UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

EDUARDO GASPARIN

MANIPULAÇÃO E VISUALIZAÇÃO DE DADOS GEOGRÁFICOS BASEADO EM UMA ARQUITETURA SERVERLESS

SANTA HELENA

2023

EDUARDO GASPARIN

MANIPULAÇÃO E VISUALIZAÇÃO DE DADOS GEOGRÁFICOS BASEADO EM UMA ARQUITETURA SERVERLESS

Manipulation and Visualization of Geographic Data Based on a Serverless Architecture

Trabalho de Conclusão de Curso apresentado como requisito para obtenção do título de Bacharel em Ciência da Computação da Universidade Tecnológica Federal do Paraná (UTFPR).

Orientador: Prof. Doutor Thiago França Naves
Coorientador: Prof. Doutor Anderson Sandro Rocha

SANTA HELENA

2023

EDUARDO GASPARIN

MANIPULAÇÃO E VISUALIZAÇÃO DE DADOS GEOGRÁFICOS BASEADO EM UMA ARQUITETURA SERVERLESS

Trabalho de Conclusão de Curso apresentado como requisito para obtenção do título de Bacharel em Ciência da Computação da Universidade Tecnológica Federal do Paraná (UTFPR).

Data de aprovação: 27 de Novembro de 2023

Thiago França Naves
Doutorado em Ciência da Computação
Universidade Tecnológica Federal do Paraná

Giuvane Conti
Doutorado em Engenharia Agrícola
Universidade Tecnológica Federal do Paraná

Franck Carlos Vélez Benito
Doutorado em Informática
Universidade Tecnológica Federal do Paraná

SANTA HELENA

2023

AGRADECIMENTOS

Agradeço àqueles que contribuiram para o desenvolvimento deste trabalho com valiosas sugestões, incluindo colegas e professores, com especial destaque ao orientador Thiago França Naves e o coorientador Anderson Sandro da Rocha

Agradeço também à minha família, cujo apoio inabalável e compreensão foram fundamentais ao longo desta jornada. À minha dedicada esposa, expresso meu profundo agradecimento. Seu apoio incansável, compreensão e paciência foram a âncora que sustentou minha jornada acadêmica. Nos desafios e nas conquistas, sua presença significou tudo para mim.

A conclusão deste trabalho representa não apenas o encerramento de um ciclo acadêmico, mas também a celebração de uma jornada repleta de desafios e conquistas. Compartilho essas realizações com profundo agradecimento àqueles que de uma forma ou de outra, tornaram este feito possível.

RESUMO

A interpretação de dados geográficos enfrenta desafios notáveis devido à heterogeneidade de fontes e formatos. A disparidade nas estruturas e origens desses dados pode complicar a análise e compreensão das informações contidas nessas bases de dados, prejudicando a avaliação de cenários e os processos de tomada de decisão. Uma abordagem eficaz para atenuar esses desafios consiste na adoção de representações visuais, com ênfase na utilização de mapas interativos. Por fundamentar-se em elementos visuais intuitivos, essa abordagem torna as informações acessíveis e facilita a identificação de padrões espaciais e relações entre diferentes variáveis, contribuindo para uma interpretação mais intuitiva e eficaz. O presente trabalho delineia as especificações e etapas de implementação de um portal de dados geográficos no ambiente da Amazon Web Services (AWS), fazendo uso da arquitetura *serverless* para a construção de uma API voltada ao processamento das informações geográficas, e exposição destas por meio de uma camada de visualização e interação com o usuário. Adicionalmente, o presente trabalho explicita as vantagens relacionadas à computação *serverless* no processamento de requisições de dados em uma plataforma web, com os resultados experimentais evidenciando a eficácia da implementação proposta neste trabalho, consolidando-a como uma estratégia apropriada e vantajosa na construção deste modelo de aplicação em ambiente web.

Palavras-chave: serverless; dados geográficos; visualização; API.

ABSTRACT

The interpretation of geographical data faces significant challenges due to the heterogeneity of sources and formats. Disparity in the structures and origins of this data can complicate the analysis and understanding of the information contained in these databases, hampering the assessment of scenarios and decision-making processes. An effective approach to mitigate these challenges involves the adoption of visual representations, with an emphasis on the use of interactive maps. By being grounded in intuitive visual elements, this approach makes information accessible and facilitates the identification of spatial patterns and relationships between different variables, contributing to a more intuitive and effective interpretation. The present work outlines the specifications and implementation steps of a geographical data portal in the Amazon Web Services (AWS) environment, utilizing serverless architecture for the construction of an API dedicated to processing geographical information, and exposing it through a layer of visualization and interaction with the user. Additionally, this work elucidates the advantages related to serverless computing in processing data requests on a web platform, with experimental results demonstrating the efficiency of the implementation proposed in this work, solidifying it as a suitable and advantageous strategy in constructing this application model in a web environment.

Keywords: serverless; geographic data; visualization; API.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Dados geográficos podem incluir informações sobre a latitude e longitude de um local, limites políticos e relevos, além de atributos como recursos naturais, clima, dados demográficos e econômicos. A obtenção de dados geográficos pode apresentar uma série de desafios, principalmente em relação à qualidade dos dados, a depender da fonte e do método de coleta (CHANG, 2008; MOTA, 2016).

O padrão *shapefile* (.shp) é amplamente utilizado em sistemas de informação geográfica, do inglês *Geographic Information System* (GIS) para armazenar dados geográficos. Ele permite representar pontos, linhas e polígonos, como rios, estradas e estados, e incluir atributos associados a cada objeto geográfico. Sua popularidade se deve à compatibilidade com diversos *softwares* GIS, facilitando o compartilhamento e a transferência entre diferentes plataformas. Por meio da tecnologia GIS é possível manipular os dados geográficos armazenados em arquivos shapefile, permitindo o processamento, análise e visualização. Contudo, este formato apresenta uma considerável complexidade de manipulação, com vários arquivos inter-relacionados e com limitações, associadas à necessidade de dados precisos e habilidades técnicas para gerenciá-lo, além de restrições de armazenamento devido ao tamanho máximo permitido para um arquivo (ZHU *et al.*, 2019; SILVA SANTOS; SANTOS JUNIOR, 2021).

Apesar da importância e utilidade no uso de dados geográficos no padrão GIS para tarefas de análise e tomada de decisão, existe uma carência por soluções que explorem o processamento e visualização destes dados em ambientes com maior usabilidade e fácil acesso aos usuários finais, como os sistemas web (LÜ *et al.*, 2019). Dificuldades relacionadas à não existência de bases de dados adequadas, seja pela especificação incorretas destas ou pela falta de dados atualizados e confiáveis, o alto custo de implantação de ferramentas, além da limitação destas em termos de funcionalidade ou acessibilidade, limitam a disponibilidade de soluções para manipulação e visualização de dados e por consequência, seu uso no processo de tomada de decisão (PEREIRA; SILVA, 2001; BREUNIG *et al.*, 2020).

Outro limitante é o uso intensivo de recursos computacionais necessário ao processamento de dados geográficos, sendo necessária infraestrutura de *hardware* e *software* adequada. A utilização de uma arquitetura *serverless*, simplifica a implementação de serviços web com base em *hardware* e software (FONTANA; ZANELATTO,

2019), servindo de alternativa às opções convencionais de configuração e gerência de um servidor, seja local ou na nuvem. Na arquitetura *serverless*, o provedor é o responsável pela configuração e alocação de recursos, realizando cobranças somente pelo que for de fato utilizado (PEREIRA; SILVA, 2001; SANTACATTARINA, 2021; SHAFIEI; KHONSARI; MOUSAVI, 2022).

Assim, este trabalho visa explorar a manipulação de dados geográficos utilizando uma arquitetura *serverless* da provedora AWS[1] (WITTIG; WITTIG, 2016), visando o tratamento mais ágil deste tipo de dado com suporte a maior e melhor exploração e visualizações via plataforma *web*.

1.1 Objetivo

Expõem-se a seguir os objetivos geral e específicos que se pretende atingir com o trabalho.

1.1.1 Geral

O objetivo deste estudo é investigar a manipulação de dados geográficos por meio da utilização de uma arquitetura *serverless* fornecida pela provedora AWS.

A adoção da abordagem *serverless* visa obter vantagens como escalabilidade automática, gerenciamento simplificado de recursos e maior agilidade e eficiência no processamento dos dados geográficos, permitindo a criação de visualizações mais abrangentes por meio de uma plataforma *web*. Dessa forma, busca-se possibilitar análises mais aprofundadas e uma melhor compreensão dos padrões e relações espaciais presentes nos referidos dados.

1.1.2 Específicos

1. Fazer uma revisão bibliográfica dos trabalhos na área de processamento de dados geográficos;
2. Analisar os dados geográficos que serão coletados, em relação ao seu processamento, seus formatos, suas fontes e modos de apresentação;

[1] https://aws.amazon.com/.

3. Desenvolver uma arquitetura *web* de processamento e visualização de dados geográficos com base em *serverless*;
4. Executar testes de desempenho da ferramenta, coletando informações relacionadas ao tempo de resposta das requisições pelo usuário;
5. Analisar os resultados dos testes de desempenho da ferramenta de processamento e de visualização dos dados

1.2 Contribuições do Trabalho

A principal contribuição deste estudo consiste na validação de uma arquitetura *serverless* para o processamento de dados geográficos, com o propósito de aprimorar a capacidade de manipulação e visualização desses dados, com o intuito de facilitar a tomada de decisões. Além disso, este trabalho apresenta outras contribuições relevantes, as quais são enumeradas a seguir:

1. Proposta e validação de uma metodologia de processamento de dados geográficos de fontes diversas;
2. Criação de um banco de dados geográfico, com as informações processadas previamente;
3. Combinação de diferentes tecnologias em uma arquitetura *serverless*;
4. Validação e testes com a arquitetura gerada no processamento de dados geográficos, avaliando sua escalabilidade e desempenho.

1.3 Justificativa

Em razão de fatores como a complexidade dos dados, a heterogeneidade de fontes, a falta de padronização e a necessidade de escalabilidade em lidar com vasto volume de dados, há um deficit em arquiteturas e ferramentas eficientes para o processamento de dados geográficos e a apresentação visual destes (LI *et al.*, 2016).

Autores como Mete e Yomralioglu (2021), propuseram uma arquitetura capaz de lidar com o processamento de dados geoespaciais com pouca intervenção humana e reduzido consumo de recursos a arquitetura tecnologias *serverless*. Já Bebortta *et al.* (2020), conseguiram a partir da arquitetura *serverless* melhorar o tempo e o

desempenho de processamento de dados geoespaciais para dados hiper-espectrais de alta dimensão.

As abordagens propostas nos trabalhos apresentados anteriormente buscam viabilizar a coleta e processamento de dados geográficos, de maneira mais eficiente e escalável. Contudo, nenhum deles buscou validar a integração de dados de fontes diversas junto ao formato GIS, através do gerenciamento destes utilizando uma arquitetura *serverless*. Com isso, este trabalho justifica-se ao buscar essa validação de modo a criar visualizações com base na necessidade da análise do utilizador, auxiliando com uma apresentação precisa para o processo de tomada de decisão.

1.4 Delimitações do trabalho

Devido a amplitude do tema de processamento de dados geográficos, assim como a integração destes com dados heterogêneos, este trabalho se limita aos seguintes pontos:

1. Não serão explicados os passos de instalações de *softwares* utilizados;
2. Não será explicado o processo de aquisição e acesso ao provedor da arquitetura *serverless*
3. Não haverá comparação da ferramenta proposta com SIG's comerciais
4. Serão utilizados dados geográficos de fontes públicas, bem como aqueles obtidos em trabalhos de pesquisadores

2 REVISÃO DA LITERATURA

A revisão da literatura foi construída para revisar conceitos fundamentais das áreas correlatas necessárias para a integração de técnicas relacionadas a construção de uma arquitetura *serverless* para processamento e visualização de dados geográficos complexos. Todos os conceitos comumente encontrados na literatura são revisados com detalhes e exemplos didáticos, e indo além disso, parte do capítulo possui sínteses destes conteúdos e conhecimentos elaborados pelos autores. O propósito é apresentar conhecimentos necessários para o o uso das tecnologias necessárias, bem como os específicos que serão utilizados na integração destas e processamento do ciclo de vida dos dados.

Portanto, este capítulo é parte fundamental no processo de construção do conhecimento gerado neste trabalho.

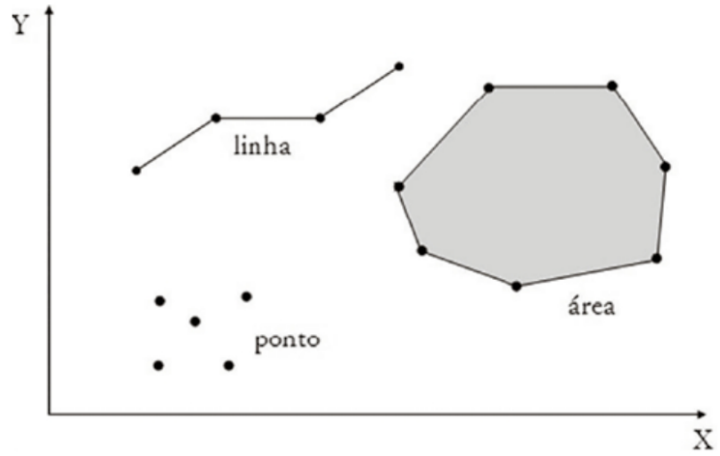
2.1 Dados Geográficos

De acordo com Mota (2016), dados geográficos são aqueles que possuem uma componente espacial associada, podendo incluir informações sobre a latitude e longitude de um local, limites políticos e relevos, além de atributos como recursos naturais, clima, dados demográficos e econômicos. A obtenção destes pode ser variada, como levantamentos de campo, sensoriamento remoto, registros administrativos e dados gerados por GIS, e são essenciais para a análise espacial, tomada de decisões e compreensão dos padrões e interações no mundo físico (TROMBETA *et al.*, 2019).

De acordo com Stein *et al.* (2021), os dados geográficos podem ser estruturados em forma vetorial e matricial, cada qual contendo características próprias, aptas à suprir as necessidades dos trabalhos de análise espacial. A estrutura vetorial é caracterizada pelo uso de geometrias, usada para a representação de objetos ou elementos da superfície terrestre, fazendo uso de apenas três primitivas, o ponto, a linha e o polígono ou área, Figura 1. A primeira das primitivas, o ponto é a estrutura menos complexa, sendo formado facilmente a partir de um par de coordenadas conhecidas, e geralmente é utilizado para representar a localização de um elemento. Já a segunda primitiva, a linha, requer ao menos dois pontos para sua representação, o que irá gerar um segmento de reta. Geralmente é utilizada para representar elementos cujo sua

espessura é irrelevante ou inexistente, como é o caso de vias, rios e delimitações políticas. O polígono, por sua vez, é formado por um conjunto de pontos e segmentos de reta que devem estar conectados, formando uma área fechada (STEIN *et al.*, 2021).

Figura 1 – Primitivas para representações vetoriais em duas dimensões.



Fonte: Adaptada de Câmara (2005).

De acordo com Câmara (2005) e Stein *et al.* (2021), é importante estar atento aos relacionamentos espaciais entre as primitivas gráficas, ou seja, da topologia dos dados vetoriais, que definirão as relações de adjacência, intersecção e cruzamento entre as primitivas gráficas. Os relacionamentos entre diferentes elementos geográficos podem ser resumidos em:

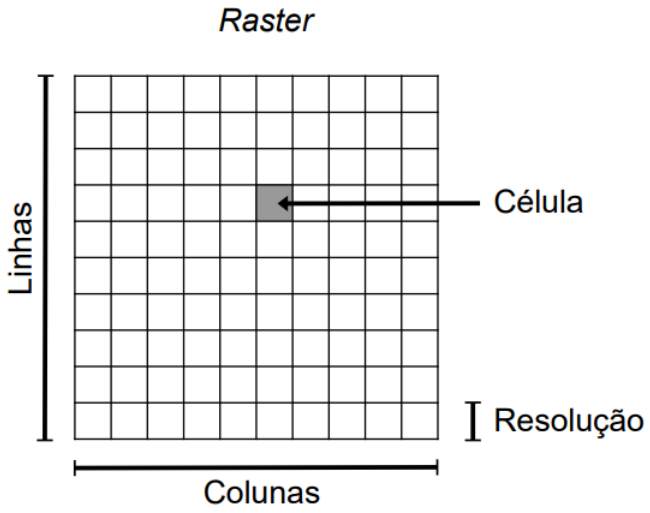
- Conectividade, atributo que define se os elementos estão interligados ou não;
- Contiguidade, que é relativa à identificação do contato entre os elementos;
- Proximidade, que está relacionada à distância entre dois elementos.

Outro fator determinante na representação dos dados geoespaciais na estrutura vetorial, é a escala, pois ela definirá a qualidade dos dados. A depender do objetivo da representação, um mesmo elemento pode ser representado por diferentes primitivas, com o qual a escala deverá ser compatível, como é o caso de um município, que embora abranja uma área especifica e determinada, devido a escala deste na representação de um mapa territorial nacional é normalmente representado por meio de um ponto.

Já a estrutura matricial, também conhecida como *raster*, é organizada por uma matriz em linhas e colunas, com cada célula formando um *pixel*, Figura 2. De acordo com Stein *et al.* (2021), nesta estrutura o tamanho do *pixel* é o fator determinante da escala, estando diretamente relacionado a sua resolução espacial, ou seja, ao tamanho da espaço real representado em cada *pixel*. Além disso, os dados alfanuméricos são

uma parte essencial dos dados espaciais e podem ser conectados a uma geometria que representa um objeto, sendo facilmente relacionados por meio de um banco de dados.

Figura 2 – Estrutura matricial (*raster*)



Fonte: Adaptada de Câmara (2005) e Stein *et al.* (2021).

Câmara (2005) cita ainda o conceito de espaço celular, que é uma estrutura matricial que associa diferentes atributos a cada célula. Essa abordagem oferece vantagens em relação às estruturas matriciais simples, pois uma única célula pode estar associada a várias informações, proporcionando uma manipulação de dados mais eficiente. Além disso, os espaços celulares são muito adequados para armazenamento em bancos de dados objeto-relacionais. Toda a estrutura de um espaço celular pode ser armazenada em uma única tabela, simplificando o manuseio dos dados em comparação com dados vetoriais ou matriciais indexados. Aplicações como álgebra de mapas e modelagem dinâmica também se tornam mais simples de implementar e operar.

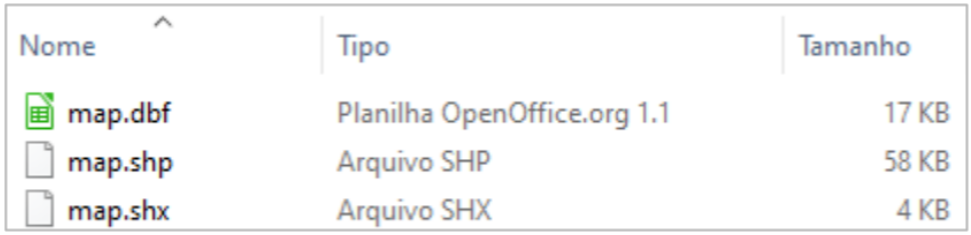
A utilização de uma estrutura ou de outra, depende das análises que se pretende realizar e dos resultados esperados.No caso de representações que demandem grande fidelidade às delimitações entre as áreas, devem priorizar o uso da estrutura vetorial, já nos casos que necessitam de processamento digital de imagens, dados ou sobreposição de camadas temáticas, deve-se priorizar o uso da estrutura matricial (STEIN *et al.*, 2021).

Todas estas informações demandam formatos especiais de arquivos para sua correta manipulação e armazenamento, neste sentido, Stein *et al.* (2021) citam os formatos criados propriamente para desenho assistido por computador, como o .dxf, o

utilizado pelo serviço do Google Earth, .kml, e o popular formato de dados vetoriais .shp (*shapefile*).

Zhu *et al.* (2019), Silva Santos e Santos Junior (2021) e Stein *et al.* (2021), destacam que o formato *shapefile* é composto por ao menos três diferentes arquivos, um .shp que armazena as características da geometria, um .dbf que armazena a tabela de atributos da geometria e um .shx que estabelece uma ligação entre os demais arquivos, Figura 3, devendo-se ter um especial cuidado com o correto inter-relacionamento entre estes arquivos, o tamanho destes e o número de atributos a serem armazenados. Também é importante destacar que os arquivos .shp e .shx, são arquivos binários não legíveis por seres humanos.

Figura 3 – Estrutura básica de um arquivo *shapefile*



Fonte: Autoria própria (2023).

Quando avaliado em relação a aplicações *desktop*, autores como Sampaio Faria (2007) e Santos *et al.* (2016), citam vantagens do formato *shapefile* em relação aos demais formatos, principalmente relacionadas a uma maior velocidade para as aplicações gerarem a representação visual dos dados contidos nestes, o menor espaço ocupado em disco, a facilidade de manipulação e edição, e sua popularidade, que se reflete em maior numero de aplicações compatíveis com seu uso.

No entanto, apesar da sua popularidade e características positivas, em aplicações *web* que exigem o constante tráfego de dados e a grande heterogeneidade entre ferramentas e dispositivos, outros formatos são mencionados na literatura. Peterson (2016), Celesti *et al.* (2017), Anderson *et al.* (2019) e Horbiński e Lorek (2022) destacam o uso do formato Geographic JavaScript Object Notation (GeoJSON).

O formato GeoJSON é uma variante geográfica do formato JavaScript Object Notation (JSON), legível por humanos, não comprimido e sem perda de informações. É atualmente adotado como padrão em projetos de mapas interativos na *web*, devido ao seu tamanho reduzido, facilidade de transformação e compatibilidade com bibliotecas como Leaflet, D3, Google Maps API, OpenLayers e Mapbox.

Em um arquivo GeoJSON, cada elemento é representado como uma tabela de dados contendo atributos espaciais, como seu tipo de elemento (ponto, linha ou polígono) sua localização (latitude e longitude), bem como atributos não espaciais associados, Figura 4. Horbiński e Lorek (2022) citam que o formato GeoJSON pode ser editado em um editor de texto básico, permitindo excluir funções e modificar propriedades sem a necessidade de software especializado.

Figura 4 – Estrutura básica de um arquivo GeoJSON.

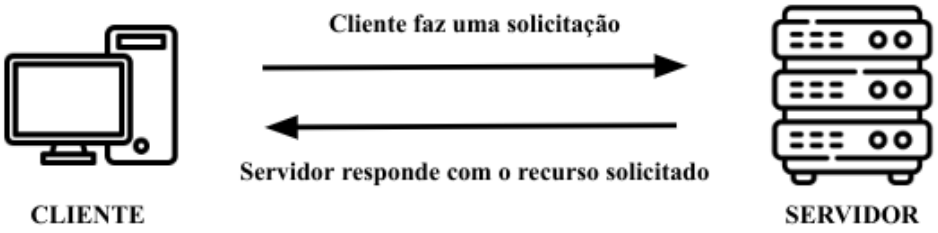
Fonte: Autoria Própria (2023).

2.2 Arquitetura Web

A arquitetura *web* é uma abordagem para o design e desenvolvimento de aplicativos e sistemas distribuídos. Ela envolve a organização e estruturação dos componentes de um sistema para garantir uma comunicação eficiente entre os diferentes elementos que o compõem.

De acordo com Alves (2015), no modelo cliente-servidor, os sistemas computacionais são divididos em duas entidades distintas: o cliente e o servidor. O cliente é geralmente um navegador da web ou um aplicativo que faz solicitações para acessar recursos na web, enquanto o servidor é o computador ou sistema responsável por processar as solicitações e fornecer as respostas correspondentes. A comunicação entre o cliente e o servidor é baseada no protocolo *Hypertext Transfer Protocol* (HTTP), por meio do qual, o cliente envia uma solicitação ao servidor, especificando o recurso desejado, geralmente por meio de um endereço no formato *Uniform Resource Locator* (URL). Ao receber a solicitação do cliente, o servidor processa esta e retorna uma resposta ao cliente contendo o recurso solicitado, que então exibe o conteúdo recebido em seu navegador ou aplicativo, Figura 5.

Figura 5 – Modelo Cliente-Servidor



Fonte: Adaptada de Alves (2015).

Para Alves (2015), a característica divisão de papéis e responsabilidades dos dispositivos no modelo cliente-servidor, proporciona uma distribuição eficiente de recursos e funções. Enquanto o servidor é planejado e projetado para prover serviços e recursos aos clientes, estes assumem a tarefa de requisitar e apresentar esses mesmos recursos. Essa segregação funcional favorece a escalabilidade, a gestão e a manutenção dos sistemas web, permitindo que os servidores sejam otimizados para lidar com múltiplas requisições simultâneas dos clientes.

Uma das vantagens do modelo cliente-servidor reside na sua capacidade de permitir o acesso a recursos e serviços em distintos locais geográficos. Os clientes têm a habilidade de requisitar recursos de servidores situados em qualquer região do globo, contanto que exista conectividade de rede, fator que tem desempenhado um importante papel no crescimento e na abrangência de serviços *web*.

2.2.1 Application Programming Interface

O conceito de *Application Programming Interface* (API) refere-se a uma arquitetura projetada para permitir a comunicação e interação entre diferentes sistemas e aplicações, inclusive por meio da *web*. Quando utilizadas em ambiente *web*, são convencionalmente chamadas de APIs remotas, e por meio de protocolos e padrões, como HTTP, URLs e formatos de dados como XML (*eXtensible Markup Language*) ou JSON, permite acesso controlado a recursos e funcionalidades específicas de um servidor *web*, incluindo dados, serviços ou outra funcionalidade que o servidor disponibilize (ALVES, 2015; REDHAT, 2023).

De acordo com Ferreira (2021), por uma API corresponder a uma coleção de regras e ferramentas, permite a interação entre diferentes aplicações, programas e plataformas, sem a necessidade de intervenção direta do usuário, ou a necessidade de

conhecimento por parte do desenvolvedor, da implementação interna do serviço a ser acessado em um determinado servidor.

Em uma API, o acesso a um recurso ou funcionalidade é realizado através de URLs específicas, denominadas *endpoints*. Para mapear as operações que podem ser executadas nesses recursos, à cada *endpoint* é associado um ou mais métodos HTTP, como *GET*, *POST*, *PUT* ou *DELETE*, que definem e delimitam as ações que podem ser realizadas no recurso (STEIN JUNIOR; SANTOS, 2019; FERREIRA, 2021).

2.2.2 Arquitetura Serverless

A popularização da internet e das aplicações *web*, impulsionou um interesse crescente no uso da computação em nuvem, principalmente devido à promessa de menores tempos de resposta, menores custos de armazenamento e computação, além de maior disponibilidade. De modo a atender esta demanda, alguns modelos de negócio surgiram, com diferentes abordagens para o problema, destacando o modelo IaaS (*Infrastructure as a Service*), que permite o compartilhamento de um *datacenter* entre várias aplicações ou clientes, mantendo o isolamento entre sistemas operacionais, o modelo PaaS (*Platform as a Service*), onde apenas a plataforma de execução é especificada, com a infraestrutura sendo transparente ao desenvolvedor, o modelo BaaS (*Backend as a Service*) que oferece soluções modulares para funções comuns de uma infraestrutura, como autenticação de usuários, integração com banco de dados, e serviços de notificações e por fim, o FaaS (*Function as a Service*), que fornece apenas um domínio para o desenvolvimento e implementação das funções responsáveis pela lógica de negócio, com o gerenciamento de recursos realizado pela plataforma, sendo o AWS Lambda um popular representante deste modelo(CASTRO *et al.*, 2019; JONAS *et al.*, 2019; BEBORTTA *et al.*, 2020).

As principais características de um serviço FaaS, também conhecido por *serverless* são descritas por Shafiei, Khonsari e Mousavi (2022) e Adzic e Chatley (2017) como:

- Supressão das informações sobre o ambiente de execução para o cliente, como máquina virtual, contêiner, sistema operacional, entre outros, tornando-as transparentes para o cliente;

- Escalabilidade como responsabilidade do provedor, com os recursos devendo estar disponíveis de acordo com a demanda;
- Cobrança somente pelos recursos utilizados pelo cliente.
- Limitação do tempo de execução máximo, com o provedor garantindo capacidade de executar a requisição assim que ela for recebida, ou o quanto antes possível.
- O provedor deve estar ciente das definições e dependências das funções adicionadas pelo cliente, como também quaisquer informações relacionadas ao estado e ambiente de execução.

De acordo com Adzic e Chatley (2017) e Shafiei, Khonsari e Mousavi (2022) a arquitetura *serverless* oferece benefícios significativos em termos de implementação de aplicações, eliminando a necessidade de gerenciar infraestrutura e lidar com preocupações de escalabilidade. Nesse modelo, o provedor de serviços é responsável por alocar os recursos necessários para atender às requisições, eliminando as limitações que uma aplicação enfrentaria em uma infraestrutura convencional baseada em máquinas virtuais ou contêineres. Em vez de manter máquinas dedicadas que muitas vezes têm baixa utilização média, o compartilhamento de recursos permite otimizar o uso e reduzir os custos relacionados à infraestrutura. Além disso, a cobrança baseada no processamento incentiva a otimização e eficiência para minimizar os custos associados.

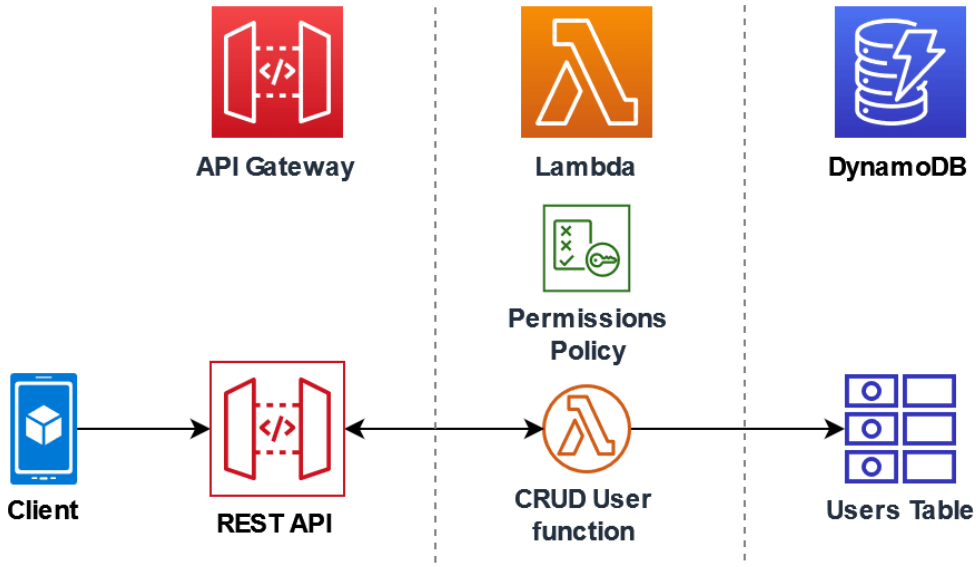
Pode-se questionar se plataformas *serverless* possuem bom desempenho para o processamento intenso de dados, visto sua dinâmica de provisionamento de recursos. Assim Lee, Satyam e Fox (2018), avaliaram diferentes ambientes *serverless*, focando em aspectos como desempenho do processador, utilização de memória e disco, escalabilidade e eficácia de custos. Seus resultados indicam que estes oferecem processamento eficiente e escalável, especialmente para cargas de trabalho temporárias. No entanto, sugerem que essas plataformas ainda precisam ser aprimoradas para lidar com cargas de trabalho mais complexas no futuro.

Já Niu *et al.* (2019) compararam o desempenho de duas plataformas *serverless*, o AWS Lambda e Google Cloud Functions, com arquiteturas tradicionais no sequenciamento de proteínas, visando avaliar estas em relação ao tempo de execução e consumo de recursos. De modo a lidar com restrições de espaço em disco, espaço em memória e número de unidades de processamento virtual, utilizaram uma abordagem paralela e distribuída. Os resultados mostraram que a arquitetura *serverless* pode proporcionar

um desempenho significativamente melhor, apresentando maior velocidade e eficiência na execução de tarefas em grandes conjuntos de dados. Por fim, observou-se que o AWS Lambda foi mais eficiente em comparação com o Google Cloud Functions.

Especificamente sobre a arquitetura *serverless* fornecida pelo AWS Lambda, Castro *et al.* (2019) e Jonas *et al.* (2019) afirmam que, por garantir completa abstração do servidor, eliminando a necessidade de gerenciamento de infraestrutura, permitem aos desenvolvedores se concentrarem apenas no código das funções. No entanto destacam algumas limitações, principalmente em relação a persistência de dados. De acordo com os autores, nestas situações, deve-se configurar e conectar sistemas de banco de dados independentes, podendo utilizarem a abordagem tradicional ou então utilizar armazenamento como serviço (*Storage as a Service*-STaaS), que embora mais cômodo por evitar a necessidade de configuração e manutenção, pode implicar em custos mais elevados ou à dependência de um fornecedor de serviço.

Figura 6 – Esquema simplificado de uso do AWS Lambda



Fonte: Amazon Web Services (2022).

Especificamente sobre a AWS Lambda, destaca-se que esta é orientada a eventos, o que significa dizer que uma função *lambda*, que nada mais é que uma função executada no serviço AWS Lambda, somente é invocada como resposta a algum evento. Os eventos podem ser desde uma requisição HTTP, uma solicitação de alguma aplicação específica ou alguma rotina periódica definida nas configurações (CASTRO *et al.*, 2019).

Em uma uma função *lambda*, uma diversidade de operações que podem ser

implementadas, como cálculos, validações, transformações de dados, acesso à bancos de dados e integração com serviços externos, bem como é suportado uma variedade de linguagens, como JavaScript, Python, Java, Golang e C#. Autores como Castro *et al.* (2019) e Bebortta *et al.* (2020), citam que em razão da plataforma ser baseada em contêineres, cada instância fornece especificações necessárias para a execução do trecho de código em um ambiente específico, possibilitando que a aplicação escale e provisione recursos automaticamente, conforme houver demanda para tal.

Quanto ao desempenho de diferentes linguagens de programação em funções *lambda*, autores como Shrestha (2019) e Hosseini e Sahragard (2019), destacam a eficácia da linguagem JavaScript. Esses estudos evidenciam a capacidade desta linguagem em apresentar baixo tempo de inicialização e um consumo eficiente de recursos no processamento de requisições. Ainda, Koskela (2022), sustenta que sua utilização em conjunto com seu *superset* TypeScript possibilita a implementação de tipagem estática, o que contribui para tornar a base de código mais robusta e clara, sem que isso implique em grandes esforços adicionais na implementação ou mesmo na migração de uma aplicação.

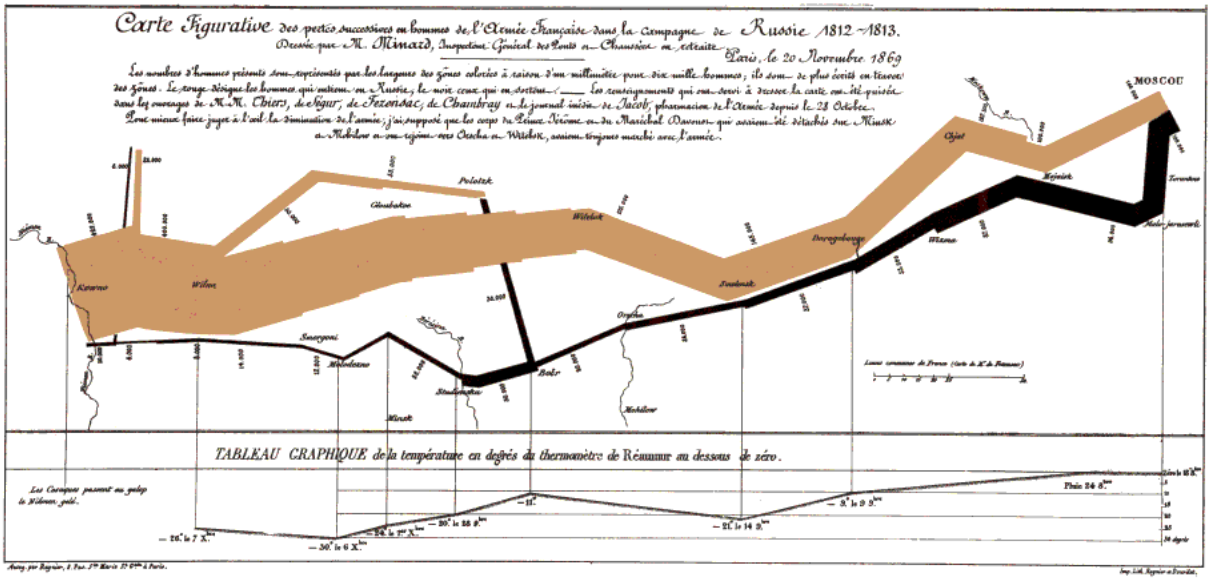
2.3 Visualização de Dados

A visualização de dados é o processo de representar informações de forma visual por meio de gráficos, mapas, diagramas e outras formas de representação. Seu objetivo é tornar os dados mais compreensíveis, facilitar a identificação de relações, anomalias, padrões e tendências que podem não ser percebidos apenas pelo uso de tabelas ou dados brutos, tornando mais fácil o processo de obtenção de *insights* relevantes.

Sharda, Delen e Turban (2019) apontam que as raízes da visualização de dados podem ser rastreadas até o século II d.C., mas os avanços significativos ocorreram principalmente nos últimos 250 anos. Dentre os pioneiros notáveis, destacam-se os trabalhos de William Playfair no final do século XVIII e de Charles Joseph Minard no início do século XIX. Playfair é reconhecido por sua contribuição na criação dos primeiros gráficos de linhas e barras, enquanto Minard se destacou ao representar visualmente as perdas sofridas pelo exército napoleônico na campanha da Rússia de 1812. Minard conseguiu retratar simultaneamente várias dimensões, como o tamanho

do exército, a direção do movimento, localizações geográficas e a temperatura externa, tudo em uma única representação gráfica, Figura 7.

Figura 7 – Representação de Minard do contingente do exercito napoleônico na campanha da Rússia de 1812



Fonte: Adaptada de Sharda, Delen e Turban (2019).

Com o advento e a popularização da internet no início do século XXI, ampliou-se consideravelmente a disponibilidade de dados e visualizações, tornando-os acessíveis a um público mais amplo. Esse cenário impulsionou a demanda por novas formas de visualização que incorporassem recursos interativos, alta qualidade gráfica, atualização em tempo real de dados e ambientes imersivos, a fim de possibilitar a eficiente e eficaz compreensão e consumo de informações (SHARDA; DELEN; TURBAN, 2019).

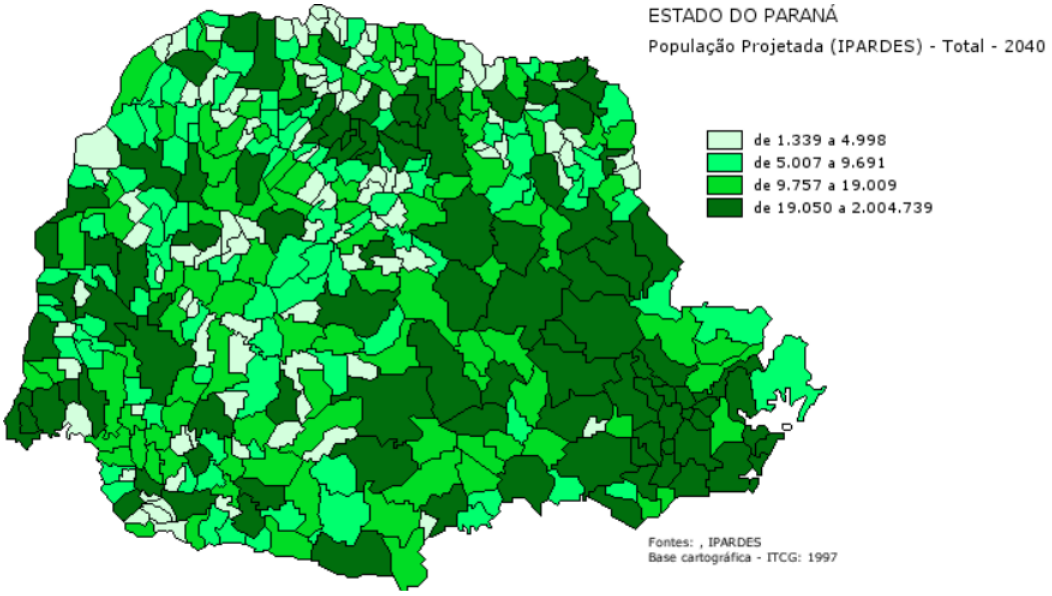
Para criar visualizações de dados eficazes, é necessário que os dados sejam processados e organizados adequadamente. Em casos que os dados se originam de fontes diversas e heterogêneas, o processo de adequação dos dados envolve a limpeza e transformação em um formato adequado, garantindo sua qualidade e confiabilidade. Além disso, deve-se atentar ao uso de técnicas de design visual, principalmente relacionadas à escolha de cores, uso adequado de escalas, eixos e representações, pois tais fatores desempenham um papel importante na criação de visualizações de dados.

Sharda, Delen e Turban (2019) ressaltam que, em relação à representação de dados, diversos diagramas e gráficos possuem características distintas que os tornam mais apropriados para responder a diferentes tipos de perguntas. Enquanto certos diagramas, como os gráficos de barras, de pizza e de linhas, são simples e de fácil compreensão, outras formas de representação mais complexas e especializadas, como

histogramas, diagramas de Gantt e *treemaps*, podem conter uma quantidade significativa de informações. Os autores ainda destacam que embora não exista uma regra universal para definição do melhor tipo de representação, é recomendado selecionar e usar aquele mais simples dentre as alternativas para facilitar a compreensão por parte do público-alvo.

Para representar dados que envolvem informações de localização, como endereços, códigos postais e coordenadas geográficas, Sharda, Delen e Turban (2019) e Horbiński e Lorek (2022) destacam o uso de mapas como uma forma adequada e informativa. Os autores enfatizam que o uso de mapas, combinado com outros diagramas e gráficos, pode ser uma ferramenta importante para representar informações demográficas e econômicas, Figura 8, proporcionando suporte à tomada de decisões.

Figura 8 – População Projetada do Estado do Paraná em 2040



Fonte: Adaptada de IPARDES (2023).

Longley *et al.* (2009) enfatizam que, devido à natureza volumosa, complexa e heterogênea das fontes de dados atuais, têm surgido técnicas interativas de geovisualização com o objetivo de torná-las compreensíveis para os usuários e evitar a sobrecarga de informações. Essas técnicas se distinguem da simples produção de mapas e da cartografia, pois envolvem o uso intensivo de ambientes computacionais interativos para a exploração dos dados, resultando na criação de múltiplas representações de dados espaciais e possibilitando a representação de mudanças ao longo do tempo.

Estudos como o de MacEachren e Kraak (2001) e Meneguette (2014) discutem

a necessidade de combinação de técnicas de visualização com análise espacial para auxiliar na exploração e comunicação de informações geográficas. Os autores ressaltam a necessidade de abordagens interdisciplinares e colaborativas para impulsionar o desenvolvimento dessa ferramenta e maximizar seu potencial na análise e tomada de decisões, discutindo os desafios que permeiam este campo, incluindo a compreensão dos processos cognitivos envolvidos na interpretação visual de dados geográficos, o desenvolvimento de técnicas eficazes de visualização, a avaliação da eficácia e usabilidade das visualizações, da interatividade e a integração de múltiplas fontes de dados geográficos.

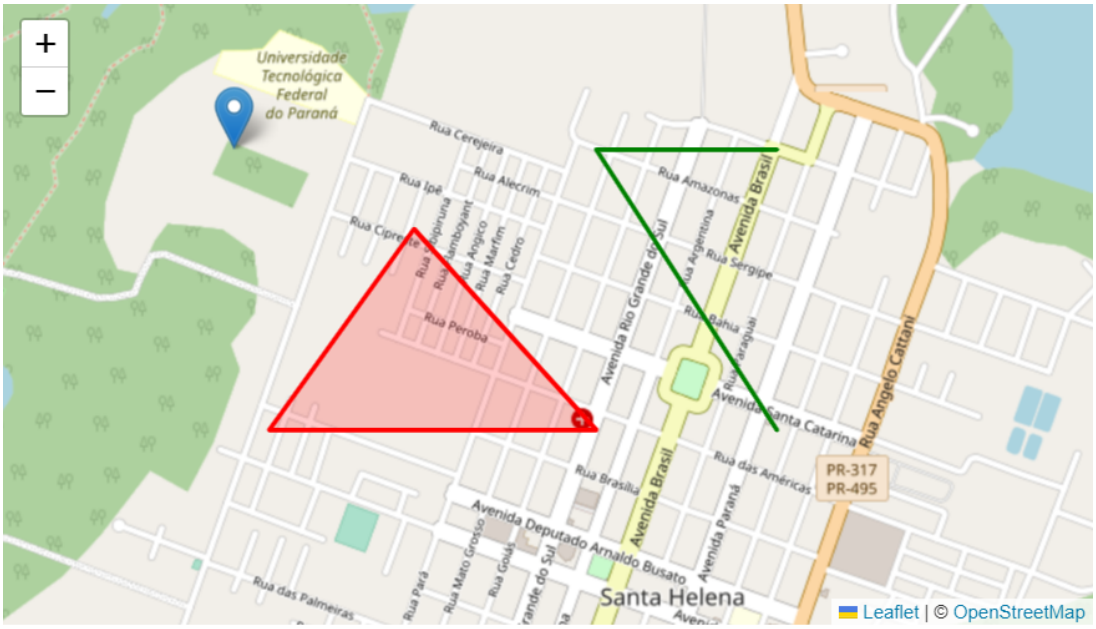
Netek, Brus e Tomecka (2019) realizaram a avaliação do desempenho de diversas bibliotecas JavaScript na criação de visualizações envolvendo grandes volumes de dados. Utilizando conjuntos de dados distintos, foram coletados os tempos de resposta no processo de construção de mapas. Os resultados indicaram que a biblioteca Leaflet apresentou um desempenho superior em relação à biblioteca OpenLayers.

Outros trabalhos, como o de Schoedon *et al.* (2016), Netek e Burian (2018), Seta e Hartomo (2020), Hynek, Kachlík e Rusňák (2021) e Arifin e Supriyatna (2023), também fazem uso da biblioteca Leaflet, fundamentando a escolha com base em critérios tais como simplicidade, desempenho, facilidade de uso, interatividade e a flexibilidade proporcionada pelo seu caráter de código aberto, que permite modificações e ajustes conforme necessidades específicas.

Sobre a biblioteca Leaflet, Horbiński e Lorek (2022), cita que esta foi projetada visando simplicidade, desempenho e usabilidade, apresentando código-fonte claro e transparente e possibilitando extensão por meio de *plugins*. Sua capacidade de suportar interações baseadas em toque, permitir que o cliente renderize arquivos no formato Scalable Vector Graphics (SVG) de forma independente, possuir um tamanho compacto de arquivos e ser suportada em diferentes plataformas *web* e móveis, tornou esta amplamente utilizada na construção de mapas interativos. Outro ponto fundamental é sua facilidade de uso, fornecendo recursos para adicionar e renderizar elementos como marcadores, polígonos e linhas, Figura 9, utilizando do formato GeoJSON para a troca de dados entre o servidor e o dispositivo cliente.

Outra ferramenta para a criação de paginas complexas e interativas é a biblioteca de construção de interface de usuário React. Segundo Gackenheimer (2015) e Fedosejev (2015), esta é uma biblioteca de JavaScript, com abordagem baseada em

Figura 9 – Renderização de um marcador, um polígono e uma linha com *Leaflet*.



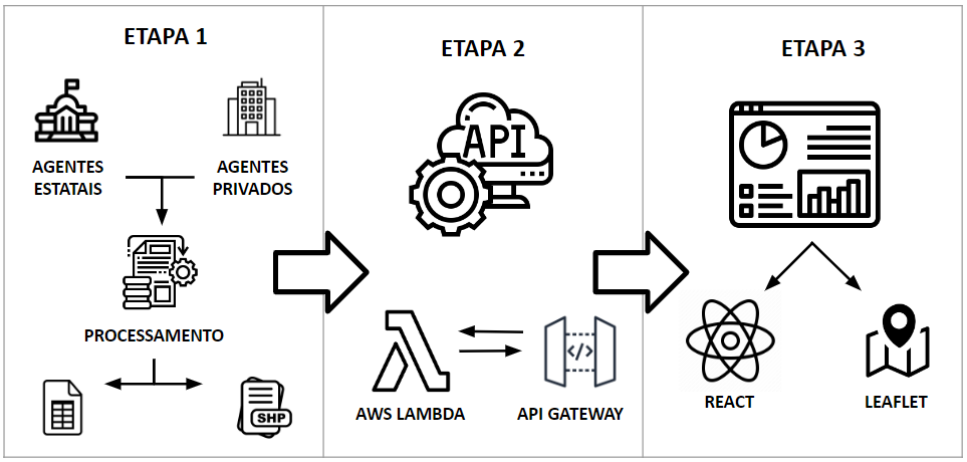
Fonte: Autoria própria (2023).

componentes, cada um responsável por uma parte específica da interface. Sua abordagem declarativa, onde os desenvolvedores descrevem como a interface do usuário deve ser exibida com base no estado da aplicação, permite que automaticamente a interface seja atualizada, conforme o estado é alterado, atualizando apenas os componentes que foram afetados, em vez de renderizar a página inteira, característica que resulta e um melhor desempenho e responsividade para o usuário. Além disso, o React é altamente flexível e pode ser combinado com outras bibliotecas e *frameworks* para construir aplicações mais complexas.

3 METODOLOGIA

Para o desenvolvimento deste estudo, cujo propósito é o desenvolvimento de uma plataforma destinada ao processamento e à visualização de dados geográficos, se faz imperativa a realização das fases delineadas na representação esquemática apresentada na Figura 10.

Figura 10 – Representação gráfica da metodologia deste trabalho.



Fonte: Autoria própria (2023).

A primeira etapa deste projeto foi dedicada à definição e obtenção de dados de fontes diversas, incluindo bancos de dados públicos mantidos por órgãos governamentais e privados. O processo de coleta é fundamental, pois representa a base sobre a qual toda a análise e tomada de decisão posterior serão construídas, devendo ser planejado e realizado com rigor metodológico, garantindo a qualidade e integridade dos dados adquiridos. Assim, esta etapa envolve a identificação de fontes confiáveis, a definição de critérios de seleção e a implementação de técnicas de coleta eficazes, que podem variar de raspagem da web a integração de APIs.

Na segunda fase, direcionou-se a atenção para a elaboração de uma arquitetura *serverless* dedicada ao processamento de dados. A seleção de tal arquitetura se justifica pela sua flexibilidade e capacidade de se adaptar a diferentes cargas de trabalho. Sua característica de escalonamento automático possibilita uma utilização eficiente e econômica de recursos computacionais, uma vez que somente aloca recursos na quantidade e tempo necessário para lidar com a carga de processamento demandada, revelando-se como uma alternativa moderna para a manipulação de volumes massivos de dados em contexto de grande escala.

Na terceira e última fase, dedicou-se à exposição dos dados processados por meio da concepção de representações visuais significativas. A partir do levantamento bibliográfico, decidiu-se pela elaboração de mapas interativos, visto que tais representações não só tornam os dados acessíveis a um público mais abrangente, mas também desempenham um importante papel na condução de análises exploratórias e na identificação de tendências e padrões. Destaca-se ainda, que a escolha desta técnica de visualização específica é influenciada pelos objetivos específicos do projeto e pelo público alvo, sendo a clareza e eficácia das representações visuais de suma importância para a interpretação e compreensão precisa dos dados a ser apresentado.

Em resumo, esse projeto seguiu um fluxo de trabalho bem definido, Figura 10, começando com o planejamento e coleta de dados, seguida pela implementação de uma arquitetura *serverless* para processamento eficiente e escalável, e concluído com a criação de visualizações que comunicam de maneira eficaz as informações derivadas dos dados processados. Cada etapa é primordial para o sucesso geral do projeto, e sua adequada integração contribui tanto para o processamento eficiente dos dados quanto para a criação de representações visuais significativas, o que, por sua vez, conduz a uma análise de dados sólida e à formulação de decisões bem fundamentadas.

No presente capítulo, todas as tecnologias e recursos empregados nos processos de tomada de decisão referentes a cada fase serão detalhados. Este capítulo se encontra estruturado em seções correspondentes a cada etapa.

3.1 Aquisição e Tratamento dos dados Geográficos

É comum encontrar na literatura, que as fontes de dados sejam heterogêneas, abrangendo desde documentos físicos, bases de dados eletrônicas, até documentos de texto digitalizados e dispositivos físicos de coleta de dados, como sensores e satélites.

Neste trabalho, a primeira etapa da coleta, consistiu da seleção dos municípios do Oeste do Paraná, visando obter informações referentes ao seu posicionamento geográfico, as suas limitações territoriais, seus vizinhos e dados de identificação. Estes dados foram extraídos do site oficial do Instituto Brasileiro de Geografia e Estatística (IBGE), por meio do portal Malha Municipal[1]. O portal fornece esses dados em arquivos

[1] https://www.ibge.gov.br/geociencias/organizacao-do-territorio/malhas-territoriais/15774-malhas.html.

shapefile que estão organizados e separados por estado. Em seguida, utilizou-se o software QGIS (2021) para realizar a transformação dos dados. Isso incluiu a exclusão de atributos e instâncias municipais não pertinentes, resultando na retenção dos cinquenta municípios que compõem a região oeste do Paraná. Como resultado, foi gerado um novo arquivo *shapefile* contendo as coordenadas necessárias para a representação vetorial desses dados por meio de mapas.

Já os atributos relacionados às características ambientais, foram obtidos do Instituto Água e Terra (IAT)[2], autarquia estadual criada pela Lei nº 20.070/19, que possui no rol de suas atribuições atividades relacionadas à gestão territorial do estado, bem como a elaboração de estudos de geoprocessamento, emissões de licenciamento e outorgas, acompanhamento e desenvolvimento de projetos de proteção e gestão dos recursos naturais presentes no território. Por meio do seu portal GEODADOS[3], foram obtidos arquivos *shapefiles* com as informações sobre os tipos de solo, a geologia, geomorfologia, declividade da superfície e clima. Utilizou-se o software de geoprocessamento QGIS (2021) para realizar a transformação dos dados, realizando a limpeza e organização das informações disponíveis, mantendo os atributos e municípios pertinentes a este projeto. Os dados transformados, foram então salvos em arquivos *shapefiles*.

Outro conjunto de dados coletados, esta relacionado à atributos de temperatura e umidade do ar, precipitação e evapotranspiração. Este dados foram obtidos a partir da base de dados ECMWF – ERA 5[4], abrangendo o período de 1990 a 2010, então processados, interpolados e inseridos na tabela de atributos por meio de ferramentas SIG, sendo então estruturados em arquivos matriciais, também conhecida como *raster*, e posteriormente gerados arquivos de imagem, no formato .png para carregamento estático na plataforma. Todo o processo é realizado manualmente por uma equipe coordenada pelo professor da Universidade Tecnológica Federal do Paraná, câmpus Santa Helena, o senhor Anderson Sandro da Rocha.

Os dados socioeconômicos e de produção agropecuária foram obtidos no site oficial do Instituto Brasileiro de Geografia e Estatística (IBGE) a partir do banco de dados e das tabelas estatísticas do SIDRA[5]. Nessa plataforma, foram adquiridos e compilados

[2] https://www.iat.pr.gov.br/.
[3] https://www.iat.pr.gov.br/Pagina/Mapas-e-Dados-Espaciais.
[4] https://www.ecmwf.int/en/forecasts/dataset/ecmwf-reanalysis-v5.
[5] https://sidra.ibge.gov.br/home/ipp/brasil.

dados quantitativos referentes aos censos agropecuários realizados nos anos de 1995, 2006 e 2017. Adicionalmente, foram coletados dados do Departamento de Economia Rural (Deral)[6], órgão estatal responsável pela coleta, análise e acompanhamento de diversas cadeias produtivas no estado. Os dados foram adquiridos diretamente no portal e estavam disponíveis em formatos .xlsx e .pdf. É relevante ressaltar que os dados de produção estavam contidos em um único arquivo, aonde as informações sobre o ano da safra, a produção em quilogramas, o valor bruto em moeda corrente e a área cultivada eram apresentadas em colunas distintas da tabela. Para adaptar esses dados às necessidades da plataforma, foram realizadas transformações e ajustes por meio de operações matemáticas nos atributos específicos de cada município. O resultado final desse processo foi a criação de um arquivo *shapefile* contendo os dados finais.

Por fim, a última categoria de dados a ser coletada refere-se às características de uso e cobertura do solo. Esses dados foram obtidos no portal da Empresa Brasileira de Pesquisa Agropecuária (Embrapa), que, em colaboração com a Agência Espacial Europeia (ESA), disponibiliza informações de monitoramento remoto por satélite obtidas por meio da missão Sentinel[7]. O processamento desses dados foi realizado utilizando o software de geoprocessamento QGIS (2021), aplicando a técnica de classificação supervisionada de imagens de satélite. Essa abordagem possibilita a categorização dos dados de acordo com seus usos e coberturas, com foco nas categorias específicas de agricultura, pastagem, áreas urbanas, corpos hídricos e solo exposto. Os dados processados foram organizados em arquivos matriciais *raster* e, posteriormente, foram gerados arquivos de imagem no formato *.png* para para carregamento estático na plataforma.

Em resumo, neste trabalho os dados, suas fontes e respectivos formatos disponibilizados, estão compilados na Tabela 1

3.2 Construção da Arquitetura *Serverless*

A arquitetura *serverless* é um paradigma de desenvolvimento de software que permite a execução de código em resposta a eventos, com provisionamento

[6] https://www.agricultura.pr.gov.br/Pagina/Departamento-de-Economia-Rural-Deral.
[7] https://www.embrapa.br/satelites-de-monitoramento/missoes/sentinel.

Tabela 1 – Fontes dos dados e formatos

Fonte	Categoria	Formato
IBGE	Limites municipais	shapefile
IBGE	Sensos Agropecuários	pdf e xlsx
Deral	Dados de produção agropecuária	pdf e xlsx
IAT	Geologia, tipo de solos, geomorfologia, topografia, clima	shapefile
ECMWF – ERA 5	Temperatura e umidade do ar, evapotranspiração e precipitação	raster
EMBRAPA /ESA	Uso e cobertura do solo	shapefile

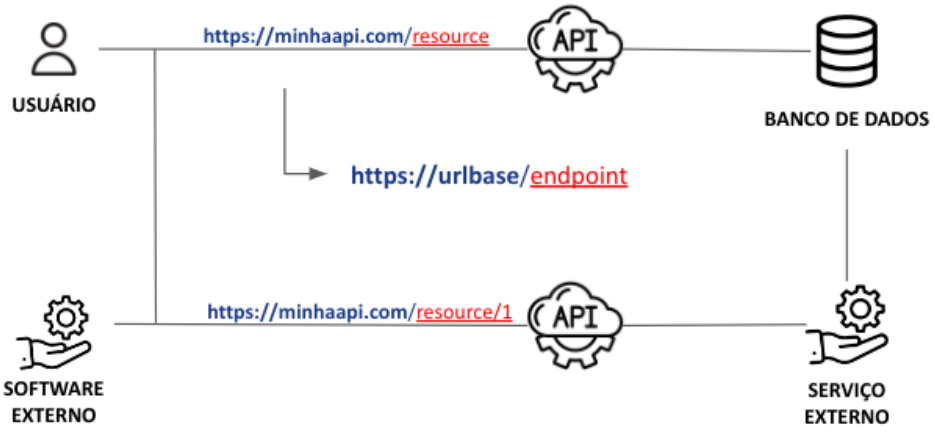
Fonte: Autoria própria (2023)

automático de recursos. Na infraestrutura da AWS, o serviço utilizado para hospedar e gerenciar o código desenvolvido é o AWS Lambda. Neste serviço, os desenvolvedores definem funções ou rotinas de código que respondem a eventos específicos, que podem incluir solicitações HTTP, gatilhos de banco de dados, eventos de armazenamento ou eventos personalizados. Quando um evento ocorre, a função é executada, com o provisionamento dos recursos necessários, sendo automaticamente realizado pela plataforma.

No contexto de uma arquitetura *serverless*, as APIs desempenham um papel central, definindo como diferentes componentes de software podem interagir uns com os outros. Em outras palavras, as APIs são responsáveis por expor funcionalidades do sistema, especificando os métodos disponíveis, os formatos de dados aceitáveis e as operações que podem ser realizadas. Nas APIs, o acesso a um recurso ou funcionalidade é efetuado por meio de URLs específicas, denominadas *endpoints*, Figura 11. Os *endpoints* atuam como pontos de acesso que estabelecem as regras de comunicação e interação da API com o ambiente externo. A definição das operações que podem ser executadas nesses recursos ocorre através da associação de um ou mais métodos HTTP, como *GET*, *POST*, *PUT* ou *DELETE*, os quais determinam e delimitam as ações que podem ser realizadas no referido recurso.

Toda a configuração e comunicação com os *endpoints* da API é realizada por meio do serviço Amazon API Gateway. Esta ferramenta atua recebendo e manipulando as requisições, designando assim qual serviço ou função precisará ser invocado, assim como retornando o resultado obtido, quando necessário. No contexto deste trabalho, quatro *endpoints* foram definidos e configurados, Tabela 2. A cada um desses *endpoints*, são associados métodos HTTP específicos, estabelecendo as operações e ações que podem ser realizadas no sistema. Essa abordagem no design oferece uma organização clara e eficiente para a comunicação e interação entre os componentes da aplicação

Figura 11 – Exemplo de um *endpoint*.



Fonte: Autoria própria (2023).

e usuários, contribuindo para a sua implementação e o desempenho otimizado do sistema.

Tabela 2 – *Endpoints* e métodos associados

Endpoint	GET	POST	PUT	DELETE
/categorias/	getAll()	create()	-	-
/mapas/	getAll()	create()	-	-
/categorias/{cat_id}	getById(id)	-	modify(id)	delete(id)
/mapas/{map_id}	getById(id)	-	modify(id)	delete(id)

Fonte: Autoria própria (2023)

A definição, configuração dos serviços necessários à aplicação, como é o caso das funções *lambda* e a associação destas no Amazon API Gateway foi realizada por meio do AWS CloudFormation. Esta ferramenta de orquestração permite criar, implantar e gerenciar a *stack* de recursos, ou seja, o conjunto de serviços e recursos necessários à aplicação, de maneira consistente, garantindo a inicialização e o provisionamento automático de toda a infraestrutura necessária.

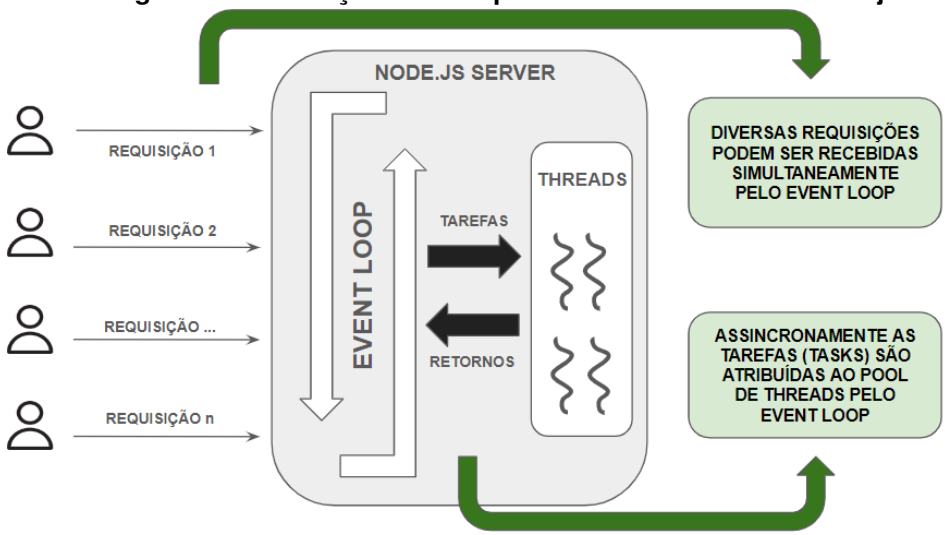
Com o ambiente AWS definido e configurado, iniciou-se o desenvolvimento da API. Para cada *endpoint*, foram associadas funções que implementam a lógica de negócios para determinar como os dados são processados, quais regras de negócios se aplicam e quais operações são executadas. Isso envolve desde acesso aos dados no banco de dados, bem como a transformação dos dados tabulares e vetoriais relacionados a cada mapa específico em um formato JSON adequado. Esses arquivos JSON são posteriormente disponibilizados como respostas às requisições realizadas aos *endpoints* correspondentes.

3.2.1 A construção da API

A construção da API deste trabalho foi realizada usando o *framework* AWS Cloud Development Kit (CDK), para a definição da infraestrutura e recursos necessários à aplicação, e a biblioteca AWS Software Development Kit (SDK), que fornece uma interface de alto nível para interação com os serviços disponíveis na AWS. Além disso, será abordada a integração com o banco de dados Amazon DynamoDB e o Amazon Simple Storage Service (S3), ambos utilizados para armazenamento de dados e ativos essenciais à operação da aplicação.

Quanto à linguagem de programação, foi adotada a combinação de JavaScript e sua extensão, o TypeScript. O TypeScript inclui à linguagem, tipagem estática e recursos de desenvolvimento que contribuem para a redução de erros e o aumento da produtividade. Ademais, por meio do interpretador Node.js, é viabilizada a execução de JavaScript no servidor, permitindo que a aplicação tire proveito das características do modelo de E/S (entrada/saída) não bloqueante de execução assíncrona, Figura 12, o que é particularmente benéfico para aplicações que lidam com múltiplas conexões simultâneas, como servidores da web. A disponibilidade de módulos e bibliotecas nesse ecossistema torna essa combinação de ferramentas uma escolha interessante e popular no desenvolvimento de aplicações web.

Figura 12 – Execução não bloqueante e assíncrona no Node.js.



Fonte: Autoria própria (2023).

Por meio do AWS CDK, é possível descrever a infraestrutura como código, o que facilita a criação, a modificação e o provisionamento de recursos necessários para

a aplicação desenvolvida. Todos os parâmetros de configuração são definidos por meio do arquivo cdk.json, Figura 13. O arquivo referido está subdividido em três seções, cada uma desempenhando um papel distinto na configuração do ambiente do projeto:

- A primeira seção, referenciada pela palavra chave 'app', é responsável por definir o comando de inicialização, indicando o processamento do arquivo 'bin/back-end-v.1.ts'.
- A segunda seção, denominada 'watch', determina os arquivos e diretórios que são monitorados durante a execução do CDK, assegurando a atualização de recursos em resposta a quaisquer modificações no código.
- A terceira seção, 'context', configura opções relacionadas a validações e comportamentos específicos, onde cada entrada pode ser configurada como 'true' ou 'false', determinando o tratamento dessas validações e comportamentos ao longo do ciclo de desenvolvimento e implantação do projeto.

Figura 13 – Definições no arquivo cdk.json.

```
{
  "app": "npx ts-node --prefer-ts-exts bin/back-end-v.1.ts",
  "watch": {
    "include": [
      "**"
    ],
    "exclude": [
      "README.md",
      ...
    ]
  },
  "context": {
    "@aws-cdk/aws-lambda:recognizeLayerVersion": true,
    "@aws-cdk/core:checkSecretUsage": true,
    "@aws-cdk/core:target-partitions": [
      "aws",
      "aws-cn"
    ],
    "@aws-cdk/aws-ec2:uniqueImdsv2TemplateName": true,
    ...
    "@aws-cdk/aws-lambda-nodejs:useLatestRuntimeVersion": true
  }
}
```

Fonte: Autoria própria (2023).

Por meio do arquivo executado na primeira sessão, Figura 14, é orquestrada a criação, o provisionamento de recursos e o gerenciamento das diferentes *stacks* que

compõem a aplicação, estabelecendo as relações de dependência entre elas, bem como as variáveis de ambiente e características comuns, como região e conta da AWS.

Figura 14 – Definições no arquivo app.

```
#!/usr/bin/env node

// realiza as importações necessárias
import * as cdk from 'aws-cdk-lib';
...
// instancia um app cdk
const app = new cdk.App();

// define as variaveis
const env: cdk.Environment = {
  account: "",
  region: "us-east-1"
}

const tags = {
  const: "Back-end_PortalGeoeste",
  team: "PortalGeoeste-1"
}

const mapsAppLayersStack = new MapsAppLayersStack(app, "
    MapAppLayer", {
  tags: tags,
  env: env,
})

const mapsAppStack = new MapsAppStack(app, "MapsApp", {
  tags: tags,
  env: env,
})
mapsAppStack.addDependency(mapsAppLayersStack)
...

const backendApiStack = new BackendAPIStack(app, "BackendApi", {
  mapsFetchHandler: mapsAppStack.mapsFetchHandler,
  mapsAdminHandler: mapsAppStack.mapsAdminHandler,
  tags: tags,
  env: env,
})
backendApiStack.addDependency(mapsAppStack)
```

Fonte: Autoria própria (2023).

As *stacks* criadas podem ser classificadas em três grupos. O primeiro grupo, identificado pelo sufixo 'LayerStack' na nomenclatura de seus arquivos, corresponde às *stacks* responsáveis por estabelecer a estrutura de dados e os métodos para para

recuperar, criar e excluir informações do banco de dados. A definição desta estrutura em arquivos distintos é útil para compartilhar código comum e bibliotecas entre várias funções *lambda*, simplificando o desenvolvimento e a manutenção, pois o código da camada pode ser atualizado centralmente, e as funções *lambda* podem se concentrar apenas em sua lógica de negócios específica, Figura 15.

Figura 15 – Definições no arquivo LayerStack

```
1  import * as lambda from "aws-cdk-lib/aws-lambda";
2  import * as cdk from "aws-cdk-lib";
3  import * as ssm from "aws-cdk-lib/aws-ssm"
4  import { Construct } from "constructs";
5
6  export class MapsAppLayersStack extends cdk.Stack {
7      readonly mapsLayers: lambda.LayerVersion
8
9      constructor(scope: Construct, id: string, props?: cdk.
       StackProps){
10         super(scope, id, props)
11
12         this.mapsLayers = new lambda.LayerVersion(this, "
       MapsLayer", {
13             code: lambda.Code.fromAsset('lambda/maps/layers/
       mapsLayer'),
14             compatibleRuntimes: [lambda.Runtime.NODEJS_16_X],
15             layerVersionName: "MapsLayer",
16             removalPolicy: cdk.RemovalPolicy.DESTROY
17         })
18         new ssm.StringParameter(this, "MapsLayerVersionArn", {
19             parameterName: "MapsLayerVersionArn",
20             stringValue: this.mapsLayers.layerVersionArn
21         })
22 }}
```

Fonte: Autoria própria (2023).

O segundo grupo, que possui no nome dos seus arquivos o sufixo ’AppStack’, Figura 16, corresponde à definição e inicialização dos dos recursos no AWS e a integração destes com uma camada do AWS Lambda. O código define detalhes, como a instanciação do banco de dados e suas propriedades, bem como as configurações de permissão e ambiente das funções *lambda* utilizadas para interação com este e com o serviço de armazenamento S3.

Figura 16 – Definições no arquivo AppStack

```
1  // realiza as importações necessárias
2  import * as lambda from "aws-cdk-lib/aws-lambda";
3  ...
4
5  export class MapsAppStack extends cdk.Stack {
6    // define as funções e variáveis
7    readonly mapsFetchHandler: lambdaNodeJS.NodejsFunction
8    readonly mapsAdminHandler: lambdaNodeJS.NodejsFunction;
9    readonly mapsDdb: dynamodb.Table
10
11   constructor(scope: Construct, id: string, props?: cdk.
       StackProps) {
12     super(scope, id, props)
13
14     // constrói os layers dos maps
15     const mapsLayerArn = ssm.StringParameter.
       valueForStringParameter(
16       this, "MapsLayerVersionArn")
17
18     // criar tabela do dynamo para o maps
19     this.mapsDdb = new dynamodb.Table(this, "MapsDdb", {
20       // define as propriedades da tabela ...
21     })
22
23     // função de acesso e captura dos dados gravados
24     this.mapsFetchHandler = new lambdaNodeJS.NodejsFunction(
25       this,
26       "MapsFetchFunction", {
27         // declara as propriedades da função lamda, como o
       runtime:
28         runtime: lambda.Runtime.NODEJS_16_X,
29         // memória alocada
30         memorySize: 128,
31         ...
32       })
33     // define e constrói outras funções declaradas ...
34 }}
```

Fonte: Autoria própria (2023).

Por fim, há a instância corresponde à criação e configuração da *stack* principal da API, Figura 17. Nesta são definidas propriedades, como os manipuladores de funções Lambda, seus recursos e integrações com serviços como Amazon API Gateway, bem como dependências e outras configurações de ambiente para essa *stack*. As funções textitlambda referenciadas neste arquivo, são as responsáveis por atuarem como um controlador de API, que recebe eventos de solicitação do Amazon API

Gateway e determina a ação a ser executada com base no recurso e no método HTTP da solicitação.

Figura 17 – Definições no arquivo da *stack* principal.

```
// realiza as importações necessárias
import * as apigateway from "aws-cdk-lib/aws-apigateway"
...

export class BackendAPIStack extends cdk.Stack {

  constructor(scope: Construct, id: string,
    props: BackendAPIStackProps) {
    super(scope, id, props)

    const api = new apigateway.RestApi(this, "BackendAPI", {
      // define as propriedades do APIGateway, como nome:
      restApiName: "BackendAPI",
      // opções de deploy
      deployOptions: {
        // regras e propriedades
        })
      }
    })

    // integra as funções lambda com o APIGateway
    const mapsFetchIntegration =
      new apigateway.LambdaIntegration(props.mapsFetchHandler)
    const mapsAdminIntegration =
      new apigateway.LambdaIntegration(props.mapsAdminHandler)

    // realiza os mapeamentos das rotas
    const mapsResource = api.root.addResource("maps")
    mapsResource.addMethod("GET", mapsFetchIntegration)
    mapsResource.addMethod("POST", mapsAdminIntegration)
    const mapIdResource = mapsResource.addResource("{id}")
    mapIdResource.addMethod("GET", mapsFetchIntegration)
    mapIdResource.addMethod("PUT", mapsAdminIntegration)
    mapIdResource.addMethod("DELETE", mapsAdminIntegration)
}}
```

Fonte: Autoria própria (2023).

Também deve-se discutir a implementação das funções *lambda* que atuam como *endpoints* da API, Figura 18, respondendo a eventos originados do Amazon API Gateway, os quais estão relacionados a diferentes rotas e métodos HTTP. A função é responsável pela criação e acesso ao banco de dados, considerando as regras de negocio e atributos dos dados definidos em um repositório separado. A função deve processar as requisições, adequar os dados aos formatos de saída esperados, realizar

tratamento de erros e exceções e retornar as respostas com os códigos de status adequados à cada desfecho.

Figura 18 – Definições de uma função *lambda*.

```
1  // realiza as importações necessárias
2  import { Map, MapRepository } from "/opt/nodejs/mapsLayer";
3  import { DynamoDB } from "aws-sdk"
4  ...
5  // cria as variáveis para interar com o banco de dados
6  const mapDdbTable = process.env.MAPS_DDB!
7  const clientDdb = new DynamoDB.DocumentClient()
8  const mapRepository = new MapRepository(clientDdb, mapDdbTable)
9
10 export async function handler(
11   event: APIGatewayProxyEvent, context: Context):
12   Promise<APIGatewayProxyResult> {
13     const lambdaRequestId = context.awsRequestId
14     const apiRequestId = event.requestContext.requestId
15     const method = event.httpMethod
16
17     // conforme a rota acessada, define as operações e respostas
18     if (event.resource === "/maps") {
19       const map = JSON.parse(event.body!) as Map
20       const mapCreated = await mapRepository.create(map)
21
22       return {
23         statusCode: 201,
24         body: JSON.stringify(mapCreated)
25       }
26     } else if (...) {
27       // realiza as demais operações e respostas adequadas
28     }
29 }
```

Fonte: Autoria própria (2023).

No arquivo correspondente ao repositório importado pela função *lambda*, Figura 19, são definidas as estruturas de dados dos objetos que serão comunicados com o banco de dados, bem como a implementação dos métodos para realizar as operações de modificação, criação, exclusão e recuperação das informações no banco de dados. A construção destes elementos separados da função *lambda*, permite modificações na estrutura dos dados e nos métodos, independente às operações na função *lambda* que referencia o repositório, trazendo mais modularidade ao projeto e facilitando eventuais alterações e manutenções no código.

Figura 19 – Repositório utilizado pela função *lambda*.

```
1  // realiza as importações necessárias
2  import { DocumentClient } from "aws-sdk/clients/dynamodb" ...
3
4  // define a estrutura dos dados
5  export interface Map {
6    id: string,
7    mapName: string,
8    geojsonUrl?: JSON,   ...
9  }
10
11 // define os métodos de interação com o banco de dados
12 export class MapRepository {
13   private ddbClient: DocumentClient
14   private ddbTable: string
15
16   constructor(ddbClient: DocumentClient, ddbTable: string) {
17     this.ddbClient = ddbClient
18     this.ddbTable = ddbTable
19   }
20   \/ criando os metodos de acesso à tabela
21   async getAll(): Promise<Map[]> {
22     const data = await this.ddbClient.scan({
23       TableName: this.ddbTable
24     }).promise()
25     return data.Items as Map[]
26   }
27   async getById(itemId: string): Promise<Map> {
28     // implementação de acesso a um item específico
29   }
30   async create(itemToPersist: Map): Promise<Map> {
31     // implementação da criação de um item
32   }
33   async exclude(itemId: string): Promise<Map> {
34     // implementação da exclusão de um item específico
35   }
36   async modify(itemId: string, itemToModify: Map): Promise<Map>
      {
37     // implementação de modificação de um item específico
38   }
39 }
```

Fonte: Autoria própria (2023).

3.2.2 Modelagem e armazenamento dos dados

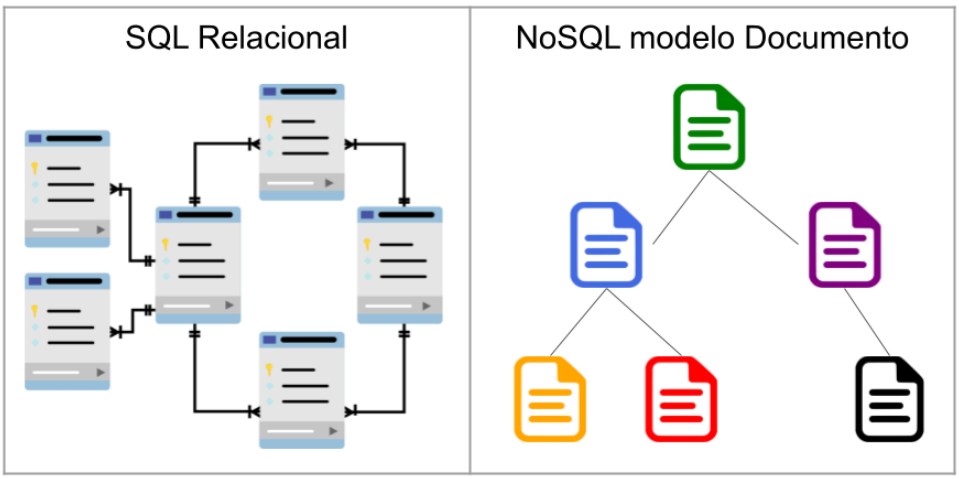
Para armazenamento de dados necessários ao funcionamento da aplicação, optou-se pelo uso conjunto do banco de dados Amazon DynamoDB para armazenamento de chaves de acesso e informações de relacionamento, e do S3 para armazena-

mento de arquivos de grande monta, como arquivos vetoriais e dados estáticos.

A utilização do *DynamoDB*, é baseada nas razões a seguir elencadas. A primeira é sua escalabilidade, permitindo o redimensionamento de instâncias de banco de dados de acordo com a demanda, garantindo um desempenho otimizado. Ademais, a hospedagem nativa da AWS fornece alta disponibilidade, com recursos de *backup* automático, replicação e tolerância de falhas, assegurando assim que os dados estejam acessíveis. Também deve-se destacar os recursos de proteção de dados em trânsito e em repouso, grupos de segurança de rede e integração com serviços de autenticação e autorização da própria AWS. Por fim, mas não menos importante, a integração com outros serviços, o gerenciamento simplificado e a economia de custos através de opções de pagamento por uso, complementam as vantagens do uso *DynamoDB* na AWS, tornando esta solução uma escolha sólida para aplicações hospedadas na nuvem.

O Amazon DynamoDB segue uma abordagem de armazenamento de dados NoSQL, onde os dados são armazenados em tabelas, com cada item representando uma única entrada de dados. Diferentemente de bancos de dados relacionais, o Amazon DynamoDB não exige um esquema de dados pré-definido, permitindo uma flexibilidade significativa na adição ou remoção de atributos em itens, Figura 20. Essa estrutura flexível é ideal para cenários em que a agilidade e a escalabilidade são fundamentais, tornando o Amazon DynamoDB uma escolha robusta para aplicações que exigem alta disponibilidade, escalabilidade horizontal e armazenamento de dados semi-estruturados.

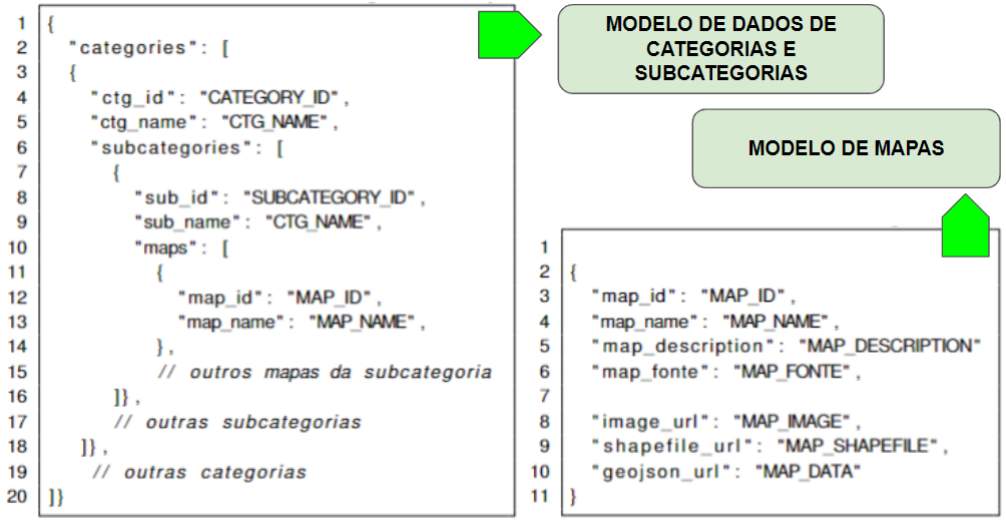
Figura 20 – Diferença entre o modelo SQL e NoSQL



Fonte: Autoria própria (2023).

No âmbito do presente projeto, foram criadas duas tabelas no banco de dados Amazon DynamoDB. A primeira tabela destina-se ao armazenamento de dados relativos às categorias das informações contidas e suas subcategorias relacionadas, por meio de uma organização hierárquica dos dados, onde as categorias são mapeadas como itens principais e as subcategorias como itens secundários. A segunda tabela, foi configurada para abrigar dados relacionados aos mapas, incluindo identificadores dos arquivos contendo os dados geográficas em diferentes formatos, fontes e legendas. Essa abordagem não relacional do Amazon DynamoDB facilita a gestão e recuperação de informações, proporcionando uma estrutura eficaz para a aplicação, Figura 21.

Figura 21 – Exemplo de estrutura de arquivos do banco de dados NoSQL



Fonte: Autoria própria (2023).

Já o uso do S3 para o armazenamento de arquivos de dados e arquivos estáticos é justificado pela sua escalabilidade e economia de custos associada ao modelo de pagamento por uso. Ao fazer uso do S3 para arquivos grandes e que não necessitam modificação ao longo da vida da aplicação, evita-se sobrecarregar o Amazon DynamoDB, que é mais apropriado para transações de baixa latência e armazenamento de metadados. Essa estrategia de uso conjunto destes dois serviços de armazenamento resulta em economia de custos significativa e uma arquitetura mais eficiente, garantindo que cada serviço seja aproveitado em seu cenário ideal, atendendo às necessidades técnicas do projeto, sem no entanto, descuidar dos aspectos econômicos deste.

3.3 Geração das Visualizações de Dados

A visualização de dados é o processo para representar informações de forma visual, por meio de gráficos, mapas e diagramas, a fim de tornar os dados mais compreensíveis, identificar relações, padrões e tendências. Essa abordagem vai além do uso de tabelas ou dados brutos, permitindo obter *insights* relevantes. Para criar visualizações eficazes, é necessário processar e organizar os dados adequadamente, especialmente quando eles vêm de fontes diversas e heterogêneas, garantindo qualidade e confiabilidade. Além disso, o uso adequado de técnicas de design visual, como cores, escalas e representações, é fundamental para uma visualização impactante e informativa.

Devido às peculiaridades geográficas dos dados, uma abordagem eficaz para representá-los é o uso de mapas interativos, os quais possibilitam a visualização de informações diversas, como dados econômicos e ambientais, por meio de representações distintas. Essa abordagem permite a geração de visualizações dinâmicas e informativas, proporcionando uma compreensão mais aprofundada e abrangente dos dados em um contexto espacial.

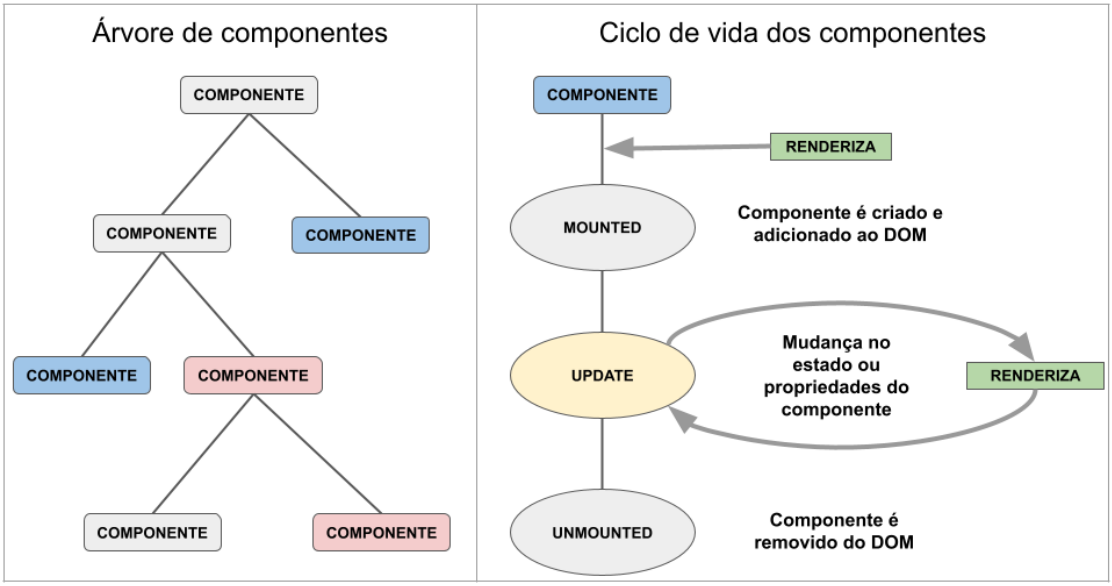
Na implementação da arquitetura *serverless*, a exploração de visualizações, especialmente por meio de mapas interativos, se torna altamente flexível e adaptável aos diferentes perfis de dados e visualizações. A principal vantagem esta na facilidade com que essas visualizações podem ser criadas e personalizadas dentro da mesma arquitetura. O uso de funções *lambda* possibilita que as requisições de dados sejam tratadas de forma dinâmica, adaptando-se automaticamente aos diversos formatos e necessidades. Por exemplo, para um mapa interativo que exibe dados geográficos sobre população, a função *lambda* pode recuperar dados específicos do banco de dados, formatá-los em um arquivo JSON e, em seguida, enviar para o *frontend* da aplicação, onde um serviço de visualização, como Leaflet, utilizado neste trabalho, é usado para criar o mapa interativo.

Para diferentes requisitos de visualização, a arquitetura *serverless* oferece a capacidade de criar visualizações altamente personalizadas sem a necessidade de reestruturação significativa. Por exemplo, uma visualização focada em dados demográficos pode utilizar a arquitetura para criar mapas temáticos detalhados, enquanto uma visualização de análise de produtividade pode utilizar a mesma infraestrutura para criar

visualizações baseadas em dados de produção e produtividade.

Referente à construção da apresentação da interface do usuário no *frontend*, está foi desenvolvida por meio do uso da biblioteca do Javascript, o React. Sua abordagem baseada em componentes, cada um responsável por uma parte específica da interface, arranjados em uma arvore de componentes, possibilitou combinar diferentes componentes para criar páginas complexas e interativas. Sua abordagem declarativa, onde os desenvolvedores descrevem como a interface do usuário deve ser exibida com base no estado da aplicação, permite que por meio do *Document Object Model* (DOM) e do *Virtual Document Object Model* (Virtual DOM), a interface seja automaticamente atualizada, conforme o estado é alterado, modificando apenas os componentes que foram afetados, em vez de renderizar a página inteira, característica que resulta e um melhor desempenho e responsividade para o usuário. Além disso, o React é altamente flexível e pode ser combinado com outras bibliotecas e *frameworks* para construir aplicações mais complexas.

Figura 22 – Estrutura do DOM de uma página e ciclo de vida de um componente



Fonte: Autoria própria (2023).

Especificamente à construção das visualizações dos dados geográficos por meio de mapas, utilizou-se a biblioteca React-Leaflet, que possibilita manipular os dados retornados pela API em formato JSON, gerando as renderizações necessárias. A escolha desta ferramenta é justificada por esta atuar como um envoltório (*wrapper*) para a biblioteca Leaflet, fornecendo recursos para renderizar mapas geográficos, adicionar elementos como marcadores, polígonos e linhas, bem como realizar interações como

zoom e panorâmica. Por meio desta biblioteca, é possível integrar facilmente recursos de construção de visualizações geográficas em aplicações React, aproveitando as funcionalidades fornecidas pela biblioteca Leaflet e a facilidade de uso e organização proporcionada pelo React.

A partir da implementação do *frontend* da aplicação, ocorrem as requisições à *endpoints* específicos da API, que, por sua vez, retorna os dados no formato JSON. Posteriormente, esses dados são submetidos a processamento, visando a segmentação das informações identificativas da visualização, como nome, legenda e fonte, e informações geográficas. Após a segmentação dos dados em variáveis apropriadas, os componentes da visualização são renderizados ao usuário, permitindo a este, interagir com os mapas gerados.

3.4 Resumo das tecnologias utilizadas

No escopo deste trabalho, diversas tecnologias foram empregadas para atender aos objetivos propostos neste trabalho. A Tabela 3 oferece um resumo das aplicações dessas tecnologias, incluindo às relacionadas aos serviços de configuração da infraestrutura, serviços de armazenamento de dados e linguagens de programação. A diversidade e a integração dessas tecnologias proporcionam uma abordagem holística para a consecução dos objetivos estabelecidos no projeto.

Tabela 3 – Tecnologias utilizadas e suas aplicações

Tecnologia/Serviço	Aplicação
AWS Lambda [8]	Função de interação com dados
AWS DynamoDB [9]	Banco de dados não relacional
AWS S3 [10]	Serviço de armazenamento de objetos
AWS APIGateway [11]	Administrador de rotas e acesso
Cloud Development Kit [12]	Definição da infraestrutura
Software Development Kit [13]	Interface de acesso aos serviços AWS
JavaScript [14]	Linguagem de programação
Typescript [15]	Extensão da linguagem JavaScript
Node.js [16]	Interpretador JavaScript
React [17]	Biblioteca Javascript para criação de interface de usuario
Leaflet [18]	Biblioteca JavaScript de criação de mapas interativos
React-Leaflet [19]	Envoltorio da biblioteca Leaflet para o React

[8] https://aws.amazon.com/pt/pm/lambda/
[9] https://aws.amazon.com/pt/pm/dynamodb/
[10] https://aws.amazon.com/pt/s3/
[11] https://aws.amazon.com/pt/api-gateway/
[12] https://aws.amazon.com/pt/cdk/
[13] https://aws.amazon.com/pt/what-is/sdk/
[14] https://developer.mozilla.org/pt-BR/docs/Web/JavaScript
[15] https://www.typescriptlang.org/
[16] https://nodejs.org/en
[17] https://react.dev/
[18] https://leafletjs.com/
[19] https://react-leaflet.js.org/

Fonte: Autoria própria (2023)

4 RESULTADOS E DISCUSSÕES

Nesta seção de resultados, são descritas as análises obtidas na avaliação da eficiência de uma arquitetura *serverless* em relação a uma arquitetura convencional na entrega de visualizações de mapas interativos. Como descrito na seção de Introdução, a hipótese central é que uma arquitetura *serverless* se mostraria mais eficaz na entrega dessas visualizações.

4.1 Apresentação das visualizações geradas

Nesta primeira parte da seção de resultados, é fornecida uma visão geral dos mapas interativos deste estudo. Conforme descrito na Seção 1 e 3, foram criadas visualizações com base em dados ambientais, geográficos e econômicos, sendo projetados para serem explorados pelos usuários. Por incorporarem recursos interativos, este tipo de visualização, possibilita aprimorar a experiência do usuário, facilitando a compreensão e análise dos dados apresentados.

No total, foram elaboradas vinte e nove visualizações gráficas de diferentes aspectos ambientais e agrícolas. As visualizações relacionadas às aspectos ambientais, incluem clima, geologia, hidrografia, pedologia e vegetação. Além disso, foram construídas visualizações relacionadas à temática agropecuária, como a produção de aves, bovinos e suínos, e a produção e produtividade média das principais culturas, como soja, milho, trigo e mandioca.

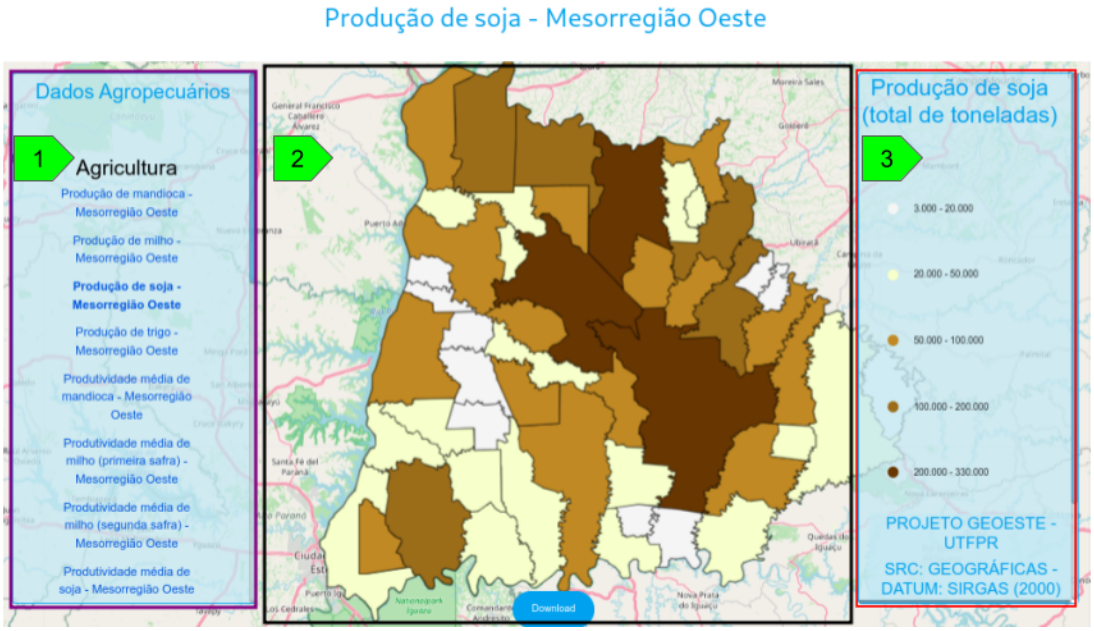
A apresentação das informações nas visualizações geradas, compreende três elementos principais, Figura 23. A primeira dessas estruturas, numerada como 1, é o menu de interação. Este menu confere aos usuários o poder de alternar entre diferentes visualizações de maneira intuitiva. Em seu interior, estão todos os mapas disponíveis, criteriosamente organizados por categorias de informações.

Outro importante componente, numerado como 2, é o próprio mapa interativo, que constitui a representação gráfica das informações temáticas e geográficas de forma acessível e facilmente interpretável. Sua posição central como componente visual na página, é justificada por ser este o elemento fundamental para a compreensão do contexto espacial.

Outra parte essencial na construção das visualizações é o campo de informações, numerado como 3. Trata-se de uma área que fornece informações adicionais e facilitando o entendimento dos dados apresentados no mapa. Nesse campo, são apresentadas as legendas, que subsidiam o entendimento sobre o que os diferentes elementos e cores no mapa significam. Além disso, neste campo é apresentado detalhes sobre a fonte primária dos dados apresentados, bem como sobre os responsáveis pela coleta e compilação destas. O principal objetivo deste tipo de informação apresentada, é conferir credibilidade e transparência à página, possibilitando ao usuário a conferência e coleta dos dados, etapas que a depender do contexto, podem ser de vital importância para a tomada de decisões adequadamente fundamentadas.

Essa formatação estrutural, possibilita a construção de visualizações eficazes dos dados geográficos, Figura 24, subsidiando assim uma compreensão completa das informações apresentadas e tornado a interação com elas uma experiência mais intuitiva e fluida.

Figura 23 – Estrutura da visualização para os dados geográficos



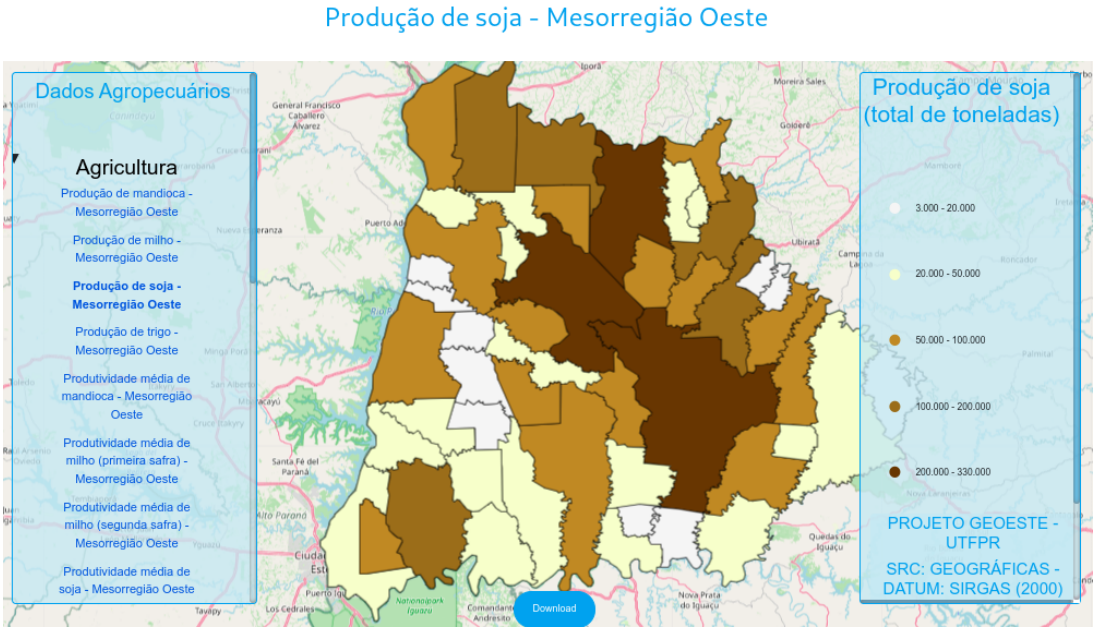
Fonte: Autoria própria (2023).

Embora a estrutura utilizada para a representação dos dados segue o padrão especificado anteriormente. É importante notar que algumas das características e propriedades desses mapas são distintas, o que resulta em algumas diferenças nas abordagens de representação geoespacial.

A primeira característica a ser discutida refere-se às delimitações geográficas

estabelecidas. Em visualizações de dados agropecuários, como a produção de soja, Figura 24, os limites administrativos dos municípios são utilizados como referência para estruturar os mapas, visto que a coleta dos dados realizada pelas fontes utilizadas neste trabalho, consideram os valores obtidos dentro dos limites de cada municipalidade.

Figura 24 – Produção de soja segmentada por município

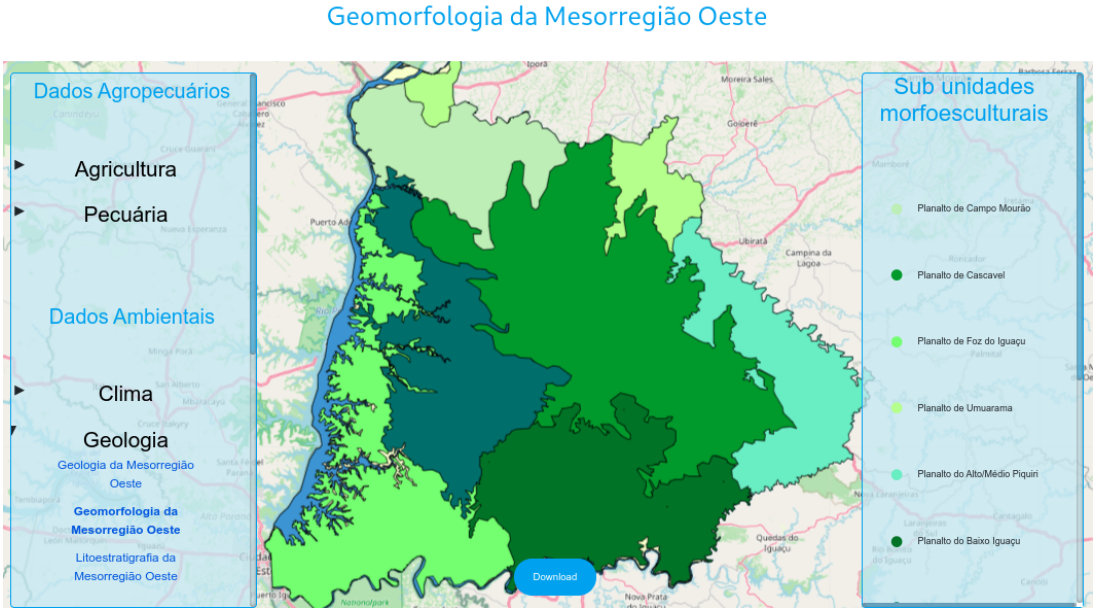


Fonte: Autoria própria (2023).

No entanto, há algumas situações onde a construção dos mapas interativos adota delimitações geográficas naturais, em detrimento das políticas, como é o caso das bacias hidrográficas e mapas de classificação de solo ou vegetação. Nessas situações, as fronteiras são definidas com base em características geofísicas e ambientais, como características físico-químicas do solo, da vegetação, ou em cursos d'água e divisores de águas. Isso proporciona uma representação mais condizente com a dinâmica do meio ambiente, tornando esses mapas especialmente valiosos para a compreensão das características físicas da região de interesse, Figura 25.

Outra distinção importante diz respeito à natureza da interação com os mapas. Dos apresentados até este ponto, todos são dinâmicos, o que permite a interatividade do usuário, e fornece a capacidade de explorar dados geográficos em tempo real e personalizar a visualização. No entanto, nem em todos os casos estão disponíveis as informações para estruturação dos dados e sua representação em ferramentas interativas. Nestes casos, faz-se o uso de mapas estáticos, sem qualquer forma de animação. No caso deste trabalho, alguns mapas estáticos são utilizados na represen-

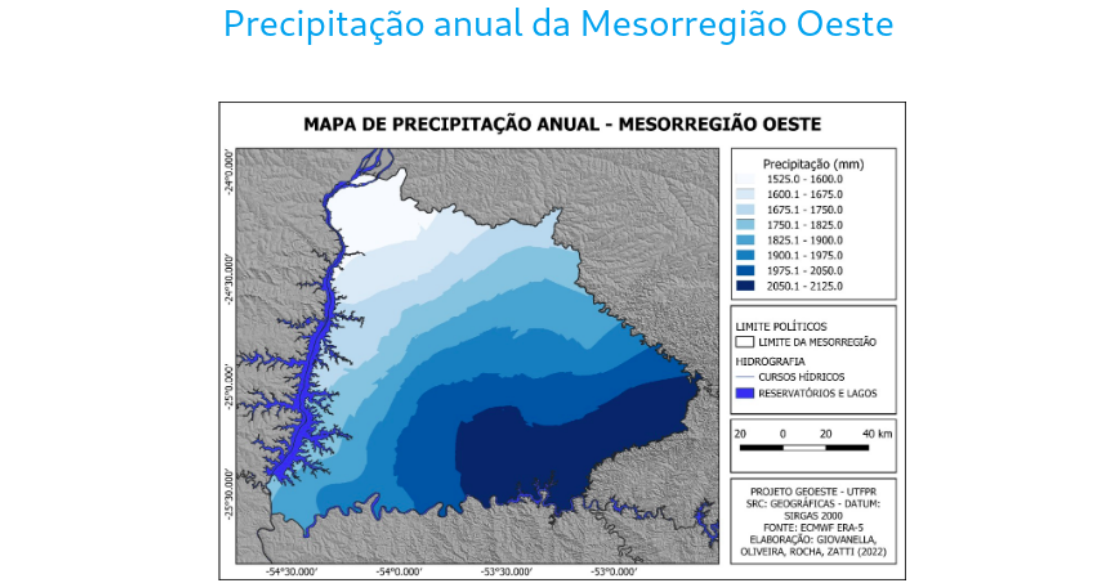
Figura 25 – Geomorfologia da Mesorregião Oeste



Fonte: Autoria própria (2023).

tação de informações climáticas específicas, como a precipitação anual na região de interesse, Figura 26. Embora sem a capacidade de interação, estas visualizações são particularmente eficazes na comunicação de padrões climáticos e podem servir como referências sólidas para análises de longo prazo.

Figura 26 – Precipitação anual da Mesorregião Oeste



Fonte: Autoria própria (2023).

Em resumo, as visualizações de dados geográficos possuem abordagens que refletem a complexidade das informações geográficas e ambientais. A escolha de delimitações geográficas e o tipo de animação, ou a ausência dela, foram decisões es-

senciais na criação de mapas que atendam às necessidades de análise e compreensão de diferentes aspectos dos dados disponíveis.

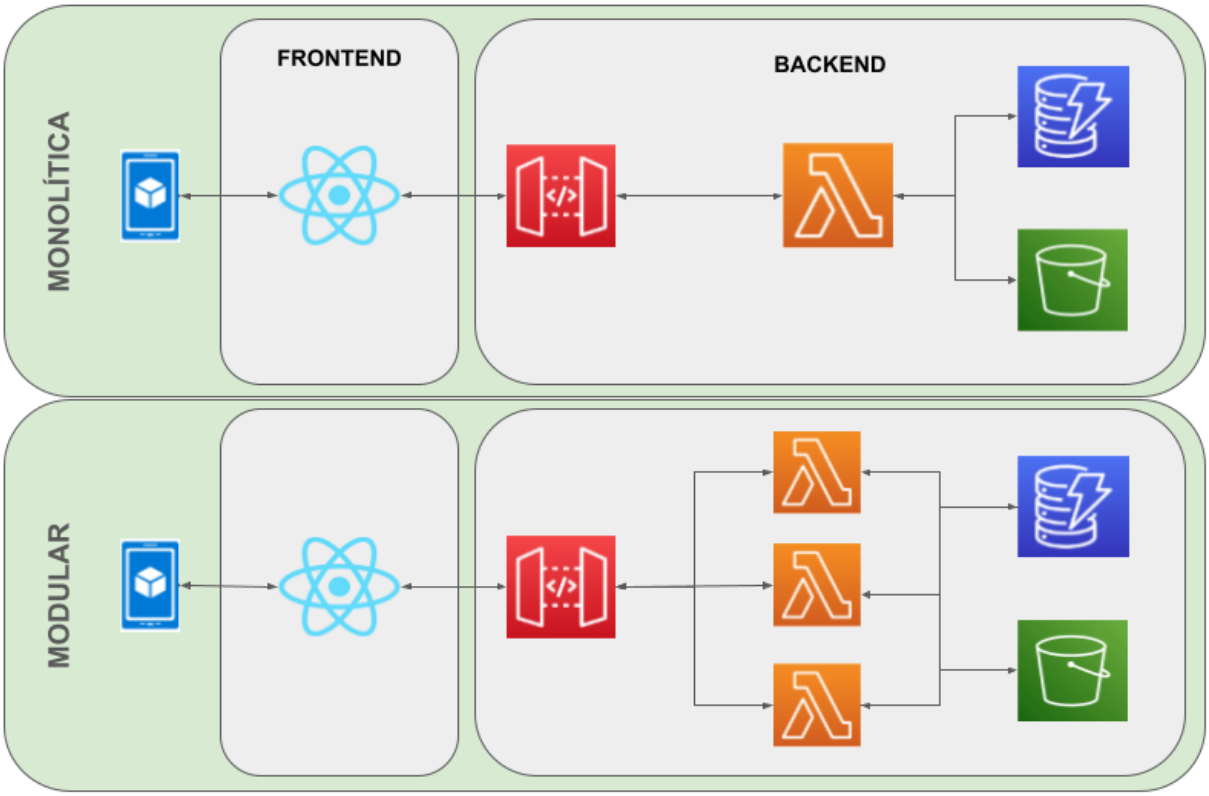
Findada a apresentação das visualizações de dados e construção destas no *frontend* da aplicação, a próxima etapa da seção de resultados, se concentrará na análise do desempenho da arquitetura *serverless* em fornecer essas visualizações, permitindo uma compreensão abrangente de sua eficácia e dos benefícios que essa abordagem traz para a entrega de informações geográficas interativas.

4.2 Comparação do desempenho da arquitetura proposta

Neste trabalho, objetivou-se desenvolver uma arquitetura *serverless* que possibilite a manipulação e visualização de dados geográficos complexos, provenientes de diferentes fontes e em formatos heterogêneos. Com a implementação da arquitetura, busca-se realizar um processamento eficiente, permitindo a disponibilização dos dados para a camada apresentação intuitiva dos dados para o usuário final, visando proporcionar benefícios significativos no processo de tomada de decisão.

De modo a avaliar o desempenho da implementação proposta, que adota uma abordagem modular com funções *lambda* separadas para diferentes funcionalidades, comparou-se esta com uma implementação monolítica, com todo o código fonte estando presente em um único ambiente de processamento composto por apenas uma função *lambda*, Figura 27. Esta última, implementada usando o *framework* Python Flask, banco de dados PostgreSQL e construída conforme o padrão *Data Access Object* (DAO). Em ambas implementações, foram coletados dados quantitativos relacionados ao tempo de processamento e à utilização de recursos de memória e o custo registrado pela AWS, o que permitirá identificar vantagens e desvantagens em termos de escalabilidade, eficiência e custos operacionais. De modo a garantir a equidade da coleta de dados, foram realizadas requisições simultâneas para ambas implementações, as quais deveriam resgatar as informações nos serviços de armazenamento, adequar ao formato especificado na sua modelagem e retornar estes como resposta. É importante destacar que foram realizadas requisições visando o retorno de dados referentes a dois conjuntos de dados distintos, denominados CI e CII, cada qual com aproximadamente 1MB de tamanho.

Figura 27 – Implementação monolítica e modular



Fonte: Autoria própria (2023).

Para análise de dados fez-se uso de algumas métricas estatísticas para compreender a distribuição e a variação dos dados. A primeira é a média, que representa a tendência central dos dados, e é obtida pelo quociente do somatório dos valores e o número de observações. A segunda métrica utilizada foi a mediana, que por sua vez representa o valor do meio em um conjunto de dados ordenados, diferenciando-se da média, pois não é afetada por valores extremos e fornece uma compreensão mais clara da distribuição dos dados. Outra medida de distribuição importante é o desvio padrão, pois este quantifica o grau de dispersão dos valores observados em relação à média. Por fim, utilizou-se um teste de comparação de médias para determinar se há evidências matemáticas de diferenças significativas entre entre as observações.

4.2.1 Tempo de processamento

Comparar o tempo de processamento entre as diferentes implementações é importante para avaliar o desempenho e a eficiência, pois esta variável impacta diretamente a capacidade de resposta de um sistema e a experiência do usuário. Além disso, o tempo de processamento está linearmente relacionado aos custos operacionais,

Tabela 4 – Estatística do tempo de processamento

	Conjunto de dados I - CI			Conjunto de dados II - CII		
	Média (ms)	Mediana (ms)	Desvio Padrão (ms)	Média (ms)	Mediana (ms)	Desvio Padrão (ms)
Modular	531,6[a]	539,8	122,3	490,4[c]	504,9	123,5
Monolítica	5258,0[b]	2880,6	6637,4	9060,2[d]	4448,3	11820,2

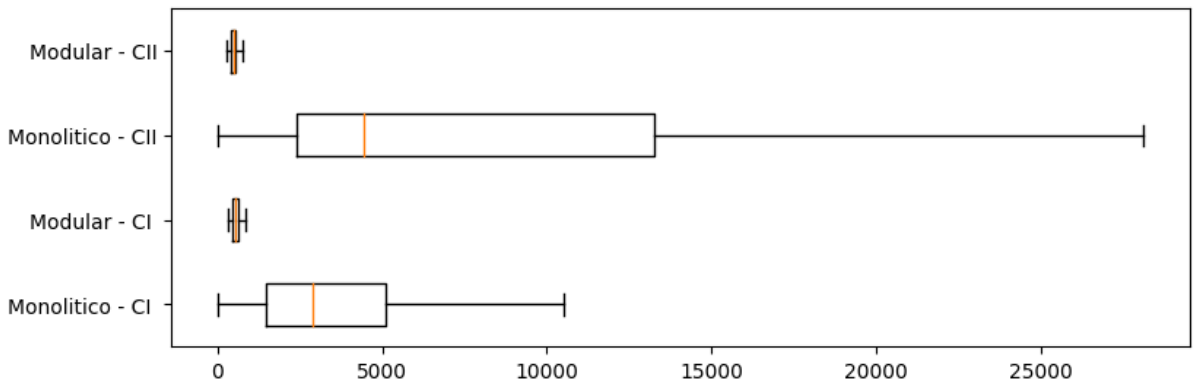
Nota: Letras diferentes indicam grupos estatisticamente distintos.

Fonte: Autoria própria (2023)

uma vez que a AWS quantifica os custos com base no tempo de execução, como será visto em uma subseção posterior.

Na Tabela 4 são apresentados os dados relativos ao tempo de processamento das requisições para os diferentes conjuntos de dados. Observa-se que a implementação monolítica apresenta valores mais elevados de média, mediana e desvio padrão. Para verificar se há diferença estatisticamente significativa entre as implementações, aplicou-se o teste de Mann-Whitney com um nível de significância α = 5%, visto que as amostras não são pareadas e não possuem distribuição normal. Para ambos conjuntos de dados, o valor obtido de *p-value* foi menor que o valor estabelecido de significância (*p-value* < α), portanto, pode-se afirmar, com 95% de confiança, que a implementação modular resulta em um tempo médio de processamento menor para os conjuntos de dados testados.

Figura 28 – Tempo de processamento em milissegundos



Fonte: Autoria própria (2023).

Na Figura 28 apresenta-se de modo resumido e visual a distribuição dos dados. Observa-se que a abordagem modular apresenta uma distribuição mais agrupada dos valores de tempo de processamento. Destaca-se que a não inclusão dos *outliers* no gráfico visou simplificar a visualização e focar nas características principais dos dados e oferecendo uma representação mais clara e concisa dos dados.

4.2.2 Uso de memória

Assim como ocorre com o tempo de processamento, comparar a memória consumida entre as diferentes implementações é importante para avaliar o desempenho da aplicação, pois esta variável esta relacionada aos custos operacionais. É importante destacar que nos serviços de computação em nuvem da AWS, os recursos de processamento são diretamente relacionados ao quanto de memória é alocada, assim para otimizar o uso das funções *lambda* e não gerar custos desnecessários, deve-se escolher um valor com base nas necessidades específicas da função.

Na Tabela 5 são apresentados os dados relativos ao uso de memória pela aplicação no processamento das requisições para os diferentes conjuntos de dados. Observa-se que a implementação monolítica apresenta valores mais elevados de média, mediana e desvio padrão. Para verificar se há diferença estatisticamente significativa entre as implementações, aplicou-se o teste de Mann-Whitney com um nível de significância α = 5%, visto que as amostras não são pareadas e não possuem distribuição normal. Para ambos conjuntos de dados, o valor obtido de *p-value* foi menor que o valor estabelecido de significância (*p-value* < α), portanto, pode-se afirmar, com 95% de confiança, que a implementação modular resulta em um tempo médio de processamento menor para os conjuntos de dados testados.

Tabela 5 – Estatística do uso de memória

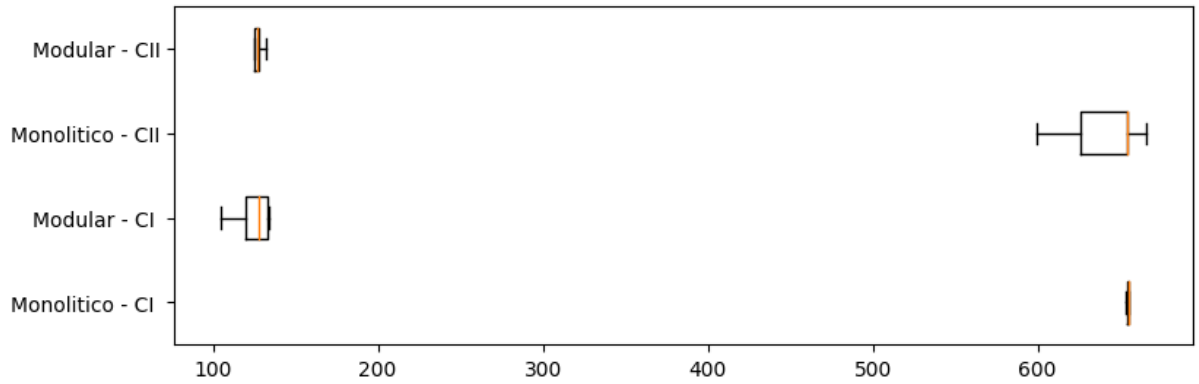
	Conjunto de dados I - CI			**Conjunto de dados II - CII**		
	Média (MB)	Mediana (MB)	Desvio Padrão (MB)	Média (MB)	Mediana (MB)	Desvio Padrão (MB)
Modular	125[a]	128	30,6	125,5[a]	127	7,3
Monolítica	650,4[b]	655	28,8	566,6[c]	654	173,9

Nota: Letras diferentes indicam grupos estatisticamente distintos.

Fonte: Autoria própria (2023)

Na Figura 29 apresenta-se de modo resumido e visual a distribuição dos dados. Observa-se que a abordagem modular apresenta uma distribuição de dados mais concentrada na faixa dos 100 aos 150 MB, enquanto a implementação monolítica apresenta os dados mais deslocada à direita, em valores acima de 600 MB. Destaca-se que a não inclusão dos *outliers* no gráfico visou simplificar a visualização e focar nas características principais dos dados e oferecendo uma representação mais clara e concisa dos dados.

Figura 29 – Uso de memória pela aplicação em MB



Fonte: Autoria própria (2023).

4.2.3 GB-segundo cobrado

O GB-segundo é uma métrica de custo utilizada pela AWS para calcular os custos de computação em serviços como AWS Lambda, e equivale ao produto entre a memória utilizada pela aplicação e o tempo de processamento. A depender da arquitetura computacional alocada à aplicação e a região desta, os custos por GB-segundo pode sofrer alterações, mas é uma métrica transparente e de fácil entendimento para os usuários entenderem e controlarem seus custos na AWS.

Na Tabela 6 são apresentados os dados relativos ao GB-segundo registrados para o processamento dos diferentes conjuntos de dados. Observa-se que a implementação monolítica apresenta valores mais elevados de média, mediana e desvio padrão. Para verificar se há diferença estatisticamente significativa entre as implementações, aplicou-se o teste de Mann-Whitney com um nível de significância α = 5%, visto que as amostras não são pareadas e não possuem distribuição normal. Para ambos conjuntos de dados, o valor obtido de *p-value* foi menor que o valor estabelecido de significância (*p-value* < α), portanto, pode-se afirmar, com 95% de confiança, que a implementação modular resulta em um GB-segundo registrado menor para os conjuntos de dados testados.

Tabela 6 – Estatística do GB-segundo

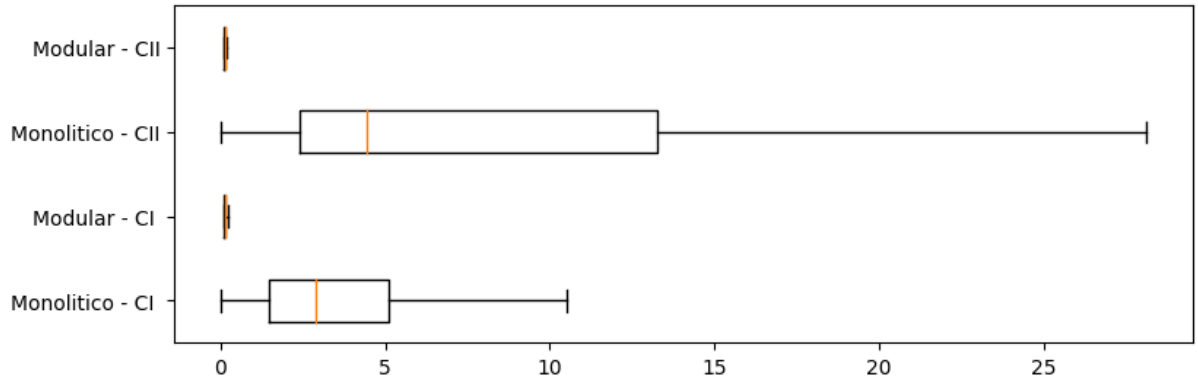
	Conjunto de dados I - CI			**Conjunto de dados II - CII**		
	Média	Mediana	Desvio Padrão	Média	Mediana	Desvio Padrão
Modular	0,133[a]	0,135	0,031	0,123[c]	0,126	0,031
Monolítica	5,259[b]	2,881	6,637	9,061[d]	4,449	11,820

Nota: Letras diferentes indicam grupos estatisticamente distintos.

Fonte: Autoria própria (2023)

Na Figura 30 apresenta-se de modo resumido e visual a distribuição dos dados. Observa-se que a abordagem modular apresenta uma variância nos valores registrados, com estes concentrados próximo ao valor de 1 GB-segundo. Destaca-se que a não inclusão dos *outliers* no gráfico visou simplificar a visualização e focar nas características principais dos dados e oferecendo uma representação mais clara e concisa dos dados.

Figura 30 – Registro do valor de GB-segundo



Fonte: Autoria própria (2023).

4.3 Comparação dos valores médios

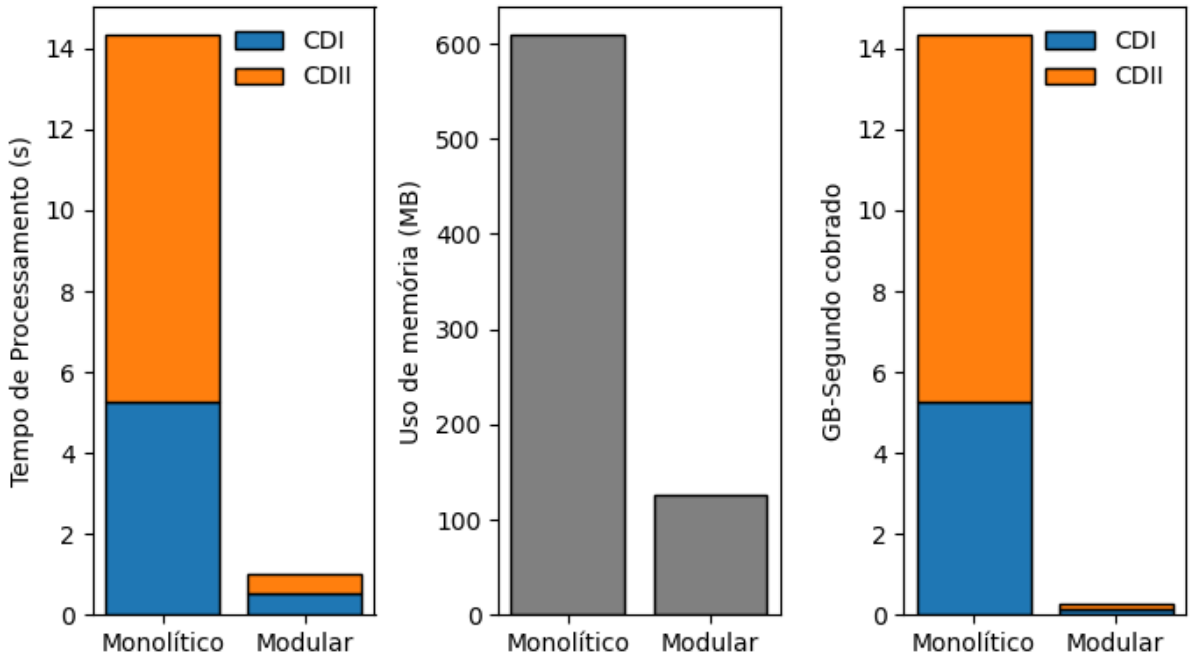
Buscando sintetizar os dados de desempenho em relação às diferentes métricas, optou-se por incluir os valores médios para o tempo de processamento, consumo de memória e GB-segundos cobrados, levando em consideração o processamento sequencial dos dois conjuntos de dados, CDI e CDII, para cada implementação, Figura 31.

Notavelmente, ao examinar os dados, destaca-se que a implementação modular apresenta valores significativamente menores para todas as métricas, o que sugere que essa implementação é vantajosa para otimizar o desempenho do aplicativo, proporcionando benefícios tangíveis em eficiência e economia de recursos computacionais e econômicos.

4.4 Discussão acerca das diferenças entre implementações

Ambas as implementações da aplicação em um ambiente *serverless* mostraram-se viáveis e passíveis de utilização, porém, por diversas razões, uma aplicação mo-

Figura 31 – Valores médios para processamento dos conjuntos de dados CDI e CDII



Fonte: Autoria própria (2023).

nolítica consolidada em uma única função *lambda* pode, apresentar maior tempo de execução em comparação com uma aplicação distribuída em várias funções. A primeira a ser discutida refere-se ao maior tamanho e complexidade, uma vez que todo o código a ser executado esta agrupado em uma única função, implicando em um tempo de carregamento e inicialização mais prolongado, e consequentemente, em uma execução mais demorada.

Outro fator que pode estar relacionado é o processamento de diversas partes da aplicação em paralelo, quando a aplicação esta dividida em várias funções *lambda*. Devido ao provisionamento e escalabilidade automática fornecida pelo serviço do AWS, diferentes etapas do processamento podem ser realizadas simultaneamente e de forma independente, o que reduz o tempo total de execução das operações. Em contrapartida, em uma aplicação monolítica, a execução não pode ser facilmente dividida em instâncias independentes e simultâneas, com funções especificas podendo causar gargalos e latências, podendo assim contribuir para um maior tempo de processamento.

Por fim, pode-se citar o aproveitamento do código, pois em aplicações divididas em funções *lambda*, pode-se reutilizar trechos de código já testados e otimizados, o que aliado as fatores acima citados, pode resultar em uma aplicação menos redundante e mais eficiente do ponto de vista do processamento. Neste sentido, se soma a melhor capacidade de gerir e definir o quantitativo de recursos adequados à cada função,

permitindo um controle mais granular e efetivo dos recursos computacionais, e evitando assim, que funções ou partes da aplicação se tornem um gargalo por falta de poder de processamento ou que seja destinado quantidades desproporcionalmente grande para a aplicação como um todo.

No entanto, é importante observar que a implementação monolítica em uma única função *lambda* proporciona uma abordagem simplificada para o processo de migração de aplicações já existentes, requerendo apenas alguns ajustes na configuração do ambiente, tornando a transição mais ágil e direta. Em contrapartida, a modularização da aplicação exige uma fatoração completa do código, o que implica na reestruturação e separação das diferentes partes do sistema, o que pode ser um processo mais complexo e demorado. A opção pela implementação monolítica pode ser uma estratégia interessante para a transição de sistemas, especialmente quando a agilidade na migração é um requisito crucial.

Tabela 7 – Comparação entre as implementações

Variável	Monolítica	Modular
Tempo de processamento	Maior	Menor
Consumo de memória	Maior	Menor
GB-Segundo cobrado	Maior	Menor
Tempo de Migração	Menor	Maior

Fonte: Autoria própria (2023)

Em resumo, Tabela 7, a utilização do serviço Lambda para o desenvolvimento de uma aplicação *serverless* mostra-se viável, sendo que a escolha pela modularização da aplicação em funções *lambda* independentes, pode oferecer benefícios significativos em termos de desempenho, escalabilidade e gerenciamento de recursos, levando a tempos de resposta menores, como é o caso da aplicação deste trabalho.

5 CONCLUSÃO

Por meio deste trabalho apresentou-se a construção de uma plataforma de visualização de dados geográficos, que fornece aos usuários uma forma intuitiva e interativa de acesso a informações geográficas sobre variados temas ambientais e agropecuárias. Esta característica possibilita a interpretação e exploração dos dados, identificação de padrões e extrapolar situações com a finalidade de melhor embasar as decisões que eventualmente fazem uso desta classe de informações.

A metodologia foi dividida em quatro etapas que contemplam diversos processos. Começou-se pela definição e levantamento das principais fontes de coleta dos dados e as variáveis que seriam apresentadas. Iniciou-se então a fase de extração dos dados, o processamento e o armazenamento no banco de dados. A terceira etapa concentrou-se então na construção da API, consistindo do seu planejamento, definição das ferramentas e serviços a serem utilizados e codificação da solução proposta. A quarta etapa concentrou-se no planejamento e elaboração da apresentação dos dados em forma de mapas interativos e sua integração com a API desenvolvida.

Com base na solução construída, aplicou-se testes de desempenho, comparando a solução modular proposta com uma implementação monolítica, coletando os dados disponibilizados pela AWS e por meio de testes estatísticos avaliando os resultados. Concluída as análises, observou-se evidencias significativas que a implementação proposta é vantajosa, apresentando tempos de processamento, discutindo-se os possíveis motivos.

Com base na solução construída, aplicou-se testes de desempenho, comparando a solução modular proposta com uma implementação monolítica, coletando os dados disponibilizados pela AWS e por meio de testes estatísticos avaliando os resultados. Após concluir as análises, evidenciou-se que a implementação proposta é vantajosa, apresentando consumo de memória e tempo de processamento médio equivalentes à 20% e 10% respectivamente, quando comparado à implementação monolítica. Com base nesses resultados, discutiu-se também os possíveis motivos do melhor desempenho e eficiência dessa implementação.

Mesmo com base nos resultados positivos, é crucial não considerar a solução como definitiva. Ajustes são necessários para lidar com as limitações de transferência e processamento de dados no serviço AWS Lambda. Além disso, é importante explorar

novas opções quanto à estrutura de armazenamento de dados, buscando formatos que minimizem a necessidade de processamento e transferência entre as camadas de processamento e visualização.

Por fim, a plataforma esta disponível para o público em geral, possibilitando seu uso, para os mais diferentes objetivos relacionados aos dados geográficos coletados, organizados e disponibilizados.

REFERÊNCIAS

ADZIC, Gojko; CHATLEY, Robert. Serverless computing: economic and architectural impact. *In*: 11TH JOINT MEETING ON FOUNDATIONS OF SOFTWARE ENGINEERING. **Proceedings [...]** [*S. l.: s. n.*], 2017. Disponível em: https://api.semanticscholar.org/CorpusID:13384955. Acesso em: 2 out. 2023.

ALVES, William Pereira. **Projetos de Sistemas Web: Conceitos, Estruturas, Criação de Banco de Dados e Ferramentas de Desenvolvimento**. E-book. Rio de Janeiro: Editora Saraiva, 2015. ISBN 9788536532462.

AMAZON WEB SERVICES. **Tutorial: Criação de uma API REST com o Amazon API Gateway e uma função do Lambda**. 2022. Disponível em: https://docs.aws.amazon.com/pt_br/lambda/latest/dg/services-apigateway-tutorial.html. Acesso em: 10 jun. 2023.

ANDERSON, Jennings *et al.* Incorporating context and location into social media analysis: A scalable, cloud-based approach for more powerful data science. *In*: 52ND HAWAII INTERNATIONAL CONFERENCE ON SYSTEM SCIENCES, Grand Wailea, Hawaii, USA. **Proceedings [...]** Grand Wailea, Hawaii, USA: [*s. n.*], 2019. P. 2274–2283.

ARIFIN, Oki; SUPRIYATNA, Agiska Ria. Sistem Informasi Geografis Untuk Pametaan Lahan Kakao Menggunakan Leaflet JS Dan GeoJson. **Jurnal Teknoinfo**, v. 17, n. 1, p. 364–371, 2023.

BEBORTTA, Sujit *et al.* Geospatial serverless computing: Architectures, tools and future directions. **ISPRS International Journal of Geo-Information**, MDPI, v. 9, n. 5, p. 311, 2020.

BREUNIG, Martin *et al.* Geospatial data management research: Progress and future directions. **ISPRS International Journal of Geo-Information**, MDPI, v. 9, n. 2, p. 95, 2020.

CÂMARA, Gilberto. **Representação computacional de dados geográficos**. 2005. Disponível em: http://mtc-m12.sid.inpe.br/col/sid.inpe.br/iris@1912/2005/07.01.19.33/doc/cap1.pdf. Acesso em: 21 mai. 2023.

CASTRO, Paul *et al.* The rise of serverless computing. **Communications of the ACM**, ACM New York, NY, USA, v. 62, n. 12, p. 44–54, 2019.

CELESTI, Antonio *et al.* An IoT cloud system for traffic monitoring and vehicular accidents prevention based on mobile sensor data processing. **IEEE Sensors Journal**, IEEE, v. 18, n. 12, p. 4795–4802, 2017.

CHANG, Kang-Tsung. **Introduction to geographic information systems**. [*S. l.*]: Mcgraw-hill Boston, 2008. v. 4.

FEDOSEJEV, Artemij. **React. js essentials**. Birmingham, UK: Packt Publishing Ltd, 2015.

FERREIRA, Arthur G. **Interface de programação de aplicações (API) e web services**. São Paulo: Editora Saraiva, 2021.

FONTANA, Neri Burato Bez; ZANELATTO, Alexandre Davi. Arquitetura serverless baseada em eventos para aplicações WEB utilizando a AWS. **Sistemas de Informação**, Florianópolis, 2019.

GACKENHEIMER, Cory. **Introduction to React**. New York, USA: Apress, 2015.

HORBIŃSKI, Tymoteusz; LOREK, Dariusz. The use of Leaflet and GeoJSON files for creating the interactive web map of the preindustrial state of the natural environment. **Journal of Spatial Science**, Taylor & Francis, v. 67, n. 1, p. 61–77, 2022.

HOSSEINI, Mehrshad; SAHRAGARD, Omid. **AWS Lambda Language Performance**. Monografia (Graduação), University of Gothenburg, Gothenburg, SW, 2019. Disponível em: https://gupea.ub.gu.se/bitstream/handle/2077/62454/gupea_2077_62454_1.pdf?sequence=1&isAllowed=y. Acesso em: 3 jul. 2023.

HYNEK, Jiří; KACHLÍK, Jakub; RUSŇÁK, Vít. Geovisto: A Toolkit for Generic Geospatial Data Visualization. *In*: 16TH INTERNATIONAL JOINT CONFERENCE ON COMPUTER VISION, IMAGING AND COMPUTER GRAPHICS THEORY AND APPLICATIONS. **Proceedings [...]** Setúbal, PT: SciTePress - Science e Technology Publications, 2021. (Volume 3: IVAPP), p. 101–111. ISBN 978-989-758-488-6. DOI: 10.5220/0010260401010111. Disponível em: https://www.fit.vut.cz/research/publication/12387. Acesso em: 23 ago. 2023.

IPARDES. **População Projetada (IPARDES) - Total - 2040**. 2023. Disponível em: https://www.ipardes.pr.gov.br/Pagina/Projecao-Populacional. Acesso em: 22 abr. 2023.

JONAS, Eric *et al.* Cloud programming simplified: A berkeley view on serverless computing. **arXiv preprint arXiv:1902.03383**, 2019. Disponível em: https://www2.eecs.berkeley.edu/Pubs/TechRpts/2019/EECS-2019-3.pdf. Acesso em: 26 jul. 2023.

KOSKELA, Jani. **Serverless backend migration from JavaScript to TypeScript**. Monografia (Graduação), Hame University of Applied Sciences, Riihimäki, FI, 2022. Disponível em: https://www.theseus.fi/bitstream/handle/10024/753309/Koskela_Jani.pdf?sequence=2. Acesso em: 3 jul. 2023.

LEE, Hyungro; SATYAM, Kumar; FOX, Geoffrey. Evaluation of production serverless computing environments. *In*: INTERNATIONAL CONFERENCE ON CLOUD COMPUTING (CLOUD). **Proceedings [...]** [*S. l.: s. n.*], 2018. IEEE, p. 442–450. DOI: 10.1109/CLOUD.2018.00062.

LI, Songnian *et al.* Geospatial big data handling theory and methods: A review and research challenges. **ISPRS journal of Photogrammetry and Remote Sensing**, Elsevier, v. 115, p. 119–133, 2016.

LONGLEY, Paul A *et al.* **Sistemas e ciência da informação geográfica**. Porto Alegre: Bookman Editora, 2009.

LÜ, Guonian *et al.* Reflections and speculations on the progress in Geographic Information Systems (GIS): a geographic perspective. **International journal of geographical information science**, Taylor & Francis, v. 33, n. 2, p. 346–367, 2019.

MACEACHREN, Alan M; KRAAK, Menno-Jan. Research challenges in geovisualization. **Cartography and geographic information science**, Taylor & Francis, v. 28, n. 1, p. 3–12, 2001.

MENEGUETTE, Arlete Aparecida Correia. Geovisualização: exercícios práticos em sala de aula. **Revista Brasileira de Cartografia**, v. 66, n. 4, p. 831–841, 2014.

METE, Muhammed Oguzhan; YOMRALIOGLU, Tahsin. Implementation of serverless cloud GIS platform for land valuation. **International Journal of Digital Earth**, Taylor & Francis, v. 14, n. 7, p. 836–850, 2021.

MOTA, Evandro Roberto. **Bancos de dados geográficos: uma abordagem orientada a grafos**. Monografia (Graduação), Universidade de Brasília, Brasília, 2016. Disponível em: https://repositorio.ufsc.br/handle/123456789/223655. Acesso em: 17 mai. 2023.

NETEK, Rostislav; BRUS, Jan; TOMECKA, Ondrej. Performance testing on marker clustering and heatmap visualization techniques: a comparative study on javascript mapping libraries. **ISPRS international journal of geo-information**, MDPI, v. 8, n. 8, p. 348, 2019.

NETEK, Rostislav; BURIAN, Tomas. WebGIS 2.0 Platform For Earthquakes Visualization. **International Multidisciplinary Scientific GeoConference: SGEM**, Surveying Geology & Mining Ecology Management (SGEM), v. 18, n. 2.2, p. 537–542, 2018.

NIU, Xingzhi *et al.* Leveraging Serverless Computing to Improve Performance for Sequence Comparison. *In*: ACM INTERNATIONAL CONFERENCE ON BIOINFORMATICS, COMPUTATIONAL BIOLOGY AND HEALTH INFORMATICS. **Proceedings [...]** Niagara Falls, NY, USA: Association for Computing Machinery, 2019. P. 683–687. ISBN 9781450366663. DOI: 10.1145/3307339.3343465. Disponível em: https://doi.org/10.1145/3307339.3343465. Acesso em: 21 jun. 2023.

PEREIRA, Gilberto Corso; SILVA, Barbara-Christine Nentwig. **Geoprocessamento e urbanismo**. Salvador, 2001. P. 97–137. Disponível em: https://repositorio.ufba.br/handle/ri/7961. Acesso em: 21 mai. 2023.

PETERSON, Michael P. **Mapping GeoRSS Feeds and the Shift from KML to GeoJSON**. [*S. l.*], 2016. Disponível em: https://cartogis.org/docs/proceedings/2016/Peterson.pdf. Acesso em: 20 abr. 2023.

QGIS. **QGIS - A Free and Open Source Geographic Information System**. 2021. Disponível em: https://www.qgis.org. Acesso em: 2 jul. 2023.

REDHAT. **What are Application Programming Interfaces?** 2023. Disponível em: https://www.redhat.com/pt-br/topics/api/what-are-application-programming-interfaces. Acesso em: 2 jul. 2023.

SAMPAIO FARIA, Nuno André de. **Suporte à edição cooperativa de Informação Geográfica em ambiente Web**. Dissertação (Mestrado), Universidade do Minho, Guimarães, PT, 2007. Disponível em: https://hdl.handle.net/1822/6351. Acesso em: 21 mai. 2023.

SANTACATTARINA, Ricardo. **Implementação de uma API em arquitetura serverless no ambiente da AWS para predição de batimetria em estuários através da aplicação de métodos de aprendizado de máquina em imagens multiespectrais de satélites**. Monografia (Graduação), Universidade Federal de Santa Catarina, Araranguá, 2021. Disponível em: https://repositorio.ufsc.br/handle/123456789/223655. Acesso em: 17 mai. 2023.

SANTOS, Ademir Pereira dos *et al.* Cidade digital: a construção de shapefiles públicos como ambiente de ensino, pesquisa e extensão. **Blucher Design Proceedings**, v. 3, n. 1, p. 272–277, 2016.

SCHOEDON, Alexander *et al.* Interactive Web-based Visualization for Accessibility Mapping of Transportation Networks. **EuroVis (Short Papers)**, Groningen, NL, p. 79–83, 2016. Disponível em: https://www.researchgate.net/publication/299470459. Acesso em: 12 jun. 2023.

SETA, Pramudhita Tunjung; HARTOMO, Kristoko Dwi. Mapping land suitability for sugar cane production using k-means algorithm with leaflets library to support food sovereignty in central java. **Khazanah Informatika: Jurnal Ilmu Komputer dan Informatika**, v. 6, n. 1, p. 15–25, 2020.

SHAFIEI, Hossein; KHONSARI, Ahmad; MOUSAVI, Payam. Serverless computing: a survey of opportunities, challenges, and applications. **ACM Computing Surveys**, ACM New York, NY, v. 54, 11s, p. 1–32, 2022.

SHARDA, Ramesh; DELEN, Dursun; TURBAN, Efraim. **Business Intelligence e Análise de Dados para Gestão do Negócio-4**. 4. ed. Porto Alegre: Bookman Editora, 2019. 584 p.

SHRESTHA, Sachin. **Comparing Programming Languages used in AWS Lambda for Serverless Architecture**. Monografia (Graduação), Metropolia University of

Applied Sciences, Helsinque, FI, 2019. Disponível em: https://urn.fi/URN:NBN:fi:amk-2019070117607. Acesso em: 30 jun. 2023.

SILVA SANTOS, Wellington José da; SANTOS JUNIOR, Antônio Carlos Pereira dos. **SIG e banco de dados relacionais e geográficos**: Gerando mapas dinâmicos para o gerenciamento hidrômetros em Cáceres-MT. Rio de Janeiro: Editora e-Publicar, 2021. v. 1.

STEIN, Ronei T. *et al.* **Geoprocessamento**. E-book. Porto Alegre, RS: Grupo A, 2021.

STEIN JUNIOR, Antonio Ricardo Medronha; SANTOS, Maicon dos. **Arquitetura REST API e desenvolvimento de uma aplicação Web Service**. Projetos e relatórios de estágios, 1(1), 1-59, 2019. Disponível em: https://raam.alcidesmaya.com.br/index.php/projetos/article/view/97. Acesso em: 12 ago. 2023.

TROMBETA, Letícia R. A. *et al.* **Geoprocessamento**. E-book. Porto Alegre, RS: Grupo A, 2019. ISBN 9786581492120.

WITTIG, Michael; WITTIG, Andreas. **Amazon web services in action**. 1. ed. São Paulo: Novatech Editora, 2016. 512 p.

ZHU, Junxiang *et al.* Integration of BIM and GIS: IFC geometry transformation to shapefile using enhanced open-source approach. **Automation in construction**, Elsevier, v. 106, p. 102859, 2019.