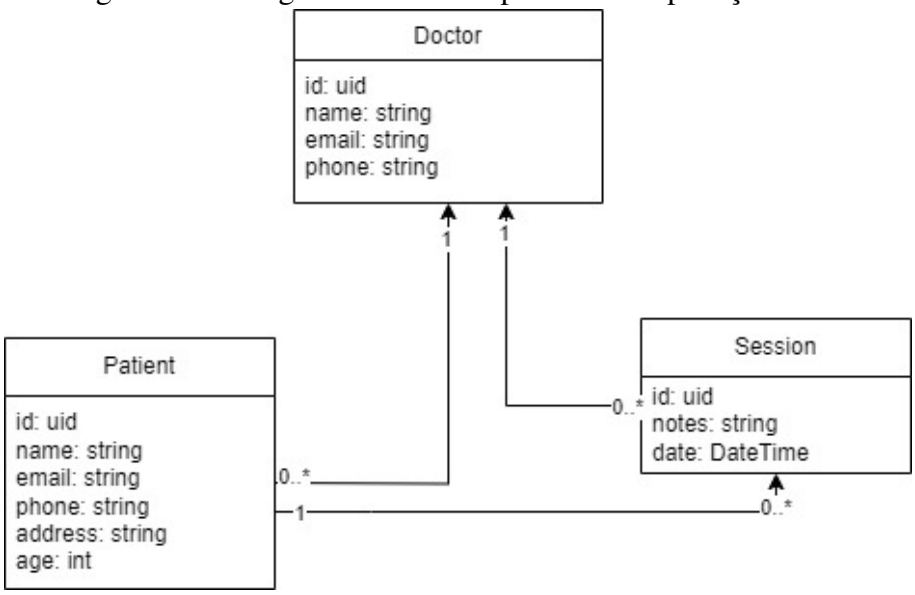
Com a definição dos requisitos iniciais, é importante estabelecer as classes de cada componente da aplicação e como estes são relacionados entre si. De acordo com os requisitos funcionais, três classes diferentes são esperadas para o funcionamento completo da aplicação. É possível dividir os objetos entre `Doctor`, `Patient` e `Session`.

A classe `Doctor` representa a entidade de um psicólogo, o usuário final, contendo um nome completo, um e-mail (único na plataforma) e um número de telefone. A criação de um usuário na plataforma simboliza a criação de uma nova instância do objeto `Doctor`. A entidade `Doctor` possui zero ou mais `Patient` e `Session`, representando cada um dos pacientes e sessões do psicólogo.

A entidade `Patient`, por sua vez, representa um paciente. Cada paciente é composto por um nome completo, e-mail (também único), telefone, endereço e idade. Cada paciente é obrigatoriamente relacionado a um `Doctor`, podendo ter zero ou mais `Session`, para cada sessão de terapia.

Por último, a entidade `Session` caracteriza uma sessão, a qual relaciona um único `Doctor` a um único `Patient`. Cada sessão contém notas e uma data. A Figura 4.1 apresenta a relação das classes presentes no projeto final bem como suas relações.

Figura 4.1 – Diagrama de classes presente na aplicação final.



Fonte: Próprio Autor

4.3 Componentes e Fluxos

Nesta seção, são explorados os componentes da AWS utilizados para o desenvolvimento deste projeto, bem como os fluxos da aplicação. Cada componente desempenha um papel fundamental na arquitetura, desde o armazenamento de dados até a execução de funções de *back-end* e gerenciamento de usuários. Com isso, são descritos cada um desses componentes, ilustrando suas funções específicas dentro do sistema. Além disso, são explicados os fluxos da aplicação, apresentando como as solicitações dos usuários são processadas pelo sistema e como os dados se movem entre os diferentes componentes.

4.3.1 AWS Amplify

O componente AWS Amplify é encarregado por hospedar a interface do usuário da plataforma final, compilando e realizando o *deploy* do código *front-end*. Ao criar um *app* no AWS Amplify pelo *console* da AWS é possível selecionar em qual ferramenta o código fonte será hospedado, como Github, Bitbucket, AWS CodeCommit e outros. Para o projeto final, o serviço escolhido para criar o repositório do *front-end* foi o Github, por sua facilidade de integração com a AWS.

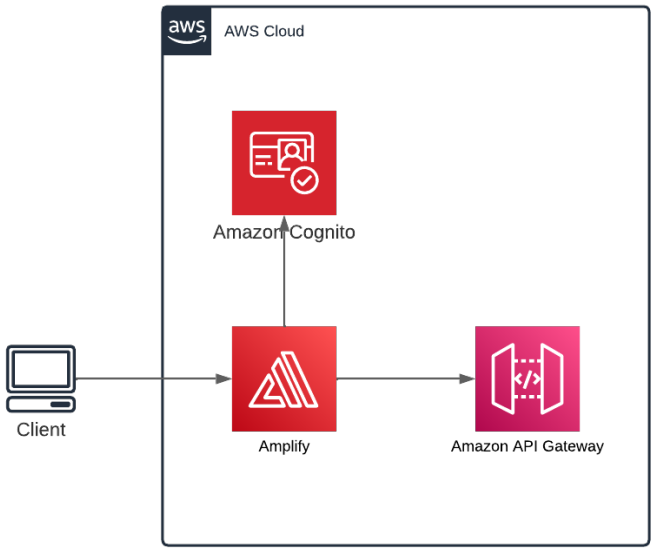
Ao conectar com o Github, qualquer modificação no código reflete em um novo artefato de compilação, permitindo um *front-end* sempre atualizado. O código *front-end*

apresenta somente uma forma de aprovação visual do projeto, não representando o foco do trabalho final.

Uma das principais vantagens da utilização do serviço AWS Amplify é a facilidade de *deploy* e a presença de componentes *cloud-connected*, isto é, componentes pré-conectados com serviços da AWS. Por exemplo, para a aplicação final foi utilizado um componente visual conectado ao Amazon Cognito encarregado de manusear o *login* e criação de conta do projeto. Os componentes, criados pela própria AWS, podem ser encontrados em (AMAZON, 2023k).

A Figura 4.2 apresenta o diagrama dos componentes conectados ao AWS Amplify. Por ser o componente de hospedagem da interface com o usuário, este serviço é conectado diretamente ao usuário final (representado pelo ícone de computador na Figura). O AWS Amplify, através dos componentes *cloud-connected*, é relacionado com o serviço de autenticação e autorização Amazon Cognito, gerenciando a criação de novos usuários e a tela de *login*. O código *front-end* é encarregado de realizar as solicitações HTTP para o componente Amazon API Gateway, por isso o componente AWS Amplify é conectado ao Amazon API Gateway no diagrama.

Figura 4.2 – Diagrama dos componentes da AWS conectados ao AWS Amplify na arquitetura final do projeto.



Fonte: Próprio Autor

4.3.2 Amazon Cognito

O Amazon Cognito é o serviço moldado para facilitar a autenticação, autorização e gerenciamento de usuários para aplicações em ambiente *web* e *mobile*. A principal funcionalidade deste serviço utilizado no aplicativo é o *pool* de usuários, responsável pelo processo de cadastro e armazenamento dos usuários.

O Amazon Cognito desempenha um papel vital ao garantir que apenas usuários autenticados possam acessar os respectivos dados e recursos. Ao criar o serviço Amazon Cognito na plataforma AWS é necessário selecionar como o usuário realizará o *login*, podendo ser com um e-mail, *username*, ou telefone. Na aplicação foi selecionado o *login* via e-mail para facilitar o cadastro dos psicólogos.

Após, é possível decidir quais atributos são obrigatórios para a criação de um novo usuário na aplicação. A fim de cumprir todos os requisitos da classe `Doctor`, é selecionado como obrigatório um nome, sobrenome, e-mail e número de telefone. É importante ressaltar que todo armazenamento de senha dos usuários é manuseado internamente pelo componente Amazon Cognito, facilitando o desenvolvimento da plataforma e garantindo uma aplicação segura.

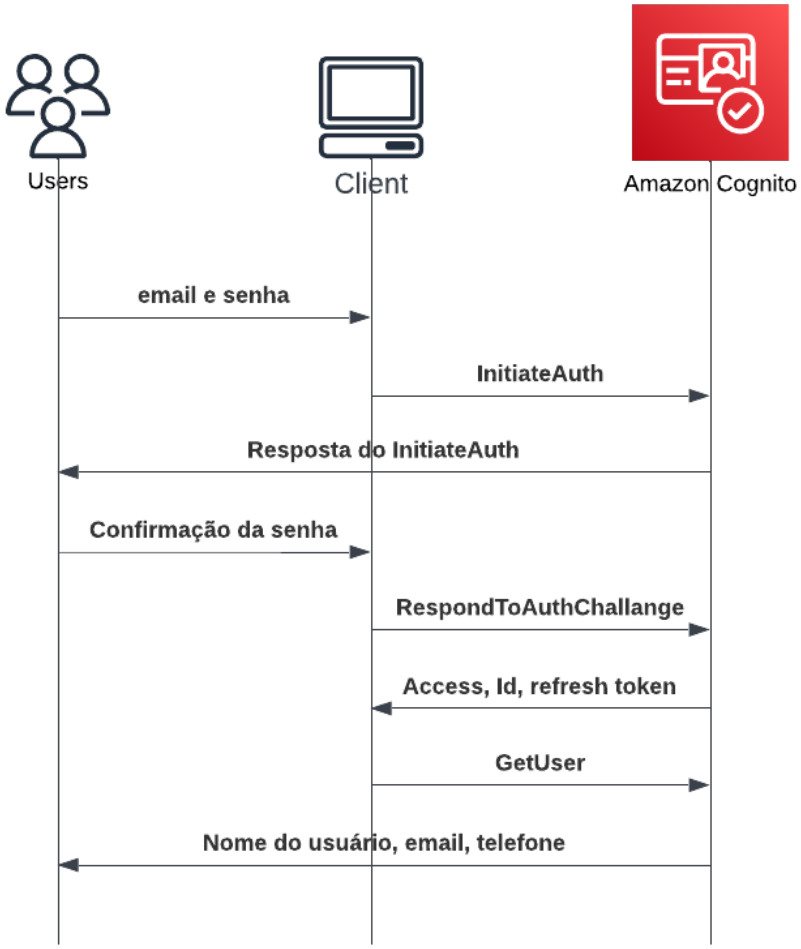
Quando um usuário realiza um *login*, ocorrem as seguintes solicitações ao Amazon Cognito:

1. `InitiateAuth`. Essa requisição é utilizada para iniciar a autenticação do usuário. A solicitação contém detalhes como o identificador do *pool* de usuários, o tipo de fluxo de autenticação e suas credenciais (AMAZON, 2023n);
2. `RespondToAuthChallenge`. Esse método é usado como confirmação no processo de *login* (AMAZON, 2023k). No projeto, este passo realiza a revalidação da senha;
3. `GetUser`. Essa é uma requisição feita ao *endpoint* `https://cognito-idp.us-east-1.amazonaws.com` utilizando o *token* retornado da etapa anterior. O objetivo desta chamada é buscar todas as informações do usuário atual, como nome de usuário, e-mail e telefone (AMAZON, 2023n).

A Figura 4.3 apresenta o fluxo de autenticação via e-mail e senha utilizado no aplicativo com os termos esclarecidos acima. Se a autenticação do usuário for bem-sucedida, o Amazon Cognito retornará um resultado que inclui o *token* de identidade, de acesso e

de atualização.

Figura 4.3 – Fluxo dos dados entre o usuário, cliente e o serviço Amazon Cognito no processo de login na aplicação.



Fonte: Próprio Autor

- *Token* de identidade (`IDToken`). Fornecido para o cliente após o processo de autenticação, contendo informações sobre a identidade do usuário, como o nome, e-mail e telefone;
- *Token* de acesso (`AccessToken`). Usado para autorizar o acesso à API ou recurso específico, identificando o usuário no *back-end* da aplicação. Ele contém detalhes sobre o `id` único do usuário e suas permissões. O `AccessToken` tem um tempo de vida limitado, geralmente de uma hora, após a qual precisa ser renovado;
- *Token* de atualização (`RefreshToken`). Solicita novos *tokens* automaticamente, facilitando a usabilidade do usuário final.

É importante ressaltar que o `AccessToken` é aplicado no *header* das requisições HTTP que necessitam de autenticação. Sendo utilizado pelo Amazon API Gateway para

determinar qual usuário está realizando a requisição e quais as suas permissões. O funcionamento do Amazon API Gateway e como é checado o `AccessToken` será explicado na seção 4.3.3.

4.3.3 Amazon API Gateway

O Amazon API Gateway é o serviço da AWS que facilita o desenvolvimento, a implementação e o gerenciamento de APIs de maneira segura e escalável, sendo responsável por processar as solicitações HTTP e encaminhá-las para a função Lambda adequada. O Amazon API Gateway atua como a "porta de entrada" para o *back-end* da aplicação desenvolvida, uma vez que todas as requisições dos usuários passam por ela (AMAZON, 2023b).

As requisições HTTP são feitas através do AWS Amplify, facilitando a conectividade do *front-end* com o *back-end*. Ao criar uma API é necessário escolher um tipo de API REST, dentre HTTP API e REST API. Embora ambas permitam criar APIs para a aplicação final, existem diferenças em relação à funcionalidade, desempenho, uso previsto e preço.

A HTTP API é uma opção mais simples e econômica para criar APIs em nível de produção, oferecendo menor latência comparado à REST API. A HTTP API é ideal para processar *payloads* de solicitações mais simples e para se integrar diretamente com os serviços da AWS (Amazon, 2023).

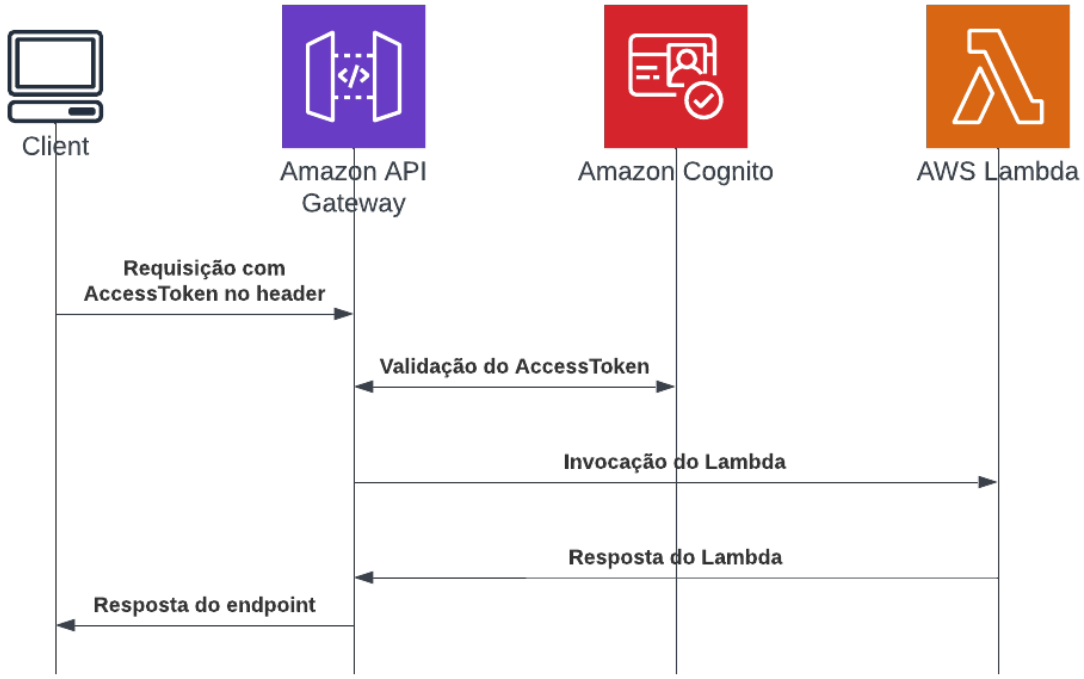
Por outro lado, a HTTP API não oferece todas as características disponíveis para a REST API que serão utilizadas no projeto. A REST API possibilita testar as invocações da API diretamente pela plataforma AWS, auxiliando o desenvolvimento de suas funcionalidades. A existência de armazenamento em *cache* das respostas dos *endpoints* — não existente na configuração HTTP API — reduzem o número de chamadas à API, melhorando a latência e precificação da aplicação (AMAZON, 2023d).

4.3.3.1 ***Autorização***

A conexão entre o Amazon API Gateway e o Amazon Cognito via *Cognito user pools authorizer* é fundamental para a segurança do sistema, funcionando como um sistema de autenticação que protege os *endpoints* da API, assegurando que somente usuários autorizados possam acessar esses recursos.

Após realizar o *login* na plataforma, todas as requisições ao Amazon API Gateway são enviadas com um *token* de acesso (`AccessToken`) no cabeçalho da solicitação. Quando a requisição é recebida pela API, é checado este cabeçalho utilizando um *authorizer*. O *authorizer* é encarregado de descriptografar o *token* de acesso e está configurado para se comunicar diretamente com o *user pool* (banco de dados dos usuários existentes da plataforma) do Amazon Cognito. O *user pool*, então, verifica a assinatura do *token* e valida se o *token* ainda não expirou. Se a validação for bem-sucedida, o *pool* de usuários retorna as reivindicações do *token* para o usuário, permitindo ou negando o acesso ao *back-end* (AMAZON, 2023a). A Figura 4.4 apresenta o fluxo de validação das requisições na aplicação.

Figura 4.4 – Fluxo dos dados entre cliente, Amazon API Gateway, Amazon Cognito e AWS Lambda em requisições autenticadas.



Fonte: Próprio Autor

4.3.3.2 ***Rotas***

A fim de compreender o comportamento do Amazon API Gateway no sistema final, é necessário apresentar a estrutura dos *endpoints* criados, bem como seus métodos e respostas. Os *endpoints* são divididos em três áreas: `Doctor`, `Patients` e `Sessions`.

1. **`Doctor`**

   A subdivisão relacionada ao `Doctor` é encarregada de gerenciar o perfil do usuário

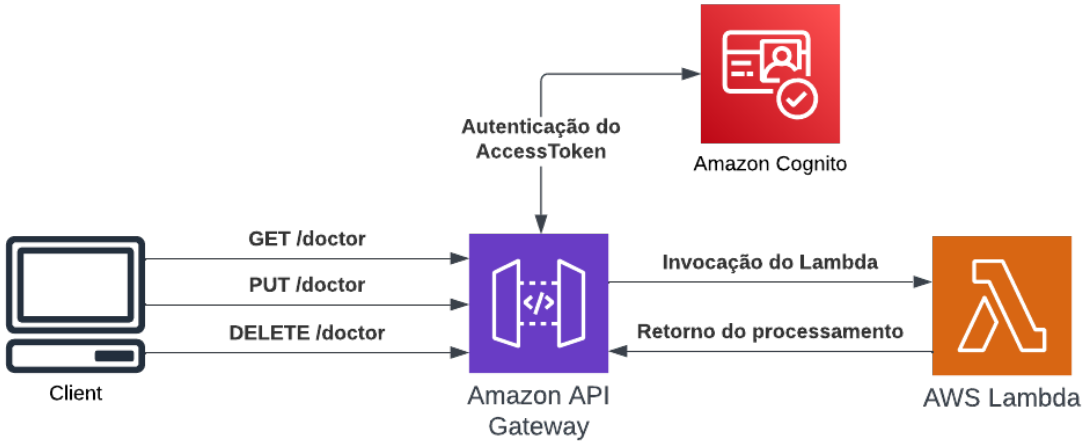
logado na plataforma, ou seja, o psicólogo que está utilizando a aplicação. Utilizando os *endpoints* relacionados ao `Doctor`, o usuário é capaz de buscar as informações do seu próprio usuário, atualizar seus dados pessoas e deletar seu usuário. Todos estes *endpoints* são autenticados via o `AccessToken` enviado no *header* da requisição.

- **Consultar seu usuário**: (`GET /doctor`) esta rota é encarregada de acessar e buscar as informações do usuário logado na aplicação. Com o `AccessToken` enviado na requisição, é possível identificar o usuário no *user pool*, sendo permitido buscar todas as informações do usuário, como o nome, sobrenome, e-mail e telefone. Caso a requisição não contenha um `AccessToken` válido, o *endpoint* retorna o código de erro `401 Unauthorized` ao *front-end*. Se a requisição for bem-sucedida, é retornado o código `200 OK`, juntamente com as informações do usuário em formato JSON (padrão de intercâmbio de dados entre cliente e servidor; sigla de *JavaScript Object Notation*).
- **Atualizar seu usuário**: (`PUT /doctor`) rota utilizada para atualizar o usuário logado na plataforma. Para sua utilização correta, é necessário enviar no *body* da requisição as informações do usuário com os dados a serem atualizados. Caso a requisição seja enviada com algum campo do *body* equivocado, ou faltante, é retornado o código de erro `400 Bad Request`, indicando ao usuário que a atualização foi falha. Assim como o `GET` apresentado no item acima, caso o *token* enviado no *header* seja inválido, é retornado `401 Unauthorized` ao *front-end*.
- **Deletar seu usuário**: (`DELETE /doctor`) esta rota consiste na deleção do usuário. O comportamento desta rota consiste em remover o usuário do *pool* de usuários do Amazon Cognito, retornando o código `200 OK` se bem-sucedido. O usuário a ser deletado é identificado pelo `AccessToken` enviado no *header* da requisição.

A fim de abranger todos esses comportamentos diferentes dos *endpoints* do `Doctor`, é necessário um processamento dos dados enviados nas requisições. Neste momento, é utilizado a primeira função Lambda do projeto. Esta função, chamada de Lambda Doctor (mais na seção 4.3.4), é encarregada de receber as rotas `/doctor`, processá-las e enviá-las ao destino correto, para, após, retornar os

dados ao Amazon API Gateway. Na Figura 4.5, é apresentado o fluxo e os *endpoints* da subdivisão `Doctor`.

Figura 4.5 – Diagrama das requisições `GET`, `PUT` e `DELETE` à rota `/doctor`, partindo do cliente, passando pelos componentes Amazon API Gateway, Amazon Cognito e AWS Lambda.



Fonte: Próprio Autor

2. **`Patients`**

   Os *endpoints* com o caminho `/patients` são relacionados às propriedades dos pacientes, podendo criar, editar, ou deletar um paciente. Assim como os *endpoints* associados ao `Doctor`, estas rotas também requisitam o *token* de acesso para autenticação da requisição.

   - **Consultar todos pacientes**: (`GET /patients`) este *endpoint* é encarregado de buscar os dados de todos os pacientes do psicólogo logado no aplicativo. Através do uso de uma função Lambda, esta requisição acessa o banco de dados Amazon DynamoDB, buscando todos os pacientes do psicólogo autenticado (mais sobre o acesso ao banco de dados na seção 4.3.7). A identificação do psicólogo logado é realizada com a decodificação do *token* de acesso, o qual contém o identificador único do usuário. Com este identificador único, é possível buscar somente os pacientes do psicólogo correto, garantindo a proteção dos dados dos outros pacientes. Se a requisição for bem-sucedida, é retornado o código `200 OK` juntamente com a lista dos pacientes para o *front-end*.
   - **Consultar paciente por `id`**: (`GET /patients/:id`) este *endpoint* é responsável pela busca dos dados de um único paciente. O paciente a ser retornado é identificado pelo caminho indicado na rota. Por exemplo, caso o

usuário chame o *endpoint* `/patients/3` é buscado o paciente cujo `id` é 3. O identificador do paciente é atribuído dentro da função Lambda encarregada. Assim como a rota `/patients`, o *token* de acesso indicado no cabeçalho da requisição é utilizado para autorizar o psicólogo. Para o retorno bem-sucedido do *endpoint*, é necessário que paciente informado na rota pertença ao psicólogo assinalado no *token*. O *endpoint* retorna `200 OK`, juntamente com os dados do paciente (nome completo, e-mail, telefone, endereço e idade). Caso o psicólogo indicado no *token* de acesso seja diferente do psicólogo do paciente na rota, é retornado o erro `401 Unauthorized`.

- **Criar paciente**: (`POST /patients`) rota é utilizada para a criação de um novo paciente. É necessário indicar as informações do novo paciente no corpo da requisição em formato JSON, contendo os dados de nome, sobrenome, e-mail, telefone, endereço e idade. Destes dados somente a idade é identificada como numeral, todas outras informações são enviadas como *string*. O *token* de acesso é utilizado para a atribuição do identificador do psicólogo responsável por este novo paciente. Após a autorização do *token* de acesso, esta requisição é enviada para uma função Lambda específica, a qual é responsável pela atribuição do identificador único do psicólogo ao objeto `Patient` a fim de incluir o novo paciente na tabela no banco de dados. Caso a criação do novo paciente seja bem-sucedida, o Amazon API Gateway retorna ao *front-end* o código `201 Created`, juntamente com o novo paciente. Se o objeto enviado no campo *body* da requisição for inválido, é retornado `400 Bad Request`.
- **Editar paciente**: (`PUT /patients/:id`) a edição do paciente é realizada através desta rota. O paciente a ser editado é indicado na requisição do próprio *endpoint*. Assim como o método de buscar um paciente com o `id`, é necessário que o psicólogo indicado no *token* de acesso seja relacionado ao paciente indicado na chamada do *endpoint*. No campo *body* da requisição é necessário conter os dados atualizados do paciente, uma vez que estas informações substituirão os dados no Amazon DynamoDB. Os dados indicados no campo são verificados no AWS Lambda específico para a edição de somente valores válidos. É retornado `400 Bad Request` caso a requisição contenha dados inválidos; `200 OK` com os dados atualizados, caso requisição seja bem-sucedida; `401 Unauthorized`, caso *token* de acesso seja inválido ou

não pertença ao psicólogo do paciente a ser editado.

- **Deletar paciente**: (`DELETE /patients/:id`) esta rota é responsável pela remoção de um paciente da aplicação. A deleção de um paciente consiste na exclusão de seus dados do banco de dados. É necessário apontar o paciente a ser deletado no próprio caminho da requisição, juntamente com o *token* de acesso no *header*. O código `204 No Content` é retornado ao *front-end* caso a exclusão seja bem-sucedida. Não é necessário adicionar nenhum dado do paciente no *body* da requisição, e nenhum dado é retornado ao *front-end* além do código de erro.

3. **`Sessions`**

   Os *endpoints* relacionados às sessões de terapia são identificados através do caminho `/patients/:id/sessions`. Ao contrário dos *endpoints* apresentados acima, as rotas das sessões encontram-se dentro do caminho `/patients`. A determinação do `id` é necessária para a busca das sessões divididas por paciente, facilitando a pesquisa no banco de dados. O *token* de acesso é obrigatório em todas as requisições das sessões.

   - **Buscar sessões por paciente**: (`GET /patients/:id/sessions`) a partir da indicação do `id` do paciente, é retornado todas suas sessões. A criação de um *endpoint* somente para buscar os dados das sessões é importante uma vez que esta requisição aciona outra função Lambda, individualizando sua operação e permitindo que cada função tenha somente um propósito. É retornado ao *front-end* uma lista com as sessões do paciente indicado na requisição, com o código `200 OK`.
   - **Buscar sessão por paciente**: (`GET /patients/:id/sessions/:id`) para a utilização desta rota, é necessário indicar o `id` do paciente e da sessão específica. É validado os identificadores da requisição, checando se o `id` da sessão pertence a alguma sessão válida do paciente assinalado. Caso a sessão não seja do paciente indicado, é retornado `401 Unauthorized` ao *front-end*. Se a requisição for bem-sucedida, é apresentado a sessão buscada, contendo seu identificador, notas e data.
   - **Criar sessão**: (`POST /patients/:id/sessions`) esta rota é encarregada pela criação de uma sessão associada a um paciente. Os dados da sessão devem ser passado no campo *body* da requisição, contendo a data da sessão e

suas notas. O paciente é associado via o identificador único passado no caminho da requisição. Ao utilizar este *endpoint* é acionado uma função Lambda designada à criação do objeto `Session` no banco de dados da aplicação. Ao obter uma requisição bem-sucedida é retornado ao *front-end* o código HTTP `201 Created`, juntamente com a nova sessão criada.

- **Editar sessão**: (`PUT /patients/:id/sessions/:id`) dispondo o identificador único do paciente e da sessão, é possível atualizar os dados de uma sessão. É obrigatório, para o uso correto do endpoint, indicar um `id` de uma sessão pertencente ao paciente adequado, juntamente com os dados atualizados da sessão no campo *body*. Ao chamar este *endpoint*, é verificado se a sessão informada pertence ao paciente e ao psicólogo (obtido via *token* de acesso) indicado. É comunicado ao *front-end* `200 OK` caso a requisição não tenha erros; `401 Unauthorized` caso alguma verificação de identidade falhe; e `400 Bad Request` caso o campo *body* esteja mal formado.
- **Deletar sessão**: (`DELETE /patients/:id/sessions/:id`) esta rota exclui a sessão informada no caminho da requisição. Ao chamar este *endpoint*, é realizado as verificações de identidade, retornado `401 Unauthorized` se alguma falha. O AWS Lambda conectado a este *endpoint* é responsável pela deleção da sessão do banco de dados. É retornado o código HTTP `204 No Content` ao *front-end* se a deleção for bem-sucedida.

4. **`Upload`**

   Para atingir o requisito funcional relacionado ao *upload* de arquivos, como laudos e receitas, é necessário a criação de *endpoints* específicos para estas funcionalidades. O fluxo de requisição continua com o mesmo padrão apresentado acima, sendo utilizado a mesma instância do Amazon API Gateway a fim de centralizar a comunicação do *back-end* com o *front-end*.

   - **Carregar arquivo de paciente**: (`POST /patients/:id/upload`) a diferença deste *endpoint* para os demais do `/patients` é a inclusão do sufixo `upload` à rota. Para o carregamento de arquivos à plataforma, é necessário indicar no campo *body* desta requisição o arquivo em formato *base64* (modelo de codificação de arquivos comumente utilizado para o envio de arquivos via HTTP). O AWS Lambda conectado a este *endpoint* é diferente dos outros *endpoints* relacionados ao `Patients`, uma vez que a função Lambda necessita

decodificar o arquivo enviado e encaminhá-lo ao Amazon S3. A resposta deste *endpoint* é `201 Created`, caso o armazenamento seja bem-sucedido. É importante ressaltar que o *token* de autorização enviado no *header* da requisição também é verificado.

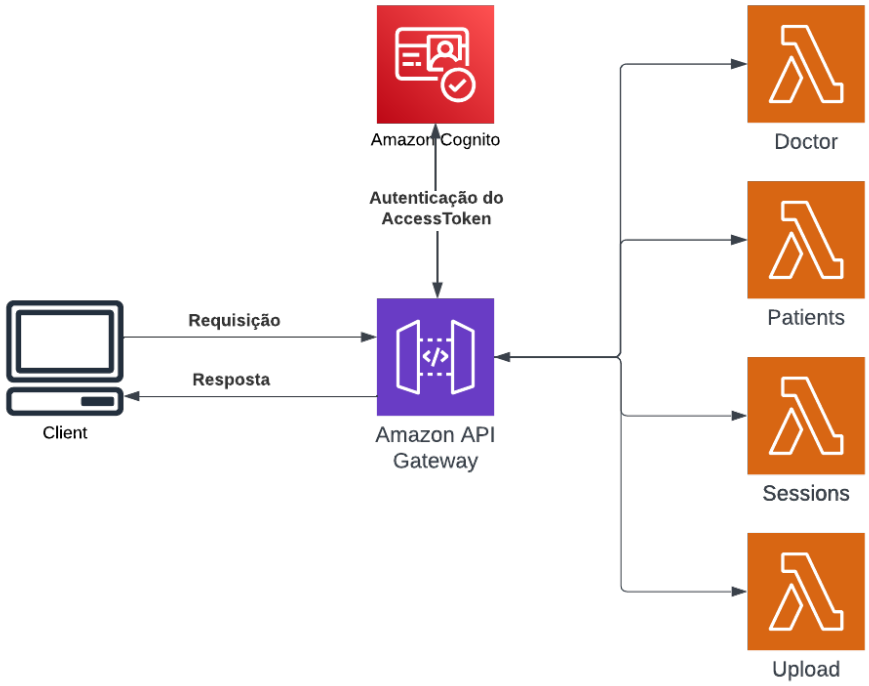
- **Buscar arquivos de paciente**: (`GET /patients/:id/upload/:id`) para realizar a busca de um arquivo no Amazon S3, é necessário realizar uma requisição informando o identificador do paciente, do arquivo e do psicólogo (via `AccessToken`). Com todos estes dados existentes, é possível realizar a leitura dos dados do arquivo do Amazon S3 e retorná-lo ao front-end em formato *base64*. Caso o documento não existir no Amazon S3, é retornado o código de erro `400 Bad Request` ao usuário final.
- **Buscar os metadados de todos os arquivos de paciente**: (`GET /patients/:id/upload`) como é necessário informar o identificador do arquivo para realizar a sua leitura (utilizando o *endpoint* descrito acima), é preciso a existência de um *endpoint* para retornar todos os metadados, como nome, tamanho e formato dos arquivos de um certo paciente. Com o nome de todos os arquivos, o usuário final consegue requisitar os dados de cada arquivo individualmente, sem a necessidade de transitar dados de arquivos desnecessários.

A Figura 4.6 apresenta o diagrama dos serviços do *back-end* da aplicação conectado ao Amazon API Gateway. Cada uma das áreas (`Doctor`, `Patients`, `Sessions` e `Upload`) utiliza sua própria função Lambda, a fim de separar e individualizar a lógica — cada função Lambda é responsável por um único domínio da aplicação —. É importante ressaltar que é responsabilidade da API separar cada requisição e encaminhar para o AWS Lambda designado. Com o propósito de compreender o fluxo inteiro da plataforma final, é necessário um estudo sobre cada uma das funções Lambdas apresentadas na Figura 4.6.

4.3.4 Lambda Doctor

O Lambda Doctor é fundamental para o comportamento da aplicação. Ele é responsável pela gestão dos usuários, ou seja, dos psicólogos, associados à plataforma. Como apresentado na seção anterior, esta função Lambda é conectada diretamente ao

Figura 4.6 – Diagrama dos serviços do *back-end* conectados ao Amazon API Gateway utilizado na aplicação.



Fonte: Próprio Autor

Amazon API Gateway, recebendo como *input* as requisições validadas com o caminho `/doctor`, juntamente com o *token* de acesso no cabeçalho. Sua função é identificar o tipo de requisição feita — `GET`, `PUT` ou `DELETE` — e encaminhar a ação correta ao Amazon Cognito.

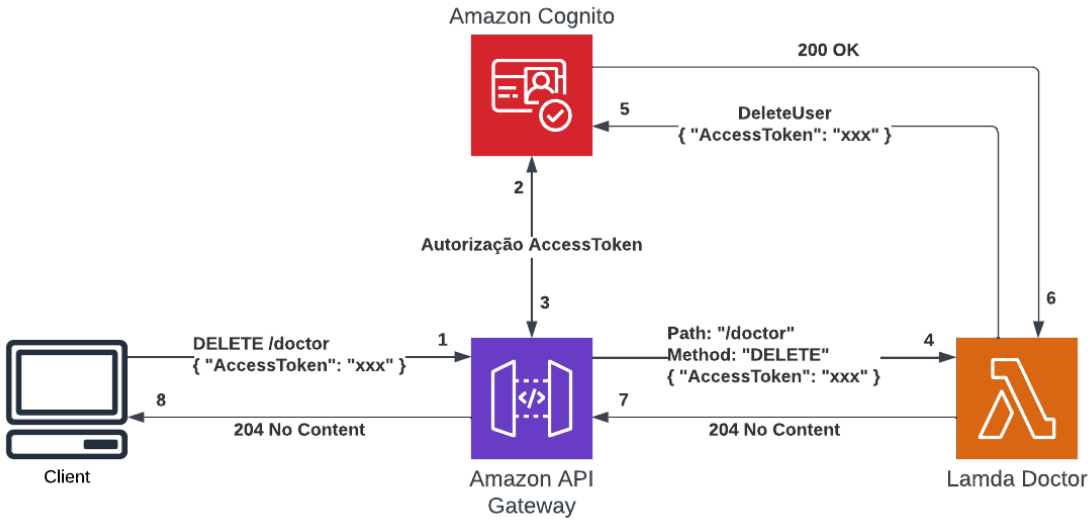
O *input* do Lambda Doctor consiste em um objeto em JSON contendo todas as informações da requisição feita ao Amazon API Gateway. Com este objeto, é possível identificar o método HTTP da requisição, seu *header*, *body* e *token* de acesso. O primeiro passo da função Lambda é ler a informação do método utilizado e agir de acordo necessário. Por isso é possível separar o comportamento do AWS Lambda em três partes:

1. `GET /doctor`. Após a função Lambda identificar que o método contido no objeto de *input* consiste em um `GET`, é necessário acessar o Amazon Cognito requisitando os dados do usuário. A comunicação com o Amazon Cognito é realizada a partir do método `GetUser` disponibilizado na API de acesso do Amazon Cognito (AMAZON, 2023e). Para o uso deste método, é necessário apresentar o *token* de acesso do usuário. Para isto, o `AccessToken` recebido no objeto de *input* é incluído na requisição do `GetUser`. O Amazon Cognito retorna à função Lambda um código HTTP, indicando sucesso ou falha, que, por sua vez é identificado e retornado à

API.

2. `PUT /doctor`. Para o método `PUT`, o Lambda Doctor aproveita do método `UpdateUserAttributes` da API do Cognito (AMAZON, 2023m). Assim como a requisição apresentada acima, o `UpdateUserAttributes` também exige o envio do `AccessToken` do usuário. A fim de identificar quais os dados serão atualizados, é necessário a inclusão de um novo objeto, `UserAttributes`, na requisição ao Amazon Cognito contendo pares chave-valor dos atributos do usuário atualizado. O Amazon Cognito, então, retorna o usuário com os campos atualizados ao AWS Lambda, que, por sua vez, encaminha estes dados ao Amazon API Gateway.
3. `DELETE /doctor`. O método `DELETE` utiliza a função `DeleteUser` do Amazon Cognito (AMAZON, 2023c). Esta função exige somente o *token* de acesso em sua requisição para um uso bem-sucedido. O Amazon Cognito indica ao AWS Lambda a deleção completa com o envio do código `200 OK` de volta. O Lambda Doctor é encarregado de mapear este código para `204 No Content` e retornar à API. A Figura 4.7 apresenta o fluxo completo de uma requisição `DELETE /doctor` ao Amazon API Gateway do projeto.

Figura 4.7 – Diagrama de uma requisição bem-sucedida do método `DELETE` à rota `/doctor`. Os numerais indicam a ordem do fluxo do dado da requisição. Objeto JSON no passo 1 e 4 são simplificados a fim de melhor visualização.



Fonte: Próprio Autor

4.3.5 Lambda Patients

O Lambda Patients é encarregado de criar, editar, buscar e excluir pacientes. O Amazon API Gateway redireciona os caminhos `/patients` e `/patients/:id` para esta função Lambda a fim de serem processados. O comportamento deste AWS Lambda consiste em decodificar o *token* de acesso recebido, identificar o método HTTP utilizado no objeto de *input*, processar o identificador do paciente caso presente na rota, integrar com o banco de dados Amazon DynamoDB e mapear a resposta para ao Amazon API Gateway.

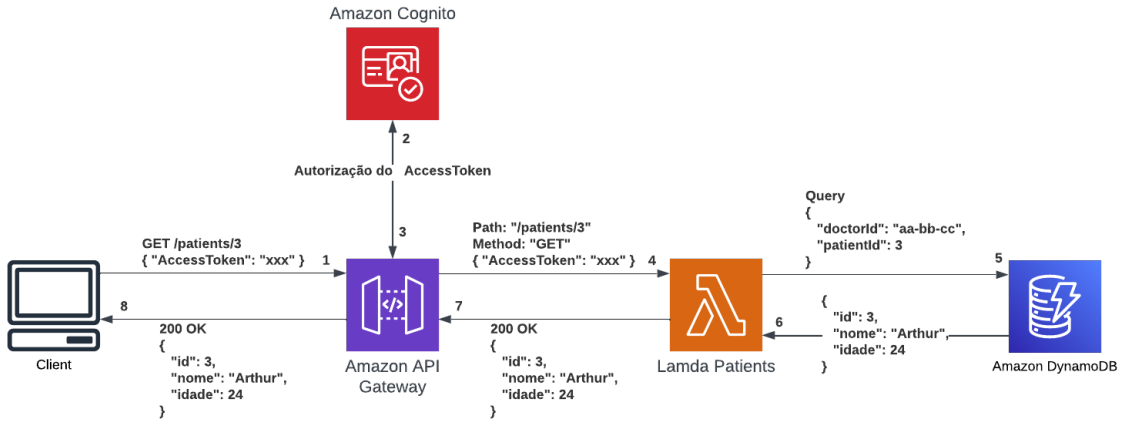
A fim de checar que os pacientes acessados no Amazon DynamoDB pertencem ao usuário que está fazendo a requisição é necessário decodificar o `AccessToken` recebido no cabeçalho. O *token* de acesso é estruturado como uma *string* e codificada em *base64*, incluindo informações como o emitente do *token*, o destinatário, a data de expiração e os dados do usuário (OKTA, 2023). Dentre as informações contidas no *token* de acesso, é possível identificar o `id` do psicólogo logado na plataforma e, com isso, buscar no banco de dados somente os pacientes que contenham este `id` do psicólogo.

Como o método HTTP é encaminhado no *input*, basta uma comparação simples para discernir cada requisição. Após esta comparação é possível manusear cada método unicamente.

1. `GET /patients`. Com a seleção do método `GET`, é necessário identificar se a requisição busca todos os pacientes ou somente um único específico. Este reconhecimento é feito checando a rota recebida pelo AWS Lambda. Caso a rota contenha o identificador único após `/patients` é realizado uma operação de busca no Amazon DynamoDB por somente o paciente com este `id`. Caso nenhum identificador seja enviado é retornado todos os pacientes do psicólogo pertencente ao `AccessToken` enviado no cabeçalho da requisição. O comportamento da busca por todos os pacientes consiste em utilizar o método `Query` da API de acesso do Amazon DynamoDB (AMAZON, 2023j), informando o `id` do psicólogo (mais sobre as operações sobre o banco de dados na seção 4.3.7). Com a resposta do banco de dados, o Lambda Patients repassa os pacientes novamente à API.
2. `GET /patients/:id`. O comportamento do AWS Lambda para esta requisição é relativamente similar à apresentada acima. Após a identificação do método, é possível ler o `id` enviado na rota através da propriedade `pathParameters` contida no objeto enviado do Amazon API Gateway ao *input* do Lambda Pati-

ents. Esta propriedade consiste em um objeto de pares chave-valor, listando todas as variáveis presente na rota. O método do Amazon API Gateway utilizado corresponde, novamente, ao `Query`. O diferencial do método para buscar todos os pacientes, é a apresentação do identificador do paciente juntamente com o `id` do psicólogo. A Figura 4.8 apresenta o fluxo, a partir do *front-end*, a requisição ao *endpoint* `/patients/:id`.

Figura 4.8 – Fluxo de dados relacionado à requisição `/patients/:id`, com o `id` igual a 3. Os numerais representam a ordem dos acontecimentos. Os dados apresentados são simplificações.



Fonte: Próprio Autor

3. `POST /patients`. O *input* da função Lambda para uma requisição com método `POST` contém informações cruciais sobre o objeto a ser criado dentro da propriedade *body*. O comportamento do Lambda Patients para este método consiste em ler e analisar os dados do campo *body*. O método utilizado para a criação de um item no banco de dados consiste em `PutItem` (AMAZON, 2023i). Para sua utilização, é necessário adicionar as propriedades do item a ser criado em sua requisição. Por isso, o JSON decodificado do campo *body* é enviado nesta requisição para o banco de dados. A fim de evitar a criação de propriedades desnecessárias no Amazon DynamoDB, é enviado somente os campos do nome, sobrenome, e-mail, telefone, endereço e idade no JSON. Juntamente com essas informações, é encaminhado o identificador do psicólogo presente no *token* de acesso na criação do paciente.
4. `PUT /patients/:id`. Para editar um paciente existente no banco de dados, a função Lambda recebe no *input* as novas informações do paciente no campo *body* e o `id` no `pathParameters`. Com esses dados, o método `UpdateItem` da API do Amazon DynamoDB (AMAZON, 2023l) é utilizado. O identificador do paciente é informado no campo `key` deste método, e os dados no campo

`AttributeUpdates`. O identificador do psicólogo é enviado em conjunto com o `id` do paciente a fim de afirmar que o psicólogo emissor é relacionado ao paciente. Caso nenhum paciente com a `key` enviada seja encontrado, o Amazon DynamoDB informa ao AWS Lambda o código de erro `400 Bad Request`. Caso a alteração seja bem-sucedida, o Amazon DynamoDB retorna `200 OK`, juntamente com o item alterado. Essas mensagens, seja de erro ou de sucesso, são repassadas novamente ao Amazon API Gateway.

5. `DELETE /patients/:id`. O último método avaliado pelo AWS Lambda é encarregado da deleção de um paciente. Ao avaliar o identificador enviado na rota, o AWS Lambda é capaz de excluir corretamente a instância do item no banco de dados utilizando o método `DeleteItem` (AMAZON, 2023b).

Com a análise e decodificação dos campos necessários, o Lambda Patients age como um meio campo entre o Amazon API Gateway e o Amazon DynamoDB. Uma das vantagens desta arquitetura com o AWS Lambda orquestrando as requisições é a facilidade de adaptar a função Lambda a novos métodos e integrações. Por exemplo, caso seja adicionado o método `DELETE` com a rota `/patients` a fim de deletar todos os pacientes de um psicólogo, basta adicionar um novo `if` na função Lambda e desenvolver esta nova funcionalidade.

4.3.6 Lambda Sessions

O Lambda Sessions realiza as operações de CRUD — criar, ler, atualizar e deletar — no banco de dados, garantindo a integridade e segurança das sessões de terapia. Além disso, é responsável por validar e processar as informações fornecidas na requisição do usuário, garantindo que as operações sejam executadas corretamente.

O seu comportamento é similar ao Lambda Patients, identificando o método e a rota chamada, processando o *input* corretamente e executando a ação no Amazon DynamoDB. A principal diferença consiste nas rotas utilizadas, contendo o identificador do paciente e da sessão específica.

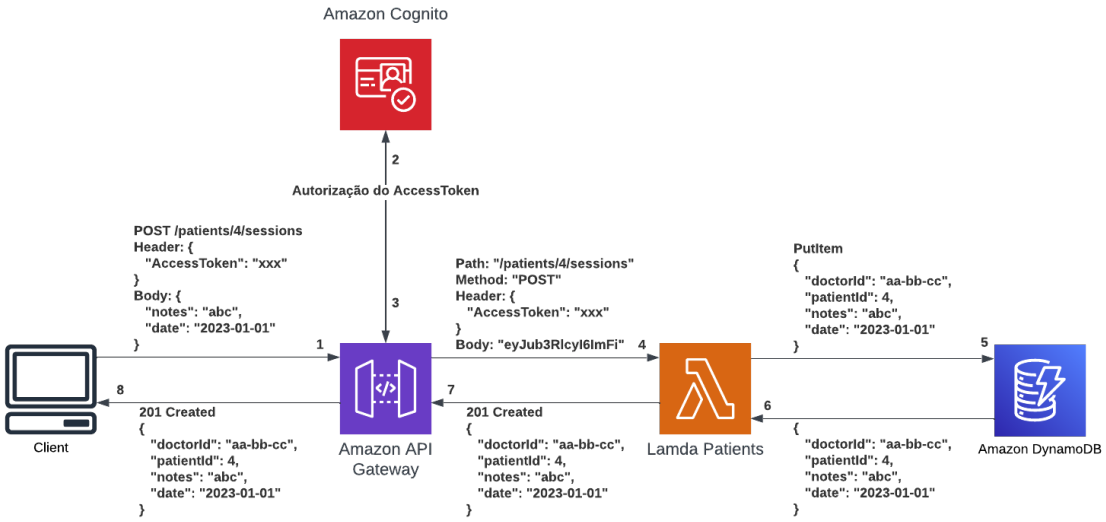
Para a busca de todas sessões de um determinado paciente, é utilizada a rota `/patients/:id/sessions` com o método `GET`. Assim como as funções Lambdas descritas anteriormente, o psicólogo é identificado via a decodificação do *token* de acesso enviado no cabeçalho do input. Com o psicólogo e o paciente — identificado via `id`

na rota — é possível realizar a ação `Query` da API do DynamoDB (AMAZON, 2023j), informando juntamente o `id` do psicólogo e do paciente a pesquisa no banco de dados.

A busca de uma sessão específica é realizada com a execução da rota `/patients/:id/sessions/:id`. Agora, além do *token* de acesso, é necessário informar o identificador do paciente e da sessão específica. A busca no banco de dados é novamente feita através do método `Query`, porém, neste caso, é informado o `id` do psicólogo, paciente e sessão. Os métodos de busca do Amazon DynamoDB, se bem-sucedidos, retornam o código `200 OK` com os dados requisitados. Caso o item não seja encontrado na tabela, ou a sessão seja de outro paciente, é retornado o código `400 Bad Request` ao AWS Lambda. Como as sessões no Amazon DynamoDB são rastreadas a partir da junção do `id` do psicólogo, do paciente e da sessão (mais sobre na seção 4.3.7), a busca contendo algum desses valores inválidos não encontrará nenhum item.

A rota `/patients/:id/sessions` com o método `POST` efetua a criação de uma nova sessão no Amazon DynamoDB. Os dados a serem criados são enviados no campo *body* da requisição HTTP. O Lambda Sessions, por sua vez, é encarregado de decodificar este campo e enviá-lo ao banco de dados. A implementação da criação de uma nova instância no banco de dados é similar ao apresentado no Lambda Patients, ambos utilizando o método `PutItem` da API do Amazon DynamoDB (AMAZON, 2023i). A fim de evitar a adição de propriedades inválidas no banco de dados, o Lambda Sessions encaminha somente os dados `notes` e `date` da sessão a ser criada. A Figura 4.9 apresenta o fluxo dos dados na execução do método `POST` à rota `/patients/4/sessions`.

Figura 4.9 – Fluxo dos dados na execução de um `POST` à rota `/patients/4/sessions`.



Fonte: Próprio Autor

A edição e deleção de uma sessão utilizam o mesmo comportamento do descrito no Lambda Patients. Ambas ações recebem na rota da requisição o identificador do paciente e da sessão, podendo, então, determinar corretamente em qual item do banco de dados é necessário agir. Para a edição, os dados da sessão atualizados são recebidos pelo AWS Lambda no campo *body* do *input*. Após a decodificação, é enviado o pedido de edição da sessão com os novos dados `notes` e `date`. Para a deleção é necessário somente informar o identificador do psicólogo — determinado via *token* de acesso —, do paciente e da sessão — identificados na rota —.

4.3.7 Amazon DynamoDB

O Amazon DynamoDB é o serviço de banco de dados NoSQL escolhido para o armazenamento dos dados da plataforma. O banco de dados é gerenciado pela AWS, facilitando sua conexão com outros serviços, como AWS Lambda e Amazon S3. Por fornecer baixa latência consistente em grandes escalas (AMAZON, 2023e), seu uso facilita o acesso aos dados da aplicação. Além disso, o Amazon DynamoDB é flexível, permitindo a criação de tabelas que podem armazenar e recuperar larga quantidade de dados, permitindo o desenvolvimento de uma plataforma escalável.

Para a implementação dos dados da aplicação no Amazon DynamoDB, optou-se pelo uso de *Single Table Design*, isto é, todos os diferentes formatos de dados (como pacientes e sessões) são armazenados na mesma tabela. O uso do *Single Table Design* permite que as operações de leitura e gravação sejam altamente eficientes, mesmo à medida que o volume de dados aumenta (JAMES BESWICK, 2021). Sua utilização promove uma redução no custos, uma vez que é eliminado a necessidade de leituras e gravações em várias tabelas para obter ou atualizar informações relacionadas. Com o propósito de modelar os dados da aplicação em somente uma tabela, é necessário investigar os padrões de acesso, ou seja, quais requisições ao banco de dados serão feitas (FOWLER, 2002). A Tabela 4.3 exibe todas os modos que os dados serão acessados no Amazon DynamoDB, apresentando sua rota, tipo de operação (leitura ou escrita), quantidade de itens (único ou múltiplo) e seus filtros necessários.

Com a definição dos padrões de acesso é possível determinar um modelo para o armazenamento dos itens na tabela do Amazon DynamoDB. Como já apresentado na seção 2.3.3, os itens são armazenados e lidos através do uso de uma chave primária única, composta por uma chave de partição (PK - *partition key*) e de uma chave de classificação

Tabela 4.3 – Tabela com os padrões de acesso aos dados do Amazon DynamoDB.

Padrão de acesso	**Leitura ou escrita**	**Tipo (item único ou múltiplos itens)**	**Filtros**
Buscar paciente *P* do psicólogo	Leitura	Único	Paciente = *P* Psicólogo = usuário autenticado
Buscar todos pacientes do psicólogo	Leitura	Múltiplo	Psicólogo = usuário autenticado
Criar paciente do psicólogo	Escrita	Único	Psicólogo = usuário autenticado
Atualizar paciente *P* do psicólogo	Escrita	Único	Paciente = *P* Psicólogo = usuário autenticado
Deletar paciente *P* do psicólogo	Escrita	Único	Paciente = *P* Psicólogo = usuário autenticado
Buscar sessão *S* do paciente *P* do psicólogo	Leitura	Único	Sessão = *S* Paciente = *P* Psicólogo = usuário autenticado
Buscar todas sessões do paciente *P* do psicólogo	Leitura	Múltiplo	Paciente = *P* Psicólogo = usuário autenticado
Criar sessão do paciente *P* do psicólogo	Escrita	Único	Paciente = *P* Psicólogo = usuário autenticado
Atualizar sessão *S* do paciente *P* do psicólogo	Escrita	Único	Sessão = *S* Paciente = *P* Psicólogo = usuário autenticado
Deletar sessão *S* do paciente *P* do psicólogo	Escrita	Único	Sessão = *S* Paciente = *P* Psicólogo = usuário autenticado

(SK - *sort key*). Por se tratar de um banco de dados NoSQL, o conceito de `JOIN` — união de dados via uma propriedade em comum — não existe. Para solucionar este problema, a identificação dos dados é feita a partir da união dos identificadores de cada objeto. Deste modo, as modelagem das classes, e suas propriedades, são apresentadas na tabela 4.4.

Tabela 4.4 – Modelagem das classes `Patients` e `Sessions` no Amazon DynamoDB.

Classe	**Chave primária**		**Atributos**
	PK	SK	

`Patients`	`DOCTOR#<DoctorID>`	`PATIENT#<PatientID>`	`FirstName`
			`LastName`
			`Email`
			`Phone`
			`Address`
			`Age`

`Sessions`	`DOCTOR#<DoctorID>` `#PATIENT_S#<PatientID>`	`SESSION#<SessionID>`	`Notes` `Date`

Como é possível observar, a chave primária de `Patients` (`DOCTOR#<DoctorID>PATIENT#<PatientID>`) consiste na união do identificador do psicólogo e do paciente, a fim de afirmar que o usuário que está fazendo a requisição ao banco de dados tem autorização para buscar o paciente específico. A chave primária de `Sessions` (`DOCTOR#<DoctorID>PATIENT_S#<PatientID>SESSION#<SessionID>`) indica o `id` do psicólogo, do paciente e da sessão. A seguir é apresentado a busca a ser feita ao banco de dados, seguido do método da API do Amazon DynamoDB utilizado e seus parâmetros.

1. Buscar todos os pacientes do psicólogo `DoctorID`
    - **Operação**: `Query`
    - **Condições**: `PK = DOCTOR#<DoctorID> and begins_with(SK, PATIENT#)`, onde `<DoctorID>` representa o identificador do psicólogo e `begins_with` uma função disponível na API do Amazon DynamoDB que checa se o valor do atributo começa com uma *substring* específica (AMAZON, 2023f), ou seja, checa se a SK começa com a *string* `PATIENT#`. Com essas condições é retornado todos os pacientes pertencentes ao psicólogo cujo `id` é `DoctorID`.
2. Buscar paciente `PatientID` do psicólogo `DoctorID`
    - **Operação**: `Query`

- **Condições**: `PK = DOCTOR#<DoctorID> and SK = PATIENT#<PatientID>`, onde `<DoctorID>` é o `id` do psicólogo e `<PatientID>` o do paciente. Utilizando os parâmetros desta forma, o paciente somente será encontrado se o psicólogo correto requisitar a busca, uma vez que o seu `id` é aplicado na busca.

3. Buscar todas as sessões do paciente `PatientID` do psicólogo `DoctorID`

   - **Operação**: `Query`
   - **Condições**: `PK = DOCTOR#<DoctorID>#PATIENT_S#<PatientID> and SK = begins_with(SK, SESSION#)`, onde o PK é composto pela *string* `DOCTOR`, seguido pelo identificador único do psicólogo `<DoctorID>`, a *string* `PATIENT_S` e o identificador único do paciente `<PatientID>`. É buscado todos os itens que apresentam a PK descrita e SK que começa com a *string* `SESSION#`.

4. Buscar sessão `SessionId` do paciente `PatientID` do psicólogo `DoctorID`

   - **Operação**: `Query`
   - **Condições**: `PK = DOCTOR#<DoctorID>#PATIENT_S#<PatientID> and SK = SESSION#<SessionId>`, onde `<SessionId>` representa o identificador único da sessão. Representando um item com a união dos identificadores únicos facilita sua busca e promove um ambiente onde somente o psicólogo correto tem acesso aos seus pacientes; e seus pacientes somente acesso à suas sessões. Para realizar a busca de uma sessão específica é necessário ter conhecimento do `id` da sessão, do paciente e do psicólogo.

Para a criação de um item no Amazon DynamoDB é utilizado as seguintes representações:

1. Criar um paciente do psicólogo `DoctorID`

   - **Operação**: `PutItem`
   - **Atributos**:

```
{
    "PK": "DOCTOR#<DoctorID>",
    "SK": "PATIENT#56789",
    "FirstName": "Bob",
    "LastName": "Kurose",
```

```
    "Email": "bob.kurose@email.com",
    "Phone": "555133086000",
    "Address": "Av. Paulo Gama, 110",
    "Age": 24
}
```

O exemplo dos atributos acima apresenta a criação de um novo paciente *Bob Kurose* com o identificador do psicólogo `DoctorID` (decodificado do `AccessToken`), e do `PatientID` igual a *56789* (criado no Lambda Patients). Os atributos do paciente enviado são `FirstName`, `LastName`, `Email`, `Phone`, `Address` em texto e `Age` em número.

2. Criar uma sessão do paciente `PatientID` do psicólogo `DoctorID`

   - **Operação**: `PutItem`
   - **Atributos**:

```
{
    "PK": "DOCTOR#<DoctorID>#PATIENT_S#<PatientID>",
    "SK": "SESSION#abcde",
    "notes": "Lorem ipsum dolor sit amet...",
    "date": "2024-01-01"
}
```

Para a criação de uma nova sessão, é identificado o `DoctorID` pelo `AccessToken`, o `PatientID` pela rota da requisição HTTP e o `SessionID` é criado dentro do Lambda Sessions. É enviado à operação `PutItem` os atributos `Notes` e `Date` em texto.

A fim de efetuar a atualização de um item no banco de dados é necessário informar qual item será modificado, juntamente com as novas informações. A edição de um paciente e de uma sessão apresentam similaridades em relação a operação.

1. Atualizar paciente `PatientID` do psicólogo `DoctorID`

   - **Operação**: `UpdateItem`
   - **Condição**: `PK = DOCTOR#<DoctorID>` e `SK = PATIENT#<PatientID>`

- **Atributos**:

```
{
    "FirstName": "Bob",
    "LastName": "Kurose",
    "Email": "bob.kurose@email.com",
    "Phone": "555133086000",
    "Address": "Av. Paulo Gama, 110",
    "Age": 24
}
```

A operação acima descreve a atualização do paciente `PatientID` e psicólogo `DoctorID`. Nos atributos para a atualização de um item não é necessário exibir a PK e a SK do paciente, esses identificadores são apresentados no campo de condição.

2. Atualizar sessão `SessionID` do paciente `PatientID` do psicólogo `DoctorID`

- **Operação**: `UpdateItem`
- **Condição**: `PK = DOCTOR#<DoctorID>#PATIENT_S#<PatientID>` e `SK = SESSION#<SessionID>`
- **Atributos**:

```
{
    "notes": "Item atualizado",
    "date": "2024-02-02"
}
```

A atualização apresentada consiste na edição da sessão `SessionID` do paciente `PatientID`, do psicólogo `DoctorID`. Caso algum desses identificadores não existam, ou a sessão seja de outro paciente, ou o paciente de outro psicólogo, o item a ser editado não será encontrado e, por isso, nenhuma operação será aplicada ao banco de dados.

Para a deleção de um item é utilizado o método `DeleteItem` do Amazon DynamoDB, como apresentado nos itens a seguir.

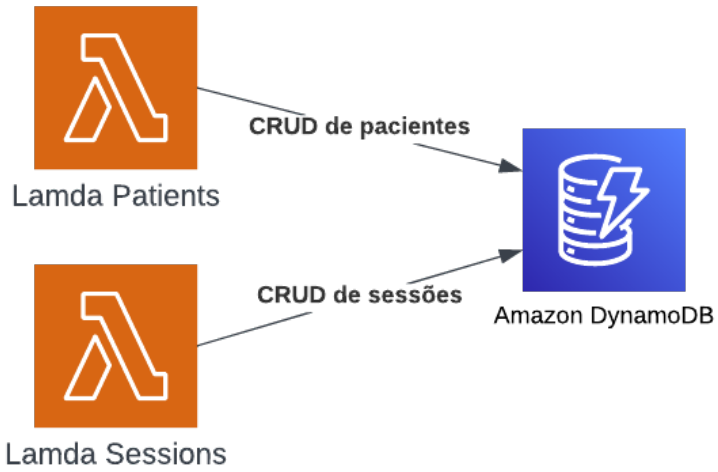
1. Deleção de um paciente `PatientID` do psicólogo `DoctorID`

- **Operação**: `DeleteItem`
- **Condição**: `PK = DOCTOR#<DoctorID>` e `SK = PATIENT#<PatientID>`

2. Deleção de uma sessão `SessionID` do paciente `PatientID` do psicólogo `DoctorID`
   - **Operação**: `DeleteItem`
   - **Condição**: `DOCTOR#<DoctorID>#PATIENT_S#<PatientID>` e `SK = SESSION#<SessionID>`

A Figura 4.10 apresenta a relação do Lambda Patients e do Lambda Sessions com o serviço Amazon DynamoDB na plataforma final. Ambas as funções Lambdas agem como orquestrador dos dados, gerenciando a criação, leitura, edição e exclusão (CRUD, do acrônimo do inglês *create*, *read*, *update* e *delete*) dos itens no Amazon DynamoDB.

Figura 4.10 – Diagrama das ações do Lambda Patients e Sessions sobre o serviço Amazon DynamoDB.



Fonte: Próprio Autor

4.3.8 Lambda Upload

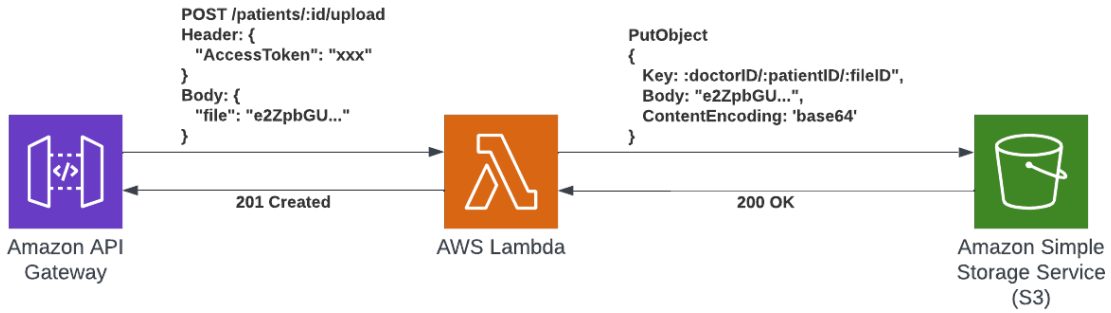
O Lambda Upload é encarregado de gerenciar o armazenamento de documentos e imagens na aplicação final. Assim como os outros AWS Lambdas descritos neste projeto, esta função recebe as requisições do usuário através do Amazon API Gateway, analisando-as para determinar se o usuário deseja salvar ou ler um arquivo. Com essa determinação, o Lambda Upload é responsável por se conectar ao serviço Amazon S3 a

fim de efetuar a operação de armazenamento do arquivo. Três endpoints do Amazon API Gateway são encaminhados para esta função Lambda:

1. `POST /patients/:id/sessions/upload`

   Com a identificação desta requisição, o Lambda Upload é responsável de encaminhar o arquivo enviado no *body* da requisição para o Amazon S3. Como o documento enviado já está no formado *base64*, nenhuma transformação é necessária ao arquivo. É utilizado o método `PutObject` da API do Amazon S3 para arquivar o documento dentro do serviço da AWS, informando o nome do arquivo, o modo de codificação e seus dados. Para separar os documentos de cada usuário, os arquivos são salvos unindo os identificadores do psicólogo e do paciente. Por exemplo, se o psicólogo com `id` *abc* arquivar um documento relacionado ao paciente com `id` *def*, o arquivo será salvo com o nome `abc/def/:id`, onde o `:id` é um identificador único de cada documento criado dentro da função Lambda. O mapeamento dos documentos utilizando os identificadores do paciente e do psicólogo é importante para realizar a busca e leitura dos arquivos corretamente. Com a ação de *upload* bem-sucedida, esta função Lambda retorna `201 Created` ao Amazon API Gateway. A Figura 4.11 apresenta o fluxo de uma requisição de *upload* de um arquivo, começando no Amazon API Gateway, passando pelo AWS Lambda para ser armazenado no diretório correto no Amazon S3.

Figura 4.11 – Diagrama de uma requisição de *upload* de um arquivo na plataforma final, indicando os serviços utilizados no processo.



Fonte: Próprio Autor

2. `GET /patients/:id/sessions/upload`

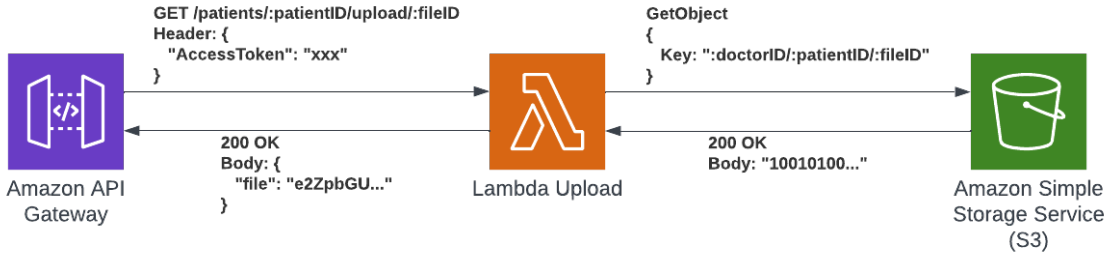
   O Lambda Upload, ao identificar esta requisição, utiliza o método `ListObjectsV2` para ler os metadados de todos os documentos de um certo paciente. A fim de realizar uma busca correta dos arquivos, é necessário informar ao método o prefixo do diretório a ser buscado, isto é, para o paciente

com `id` *def*, do psicólogo *abc*, é informado o prefixo *abc/edf*. Com isto todos os metadados dos arquivos dentro deste diretório serão retornados. Os metadados são importantes para realizar o *download* completo dos dados de um certo arquivo.

3. `GET /patients/:id/sessions/upload/:id`

   Este endpoint é utilizado em conjunto ao descrito acima, uma vez que com o identificador de cada arquivo, é possível realizar o seu *download*. Conhecendo o `id` do paciente e do documento correto, o Lambda Upload aciona o método `GetObject` da API do Amazon S3. Para uma busca bem-sucedida, o AWS Lambda é encarregado de montar o prefixo do arquivo a ser lido utilizando o `id` do psicólogo, do paciente e do documento, no formato `:doctorID/:patientID/:fileID`. O Amazon S3 retorna o arquivo em binário à função Lambda, que, por sua vez, utiliza o método `transformToString` disponibilizado pelo Amazon S3 para transformar o documento em *base64* a fim de retorná-lo ao Amazon API Gateway. O código `200 OK` é enviado juntamente com o arquivo em *base64* ao *front-end*. A Figura 4.12 apresenta o diagrama de uma requisição de *download* de um arquivo. É possível observar que os dados retornados do Amazon S3 são em binário e os retornados do AWS Lambda são em *base64*, indicando que a função Lambda é responsável por essa decodificação. Para o *front-end* identificar o documento corretamente, basta sua decodificação de *base64* para o formato correto.

Figura 4.12 – Fluxo de dados da requisição de busca de um arquivo específico. Indicando o formato dos dados de entrada e saída de cada serviço da AWS utilizado no projeto final.



Fonte: Próprio Autor

Assim como as outras funções Lambdas conectadas ao Amazon API Gateway, o Lambda Upload é somente invocado se o *token* de acesso enviado à API seja válido, ajudando na redução de custos, uma vez que, é cobrado somente pelo tempo de execução de um AWS Lambda.

4.3.9 Amazon S3

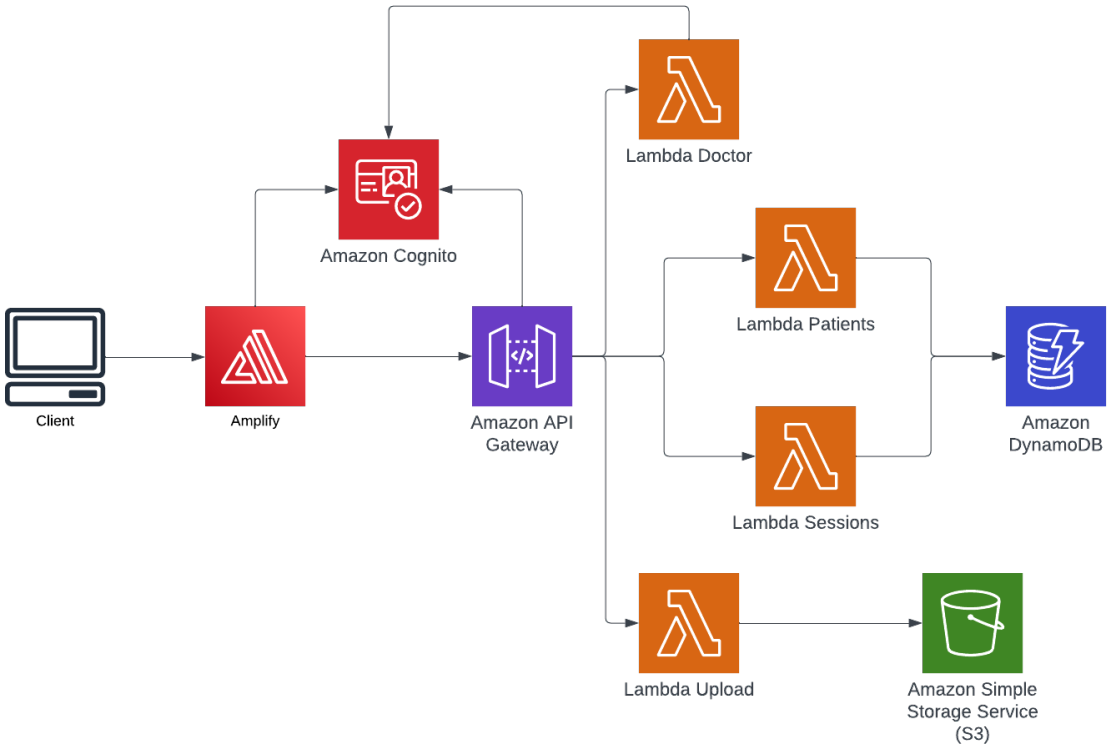
O Amazon S3 é o serviço de armazenamento de objetos utilizado no projeto. Sua principal função é salvar os documentos, como laudos e imagens, de cada paciente. Sua estrutura interna é dividida em *buckets*, isto é, pastas na nuvem onde é possível agrupar diversos arquivos – desde textos e imagens até vídeos (AMAZON, 2023j).

Optou-se por criar um único *bucket* no Amazon S3, uma vez que todos os arquivos armazenados estão relacionados aos pacientes, resultando em uma gestão simplificada e eficiente. É importante destacar que, caso o aplicativo precise lidar com outros tipos de arquivos que possuam regras de negócio diferentes ou requisitos de segurança mais rigorosos, seria recomendado criar um novo *bucket*. Ao separar diferentes tipos de arquivos em *buckets* diferentes, é possível aplicar políticas específicas de controle de acesso e criptografia para cada *bucket*, simplificando sua responsabilidade e operação.

5 RESULTADOS E AVALIAÇÃO

O objetivo deste Capítulo é fornecer os resultados dos conceitos e da arquitetura *serverless* desenvolvida anteriormente, permitindo uma avaliação do trabalho. Com o estudo de cada um dos componentes presentes na arquitetura final juntamente com seus fluxos, é possível ter uma visão geral da arquitetura desenvolvida. A Figura 5.1 demonstra o resultado da arquitetura *serverless* implementada no presente trabalho.

Figura 5.1 – Arquitetura *serverless* para um sistema de gerenciamento de atendimentos psicológicos.



Fonte: Próprio Autor

A avaliação objetiva da arquitetura é realizada através da apresentação do tempo de resposta médio para os fluxos da aplicação, bem como uma estimativa de custo para a implementação de cada serviço da AWS utilizado. Com essas duas métricas, é possível analisar a viabilidade do presente trabalho.

5.1 Tempo de resposta do sistema

A apresentação do desempenho da aplicação é fundamental para a análise da arquitetura desenvolvida. A demonstração quantitativa do tempo de resposta do sistema

ajuda a verificar a validade da solução proposta, ilustrando que a solução funciona como esperado e satisfaz as necessidades para as quais foi projetada. De acordo com estudo realizado em 2004 (NAH, 2004), o tempo de espera tolerável para a busca de informações em um *site* é de aproximadamente 2 segundos. O mesmo artigo demonstra que 0,1 segundo é o limite para que o usuário sinta que o sistema está reagindo instantaneamente.

Com o estudo do tempo de resposta de cada fluxo da aplicação, é possível analisar áreas da arquitetura que podem ser aprimoradas. A seguir, é apresentado o tempo de resposta para cada um dos *endpoints* desenvolvidos no presente trabalho. Os testes foram realizados através da interface visual do Amazon API Gateway e foram realizados três vezes consecutivas a fim de obter resultados mais consistentes.

5.1.1 Fluxo dos usuários

No que diz respeito à busca de um usuário, que corresponde a uma consulta aos dados do usuário no serviço do Amazon Cognito, o tempo médio de resposta é o mais rápido dentre as operações relacionadas ao psicólogo, com $123,33\ ms$. A atualização das informações do usuário, que também envolve a interação com o serviço Amazon Cognito, apresenta um tempo médio de resposta de $138,66\ ms$. Em relação à operação de deleção de um usuário, que envolve mais passos internos do Amazon Cognito, como a eliminação de dados associados ao usuário, o tempo médio de resposta permanece razoável, sendo $324,66\ ms$. A Tabela 5.1 apresenta os tempos de resposta para três operações seguidas realizadas no fluxo do psicólogo.

Tabela 5.1 – Tempo de resposta para a busca, atualização e deleção de um usuário realizada três vezes consecutivas, juntamente com a média aritmética dos valores obtidos.

	1ª operação	**2ª operação**	**3ª operação**	**Média**
Buscar usuário	150 ms	104 ms	116 ms	123.33 ms
Atualizar usuário	128 ms	143 ms	145 ms	138.66 ms
Deletar usuário	300 ms	322 ms	352 ms	324.66 ms

5.1.2 Fluxo dos pacientes

Examinando a Tabela 5.2 , é possível notar a eficácia do fluxo que envolve uma chamada ao *endpoint* do Amazon API Gateway, a invocação do AWS Lambda e o acesso ao Amazon DynamoDB.

A operação com o menor tempo médio de resposta consiste em buscar um paciente específico por `id`, com $28,66\ ms$. Os testes de busca de todos os pacientes foram realizados com um psicólogo contendo 20 pacientes, obtendo um tempo de resposta médio de $30,33\ ms$. A criação, atualização e deleção de um paciente também demonstraram tempos de resposta altamente eficientes, variando de $33\ ms$ a $43\ ms$.

Tabela 5.2 – Tempo de resposta da operações do fluxo relacionados aos pacientes, consistindo da busca, criação, atualização e deleção de um paciente. Teste realizado três vezes consecutivas, juntamente com a média aritmética dos valores obtidos.

	1ª operação	**2ª operação**	**3ª operação**	**Média**
Buscar todos pacientes	$30\ ms$	$31\ ms$	$30\ ms$	$30.33\ ms$
Buscar paciente por `id`	$26\ ms$	$28\ ms$	$32\ ms$	$28.66\ ms$
Criar paciente	$30\ ms$	$56\ ms$	$43\ ms$	$43\ ms$
Atualizar paciente	$42\ ms$	$44\ ms$	$35\ ms$	$40.33\ ms$
Deletar paciente	$39\ ms$	$32\ ms$	$28\ ms$	$33\ ms$

5.1.3 Fluxo das sessões

A operação de buscar uma sessão pelo seu identificador apresentou o tempo de resposta mais rápido, com $36,66\ ms$. Operações de criação e atualização de sessão tiveram tempos médios de resposta um pouco mais altos de, $44\ ms$ e $42\ ms$, respectivamente. Entretanto, ainda assim demonstram um bom desempenho, considerando que ambas as operações envolvem armazenamento no banco de dados. Ao se tratar da operação buscar todas as sessões, testada com um paciente contendo 50 sessões, o tempo médio de resposta foi de $51,33\ ms$. Este é um tempo de retorno satisfatório, considerando a quantidade de dados envolvidos. Caso o tempo de resposta para múltiplos dados representasse um problema na aplicação final, seria possível paginar a respostas, retornando as sessões em partes. O fluxo de deleção de uma sessão mostrou um tempo médio de resposta de $38\ ms$.

A análise dos fluxos de pacientes e sessões apresentam resultados eficientes, mesmo com um volume considerável de dados, evidenciando a eficácia do fluxo entre o Amazon API Gateway, o AWS Lambda e o Amazon DynamoDB. A Tabela 5.3 apresenta os resultados mencionados acima.

Tabela 5.3 – Tempo de resposta para criar, atualizar, deletar e buscar uma sessão realizada três vezes consecutivas, juntamente com a média aritmética dos valores obtidos.

	1ª operação	**2ª operação**	**3ª operação**	**Média**
Buscar todas sessões	38 ms	41 ms	75 ms	51.33 ms
Buscar sessão por `id`	32 ms	37 ms	41 ms	36.66 ms
Criar sessão	31 ms	57 ms	45 ms	44.33 ms
Atualizar sessão	40 ms	45 ms	41 ms	42 ms
Deletar sessão	39 ms	40 ms	35 ms	38 ms

5.1.4 Fluxo dos documentos

A operação de busca dos metadados do arquivos foram realizadas com um usuário contendo 18 arquivos, apresentando um tempo médio de resposta de $103,66\ ms$. A ação de buscar o arquivo por `id` obteve o tempo médio de resposta foi um pouco maior, com $148,66\ ms$ para um arquivo de 161 KB. A operação de carregar um arquivo, que envolve o envio de um objeto para o armazenamento Amazon S3, teve um tempo médio de resposta de $647,33\ ms$ para o mesmo arquivo de 161 KB. Embora essa operação tenha o tempo médio de resposta mais alto, ainda é um tempo bastante eficiente levando em consideração o tamanho do arquivo a ser carregado. A Tabela 5.4 demonstra os valores apresentados.

É possível observar uma diminuição no tempo de resposta entre a primeira e a terceira tentativa nos testes de carregar o arquivo, proveniente do efeito do *cold start* da função Lambda. O *cold start* ocorre quando uma função Lambda é executada após estar inativa por um tempo, resultando em um tempo de inicialização mais longo (AMAZON, 2023g).

Esses resultados mostram que o fluxo de documentos funciona de forma eficaz, permitindo a recuperação e carregamento de arquivos com tempos de resposta aceitáveis. Isto é especialmente relevante dada a importância de poder lidar com documentos de forma eficiente em um sistema de gerenciamento de atendimentos psicológicos.

Tabela 5.4 – Tempo de resposta para buscar os metadados, carregar um arquivo e buscar um arquivo realizado três vezes consecutivas, juntamente com a média aritmética dos valores obtidos.

	1ª operação	**2ª operação**	**3ª operação**	**Média**
Buscar metadados	64 ms	165 ms	82 ms	103.66 ms
Buscar arquivo por `id`	217 ms	144 ms	85 ms	148.66 ms
Carregar arquivo	888 ms	841 ms	213 ms	647.33 ms

5.2 Precificação

O custo de operação do sistema é uma ótima métrica na apresentação da viabilidade da aplicação. Os serviços disponíveis pela AWS possuem diferentes estruturas de custo, que dependem de vários fatores, como consumo de recursos, armazenamento de dados, transferência de informações, entre outros. O ambiente de desenvolvimento da AWS possibilita uma estimativa de custo de cada serviço, apresentando seus cálculos (AWS, 2024). Nesta seção, é demonstrado o custo aproximado para a arquitetura desenvolvida, separado por serviço.

5.2.1 AWS Amplify

O custo do AWS Amplify é divido em duas partes: tempo de *build* (processo de transformar o código-fonte em um formato executável) e de hospedagem (armazenamento dos arquivos em um servidor). Aproxima-se que o tempo médio para o *build* do *front-end* seja de 10 minutos, e que o código seja atualizado 5 vezes por mês. Como o custo de um minuto de *build* do AWS Amplify é de US$ 0,01 (AMAZON, 2023l), temos que,

$$10 \text{ minutos por } \textit{build} \times 5 \textit{ builds} \text{ por mês} \times \text{US\$ } 0,01 \text{ por minuto} = \text{US\$ } 0,50 \text{ por mês}$$

Para o custo da hospedagem, presume-se que o tamanho médio da página solicitada seja de 1 MB e que o número de usuários mensal seja de 15.000 (500 usuários por dia). O valor por megabyte de hospedagem no AWS Amplify é de US$ 0,000146 (AMAZON, 2023l), logo,

$$1 \text{ MB por usuário} \times 15.000 \text{ usuários por mês} \times \text{US\$ } 0,000146 \text{ por MB} = \text{US\$ } 2,20 \text{ por mês}$$

Desse modo, o custo do AWS Amplify no projeto final é de US$ 2,70 mensal (US$ $0,50 + $ US$ $ $2,20$).

5.2.2 Amazon Cognito

A precificação do Amazon Cognito é dependente do número de MAU (*monthly active users*), isto é, o número de usuários que, dentro de um mês, realizaram alguma operação de identidade, como registro, *login*, atualização de *token* ou alteração de senha. O número estimado de MAU para o cálculo da precificação é de 500, representando o número de psicólogos com conta na aplicação final. A AWS oferece o serviço de *user pool* do Amazon Cognito de graça para 50.000 MAUs, por isso, este serviço não apresenta custo na arquitetura final. Após 50.000 MAUs, é cobrado US$ 0,0055 para cada novo usuário (AMAZON, 2023d).

5.2.3 Amazon API Gateway

O cálculo para o serviço Amazon API Gateway é baseado no número de solicitações à API. Uma requisição à API pode representar uma busca de paciente, de sessão, ou até mesmo o *upload* de um arquivo, ou a criação de uma nova sessão. Para estimar o número de requisições, presume-se que a plataforma tenha 500 psicólogos ativos, e, que cada um deles realize, em média, 15 requisições por hora durante o horário comercial (8 horas diárias). Com isto, é estimado 60.000 requisições por dia, 1.825.000 requisições por mês. O custo de uma solicitação ao REST API do serviço Amazon API Gateway é de US$ 0,0000035 (AMAZON, 2023b). Com isso, temo que

$$1.825.000 \text{ requisições por mês} \times \text{US\$ } 0,0000035 \text{ por requisição} = \text{US\$ } 6,39 \text{ por mês}$$

Logo, o custo mensal do serviço de API é de US$ 6,39. Caso o aplicativo desenvolvido utilizasse uma HTTP API (apresentado na seção 4.3.3), este custo diminuiria para US$ 1,82.

5.2.4 AWS Lambda

Como o AWS Lambda é cobrado pelo custo computacional e pelo número de requisições, é necessário uma estimativa da alocação de memória e do tempo de resposta

médio das funções Lambdas, juntamente com o número de invocações deste serviço por mês. Assumido 60.000 requisições por dia (1.825.000 por mês), tempo de execução médio de 0,12 segundo e 0,125 GB de alocação de memória por invocação, temos 219.000 segundos de computação por mês ($1.825.000$ requisições $\times$ $0,12$ segundo por requisição) e 27.375 GB-segundo utilizado ($0,125$ GB $\times$ 219.000 segundos). O gigabyte-segundo de execução da função Lambda é US$ 0,0000166667 (AMAZON, 2023n), logo,

$$27.375 \text{ GB-s por mês} \times \text{US\$ } 0,0000166667 \text{ por GB-s} = \text{US\$ } 0,46 \text{ por mês}$$

O valor por requisição ao AWS Lambda é de US$ 0,0000002 (AMAZON, 2023n), assim,

$$1.825.000 \text{ requisições por mês} \times \text{US\$ } 0,0000002 \text{ por requisição} = \text{US\$ } 0,36 \text{ por mês}$$

Com isso, o custo mensal de todas funções Lambdas utilizadas no projeto é de US$ 0,82 (US$ $0,46$ + US$ $0,36$).

5.2.5 Amazon DynamoDB

O Amazon DynamoDB é precificado de acordo com a quantidade de dados armazenados. Presume-se que seja utilizado 5 GB de armazenamento do Amazon DynamoDB, que representa 500.000 itens de 10 KB cada. Como o valor de 1 GB do Amazon DynamoDB custa US$ 0,25 (AMAZON, 2023e), temos

$$5 \text{ GB por mês} \times \text{US\$ } 0,25 \text{ por GB} = \text{US\$ } 1,25 \text{ por mês}$$

5.2.6 Amazon S3

Supondo que cada psicólogo na aplicação tenha 20 pacientes e, para cada paciente, seja armazenado dois arquivos de, em média, 500 KB cada. Esses valores representam 10 GB em armazenamento de documentos, como laudos e imagens, para 500 psicólogos ativos. O valor para o gigabyte de uso do Amazon S3 é de US$ 0,023 (AMAZON, 2023h).

Deste modo,

$$10 \text{ GB por mês} \times \text{US\$ } 0,023 \text{ por GB} = \text{US\$ } 0,23 \text{ por mês}$$

5.2.7 Custo total

Com isto, o custo total para manter a arquitetura descrita no presente trabalho funcionando interruptamente é de aproximadamente US$ 11,39 mensal. Este valor representa uma das principais vantagens de um software baseado em uma arquitetura *serverless*, uma vez que sem o provisionamento de servidores, é cobrado somente o processamento realmente usado. Além disso, a infraestrutura pode escalar automaticamente para atender a demanda e reduzir quando não é necessária, permitindo um uso mais eficiente dos recursos, resultando em uma economia de custos.

6 CONCLUSÃO

O presente trabalho propõe o desenvolvimento e avaliação de um sistema de gerenciamento de atendimentos psicológicos baseado na arquitetura *serverless*. O objetivo do projeto consiste em apresentar uma arquitetura eficiente e de baixo custo para auxiliar os psicólogos a gerenciarem seus pacientes e sessões. Utilizando os serviços da AWS, como AWS Lambda, Amazon API Gateway, Amazon DynamoDB, e outros, foi possível implementar uma aplicação segura, flexível e resiliente. O projeto foi avaliado por meio do tempo de resposta de cada um de seus fluxos e com uma estimativa de custo mensal.

De acordo com os resultados do estudo, a aplicação desenvolvida apresentou tempos de resposta rápidos e consistentes para todas as operações. A eficiência na resposta é fundamental para a usabilidade da plataforma, demonstrando a aplicabilidade de uma arquitetura *serverless* para o projeto final. Em relação ao custo, a análise revelou que a solução apresenta um custo operacional baixo, devido a implementação *serverless*, na qual é cobrado apenas pelos recursos utilizados. Essa otimização de custos é fundamental para a viabilidade do sistema, tornando-o acessível a uma vasta gama de profissionais da área de psicologia.

O trabalho apresentou uma arquitetura *serverless* para um sistema de gerenciamento de atendimentos psicológicos, demonstrando eficiência tanto em termos de desempenho quanto de custos. Ao facilitar tarefas de gerenciamento e melhorar a organização das consultas e informações dos pacientes, este sistema pode se tornar uma ferramenta proveitosa para profissionais de psicologia. A abordagem *serverless* mostrou-se benéfica em termos de escalabilidade, eficiência e economia. Diante dessas conclusões, é perceptível o potencial que a utilização de tecnologias em nuvem e a arquitetura *serverless* oferecem não só para a área da psicologia, mas para diversas outras aplicações na área da saúde.

Olhando para o futuro, há várias áreas potenciais para a ampliação e aprimoramento deste trabalho. Por utilizar o provedor de *cloud* AWS, vários componentes estão disponíveis para a integração com a arquitetura proposta. Serviços como Amazon Chime, de video-chamadas, ou Amazon SES, de envio de e-mail, são só alguns exemplos de funcionalidades que podem ser adicionadas. Além disso, trabalhos futuros podem realizar estudos comparativos entre a arquitetura *serverless* e as arquiteturas tradicionais para avaliar as diferenças de desempenho e custo nesse contexto.

REFERÊNCIAS

AMAZON. **Add social sign-in to a user pool**. 2023. Acesso em: 05 de jan. de 2024. Disponível em: <https://docs.aws.amazon.com/cognito/latest/developerguide/cognito-user-pools-configuring-federation-with-social-idp.html>.

AMAZON. **Amazon API Gateway**. 2023. Acesso em: 05 de jan. de 2024. Disponível em: <https://aws.amazon.com/api-gateway/>.

AMAZON. **Amazon CloudWatch**. 2023. Acesso em: 05 de jan. de 2024. Disponível em: <https://aws.amazon.com/cloudwatch/>.

AMAZON. **Amazon Cognito**. 2023. Acesso em: 05 de jan. de 2024. Disponível em: <https://aws.amazon.com/cognito/>.

AMAZON. **Amazon DynamoDB**. 2023. Acesso em: 05 de jan. de 2024. Disponível em: <https://aws.amazon.com/dynamodb/>.

AMAZON. **Amazon DynamoDB resources**. 2023. Acesso em: 05 de jan. de 2024. Disponível em: <https://aws.amazon.com/dynamodb/resources/>.

AMAZON. **Amazon Relational Database Service**. 2023. Acesso em: 05 de jan. de 2024. Disponível em: <https://aws.amazon.com/rds/>.

AMAZON. **Amazon S3**. 2023. Acesso em: 05 de jan. de 2024. Disponível em: <https://aws.amazon.com/s3/>.

AMAZON. **Amazon S3 pricing**. 2023. Acesso em: 05 de jan. de 2024. Disponível em: <https://aws.amazon.com/s3/pricing/>.

AMAZON. **Amazon S3 Storage Classes**. 2023. Acesso em: 05 de jan. de 2024. Disponível em: <https://aws.amazon.com/s3/storage-classes>.

AMAZON. **Amplify Dev Center**. 2023. Acesso em: 05 de jan. de 2024. Disponível em: <https://ui.docs.amplify.aws/>.

AMAZON. **AWS Amplify**. 2023. Acesso em: 05 de jan. de 2024. Disponível em: <https://aws.amazon.com/amplify/>.

AMAZON. **AWS Amplify Hosting**. 2023. Acesso em: 05 de jan. de 2024. Disponível em: <https://aws.amazon.com/amplify/hosting>.

AMAZON. **AWS Lambda**. 2023. Acesso em: 05 de jan. de 2024. Disponível em: <https://aws.amazon.com/lambda/>.

AMAZON. **AWS X-Ray**. 2023. Acesso em: 05 de jan. de 2024. Disponível em: <https://aws.amazon.com/xray/>.

AMAZON. **Banco de dados**. 2023. Acesso em: 05 de jan. de 2024. Disponível em: <https://docs.aws.amazon.com/pt_br/whitepapers/latest/aws-overview/database.html>.

Amazon. **Choosing between REST APIs and HTTP APIs**. 2023. Acesso em: 05 de jan. de 2024. Disponível em: <https://docs.aws.amazon.com/apigateway/latest/developerguide/http-api-vs-rest.html>.

AMAZON. **Control access to a REST API using Amazon Cognito user pools as authorizer**. 2023. Acesso em: 05 de jan. de 2024. Disponível em: <https://docs.aws.amazon.com/apigateway/latest/developerguide/apigateway-integrate-with-cognito.html>.

AMAZON. **DeleteItem**. 2023. Acesso em: 05 de jan. de 2024. Disponível em: <https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_DeleteItem.html>.

AMAZON. **DeleteUser**. 2023. Acesso em: 05 de jan. de 2024. Disponível em: <https://docs.aws.amazon.com/cognito-user-identity-pools/latest/APIReference/API_DeleteUser.html>.

AMAZON. **Enabling API caching to enhance responsiveness**. 2023. Acesso em: 05 de jan. de 2024. Disponível em: <https://docs.aws.amazon.com/apigateway/latest/developerguide/api-gateway-caching.html>.

AMAZON. **GetUser**. 2023. Acesso em: 05 de jan. de 2024. Disponível em: <https://docs.aws.amazon.com/cognito-user-identity-pools/latest/APIReference/API_GetUser.html>.

AMAZON. **Key condition expressions for the Query operation**. 2023. Acesso em: 05 de jan. de 2024. Disponível em: <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Query.KeyConditionExpressions.html>.

AMAZON. **Lambda execution environments**. 2023. Acesso em: 05 de jan. de 2024. Disponível em: <https://docs.aws.amazon.com/lambda/latest/operatorguide/execution-environments.html>.

AMAZON. **Lambda executions**. 2023. Acesso em: 05 de jan. de 2024. Disponível em: <https://docs.aws.amazon.com/whitepapers/latest/security-overview-aws-lambda/lambda-executions.html>.

AMAZON. **PutItem**. 2023. Acesso em: 05 de jan. de 2024. Disponível em: <https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_PutItem.html>.

AMAZON. **Query**. 2023. Acesso em: 05 de jan. de 2024. Disponível em: <https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_Query.html>.

AMAZON. **RespondToAuthChallenge**. 2023. Acesso em: 05 de jan. de 2024. Disponível em: <https://docs.aws.amazon.com/cognito-user-identity-pools/latest/APIReference/API_RespondToAuthChallenge.html>.

AMAZON. **UpdateItem**. 2023. Acesso em: 05 de jan. de 2024. Disponível em: <https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_UpdateItem.html>.

AMAZON. **UpdateUserAttributes**. 2023. Acesso em: 05 de jan. de 2024. Disponível em: <https://docs.aws.amazon.com/cognito-user-identity-pools/latest/APIReference/API_UpdateUserAttributes.html>.

AMAZON. **User pool authentication flow**. 2023. Acesso em: 05 de jan. de 2024. Disponível em: <https://docs.aws.amazon.com/cognito/latest/developerguide/amazon-cognito-user-pools-authentication-flow.html>.

AMAZON. **What is AWS**. Filbert Pub., 2023. Disponível em: <https://aws.amazon.com/pt/what-is-aws/>.

APPHEALTH. **AppHealth**. 2022. Https://www.apphealth.com.br/. Accessed: 2023-08-15.

ASSOCIATION, A. P. **Psychologists struggle to meet demand amid mental health crisis: 2022 COVID-19 practitioner impact survey.** 2022. Https://www.apa.org/pubs/reports/practitioner/2022-covid-psychologist-workload. Accessed: 2023-08-15.

AWS. **AWS Pricing Calculator**. 2024. Acesso em: 05 de jan. de 2024. Disponível em: <https://calculator.aws/>.

CHAND, B. P. S. **Serverless Architecture for Bulk Email Management**. 2021.

DIAS, R. D. S. et al. Telemental health in brazil: past, present and integration into primary care. **Archives of Clinical Psychiatry (São Paulo)**, Faculdade de Medicina da Universidade de São Paulo, v. 42, n. 2, p. 41–44, Mar 2015. ISSN 0101-6083. Disponível em: <https://doi.org/10.1590/0101-60830000000046>.

ELHEMALI, M. et al. Amazon DynamoDB: A scalable, predictably performant, and fully managed NoSQL database service. In: **2022 USENIX Annual Technical Conference (USENIX ATC 22)**. Carlsbad, CA: USENIX Association, 2022. p. 1037–1048. ISBN 978-1-939133-29-46. Disponível em: <https://www.usenix.org/conference/atc22/presentation/elhemali>.

FOWLER, M. **Patterns of Enterprise Application Architecture**. [S.l.]: Addison-Wesley Professional, 2002.

FOWLER, M. **Microservices**. 2014. Disponível em: <https://martinfowler.com/articles/microservices.html>.

GARTNER. **The CIO's Guide to Serverless Computing**. 2022. Disponível em: <https://www.gartner.com/smarterwithgartner/the-cios-guide-to-serverless-computing>.

GOOGLE. **App Engine**. 2023. Acesso em: 05 de jan. de 2024. Disponível em: <https://cloud.google.com/appengine>.

GOOGLE. **Gmail**. 2024. Acesso em: 05 de jan. de 2024. Disponível em: <https://www.google.com/gmail/about/>.

JAMES BESWICK. **Creating a single-table design with Amazon DynamoDB**. 2021. Acesso em: 05 de jan. de 2024. Disponível em: <https://aws.amazon.com/blogs/compute/creating-a-single-table-design-with-amazon-dynamodb/>.

KHAN, W. et al. **SQL and NoSQL Databases Software architectures performance analysis and assessments – A Systematic Literature review**. 2022.

KUMARI, A.; SAHOO, B. Serverless architecture for healthcare management systems. In: ____. [S.l.: s.n.], 2022.

LUXNER, T. **Cloud computing stats: Flexera 2023 State of the cloud report**. Flexera, 2023. Disponível em: <https://www.flexera.com/blog/cloud/cloud-computing-trends-flexera-2023-state-of-the-cloud-report/>.

MICROSOFT. **App Service**. 2023. Acesso em: 05 de jan. de 2024. Disponível em: <https://azure.microsoft.com/en-us/products/app-service>.

MICROSOFT. **Microservice architecture style**. 2024. Disponível em: <https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices>.

Moodle. **Moodle**. 2023. Acesso em: 05 de jan. de 2024. Disponível em: <https://moodle.org/>.

MURTY, J. S3 applications. In: ____. **Programming Amazon Web Services S3, EC2, Sqs, fps, and simpledb**. [S.l.]: O'Reilly, 2008.

MüHLEN, D. von. Psicologia virtual: Resistência e possibilidades. **Universidade Luterana do Brasil**, 2020.

NAH, F. A study on tolerable waiting time: how long are web users willing to wait? **Behaviour & IT**, v. 23, 01 2004.

NISHIHIRA, A. et al. Psight: uma proposta de software de gestão de agendamentos para otimização de atendimentos de profissionais da psicologia. São Caetano do Sul, 2020.

NISHIMIYA, K.; IMAI, Y. **Serverless Architecture for Service Robot Management System**. 2021.

OKTA. **Access Tokens**. 2023. Acesso em: 05 de jan. de 2024. Disponível em: <https://auth0.com/docs/secure/tokens/access-tokens>.

PSICOMANAGER. **PsicoManager**. 2023. Https://www.psicomanager.com.br/. Accessed: 2023-08-15.

RESEARCH, A. M. **Serverless Architecture Market by Deployment Model, Application, Organization Size, and Industry Vertical: Global Opportunity Analysis and Industry Forecast, 2018-2025**. 2019. Disponível em: <https://www.researchandmarkets.com/reports/4828585/serverless-architecture-market-by-deployment>.

RIMAL, B. P. et al. Architectural requirements for cloud computing systems: An enterprise cloud approach. **Journal of Grid Computing**, v. 9, n. 1, p. 3–26, Mar 2011. ISSN 1572-9184. Disponível em: <https://doi.org/10.1007/s10723-010-9171-y>.

Salesforce. **Salesforce**. 2023. Acesso em: 05 de jan. de 2024. Disponível em: <https://www.salesforce.com/>.

SALIM, Z. et al. Organisational-level assessment of cloud computing adoption: Evidence from the australian smes. **Journal of Global Information Management**, v. 28, p. 73–89, 04 2020.

SHAFIEI, H.; KHONSARI, A.; MOUSAVI, P. **Serverless Computing: A Survey of Opportunities, Challenges and Applications**. 2021.

TASNIM, R. et al. A comparative study on three selective cloud providers. **International Journal on Cybernetics & Informatics**, Academy and Industry Research Collaboration Center (AIRCC), v. 11, n. 4, p. 167–178, ago. 2022. ISSN 2277-548X. Disponível em: <http://dx.doi.org/10.5121/ijci.2022.110413>.

TAVEIRO, F. T. Análise do impacto de um requisito não funcional relacionado a usabilidade. **Instituto Federal de Educação, Ciência e Tecnologia de São Paulo**, 2016.

VERONA, H. C. Resolução cfp n° 011/ 2012. **Conselho Federal de Psicologia**, Brasília, DF, 2012. Disponível em: <https://site.cfp.org.br/wp-content/uploads/2012/07/Resoluxo_CFP_nx_011-12.pdf>.

WEN, J. et al. **Rise of the Planet of Serverless Computing: A Systematic Review**. 2022.

WEN, J.; CHEN, Z.; LIU, X. **Software Engineering for Serverless Computing**. 2022.