Universidade da Beira Interior
Departamento de Informática

Sistema Baseado em Microsserviços com Arquitetura sem Servidor

Elaborado por:

Cláudio Redondo

Orientador:

Professor Doutor Tiago Miguel Carrola Simões

1 de julho de 2023

Agradecimentos

A conclusão deste trabalho, bem como da grande maior parte da minha vida académica não seria possível sem a ajuda dos meus colegas de curso e do meu orientador Tiago Simões e de todos os meus professores ao longo do meu percurso académico. Desde já o meu agradecimento por toda a ajuda!

Conteúdo

Lista de Figuras

Lista de Excertos de Código

Acrónimos

IoT	*Internet of Things*
JSON	*JavaScript Object Notation*
Amazon RDS	Amazon Relational Database Service
API	*Application Programming Interface*
Amazon ECS	Amazon Elastic Compute Cloud
AWS	*Amazon Web Services*
VM	Maquina Virtual
CPU	Central Processing Unit
E/S	Entrada/Saída

Capítulo

1

Introdução

O seguinte relatório é referente ao projeto, da Unidade Curricular de Projeto, que consiste na criação de um sistema baseado em microsserviços com arquitetura *serverless*. Neste capitulo introdutório, serão apresentadas as motivações, onde se enquadra o projeto, os objetivos e a abordagem na criação. Por fim, apresentaremos toda a estrutura e organização do relatório de projeto.

1.1 Enquadramento e Motivação

Em contexto com o projeto apresentado, assistimos a uma crescente popularidade dos microsserviços e das arquiteturas *serverless* no desenvolvimento de *Software*, devido a seus sistemas serem escaláveis e flexíveis. Tendo em conta, a crescente popularidade e o facto de estes refletirem benefícios, como a sua flexibilidade quanto as tecnologias utilizadas, a sua escalabilidade flexível possibilitando um melhor dimensionamento quanto à capacidade de carga e estas terem uma implementação independente e ágil, permitindo uma implementação mais rápida dos microsserviços, fez com que o uso de microsserviços e arquitetura *serverless* fossem opções a considerar para a construção de serviços como um gestor de eventos.

1.2 Objetivos

O objetivo principal, é o estudo de tecnologias de microsserviços e a implementação de um sistema baseado em microsserviços com arquitetura sem servidor, para a gestão de eventos. Tendo em vista o objetivo principal, foram propostos objetivos secundários sendo esses:

- Revisão do estado da arte relacionado a microsserviços e arquitetura *serverless*;
- Seleção ou proposta da arquitetura de microsserviços subjacente a um sistema que gerência eventos. Essa proposta deve incluir serviços, dependências e protocolos de comunicação;
- Implementação de microsserviços utilizando uma linguagem de programação adequada;
- Integração e implantação de microsserviços através de uma tecnologia *serverless*;
- Implantação da aplicação numa plataforma de nuvem (por exemplo, ***Amazon Web Services*** **(AWS)**), **Google Cloud**).

1.3 Abordagem

A abordagem utilizada neste projeto, passa pela criação de microsserviços, usando a tecnologia *serverless* Lambda function, fornecida pela AWS, que possibilitará a gestão dos eventos pretendidos com a ajuda de outros dois serviços desta, um que armazenará os dados (**Amazon DynamoDB**) e outra que permitirá a comunicação entre utilizador e as funções lambda (**Amazon API Gateway**). Utilizando os serviços descritos acima, será possível a criação de vários microsserviços que permitam a gestão de eventos.

1.4 Organização do Relatório de Projeto

Este relatório de projeto encontra-se estruturado da seguinte forma:

1. O primeiro capítulo – **Introdução** – apresenta o projeto, a motivação, o enquadramento, os seus objetivos e a respetiva organização do relatório de projeto;
2. O segundo capítulo – **Tecnologias Utilizadas e Estado da Arte** – enuncia algumas tecnologias utilizadas na criação de microsserviços e explicita o que são microsserviços e arquiteturas *serverless*;
3. O terceiro capítulo – **Engenharia de Software** – é descrita a abordagem feita para desenvolver os microsserviços para a gestão de eventos;

4. O quarto capítulo – **Implementação** – Demonstração de como criar os serviços e algumas considerações quanto à segurança e a ligação entre estes.

5. O quinto capítulo – **Demonstração e Validação** – explica as funcionalidades disponíveis e como interagir com estas;

6. O sexto capítulo – **Conclusões e Trabalho Futuro** – conclusão de todo o trabalho realizado e apresentação de possíveis trabalhos futuros.

Capítulo

2

Tecnologias Utilizadas e Estado da Arte

2.1 Introdução

Neste capitulo, introduzimos tecnologias usadas na criação de microsserviços e uma análise às arquiteturas baseadas em microsserviços, às arquiteturas monolíticas e às arquiteturas *serverless*.

2.2 Tecnologias Relacionadas

Nesta secção apresentamos algumas plataformas e tecnologias que podem ser usadas para desenvolver projetos semelhantes ao realizado, algumas delas como **Google Cloud Platform**, **Amazon Relational Database Service** ou **Amazon Elastic Container Service**.

2.2.1 Google Cloud Platform

Google Cloud Platform é uma plataforma da **Google Cloud** que reúne um conjunto de serviços de computação em nuvem, entre eles serviços de armazenamento e base de dados, serviços de identidade e segurança, serviços de*Internet of Things* (IoT), plataformas de *Application Programming Interface* (API) entre outras. Esta tem serviços semelhantes, a outras plataformas, de outros fornecedores, como a **Microsoft Azure**, a **Oracle Cloud** e a plataforma utilizada neste projeto, a **Amazon Web Services**. Esta e as referidas para além do mesmo propósito tem também algumas tecnologias bem

parecidas como a Funções do **Google Cloud**, da **Google**, e a **AWS Lambda**, da **Amazon**, utilizada neste projeto.([4–6])

2.2.2 Amazon Relational Database Service (Amazon RDS)

A **Amazon Relational Database Service** é um serviço web, projetado para simplificar a configuração, a operação e o escalonamento de uma base de dados relacional "em nuvem".Podendo o utilizador gerir os seus dados por **MySQL**, **PostgreSQL**, **MariaDB**, **Oracle BYOL** ou **SQL Server**. A **Amazon** oferece ainda outras opções de serviços de base de dados "em nuvem", entre elas a **Amazon DynamoDB** que será mais adiante referenciada e na qual foram hospedados os dados do projeto.([7])

2.2.3 Amazon Elastic Container Service (Amazon ECS)

O Amazon Elastic Container Service (Amazon ECS) é um serviço de orquestração de *containers* altamente escalável e de elevado desempenho, que permite executar e escalar facilmente aplicações em *containers* nos serviços AWS. Este elimina a necessidade de instalar e operar os serviços de orquestração de *containers*, de gerir e dimensionar *clusters* de máquinas virtuais (Maquina Virtual (VM)) ou agendar *containers* nessas. Com uma simples chamada de API, este permite iniciar e parar uma aplicação, consultar o seu estado e aceder a outras funcionalidades da AWS.([7])

2.2.4 Containers

Container é uma unidade de implementação que empacota um ou mais serviços isolados, com as suas dependências e configurações, bibliotecas, *frameworks*, tempos de execução e o código do serviço. Cada uma destas unidades ou *containers* é executado num espaço isolado, compartilhando o mesmo Kernel do sistema operacional subjacente, mas separado dos outros *containers*. Alguns dos benefícios do uso destes são o isolamento(cada container é isolado dos outros, ou seja, um não afeta diretamente os outros), portabilidade (estes podem ser facilmente entre outros ambientes por serem empacotados em imagens), escalabilidade(estes podem ser dimensionados horizontalmente, o que significa que vários *containers* de um mesmo serviço podem ser executados simultaneamente para lidar com cargas menores ou maiores de trabalho), com uma gestão simplificada (estes são gerenciados por orquestradores de *containers* , como o Amazon Elastic Compute Cloud (Amazon ECS), que automatizam tarefas de implementação, escalabilidade, monitoramento e recuperação de falhas). Em resumo, estes permitem que

os desenvolveres construam e implementem sistemas distribuídos de forma mais flexível, eficiente e confiável.([8])

2.3 Tecnologias Utilizadas

2.3.1 Amazon API Gateway

A **Amazon API Gateway**, serviço fornecido pela AWS, possibilita a criação, implementação e gestão de API's, no qual desempenham o papel de *gateway* seguro para expor os serviços de *backend*. Sendo, algumas das características e funcionalidades-chave, deste serviço, a criação e gestão de API's, implementação de API's, segurança e Autorização, integração com serviços AWS, Monitoramento e Registo. Em suma, o **Amazon API Gateway** foi utilizado no desenvolvimento deste trabalho para a criação de API's para fazerem a ligação entre as respostas geradas pelos microsserviços gerenciados pelas várias funções Lambdas criados e o cliente, expondo as respostas vindas dos serviços *backend* (funções lambda) ao cliente e gerenciando os vários pedidos do cliente com as várias funções lambda.

2.3.2 Amazon DynamoDB

O **Amazon DynamoDB** é uma base de dados *Key-value* e de documentos que oferece um alto desempenho. Este é totalmente gerenciado, multirregional, *multimaster* com segurança integrada, serviço de *backup*, serviço de reposição e *cache*. Esta permite lidar com mais de 10 triliões de solicitações diárias e suporta picos de mais de 20 milhões de solicitações por segundo, possibilitando uma latência baixa em qualquer escala, ideal para aplicações móveis, aplicações web, jogos, tecnologias de anúncios, IoT entre outros. Empresas como a **Samsung**, **Toyota** e **Capital One** utilizam estas, para suportar as cargas de trabalho críticas diárias.([7])

2.3.3 AWS Lambda

O **AWS Lambda** é um serviço de computação sem servidor fornecido pela AWS. Este serviço permite executar código sem precisar de provisionar ou gerir servidores, permitindo escrever funções em diversas linguagens tais como **Python**, **JavaScript**, entre outras. Podendo esta ser configurada para ser executada automaticamente por outros serviços da AWS ou executados diretamente em qualquer aplicação "web"ou móvel. Devido ao seu dimensionamento automático em resposta as solicitações recebidas e a sua integração

com outras ferramentas da **Amazon** torna-se uma ferramenta útil na criação de microsserviços.([7])

2.4 Estilos Arquiteturais para o Desenvolvimento de Software

Neste capítulo, é apresentada uma visão geral dos microsserviços, das arquiteturas monolíticas e das arquiteturas *Serverless*, discutindo os seus conceitos fundamentais e a sua importância no desenvolvimento de *software* moderno.

2.4.1 Introdução aos Microsserviços

Nesta secção, são introduzidas as arquiteturas monolíticas e não monolíticas e as principais diferenças entre elas. ([1])

2.4.1.1 Arquiteturas Monolíticas

A arquitetura monolítica é um estilo de arquitetura de *software* em que uma aplicação é construída como um único e grande bloco de código. Onde, todas as funcionalidades estão agrupadas num único projeto, conforme demonstrado na figura seguinte 2.1, geralmente num único código-fonte, executadas num único processo ou servidor.

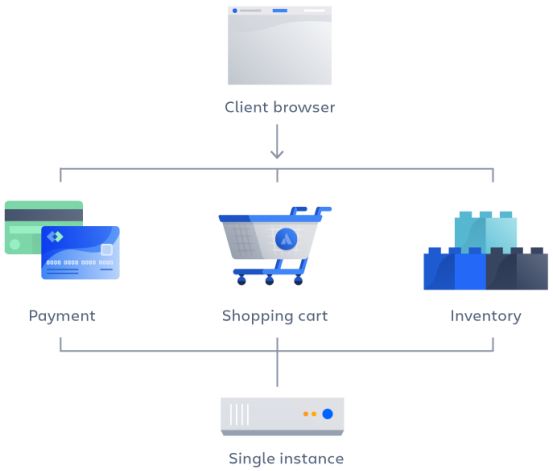

Figura 2.1: Arquitetura Monolítica. (Imagem Adaptada de [1])

Esta tem como vantagens:

- Fácil implementação e desenvolvimento-Por esta, usar apenas um ficheiro ou diretoria facilita a implementação e o seu desenvolvimento;
- Desempenho- uma API pode executar a mesma função que várias API's de microsserviços;
- Teste simplificado e fácil depuração - como a aplicação é uma unidade única facilita a realização de testes e como seguimos as solicitações e encontramos os itens que queremos.

E algumas das desvantagens são:

- Velocidade de desenvolvimento mais lenta - Por ser uma aplicação grande e monolítico torna que este seja mais complexo e lento;
- Escalabilidade- Os seus componentes não podem ser escaláveis individualmente;
- Confiabilidade - Em caso de erro em alguma funcionalidade, este pode afetar a disponibilidade de toda a aplicação;
- Barreira para adoção de tecnologias - Qualquer alteração na estrutura ou na linguagem afeta toda a aplicação;
- Falta de flexibilidade - Apenas podem ser usadas tecnologias inicialmente escolhidas e implementadas na arquitetura em questão;
- Implementação- uma pequena alteração no monólito requer a reimplementação de todo o monólito.

2.4.1.2 Arquiteturas baseadas em Microsserviços

Como referido anteriormente, as arquiteturas não monolíticas ou de microsserviços consistem na divisão de serviços, ou funcionalidades da aplicação para que estes sejam independentes e autónomos, conforme demonstrado na figura (2.2).

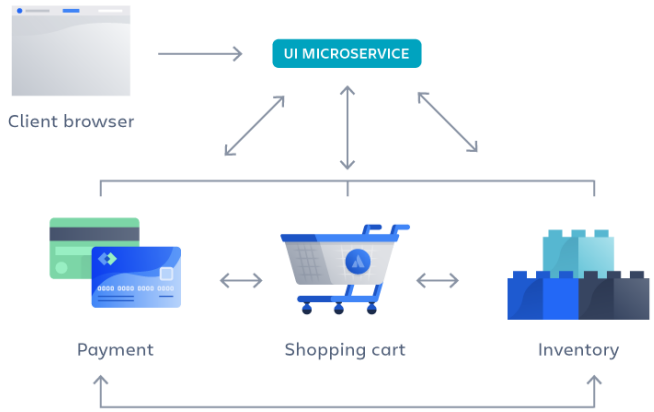

Figura 2.2: Arquitetura não Monolítica. (Imagem adaptada de [1])

Possibilitando que os diversos serviços possam ser criados de diversas formas e usando linguagens de programação diferentes, e permite também que estes sejam escaláveis autonomamente. Estas têm como vantagens:

- Agilidade- promove maneiras ágeis de trabalhar com equipas de pequena dimensão, possibilitando que cada uma seja encarregue de uma funcionalidade;
- Escalabilidade flexível- se os microsserviços atingirem a capacidade de carga, novas instâncias desse serviço podem ser criadas para aliviar a pressão sobre estes;
- Implementação continua, sustentável e testável- As equipas podem experimentar novos recursos e reverter se algo não funcionar, facilitando a atualização do código, e a correção isolada de falhas ou *bugs* em serviços individuais;
- Implementável com independência - como estes são individuais, permitem uma implementação independente rápida e fácil dos recursos;
- Flexibilidade tecnológica- permite que cada equipa use as ferramentas que quiserem;
- Alta confiabilidade - as equipas podem implementar alterações para um serviço específico, sem precisar de derrubar a aplicação.

E algumas das desvantagens são:

- Expansão do desenvolvimento- os microsserviços adicionam complexidade ao desenvolvimento em comparação a estruturas monolíticas. Se a expansão do desenvolvimento não for gerida adequadamente, a velocidade de desenvolvimento fica mais lenta e o desempenho operacional não será o melhor;
- Escalabilidade flexível- Custo exponencial em infraestruturas, cada novo microsserviço pode ter o custo próprio para o conjunto de testes, esquema de implementação, infraestrutura de hospedagem, ferramentas de monitoramento, entre outros;
- Desafios de depuração -cada microsserviço tem um conjunto de registos próprios, tornando a depuração mais complexa.

Como referido, uma arquitetura de microsserviços (geralmente chamada microsserviços) é um estilo de desenvolvimento de *software*. Esta consiste na separação do *software* ou aplicação em diversas partes, todas elas independentes e cada uma com a sua própria responsabilidade e objetivo. Diferenciando-se de arquiteturas de serviços monolíticos que consistem na execução dos processos num único serviço. Com esta abordagem podemos solicitar diversos microsserviços para fornecer uma resposta objetiva e eficiente ao cliente.(Figura 2.3)

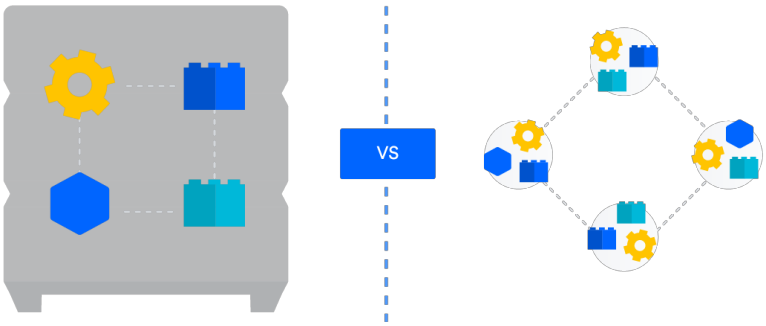

Figura 2.3: Arquitetura Monolítica e não Monolítica. (Imagem Adaptada de [1])

2.4.2 Arquiteturas *Serverless*

Uma arquitetura *Serverless* é uma maneira de criar e executar aplicações e serviços sem precisar de ter nenhuma infraestrutura própria para a sua ges-

tão. Esta arquitetura evoluiu ao longo do tempo, as primeiras implementações desta foram realizadas em servidores físicos, onde apenas podia existir uma instância do sistema operativo em execução num determinado momento. Mais tarde, alternando operações de Central Processing Unit (CPU) e Entrada/Saída (E/S), foi possível vários computadores virtuais executarem o mesmo *hardware* em simultâneo. Com o surgimento da "nuvem"e o surgimento posterior, em 2006, da tecnologia de containers Linux possibilitou uma grande mudança nas arquiteturas de *software* existentes, apesar de a tecnologia de containers ainda apresentar algumas lacunas na sua implementação. Na figura 2.4, observamos a evolução das infraestruturas apresentadas.

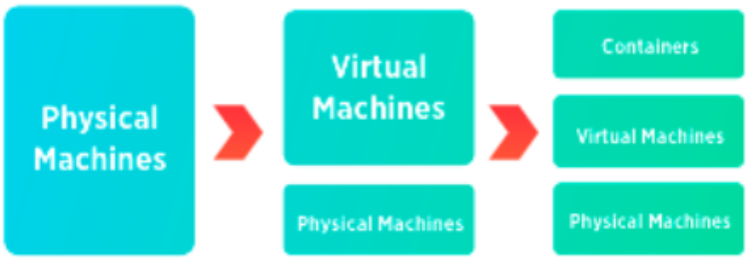

Figura 2.4: Figura Representativa da Evolução das Infraestruturas. (Imagem Adaptada de [2])

Em 2013, após o surgimento do conceito de microsserviços, no ano anterior, a empresa **Docker**, melhora a tecnologia de containers já existente preenchendo as suas lacunas, a acessibilidade, a usabilidade, e serviços de suporte no seu ecossistema. Iniciando a sua popularização e possibilitando a decomposição de sistemas monolíticos em serviços individuais. Mais tarde, em 2015, a AWS introduziu o **AWS Lambda** que poderia economizar ainda mais os custos de implementação de *software* executado e microsserviços, interrompendo-os quando este não estiverem sobe carga. A figura 2.5 apresenta a evolução das metodologias de *software* enunciadas anteriormente.

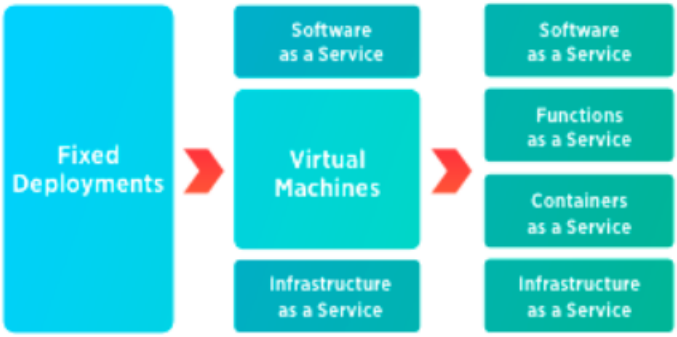

Figura 2.5: Figura Representativa da Evolução das Metodologias de Implementação de *Software*. (Imagem Adaptada de [2])

Permitindo agora que as plataformas automatizem todo o processo de criação, implementação e início de serviço sob demanda, assim o utilizador apenas precisa de registar as suas funções e os seus recursos. Estas funções, podem ainda, ser classificadas em dois tipos, funções acionadas pela solicitação de um cliente ou acionadas em segundo plano por um *trigger*.Na figura 2.6, observamos um exemplo de arquitetura *serverless*, baseada em microsserviços, com uso de tecnologias AWS.([2, 3, 9])

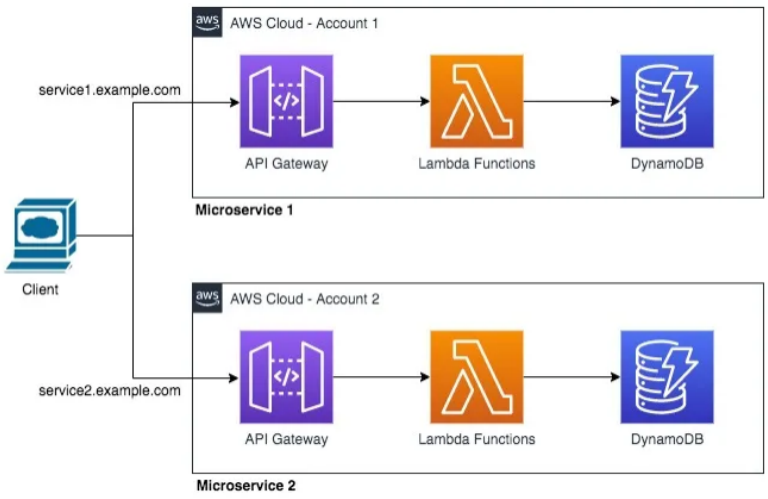

Figura 2.6: Figura Representativa de uma Arquitetura *Serverless*. (Imagem Adaptada de [3])

2.5 Conclusão

Por fim, neste capitulo foram abordadas algumas plataformas e algumas das suas ferramentas que possibilitam uma mais fácil criação de microsserviços, como o **Amazon Relational Database Service (Amazon RDS)** ou o **Amazon API Gateway**. Por fim, abordamos alguns conceitos que nos permitam perceber o objetivo deste trabalho e as arquiteturas utilizadas para sua realização, tais como microsserviços, arquiteturas monolíticas e arquiteturas *serverless*.

Capítulo

3

Engenharia de Software

3.1 Introdução

Neste capítulo, é apresentada a metodologia usada para criar microsserviços para uma aplicação de gestão de eventos. Este, contém os requisitos funcionais e não funcionais do projeto, os diagramas de caso de uso e de atividade, a modelação da base de dados e a arquitetura geral dos microsserviços.

3.2 Análise de Requisitos

A análise de requisitos visa a identificar, a analisar e a descrever todos os requisitos que o sistema exige. A seguinte análise está dividida em dois tipos de requisitos: funcionais e não funcionais.

3.2.1 Requisitos Funcionais

Os requisitos funcionais descrevem o que o sistema poderá fazer, como irá funcionar e as funcionalidades disponíveis para o utilizador. Os requisitos são os seguintes:

- A aplicação deverá possibilitar o registo de utilizadores;
- A aplicação permitirá que o utilizador se autentique, para usar os diversos serviços;
- A aplicação permitirá criar um evento;
- A aplicação permitirá eliminar um evento;

- A aplicação permitirá editar um evento;
- A aplicação permitirá observar os dados do evento;
- A aplicação permitirá criar um local;
- A aplicação permitirá eliminar um local;
- A aplicação permitirá observar dados do local.

3.2.2 Requisitos não Funcionais

Os requisitos não funcionais, do projeto, estabelecem as características e qualidades que vão alem das funcionalidades.

- O sistema será suportado por uma API, fornecida pela AWS.
- O sistema terá de ter uma base de dados, para armazenamento de dados.
- As funções Lambda puderam ser implementadas utilizando linguagens de programação que a suportem, como exemplo, **Java**, **Go**, **PowerShell**, **Node.js**, **Python** e **Ruby**.
- os dados armazenados não poderão ser divulgados nem partilhados.
- Os serviços terão de ser multiutilizador
- Os serviços têm de ter mecanismos de segurança na utilização das API's, por exemplo, via *token*.

3.3 Diagramas de Casos de Uso

Para facilitar a identificação das funcionalidades que deverão estar implementadas pelos microserviços para a gestão de eventos, foram criados diferentes casos de uso. Os casos de uso, ilustrados nesta secção, ilustram as funcionalidades disponibilizadas pelos microserviços criados, e como eles se podem relacionar entre si. Na Figura 3.1 é ilustrado as principais funcionalidades dos microserviços fornecidos. Nesta, pode-se verificar que o utilizador após utilizar os serviços de autenticação poderá usar outros dois serviços disponíveis, o gestor de eventos e o gestor de locais, onde ambos produziram várias interações com as bases de dados fornecidas para cada gestor de eventos.

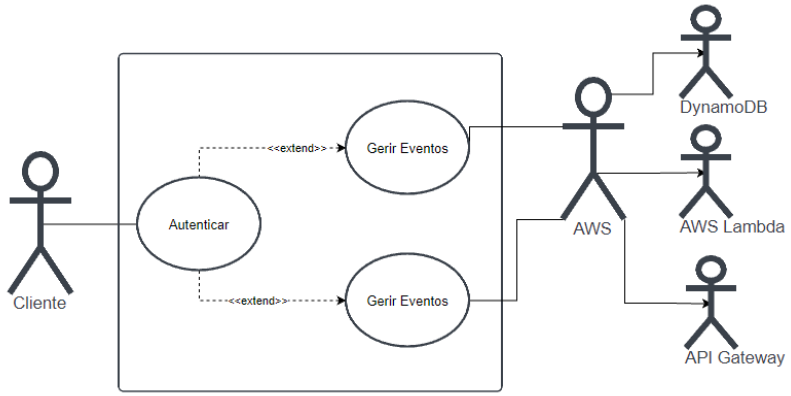

Figura 3.1: Caso de Uso Referente aos Microsserviços.

As funcionalidades fornecidas pelo microsserviço de autenticação serão, criar utilizador para que este possa utilizar os serviços desenvolvidos, verificação de identidade, que permitirá verificar se os dados passados pelo utilizador são os corretos, e se este pode usar os serviços seguintes (Figura 3.2). Estes permitiram que, os seus utilizadores, utilizem este serviço para registo e autenticação numa aplicação *web*, *website*, ou uma aplicação móvel.No qual apenas será necessário fazer a comunicação com a API em formato ***JavaScript Object Notation* (JSON)**.

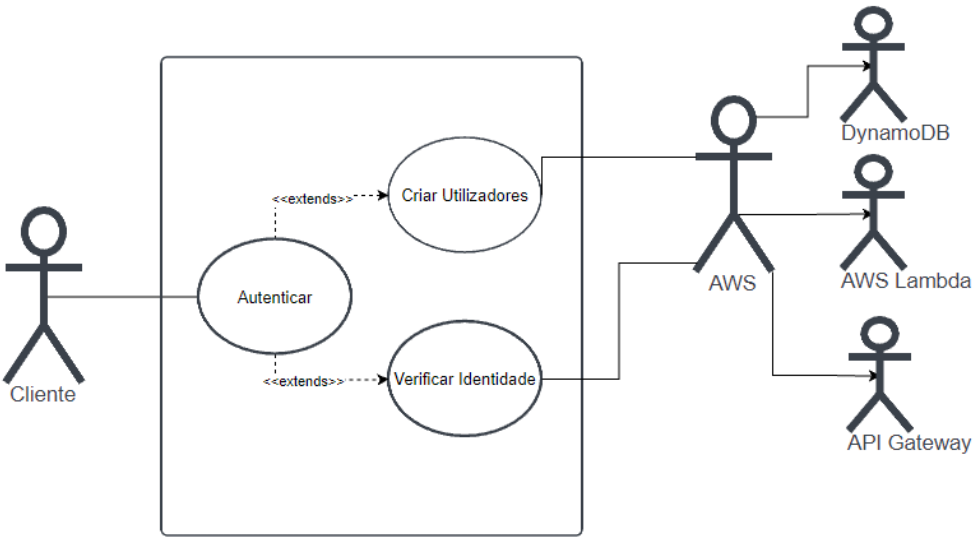

Figura 3.2: Caso de Uso Referente ao Serviço de Autenticação.

Na figura 3.3 são ilustradas as funcionalidades referentes à gestão de eventos. As funcionalidades referidas são a criação de eventos e a eliminação de eventos da base de dados, editar eventos já criados e ver os eventos criados pelo utilizador. Caso o utilizador seja verificado pela operação de autenticação, como visto na figura 3.1, este poderá utilizar estas funcionalidades enviando uma solicitação á API, num formato JSON, pré-definido, permitindo a utilização destes e o acesso esperado à base de dados fornecida para este serviço.

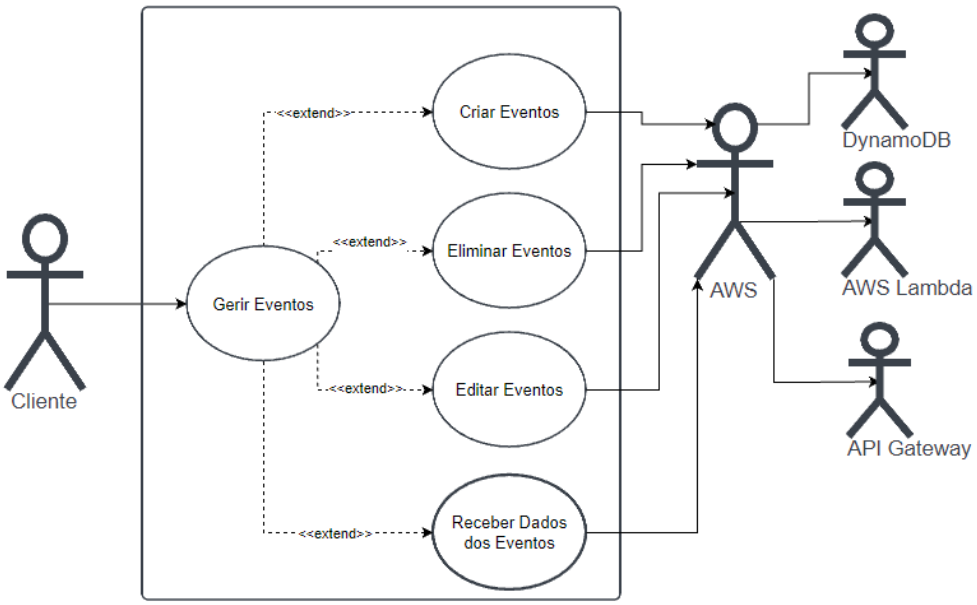

Figura 3.3: Caso de Uso Referente ao Gestor de Eventos.

Na Figura 3.4, podemos evidenciar as funcionalidades que o gestor de locais permite. Estas são, criar um local, eliminar um local e listar os dados de um local ou de todos os locais de um dado utilizador. Para o uso destas funcionalidades, em semelhança com o gestor de eventos, o utilizador tem de ser verificado pela operação de autenticação e enviar os dados em formato JSON, pré-definido, por meio de uma solicitação á API, permitindo o uso correto destas funcionalidades e o resultado esperado pelo utilizador.

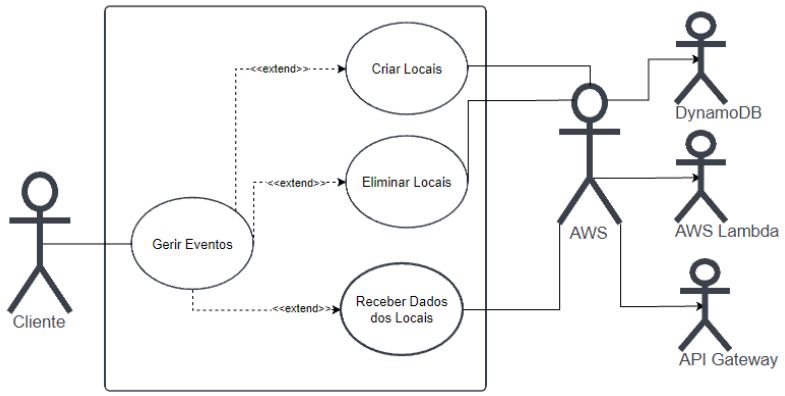

Figura 3.4: Caso de Uso Referente ao Gestor de Locais.

3.3.1 Diagrama de Atividade

Os diagramas de atividade apresentados nas figuras 3.5, (3.6) e (3.7) apresentam os microsserviços de autenticação, de gestão de eventos e de gestão de locais no ponto de vista do utilizador, apresentando o fluxo de ações que este poderá tomar. Observando a figura 3.5, concluímos que o utilizador no processo de autenticação, decide se quer se registar ou autenticar-se na sua conta. Em caso de este querer se registar, envia os seus dados de registo ao microsserviço e este retorna uma mensagem de sucesso ou insucesso, registando ou não os seus dados na base de dados, e terminará o processo. Se este optar, por autenticar-se este envia os seus dados de acesso, para que este os verifique. Caso a verificação destes seja válida o microsserviço retorna um *token* de autenticação, a ser usado nos outros microsserviços, caso esta seja inválida, recebe uma mensagem de erro.

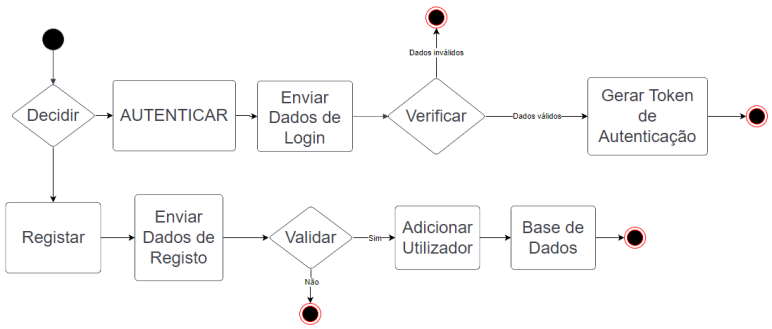

Figura 3.5: Diagrama de Atividade do Microsserviço de Autenticação.

Na figura 3.6 é nos apresentado o diagrama de atividade para o gestor de eventos. Onde, após o processo de autenticação no microsserviço de autenticação e usando o *token* por ele disponibilizado, esta recebe o *token* e faz uma operação de verificação, caso este seja válido o utilizador tem acesso às funcionalidades do gestor de eventos, caso contrário este termina o processo e nega o acesso a estas funcionalidades. Se este tiver acesso às funcionalidades, tem de escolher a funcionalidade pretendida, criar, eliminar, editar ou listar eventos, e a seguir enviar os dados pretendidos. Se os dados forem inválidos o processo terminará, caso isso não aconteça são feitas as devidas alterações na base de dados, adicionar elemento, eliminar elemento, modificar elemento ou retornar os dados, presentes na base de dados, que este solicitou.

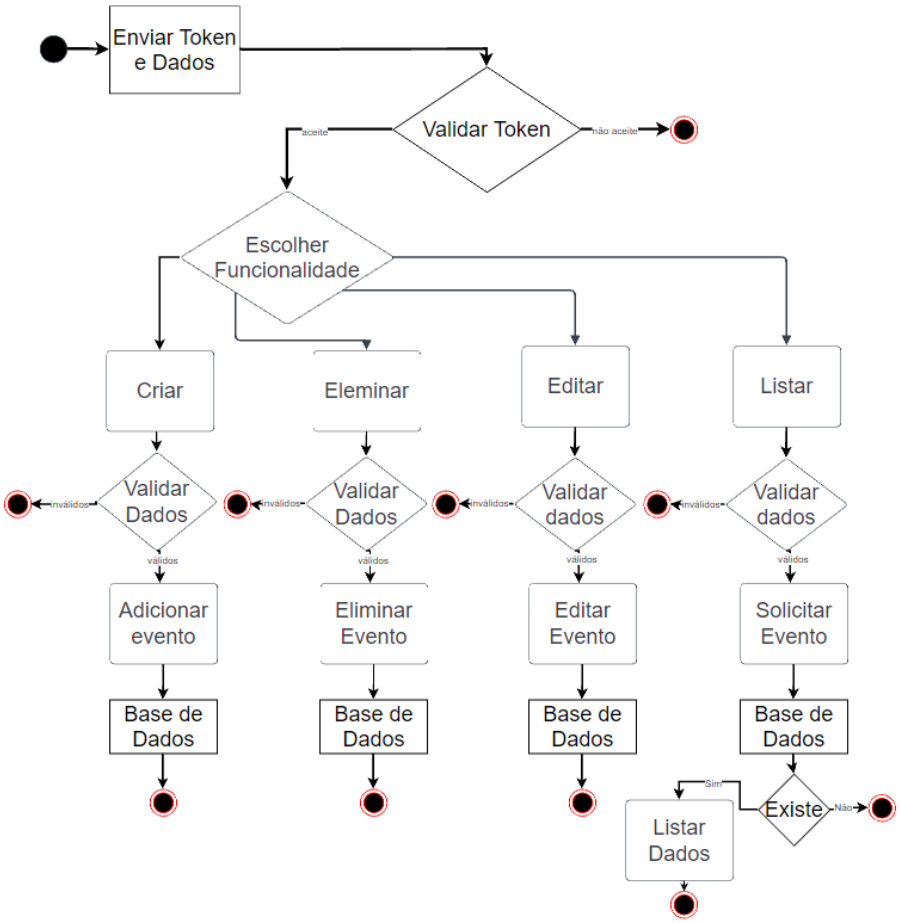

Figura 3.6: Diagrama de Atividade do Microsserviço de Gestão de Eventos.

O diagrama de atividade referente ao microsserviço de gestão de locais, figura 3.7, em semelhança com o microsserviço de gestão de eventos, valida o

token de autenticação concedendo o acesso às funcionalidades deste ou terminando o processo. Se este tiver acesso às funcionalidades (criar local, eliminar local e listar os locais solicitados), pode escolher uma e enviar os dados de solicitação em formato JSON. Permitindo efetuar as operações referentes a estas, na base de dados de locais, caso estes sejam válidos. Na funcionalidade de listar evento este retornará os dados encontrados na base de dados, se estes existirem.

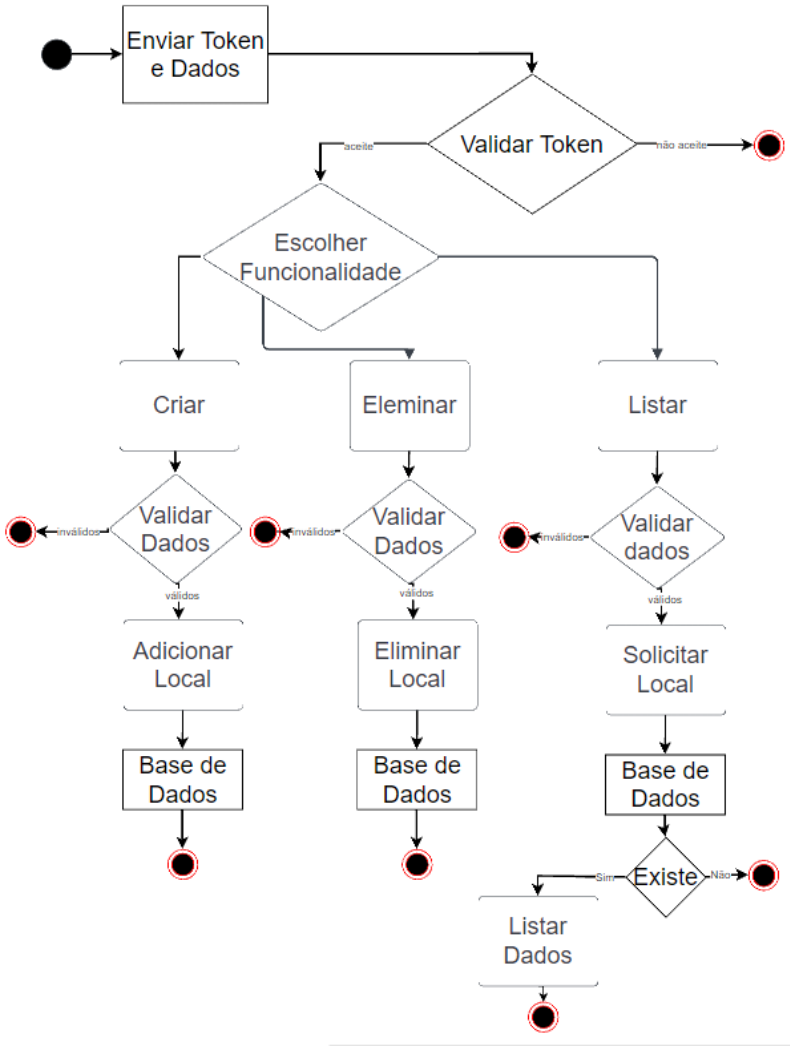

Figura 3.7: Diagrama de Atividade do Microsserviço de Gestão de Locais.

3.3.2 Modelação da Base de Dados

Os modelos de base de dados visam a esquematizar os dados necessários para um conjunto de serviços, estabelecendo também as relações existentes entre estes dados. Nesta secção são apresentados dois tipos de modelos: modelo conceptual e o modelo físico.

3.3.2.1 Modelo Conceptual

Na Figura que se segue (3.8), é ilustrado o modelo conceptual das bases de dados existentes para os microserviços apresentados. É possível observar que existem três entidades Utilizadores, Locais e Eventos Sendo que um utilizador pode ter vários eventos e vários locais, enquanto estes só podem ser atribuídos a um utilizador. E os eventos podem ter um local atribuído e os locais podem ser atribuídos a vários eventos.

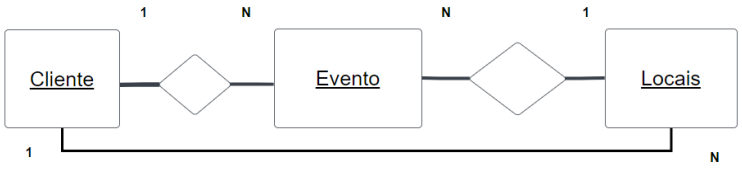

Figura 3.8: Modelo Conceptual.

3.3.2.2 Modelo Físico

No diagrama relacional da Figura 3.9, pode se verificar que a estrutura da base de dados é composta por três tabelas. A tabela de utilizador que contem a informação referente ao utilizador registado, realçando que este têm uma palavra-chave, usada para a sua autenticação, que por motivos de segurança é devidamente criptografada usando **SHA-256**, com um **Salt** gerado, sendo apenas guardado o seu valor de **Hash** e o **Salt** gerados na base de dados. Toda a informação sobre os eventos será armazenada na base de dados dos eventos, sendo referenciado, usando uma chave estrangeira, um utilizador e um local. Por fim temos a tabela dos locais, armazenada na base de dados dos locais, esta tem toda a informação dos locais, sendo referenciado por uma chave estrangeira um utilizador, podendo este estar referenciado a vários locais, e um dado local poder ser referenciado a vários utilizadores.

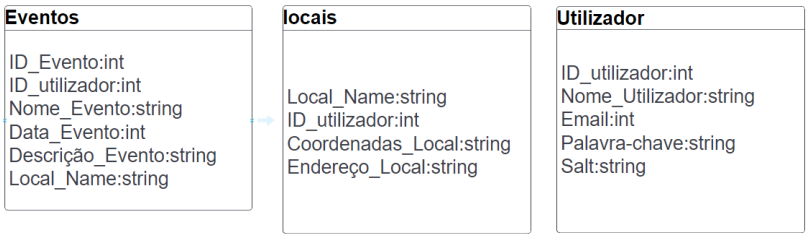

Figura 3.9: Modelo Físico.

3.3.3 Arquitetura Geral

A arquitetura geral dos microsserviços (Figura 3.10), faz uso de API's e funções lambda para a interação de vários clientes com as bases de dados referenciadas anteriormente. Esta interação é realizada, quando os diversos utilizadores recorrem a solicitações para receber dados ou executar certas funcionalidades enunciadas, sendo estas executadas pela API em conjunto com as funções lambda que por sua vez fazem as devidas alterações na base de dados ou o devido retorno dos dados observados nesta.

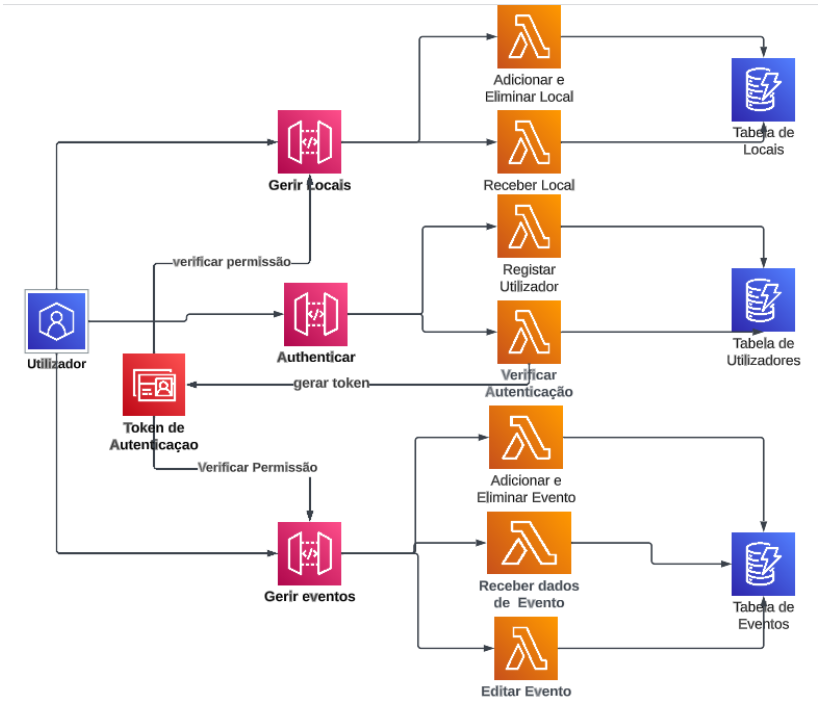

Figura 3.10: Arquitetura Geral.

3.4 Conclusões

Em suma, este capítulo enuncia os resultados do levantamento de requisitos feito para o projeto antes de se proceder à sua implementação. É de realçar que uma das etapas-chave no desenvolvimento de *Software* é levantar requisitos e deve ser realizado sempre antes da implementação de qualquer *Software*. Durante o levantamento destes são determinadas quais as funcionalidades que deverão ser implementadas e as relações entre estas.

Capítulo

4

Implementação

4.1 Introdução

Este capitulo tem como intuito explicar todas as implementações feitas neste projeto. Será demonstrado, um exemplo, como proceder à criação dos serviços geridos pela AWS, usados no projeto, e pormenorizar alguns excertos de código das funções lambda do mesmo.

4.2 Amazon API Gateway

Nesta secção, iremos demonstrar um exemplo de como criar uma API Gateway, tal como as utilizadas neste projeto. Para a criação da API devemos estar autentificados nos serviços da AWS, e procurar na barra de pesquisa destes por "API Gateway"e entrar na página inicial deste serviço. A seguir, basta clicar no botão "criar API"para prosseguir para as configurações desta.(Figura 4.1)

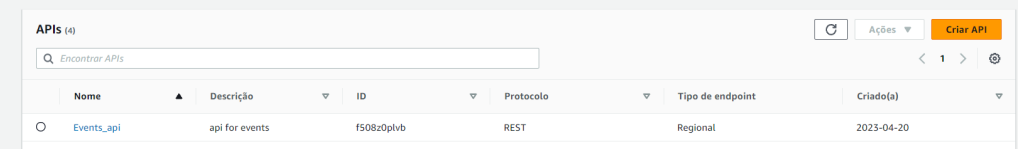

Figura 4.1: Imagem Referente ao Inicio de Criação de uma API nos Serviços AWS.

Este pode escolher entre as varias opções de API fornecidas por este ser-

viço, tais como, API HTTP, API WebSocket, API Rest e API Rest privada. Neste projeto foi criada uma API Rest.(Figura 4.2)

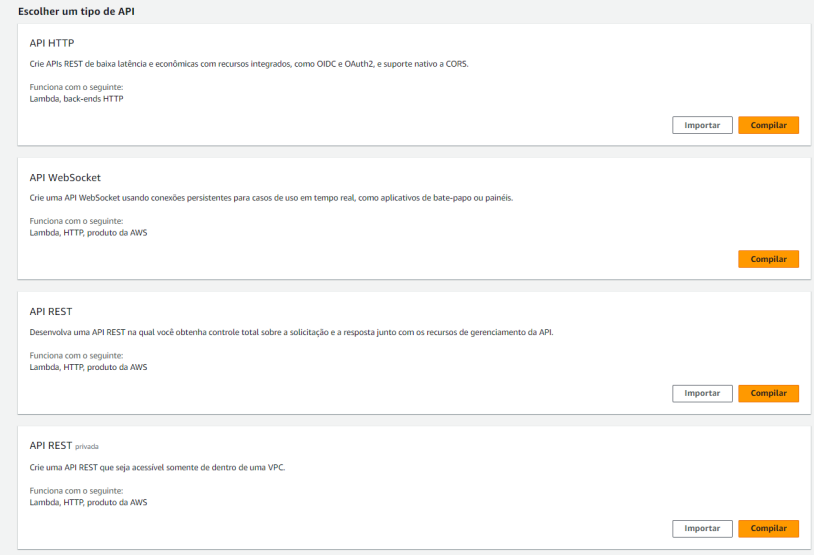

Figura 4.2: Tipo de API a ser Criada.

Na figura 4.3 é apresentada as opções utilizadas na criação desta, o utilizador tem de inserir um nome para API, e caso este queira pode adicionar uma descrição para esta. Nota: Este serviço não permite só criar as seguintes API's mas também importar, usando a opção "Import from Swagger or Open API 3 ".

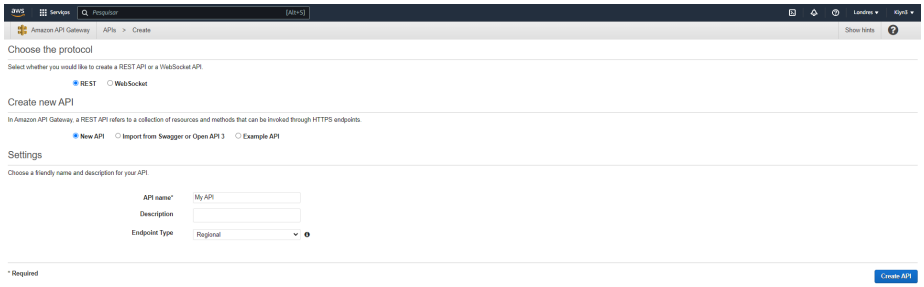

Figura 4.3: Opções Usadas na Criação da API e Opção para Importar uma API já Criada.

Com a API criada, o utilizador poderá configurá-la consoante o objetivo do seu trabalho criando métodos, recursos ou dar *deploy* nesta. Nas figuras seguintes são dados exemplos de configuração destas funções (figuras 4.4, 4.5 e 4.6)

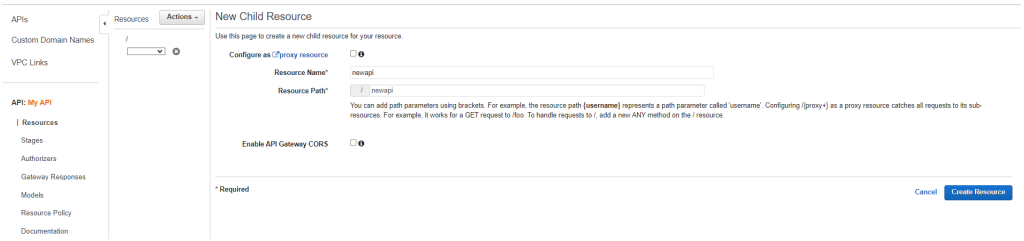

Figura 4.4: Criar Recurso numa API.

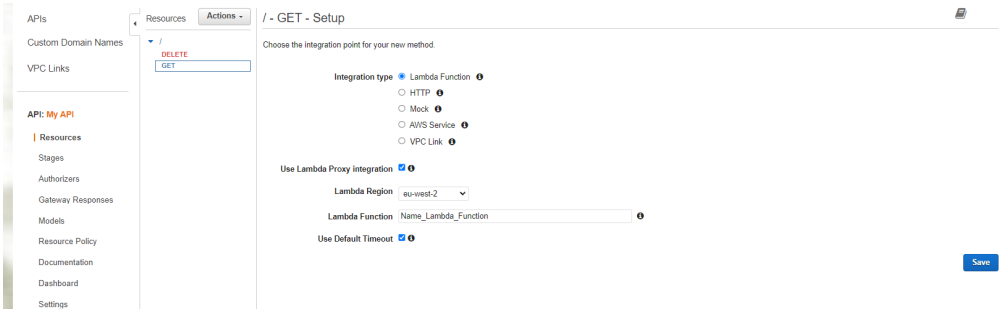

Figura 4.5: Configurar Recursos de uma API.

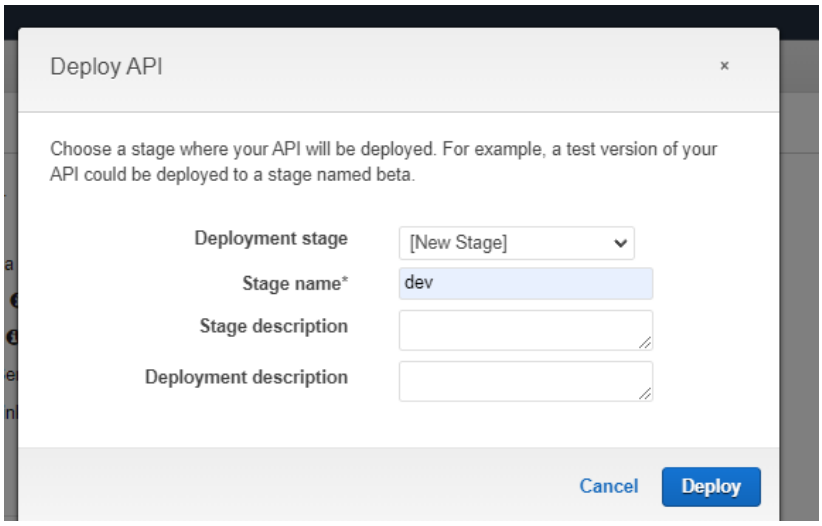

Figura 4.6: Dar *Deploy* na API.

Este serviço, permite configurações de segurança como um autorizador para as solicitações feitas a esta, no projeto implementado foi criado um autorizador que recebe um *token*. Com esse objetivo foi criado e configurado um autorizador, por meio de uma função lambda criada anteriormente, como demonstra as figuras 4.7 e 4.8.

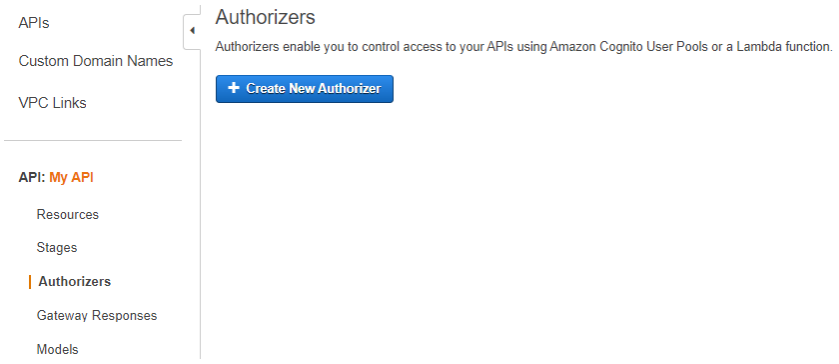

Figura 4.7: Criar Autorizador.

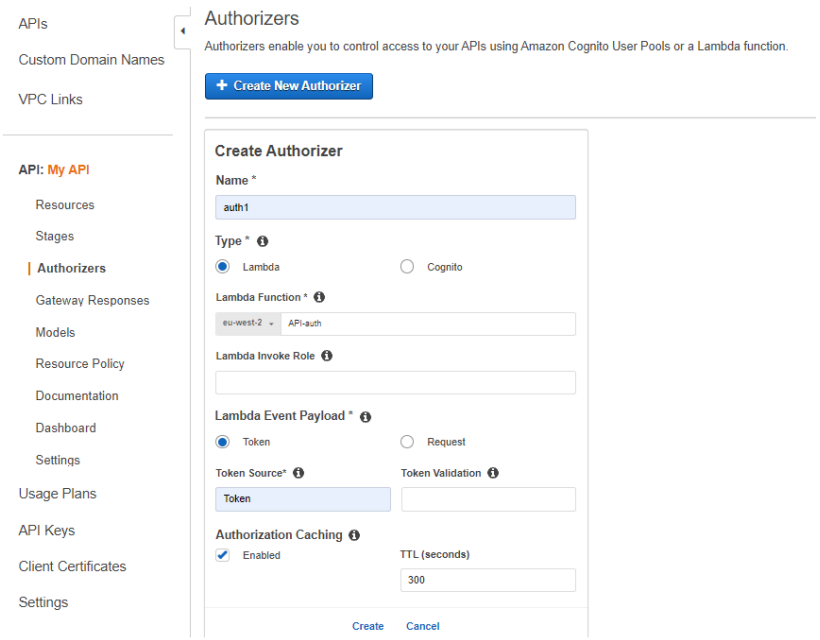

Figura 4.8: Configurar Autorizador da API.

Por fim, pode ativar este serviço nos métodos pretendidos, mudando a opção de "Authorization"da figura 4.9 para o autorizador criado.

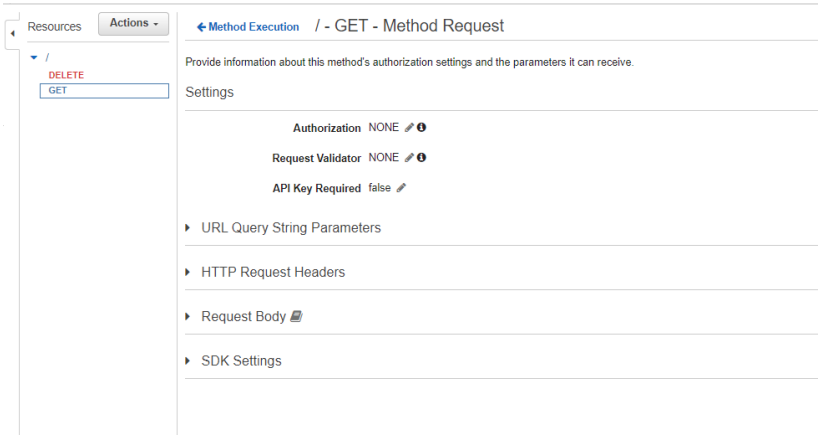

Figura 4.9: Opção para Utilizar um Autorizador num Método.

Concluindo, nesta secção foi demonstrado como criar uma API e algumas configurações feitas neste projeto.

4.3 Amazon DynamoDB

Na seguinte secção, será evidenciado o processo de criação das bases de dados. Para a sua criação terá de entrar na página inicial deste serviço e clicar na opção "criar tabela"(Figura 4.10). Em seguida, é possível configurar a nossa base de dados ou tabela, segue-se nas figuras seguintes uma demonstração de exemplo da configuração desta e algumas opções que o utilizador poderá tomar.

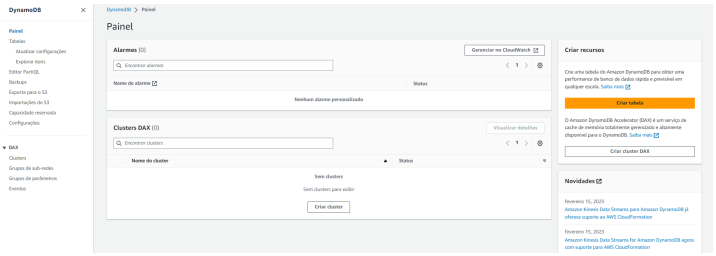

Figura 4.10: Imagem Referente ao início de Criação de uma Base de Dados Dinâmica nos Serviços AWS.

No início da criação desta o utilizador tem de dar um nome a esta e indicar uma chave de partição, podendo ainda adicionar uma chave de classificação caso este o pretenda (figura 4.11).

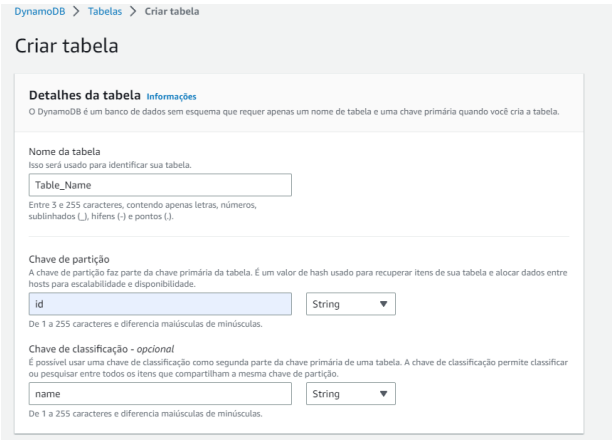

Figura 4.11: Imagem Mostra Como Dar Nome à Tabela e as Chaves.

O utilizador pode escolher entre uma configuração *standard*(Figuras 4.12) ou configurar com as opções desejadas por ele.

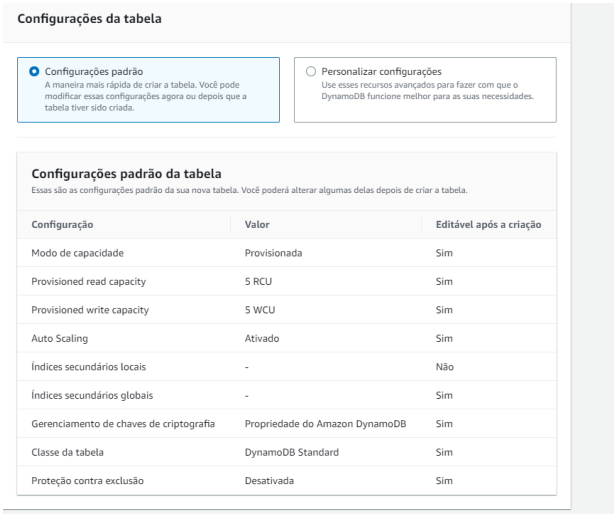

Figura 4.12: Imagem Referente à Configuração *Standard* de uma Base de Dados Dinâmica.

Segue-se também um exemplo de uma configuração personalizada e utilizada neste trabalho. Observando as figuras 4.13, 4.14 e 4.15), vemos que foi utilizada a opção "DynamoDB Standard", pois esta é recomendada para tabelas acedidas com grande frequência como no gestor de eventos. Escolhemos a opção de capacidade provisionada, permitindo que a base de dados efetue um *auto scaling* das capacidades de leitura e escrita, num valor estipulado. E por fim, demos a gestão da chave criptográfica da base de dados como propriedade da **Amazon DynamoDB**, encarregando-lhe a gestão dela.

Figura 4.13: Imagem Referente à Classe da Tabela.

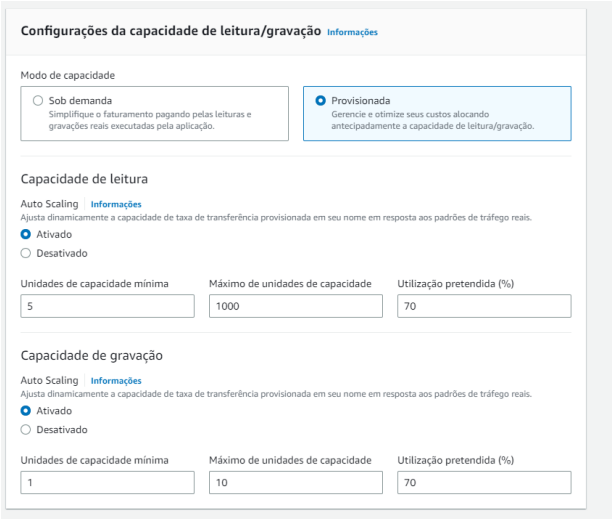

Figura 4.14: Imagem Referente à Capacidade de Leitura e Escrita nas Tabelas.

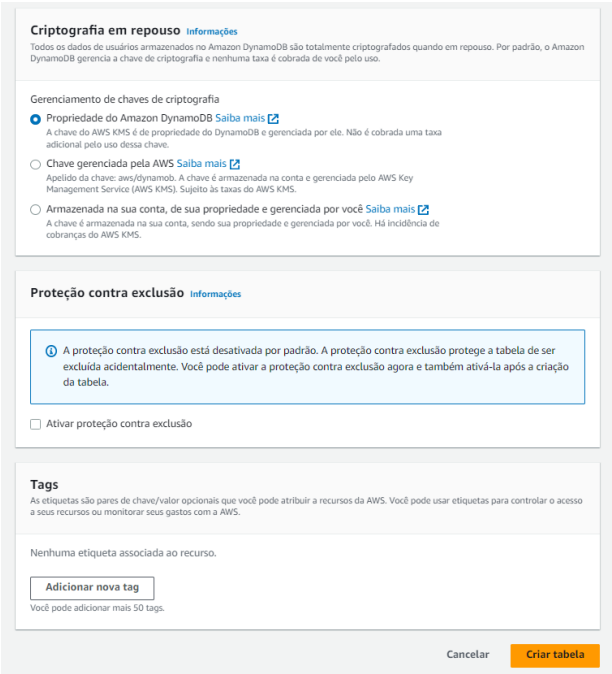

Figura 4.15: Imagem Referente à Proteção dos Dados em Repouso na Base de Dados.

Em suma, nesta secção foram descritos os passos para a criação das bases de dados usadas no projeto referente.

4.4 AWS Lambda

Nesta secção, demonstraremos como criar uma função lambda nos serviços da AWS e mostrar o ambiente onde esta nos permite programar as funções lambda e testar estas funções, partes essenciais na criação dos microsserviços. Para a criação destas funções, na página principal das funções lambda da AWS onde se encontram todas as funções lambda criadas pelo utilizador e a opção "criar função"para proceder à criação desta, como se pode observar na seguinte Figura 4.16.

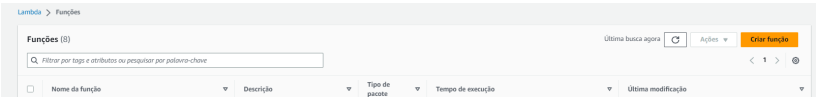

Figura 4.16: Imagem Referente ao início da Criação de uma Função Lambda nos Serviços AWS.

Na página de configuração é possível escolher opções, como a linguagem pretendida na sua programação, no caso deste projeto foi utilizado a linguagem Python, a arquitetura destas e outras opções avançadas conforme evidente nas Figuras 4.17 e 4.18, este é o exemplo de configuração utilizado no projeto. Também é possível importar uma função já criada selecionando a opção de "Imagem de container"na figura 4.17.

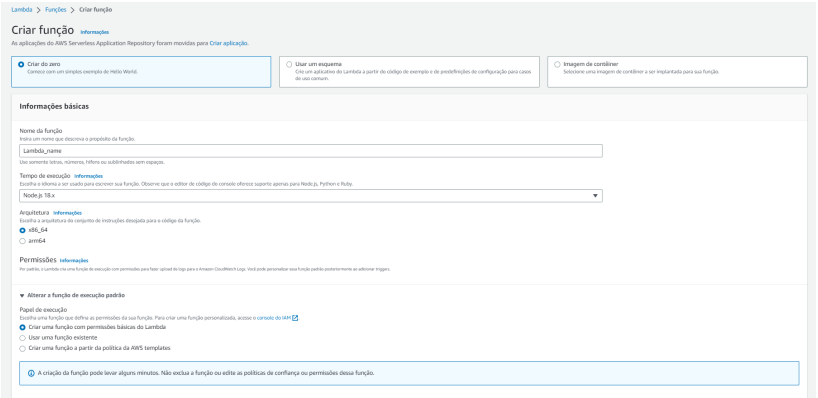

Figura 4.17: Imagem Referente à Configuração Utilizada nas Funções Lambda e Opção para Importar Função.

Figura 4.18: Imagem Referente as Opções Avançadas da Configuração das Funções Lambda.

Nas figuras seguintes, podemos observar a página principal da nossa função lambda (Figura 4.19) e o ambiente de desenvolvimento onde poderemos criar e desenvolver as nossas funções ou até mesmo importar código de funções já criadas. Esta *interface* ainda nos dá a possibilidade de realizar testes criando funções teste como podemos observar na Figura 4.20.

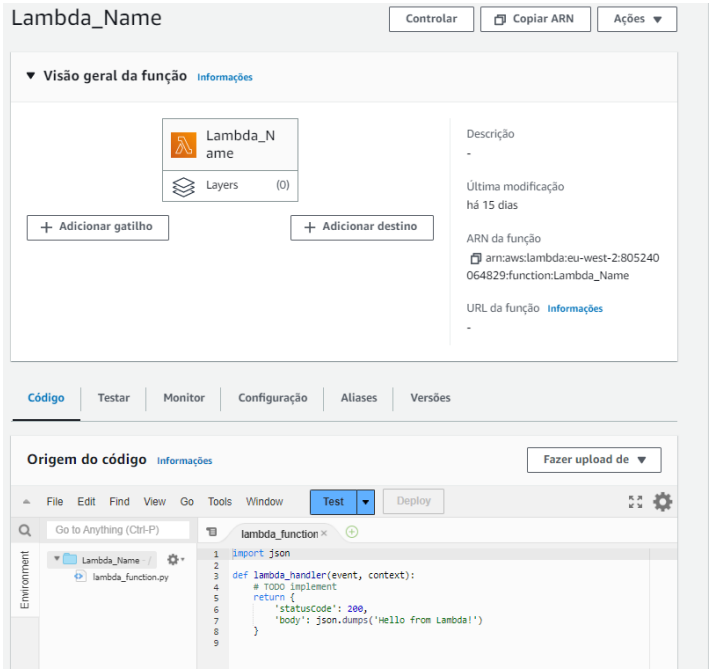

Figura 4.19: Pagina Principal da Função Lambda Criada.

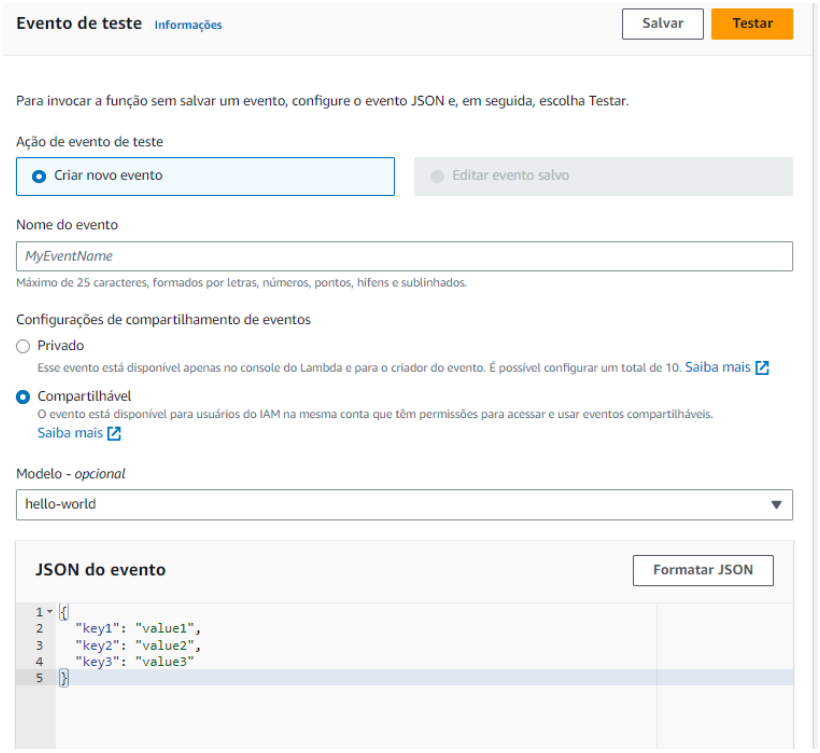

Figura 4.20: Imagem Referente à *Interface* de Teste da Função Lambda.

Em suma, esta *interface* oferece muitas outras funcionalidades, incluindo o suporte ao desenvolvimento e teste de funções lambda, sendo as descritas parte básica e essencial na implementação do projeto descrito.

4.5 Mecanismos de Segurança e de Interoperabilidade

Nesta secção são referenciados alguns elementos essenciais na conexão entre os serviços **AWS Lambda** e o serviço de base de dados **Amazon DynamoDB**, e elementos de segurança como a encriptação das palavras-chave, a criação de *token* de autenticação e a resposta do autorizador das API's referentes ao projeto.

No acesso à base de dados foi utilizado o boto3 uma biblioteca para Python que nos permite fazer algumas operações sobre a bases de dados dinâmicas. Para executarmos operações nas bases de dados primeiro temos de importar a biblioteca no código e depois identificar a tabela a aceder.(excerto de código

5)

```
import boto3

dynamodb_client = boto3.resource('dynamodb')
table = dynamodb_client.Table('local_events')
```

Excerto de Código 4.1: Código Referente à Importação da Biblioteca Boto3 e Identificação da Bases de Dados a Utilizar.

A seguir usando, como exemplo, funções descritas na documentação da biblioteca boto3 (para mais informações [10]), temos as funções para criar um elemento na tabela, eliminar um elemento da tabela ou receber um elemento da tabela (funções "putitem", "deleteitem"e "getitem"no excerto de código 21).

```
table.putitem(Item={
                'IdUser': id_user,
                'local_name': local_name,
                'local_cord': local_cord,
                'local_end': local_end,
            })

table.deleteitem(
                Key={
                    'local_name': local_name,
                }
            )

table.getitem(
        Key={
             'IdUser': id_user,
             'local_name': local_name
            }
```

Excerto de Código 4.2: Código Referente à Criação, Eleminação e Receção de um Item da Base de Dados.

No excerto seguinte podemos evidenciar o processo para encriptar a palavras-chave do utilizador. Para esse fim, foram utilizadas as bibliotecas **secrets** e **hashlib** para python, para criar um *salt random* e encriptar as palavras-chave com o *salt* gerados respetivamente.(excerto de código 3)

```
salt = secrets.token_hex(16)
hashed_password = hashlib.sha256((password + salt).encode('utf-8')).
    hexdigest()
```

Excerto de Código 4.3: Código Referente à Encriptação das Palavras-Chave dos Utilizadores.

Por fim, no excerto de código 14 é evidenciado uma das respostas do autorizador utilizados no projeto, sendo os principais aspetos deste, o valor do "principalID"onde é referido o nome do utilizador ou o seu identificador, em caso de este não ter permissão o seu identificador é "unknown", a versão deste, em "version", a ação que este realiza em caso de sucesso ("Action"), neste caso invocará a API referente, o "Effecte", se este for válido o seu valor é "Allow"caso contrário "Deny"e ainda a referência aos recursos que este tem acesso, em "Resource". Para mais informações [11].

```
authResponse = {
            "principalId": id_usuario,
            "policyDocument": {
                "Version": "2023-06-18",
                "Statement": [
                    {
                        "Action": "execute-api:Invoke",
                        "Effect": 'Allow',
                        "Resource": ["arn:aws:execute-api:eu-west
                            -2:*/*/*/*"]
                    }
                ]
            }
        }
```

Excerto de Código 4.4: Código Referente a um Exemplo de Resposta do Autorizador.

4.6 Conclusões

Em suma, neste capitulo foi dado um exemplo de como foram implementados os vários serviços, gerenciados pela AWS, utilizados para o propósito principal do projeto. Por fim, ainda foram enfatizados alguns pormenores ou excertos de código implementados nas funções lambda, entre eles a encriptação das palavras-chave, acesso as bases de dados e operações feitas sobre estas.

Capítulo

5

Demonstração e Validação

5.1 Introdução

Este capítulo, demonstra os serviços desenvolvidos e valida estes através da plataforma **Postman** (esta é uma plataforma que permite projetar, construir, testar e usar API's).

5.2 Microsserviço Responsável pelo Registo e Autenticação de Utilizadores

As demonstrações que poderemos observar nesta secção são referentes há API de autenticação, usando esta podemos criar um utilizador usando um método "POST"e enviando os dados em formato JSON (excerto de código 6), como demonstrado na seguinte figura 5.1. Caso este já exista na base de dados este não será adicionado, como exemplo na figura5.2.

```
{
  "username": "klyn31",
  "email": "val.2@gmail.com",
  "password": "value1"
}
```

Excerto de Código 5.1: *Input* Usado na Demonstração da Criação de um Utilizador.

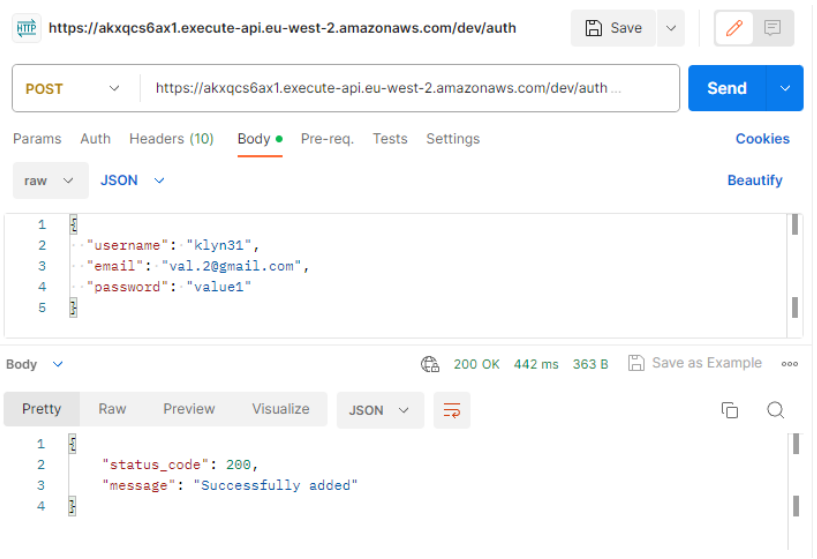

Figura 5.1: Demonstração da Criação de um Utilizador.

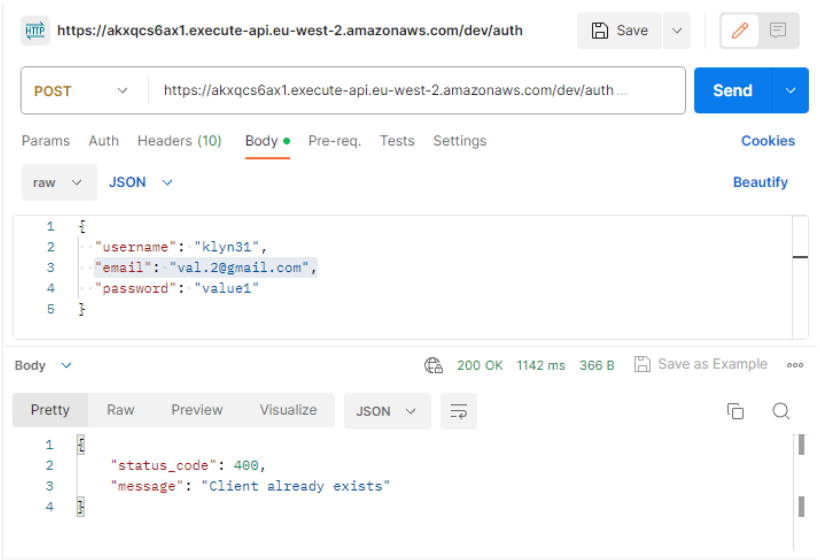

Figura 5.2: Demonstração da Criação de um Utilizador já Existente.

Numa situação de autenticação é possível utilizar a mesma API invocando o método "GET"caso os dados de autenticação sejam válidos este retorna um *token* de acesso caso este não seja o correto retorna uma mensagem de erro.(excerto de Código 5 e figuras 5.3 e 5.4)

```
{
  "username": "klyn3",
  "password": "value1"
}
```

Excerto de Código 5.2: *Input* Usado na Demonstração da Autenticação de um Utilizador.

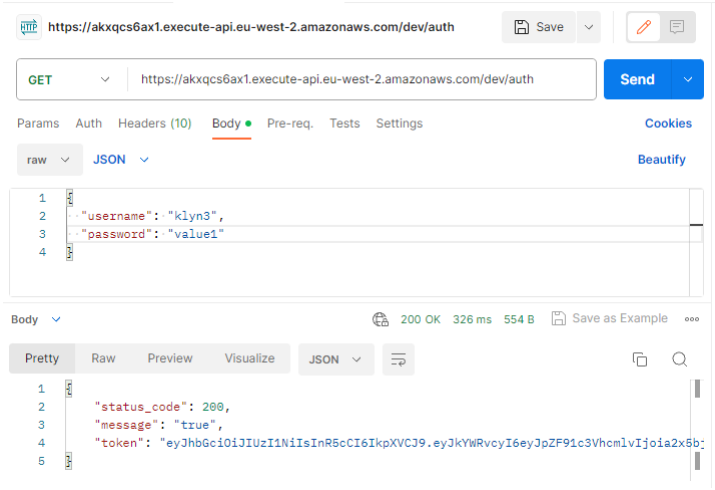

Figura 5.3: Demonstração da Autenticação de um Utilizador.

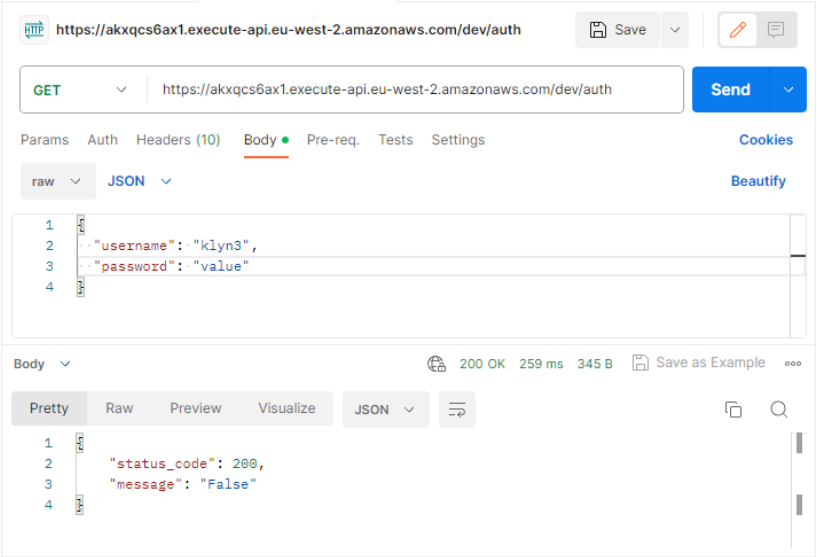

Figura 5.4: Demonstração da Autenticação de um Utilizador Invalido.

5.3 Microsserviço Responsável pela Gestão de Eventos

Nesta secção são demonstradas e validadas as funcionalidades referentes ao microsserviço de gestão de eventos que permite, criar eventos (excerto de código 10 e figura 5.5), eliminar eventos (excerto de código 6 e figura 5.6), editar eventos (excerto de código 6 e figura 5.7), receber todos os eventos de um dado utilizador (excerto de código 4 e figura 5.8) ou receber os dados de um certo evento (excerto de código 5 e figura 5.9).

```
{
  "httpMethod": "POST",
  "IdUser": 45856,
  "iDEvent": 1,
  "Event_Data": 202305282000,
  "Event_Description": "levar agua",
  "Event_Name": "caminhada"
  "Event_Local": "Fundao"
}
```

Excerto de Código 5.3: *Input* Usado na Demonstração da Criação de um Evento.

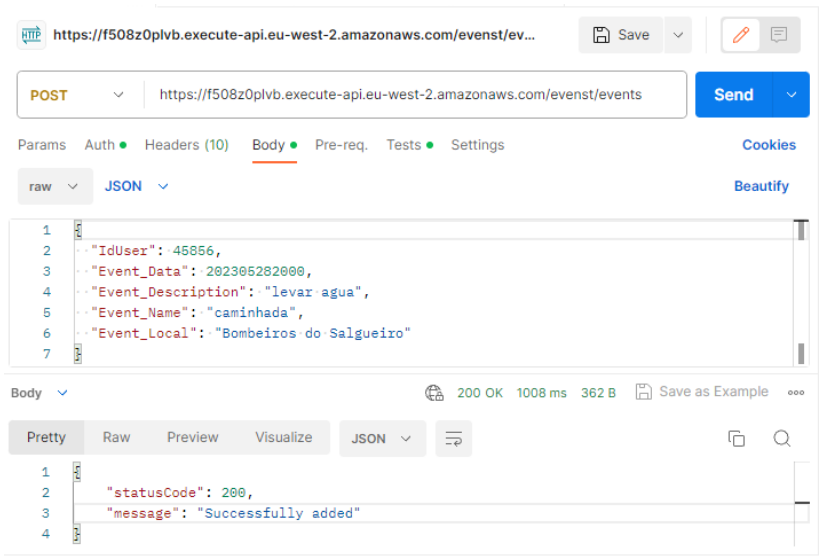

Figura 5.5: Demonstração da Criação de um Evento.

```
{
```

```
  "httpMethod": "DELETE",
  "IdUser": 45856,
  "iDEvent": 4
}
```

Excerto de Código 5.4: *Input* Usado na Demonstração da Eliminação de um Evento.

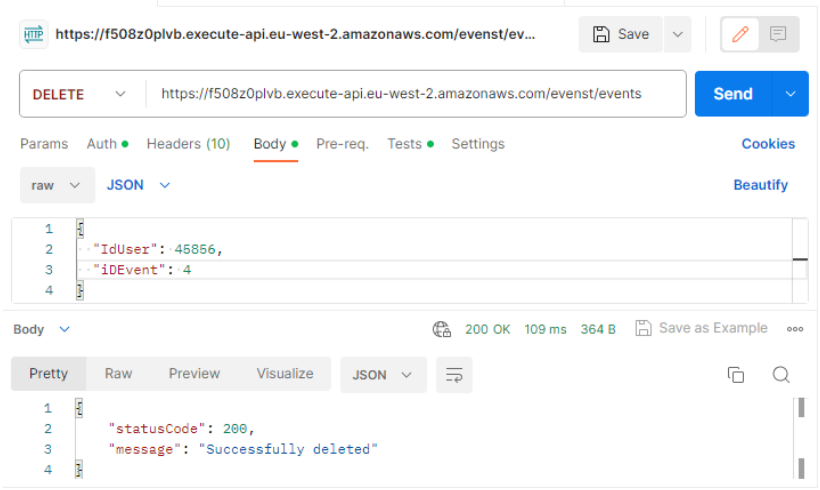

Figura 5.6: Demonstração da Eliminação de um Evento.

```
{
  "IdUser": 45856,
  "iDEvent": 3,
  "Event_Local": "Fundao"
}
```

Excerto de Código 5.5: *Input* Esado na Demonstração da Edição de um Evento.

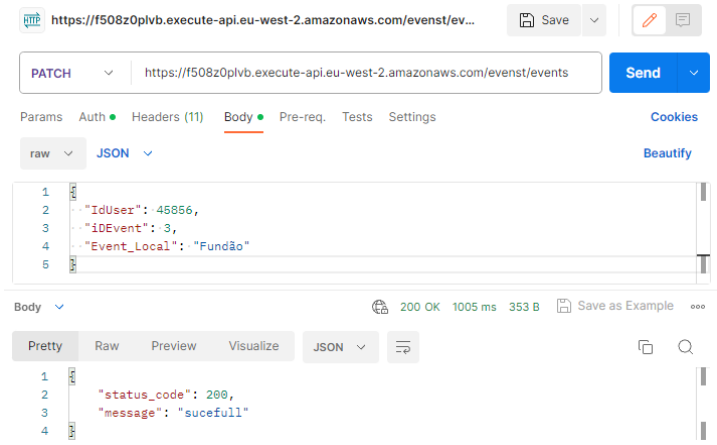

Figura 5.7: Demonstração da Edição de um Evento.

```
{
  "IdUser": 45856
}
```

Excerto de Código 5.6: *Input* Usado na Demonstração da Receção dos Dados dos Evento de um Utilizador.

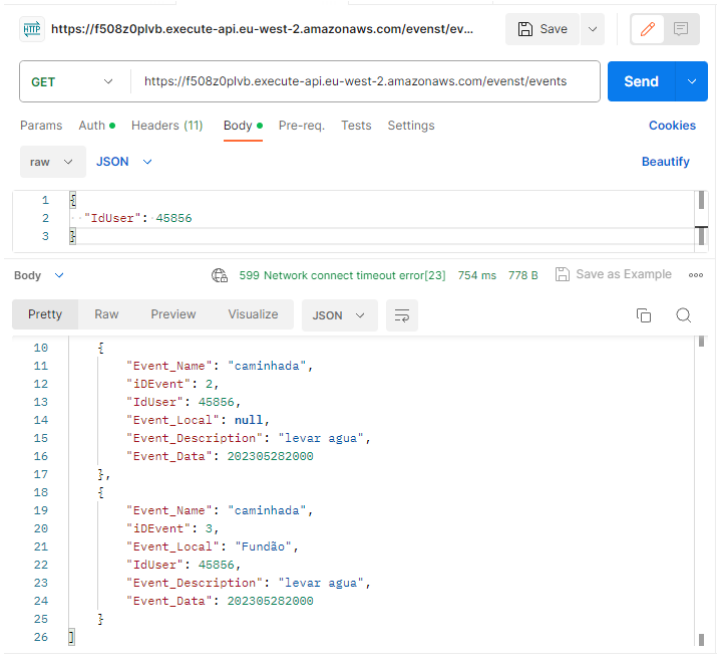

Figura 5.8: Demonstração da Receção dos Dados dos Evento de um Utilizador.

```
{
  "IdUser": 45856,
  "iDEvent": 4
}
```

Excerto de Código 5.7: *Input* Usado na Demonstração da Receção dos Dados de um Evento.

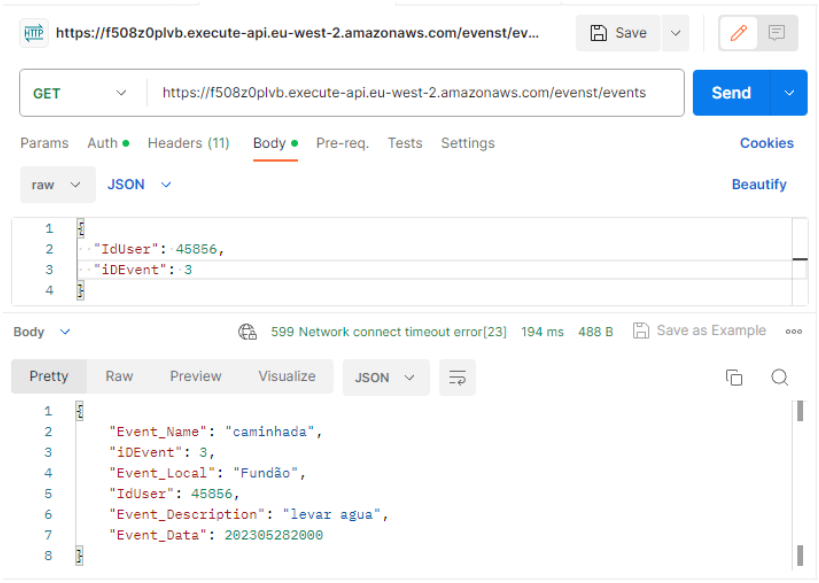

Figura 5.9: Demonstração da Receção dos Dados de um Evento.

5.4 Microsserviço Responsável pela Gestão dos Locais

Na seguinte secção, é demonstrado e testado o microsserviço de gestão de locais. As funcionalidades demonstradas são as seguintes: criar local (excerto de código 8 e Figura 5.10), eliminar local (excerto de código 6 e Figura 5.11), receber as informações dos vários locais guardados pelo utilizador (excerto de código 4 e Figura 5.12) e receber os dados de um determinado local (excerto de código 5 e Figura 5.13)

```
{
  "httpMethod": "POST",
  "IdUser": 1,
  "local_name": "lavandariax",
```

```
  "local_end": "Av. Eugenio de Andrade 19, 6230-296 Fundao",
  "local_cord": "40.14518710422179,-7.499340224276605"
}
```

Excerto de Código 5.8: *Input* Usado na Demonstração da Criação de um Local.

Figura 5.10: Demonstração da Criação de um Local.

```
{
  "httpMethod": "DELETE",
  "IdUser": 1,
  "local_name": "lavandariax"
}
```

Excerto de Código 5.9: *Input* Usado na Demonstração da Eliminação de um Local.

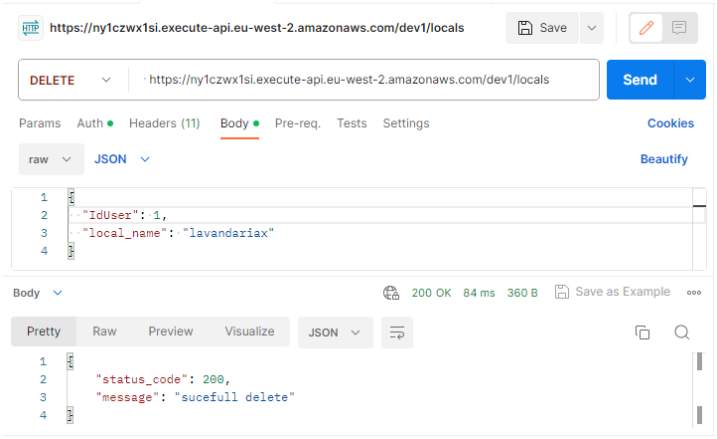

Figura 5.11: Eliminar Local

```
{
  "IdUser": 1
}
```

Excerto de Código 5.10: *Input* Usado na Demonstração da Receção dos Dados dos Locais de um Utilizador.

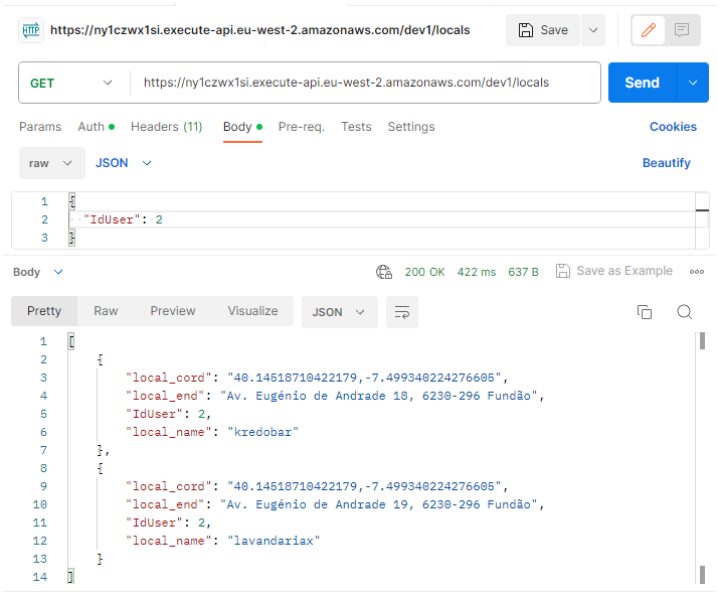

Figura 5.12: Demonstração da Receção dos Dados dos Locais de um Utilizadores.

```
{
  "IdUser": 1,
  "local_name": "lavandariax"
}
```

Excerto de Código 5.11: *Input* Usado na Demonstração da Receção dos Dados de um Local.

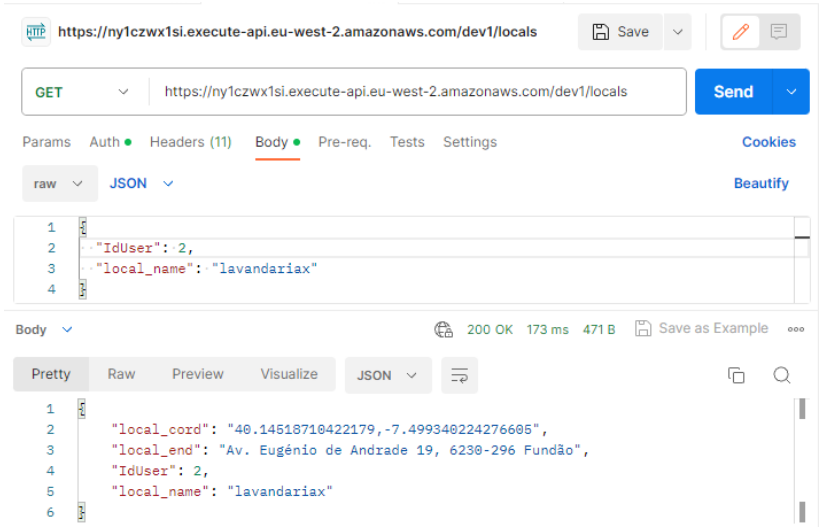

Figura 5.13: Demonstração da Receção dos Dados de um Local.

5.5 Conclusões

Concluindo, neste capitulo foram demonstrados e validados os microsserviços implementados neste projeto, usando *inputs* exemplo que podem ser utilizados pelos utilizadores para usufruir destes.

Capítulo

6

Conclusões e Trabalho Futuro

Neste capitulo final é apresentado as principais conclusões da realização deste projeto e os trabalhos futuros onde os microsserviços criados possam ser inseridos.

6.1 Conclusões Principais

Em suma, o relatório de projeto aborda a criação de um sistema baseado em microsserviços com uma arquitetura *serverless*, este refere os principais conceitos, benefícios e desafios desta abordagem. Ao longo do relatório, foi possível compreender a diferença entre arquiteturas monolíticas e não monolíticas, o que é uma arquitetura sem servidor baseada em microsserviços, seus benefícios e desvantagens. Foram introduzidas tecnologias, usadas na indústria, para a criação de microsserviços *serverless*, destacando os serviços da AWS utilizados na implementação do sistema de gestão de eventos, **AWS Lambda**, **Amazon DynamoDB** e **Amazon API Gateway**. Foi referido todo o processo de criação deste sistema, desde a engenheira de *software*, passando pela criação dos serviços na plataforma AWS e no seu desenvolvimento e terminando na demonstração e validação destes. Por fim, a criação do sistema seguinte, permitiu conhecer uma abordagem na criação de microsserviços, como utilizar as várias ferramentas oferecidas pela *AWS*, atualmente utilizadas na indústria, e adquirir conhecimento referente a microsserviços com arquitetura sem servidor.

6.2 Trabalho Futuro

Como trabalho futuro pode criar-se uma *front-end* que interligue estes três microsserviços, permitindo ao cliente uma maior facilidade no uso destes, possibilitando a criação de uma aplicação *web*, android, ou iOS para a gestão de eventos, onde os microserviços criados seriam o seu *back-end*.

Bibliografia

[1] Atlassian. Microservices vs. monolith. [Online] https://www.atlassian.com/br/microservices/microservices-architecture/microservices-vs-monolith. Último acesso a 27 de Junho de 2020.

[2] John Smith. Adapting serverless architecture. *DZone*, July 2022. [Online] https://dzone.com/articles/adapting-serverless-architecture. Último acesso a 28 de Junho de 2020.

[3] Tushar Waswani. Serverless architecture patterns in aws. *Medium*, March 2022. [Online] https://waswani.medium.com/serverless-architecture-patterns-in-aws-edeab0e46a32. Último acesso a 28 de Junho de 2020.

[4] Google Cloud. Google cloud platform documentation. [Online] https://cloud.google.com/docs?hl=pt-br. Último acesso a 20 de Junho de 2020.

[5] Microsoft. Microsoft azure documentation. [Online] https://learn.microsoft.com/en-us/azure/?product=popular. Último acesso a 20 de Junho de 2020.

[6] Amazon Web Services. Aws documentation, . [Online] https://docs.aws.amazon.com/index.html. Último acesso a 20 de Junho de 2020.

[7] Sajee Mathew and J Varia. Overview of amazon web services. *Amazon Whitepapers*, 105:1–22, 2014. [Online] https://docs.aws.amazon.com/pdfs/whitepapers/latest/aws-overview/aws-overview.pdf. Último acesso a 28 de Junho de 2020.

[8] Micro Focus. Micro focus visual cobol documentation. [Online] https://www.microfocus.com/documentation/visual-cobol/vc60/DevHub/GUID-F5BDACC7-6F0E-4EBB-9F62-E0046D8CCF1B.html. Último acesso a 20 de Junho de 2020.

[9] Amazon Web Services. Serverless architectures - learn more, . [Online] https://aws.amazon.com/pt/lambda/serverless-architectures-learn-more/. Último acesso a 20 de Junho de 2020.

[10] Amazon Web Services. Boto3 documentation: Amazon dynamodb, . [Online] https://boto3.amazonaws.com/v1/documentation/api/latest/guide/dynamodb.html. Último acesso a 26 de Junho de 2020.

[11] Amazon Web Services. Aws api gateway documentation: Use lambda authorizers, . [Online] https://docs.aws.amazon.com/apigateway/latest/developerguide/apigateway-use-lambda-authorizer.html. Último acesso a 20 de Junho de 2020.