Universidade Federal do Maranhão
Centro de Ciências Exatas e Tecnologia - CCET
Bacharelado em Ciência da Computação

DANIEL DA SILVA LIMA

Visudata: um Sistema Web para Visualização de Dados de COVID-19 e Indicadores Socioeconômicos

São Luís - MA

2023

Daniel da Silva Lima

Visudata: um Sistema Web para Visualização de Dados de COVID19 e Indicadores Socioeconômicos

Monografia apresentada ao curso de Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisitos necessários para obtenção do grau de bacharel em Ciência da Computação. Orientador: Prof. Dr. Tiago Bonini Borchartt

São Luís - MA
2023

Ficha gerada por meio do SIGAA/Biblioteca com dados fornecidos pelo(a) autor(a).
Diretoria Integrada de Bibliotecas/UFMA

Lima, Daniel da Silva.
Visudata : sistema web para visualização de dados de covid-19 e indicadores socioeconômicos / Daniel da Silva Lima. - 2024.
65 f.

Orientador(a): Tiago Bonini Borchartt.
Monografia (Graduação) - Curso de Ciência da Computação, Universidade Federal do Maranhão, São Luís, 2024.

1. Aplicações Web. 2. Covid-19. 3. Índices Econômicos. 4. Visualização de Dados. I. Borchartt, Tiago Bonini. II. Título.

DANIEL DA SILVA LIMA

VISUDATA: UM SISTEMA WEB PARA VISUALIZAÇÃO DE DADOS DE COVID-19 E INDICADORES SOCIOECONÔMICOS

Monografia apresentada ao curso de Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisitos necessários para obtenção do grau de bacharel em Ciência da Computação.

Aprovado em ___/____/____

_____

Prof. Dr. Tiago Bonini Borchartt
UFMA– Orientador

_____

Prof. Dr. Darlan Bruno Pontes Quintanilha
UFMA

_____

Prof. Me. Bruno Roberto Silva de Moraes
UFMA

São Luís, 17 de Janeiro de 2024

Dedico este trabalho à minha mãe, que sempre me deu o exemplo de perseguir as conquistas intelectuais.

Agradecimentos

Agradeço ao destino e aos bons ventos da fortuna por terem me permitido chegar ao fim desta etapa.

Agradeço aos meus companheiros de jornada neste curso de Ciência da Computação. Foram tantos que não consigo citá-los pelos nomes, mas lendo este trecho acredito que saberão quem são.

À minha namorada Alana, que conheci graças ao curso, por me apoiar, ser paciente, ajudar quando preciso, tanto emocionalmente quanto com apoio técnico.

À minha família pela compreensão da ausência nestes últimos meses de dedicação a este trabalho.

Ao meu orientador o professor Bonini por, entre outras coisas, ter sido paciente durante esta árdua jornada de elaboração do TCC.

A primeira regra é manter o espírito tranquilo. A segunda é enfrentar as coisas de frente e tomá-las pelo que realmente são. (Marco Aurélio)

RESUMO

A Covid-19 mobilizou o mundo em esforços científicos para a sua contenção e eventual supressão. Nesse contexto, a quantidade de dados relativos à Covid-19 produzida no período de 2020 e 2021 foi bastante extensa. Esses dados são relevantes para o avanço científico, no entanto são difíceis de serem entendidos ou relacionados. Por isso viu-se a necessidade da visualização desses dados de forma a relacioná-los aos dados econômicos dos países objetivando a obtenção de insights. Isso foi feito por meio de uma aplicação Web e uma API feita em Spring que seguiram as melhores práticas de programação atuais e também à luz da ciência da visualização de dados que é uma área da ciência de dados. Apresentou-se as principais características desta aplicação e por fim concluiu-se que ela atende ao que propôs uma vez que suas visões possibilitam a visualização e o entendimento dos dados de forma holística.

Palavras chave: COVID-19, Aplicações WEB, Índices Econômicos, Visualização de Dados.

ABSTRACT

The Covid-19 mobilized the world in scientific efforts for its containment and eventual suppression. In this context, the amount of data related to Covid-19 produced in the period of 2020 and 2021 was quite extensive. These data are relevant for scientific advancement, however, they are challenging to understand or relate with other kinds of data. Therefore, there was a need to visualize this data in a way that connects it to the economic data of countries, aiming to obtain insights. This was done through a web application and an API built in Spring that followed current best programming practices and also in light of the science of data visualization, which is a discipline of data science. The main features of this application were presented, and, in conclusion, it was found that it meets its purpose since its views enable the visualization and understanding of data in a holistic way.

Keywords: COVID-19, WEB Applications, Economic Index, Data Visualization.

Lista de Figuras

SUMÁRIO

1 INTRODUÇÃO

A covid-19 foi uma doença que apavorou o mundo nos últimos anos. Ao ser considerada pandemia houve um esforço de todos os países para contê-la, gerando uma alavancagem no processo de pesquisa científica e tecnológica voltada para a área.

Nesse contexto, vimos muitos sites e portais de instituições públicas e privadas, principalmente da área da saúde, colocarem informações geradas na pandemia em forma de dashboards informativos, o que facilitou e muito o entendimento dos dados produzidos nesse período por parte de quem os acessou.

1.1 Justificativa

Nos últimos anos a quantidade de dados relacionados à covid-19 que foi produzida é enorme. Da mesma forma temos dados econômicos gerados pelos países e coletados por órgãos como a ONU e o Banco Mundial. Isso caracteriza uma ótima área de pesquisa para Ciência de Dados onde a mesma pode aplicar suas técnicas para obter *insights* valiosos. No entanto, apesar desta ser uma área promissora, são poucas as plataformas que exibem os resultados destas 'tabulações de dados', principalmente voltadas para a visualização de dados de covid-19 e índices econômicos.

Na bibliografia acerca de ciência da dados, o tópico de visualização de dados é de suma importância, e segundo os autores em dois momentos principais: a visualização de dados se mostra importantíssima; antes da aplicação das métricas de pesquisas de dados, para fazer sentido de dados brutos e de quantidade elevada, e prover insights que auxiliem o pesquisador a formular sua abordagem inicial em como trabalhar com esses dados. O segundo momento de extrema importância é no fim de sua aplicação, com o objetivo de mostrar esses dados para o público de forma sucinta, e que facilite o entendimento da informação.

Por esse motivo foi necessário desenvolver uma aplicação Web para visualização de dados relacionados. Esse sistema representa uma alternativa aos softwares privados de visualização de dados e por ser web se coloca nos moldes dos sistemas utilizados pelos mais variados órgãos no mundo. Além de mostrar as

boas práticas de desenvolvimento de software para web utilizando os padrões de projeto e princípios de código limpo ao mesmo tempo que ressalta a importância de softwares desenvolvidos ‘em casa’ como alternativa às soluções prontas que servem para ‘qualquer propósito’. Essa aplicação pegou dados de índices socioeconômicos de diversos países, dados esses fornecidos pelo site[1] do Banco Mundial, e dados sobre os casos de covid-19 também dos diversos países, esses dados por sua vez foram obtidos do site[2] da *Our World in Data*.

1.2 Objetivos

O objetivo geral deste trabalho é desenvolver um sistema web para visualização de dados de índices socioeconômicos e índices de covid-19 dos diversos países, que chamaremos de *visudata*. O trabalho também busca ilustrar por meio da literatura científica disponível, a importância da visualização de dados, tanto para a ciência quanto para os negócios. E por fim mostra como o sistema se coloca como uma solução que pode ser utilizada como está ou pode servir como modelo para outros sistemas.

Objetivos específicos:

- Demonstrar a importância da visualização de dados por meio de uma revisão na literatura científica;
- Destacar a importância da aplicação da visualização de dados nos estudos envolvendo covid-19;
- Apresentar o *visudata* e suas características estruturais, de desenvolvimento e o resultado final do mesmo:
    - Mostrar o software desenvolvido pelo autor como uma alternativa relevante à visualização de dados de covid-19 e índices econômicos com o objetivo de obter insights científicos e de negócio.

[1] Disponível em: https://data.worldbank.org/
[2] Disponível em: https://ourworldindata.org/explorers/coronavirus-data-explorer

2 Revisão de literatura/referencial teórico

2.1 Visualização de Dados

A problemática que envolve a visualização de dados pode ser datada dos primórdios da sociedade quando o ser humano começou a produzir e principalmente a compartilhar informações. De acordo com Santos e Flores (2018) a necessidade de externalizar informações para as gerações futuras era expressada de acordo com o desenvolvimento tecnológico da sociedade em questão e também da época em que tal sociedade existiu. Tendo influências tanto tecnológicas como culturais. A exemplo de meios de visualizar dados podemos citar entalhes, pinturas em paredes, papiro, pergaminhos, papel e o meio digital.

2.1.1 Dados, Informação e Conhecimento

Para o entendimento deste trabalho é necessário definir dados, informação e conhecimento, assim como deixar claro a diferença entre esses conceitos.

Os dados são considerados como os blocos indivisíveis da informação. Eles também podem ser vistos como a forma bruta e crua de uma informação. Davenport e Prusak (1998) não acreditam que definir o que é dado seja uma tarefa fácil, para os autores até mesmo tentar a clássica diferenciação entre dados, informação e conhecimento resulta em um entendimento não tão preciso. Mesmo assim, os autores fornecem uma definição sobre dados a partir de suas características. Para eles os dados são simples observações sobre o estado do mundo, são facilmente estruturados e facilmente obtidos por máquinas, são frequentemente quantificados e facilmente transferíveis. Outros autores, notadamente aqueles das áreas de exatas, dão ênfase na natureza quantificável do dado ao defini-lo. Nas palavras de Setzer(1999):

> *“Definimos dado como uma sequência de símbolos quantificados ou quantificáveis. Portanto, um texto é um dado. De fato, as letras são símbolos quantificados, já que o alfabeto por si só constitui uma base numérica. Também são dados imagens, sons e animação, pois todos podem ser quantificados a ponto de alguém que entra em contato com eles ter eventualmente dificuldade*

> *de distinguir a sua reprodução, a partir da representação quantificada."*

Informação é aquilo que tem significado, tipicamente entendido em relação a um contexto específico ou um problema. Representa padrões ou estruturas de dados e usualmente comunica ou descreve alguma coisa. Enquanto que os dados são o material que compõe a informação, a informação representa significado ou padrões encontrados nos dados. Para Setzer (1999):

> "*Informação é uma abstração informal (isto é, não pode ser formalizada através de uma teoria lógica ou matemática), que representa algo significativo para alguém através de textos, imagens, sons ou animação."*

O mesmo conjunto de dados é capaz de gerar diferentes tipos de informação dependendo do contexto, necessidade ou propósito do usuário.

Já o conhecimento é um entendimento abrangente ou familiaridade com um tópico, assunto ou área de expertise. Conhecimento vai além da mera coleta ou interpretação da informação e requer uma visão abrangente, entendimento e maestria sobre um assunto. Geralmente envolve uma habilidade em aplicar informação em situações específicas ou fazer conexões entre essas informações. Para Setzer (1999), conhecimento é uma abstração interior, pessoal, de alguma coisa que foi experimentada por alguém. Já Davenport e Prusak (1998) falam que o conhecimento é uma informação valiosa da mente humana que inclui reflexão, síntese, contexto e é de difícil estruturação, difícil captura em máquinas e é frequentemente tácito, ou seja, de difícil transferência.

2.1.2 Visão

Visão é a habilidade de interpretar, entender e tomar decisões baseadas em informações visuais. Na visualização de dados a visão tem o papel crucial no design e interpretação das visualizações que são usadas para representar os dados e transmitir a informação para o usuário. Já a visualização de dados é uma disciplina importante na ciência de dados, nas palavras de Corrêa (2019):

> *“A visualização de dados é vista por muitas disciplinas como um equivalente moderno da comunicação visual. Trata-se da criação e estudo da representação visual dos dados. [...] A visualização gráfica de dados constitui uma disciplina própria dentro da ciência de dados. [...] Desse modo, situamos a visualização de dados e infografia como áreas de estudo. É nesse contexto que analisamos o percurso da visualização de dados como uma prática da ciência de dados. Em seu aspecto funcional, se trata de um recurso para garantir maior compreensão de pesquisas e análises empíricas por meio da compilação de um conjunto de soluções infográficas, ao passo que busca identificar aspectos que qualificam uma comunicação visual eficiente.”*

É necessário deixar claro a diferença entre esses dois conceitos uma vez que a visualização de dados também pode ser o processo de arranjar os dados de forma visual para que o usuário obtenha informação. Ribeiro (2009a, p. 6) nos diz que a visualização de dados é o processo que utiliza Tecnologias Computacionais para transformar dados abstratos em modelos visuais, ou seja, é a tradução criativa dos dados que em sua forma original são incapazes de carregar qualquer interpretação profunda em representações visuais reveladoras.

Portanto, até aqui temos dois significados para visualização de dados e um para a visão. Essa diferenciação é relevante pois para entendermos um sistema de múltiplas visões temos que saber exatamente no que consiste uma visão. Um sistema de múltiplas visões usa duas ou mais visões diferentes para auxiliar na investigação de dada entidade conceitual (BALDONADO, WOODRUFF, KUCHINSKY, 2000 - tradução nossa). Pode-se definir uma única visão de uma entidade conceitual como um conjunto de dados mais uma especificação em como mostrar aqueles dados visualmente. Da mesma forma pode-se dizer que visões são distintas se elas permitem que o usuário aprenda sobre diferentes aspectos da entidade conceitual ao apresentarem informações diferentes ou ao enfatizarem aspectos diferentes da mesma informação (BALDONADO et al, 2000 - tradução nossa).

2.1.3 Visualização de dados ao longo do tempo

Corrêa (2019) nos conta que a visualização de dados existe desde o tempo da pré-história. Os povos antigos, embora não soubessem a escrita na época, foram capazes de produzir obras de arte autênticas que consistiam em representações gráficas feitas nas paredes das cavernas. As antigas civilizações grega e egípcia tinham um profundo conhecimento sobre o espaço, eles criaram mapas das constelações e os movimentos do sol.

Os mapas são um dos exemplos mais óbvios de visualização de dados. Por exemplo, na época do Império Romano, os mapas eram usados para entender e planejar o movimento dos exércitos. Estratégias competitivas nasceram desses mapas. Uma das contribuições mais importantes desses impérios. Seguindo adiante no tempo podemos citar as árvores de Ramon Llull em meados do século XII, ele foi um filósofo alquimista que viveu entre a iluminação Cristã e a tecnologia árabe. Seus diagramas mecânicos de conhecimento serviram como inspiração para o desenvolvimento de sua lógica simbólica, também conhecida como o antigo testamento da computação. Nestes diagramas é onde ele usa pela primeira vez na história a estrutura orgânica de uma árvore para representar a origem e o desenvolvimento das grandes áreas do conhecimento.

No século XVIII ocorreram muitos marcos importantes na história Universal. Entre eles, podemos citar a Revolução Francesa, a colonização da América e o começo da revolução industrial. Depois da primeira metade do século, um teólogo inglês começou a trabalhar em uma das mais impressionantes visualizações de dados de todos os tempos: a carta de biografia de Joseph Priestley, um cronograma que organiza vários séculos de História: marca o começo e a queda dos impérios, as figuras mais proeminentes e os momentos mais relevantes. Pela primeira vez na história, foram utilizadas linhas para marcar a duração da vida das pessoas. Priestley revolucionou o modo como a história foi registrada.

Na história mais recente podemos citar a visualização de dados sendo utilizado nas tragédias humanas, como surtos de doença e a guerra. Temos o mapa de Charles Minard que mostra a evolução da perda de homens na marcha napoleônica contra Moscou e o mapa da cólera de Jon Snow que mostra sobrepujado em um mapa de Londres as áreas onde houveram mais casos de cólera, escurecidas no mapa de acordo com sua intensidade. Ao longo da década

de 1990 e início dos anos 2000, a mídia e as empresas incorporaram os infográficos impressos como uma maneira mais esclarecedora de representar números. A partir daí as maneiras pelas quais podemos mostrar dados avançaram exponencialmente: de infográficos e gráficos em rede a centros de comando e salas de guerra. A internet e demais mídias digitais permitiram que informação e ideias fluíssem livremente, facilitando o aprendizado de outras pessoas.

2.1.4 Importância da visualização de dados na modernidade

Nos tempos modernos, a visualização de dados se mostra uma ferramenta importantíssima. Como nos dizem Rodrigues e Dias (2017) visualizações de dados são uma ferramenta essencial para a ciência, pois nos permitem identificar padrões e tendências que podem levar a novos *insights*. O aumento da quantidade de dados disponíveis, aliado ao desenvolvimento de novas tecnologias, tornou a visualização de dados algo indispensável para a ciência e os negócios. Ela ajuda a tornar os dados mais compreensíveis e a identificar padrões e tendências que podem passar despercebidos quando os dados são apresentados em forma de tabelas ou planilhas. Braga, Alves e Leite (2021) também enfatizam a importância da visualização de dados para a ciência e os negócios. Na ciência a visualização de dados é utilizada: na análise de dados científicos onde ela mostra dados de experimentos, observações e simulações; na comunicação científica onde ela pode ser usada para comunicar resultados científicos de forma clara e concisa; na educação científica onde pode ser usada para ensinar ciência de forma mais eficaz. Nos negócios a visualização de dados é utilizada: na análise de dados empresariais onde é possível visualizar dados de marketing, vendas, finanças e outros departamentos; na tomada de decisão onde a visualização é utilizada para apoiar a tomada de decisões estratégicas e operacionais; na comunicação empresarial onde é usada para comunicar informações empresariais de forma clara e concisa. A visualização de dados tem sido muito utilizada para identificar oportunidades de mercado, para melhorar a eficiência das operações e para reduzir custos. Uma outra área moderna onde a visualização de dados se mostra essencial é nas aplicações de *big data*. O *Big Data* consiste no processamento de uma quantidade enorme de dados, geralmente utilizando técnicas de inteligência artificial e *machine learning*, com o propósito de gerar informações e achar padrões. Para Corrêa (2019) os

dados são inúteis se não é possível visualizar a informação, para ele o desafio das empresas é extrair o valor dos dados o que torna necessário ter as melhores ferramentas de visualização, ele ainda acrescenta que:

> *"A visualização de dados crescerá a importância no curto prazo. Se trata de um termo geral que descreve qualquer esforço para ajudar as pessoas a entender a importância dos dados, colocando-os em um contexto visual. Padrões, tendências e correlações que podem passar despercebidas em dados baseados em texto, podem ser mais facilmente expostos e reconhecidos com software de visualização de dados. Se falamos de Big Data, a visualização de dados é crucial para direcionar a tomada de decisões em alto nível com maior sucesso."*

Os tipos de visualizações mais utilizadas com *Big Data* são: as visualizações interativas, as visualizações multidimensionais, e as visualizações distribuídas. As visualizações interativas permitem que os usuários explorem os dados de forma interativa, usando os recursos como filtros, *zoom* e legendas. As visualizações multidimensionais permitem que os usuários visualizem dados em mais de duas dimensões, o que pode ser bem útil para visualizar dados complexos. Já as visualizações distribuídas são aquelas que são divididas em vários computadores ou monitores, elas são utilizadas quando é necessário para uma apresentação onde um público é muito grande ou um tipo de apresentação com dados bastante complexos.

2.2 COVID 19

A COVID-19 é uma doença infecciosa causada pelo coronavírus SARS-CoV2. O vírus é transmitido de pessoa para pessoa através de gotículas respiratórias produzidas quando uma pessoa infectada tosse, espirra ou fala. As gotículas podem se depositar em superfícies e objetos, e as pessoas podem ser infectadas ao tocar nessas superfícies ou objetos e depois tocar os olhos, o nariz ou a boca. De acordo com os registros a COVID-19 começou em dezembro de 2019 na cidade de *Wuhan*, na China. O vírus se espalhou rapidamente para outros países e continentes, e em março de 2020 foi declarada uma pandemia pela Organização

Mundial de Saúde (OMS) (Souza, 2023). O surto da doença afetou os países de várias maneiras. Ela causou milhões de mortes e infecções em todo o mundo. Em termos econômicos a pandemia levou a uma recessão global, com fechamento de empresas, perda de empregos e queda do PIB. A Organização para a Cooperação e Desenvolvimento Econômico - OCDE (*Organisation for Economic Co-operation and Development* - OECD) alertou sobre o impacto negativo que o Coronavírus trouxe ao mundo em relação ao sistema econômico internacional. Em março de 2020, A OCDE divulgou um relatório que estimou que a crise da COVID-19 foi capaz de reduzir pela metade o crescimento da economia mundial até aquele ano, sendo que os efeitos adversos poderiam continuar até 2022 (MAGAZZINO; MELE; MORELLI, 2021). Em termos sociais, a pandemia provocou mudanças significativas no comportamento das pessoas, com o aumento do isolamento social, do trabalho remoto e do uso de máscaras.

Os sintomas causados podem variar de leves a graves. Entre os mais comuns incluem febre, tosse seca, cansaço e dificuldade para respirar. Em alguns casos, a doença pode levar a pneumonia, insuficiência respiratória e morte. Como também causar efeitos a longo prazo, que podem incluir problemas respiratórios, cardíacos, neurológicos e psicológicos. A pandemia causou uma recessão global, com queda do PIB em todos os continentes. Em 2020, o PIB global caiu 3,5%, o maior declínio desde a Segunda Guerra Mundial. Ela também levou ao fechamento de empresas, perda de empregos e redução do consumo. Isso teve um impacto negativo na economia de todos os países, ricos e pobres. Como consequência houve também um aumento da ansiedade, da depressão e do estresse entre as pessoas. Além disso, a pandemia destacou as desigualdades sociais e econômicas existentes no mundo. Os países pobres foram mais afetados pela pandemia, e as populações mais vulneráveis foram as mais impactadas.

Desde o início da pandemia de covid-19, em dezembro de 2019, uma quantidade enorme de dados tem sido coletada sobre a doença. Esses dados incluem informações sobre casos, mortes, hospitalizações, testes, vacinas, tratamentos e etc. Os dados produzidos podem ser divididos em quatro categorias principais: dados epidemiológicos, dados clínicos, dados comportamentais, e dados de laboratório. Os dados epidemiológicos, como dito anteriormente, trazem informações sobre casos, mortes, hospitalizações, testes e outros indicadores de

saúde pública. Eles são coletados por governos, organizações de saúde e instituições de pesquisa. A categoria de dados clínicos traz informações sobre os sintomas, o curso da doença e os tratamentos recebidos por pessoas com COVID-19. Eles foram coletados principalmente por hospitais, clínicas e outros estabelecimentos de saúde. Os dados comportamentais trazem informações sobre o comportamento das pessoas, como a adesão às medidas de saúde pública, o uso de máscaras e o consumo de álcool em gel. Eles são coletados por pesquisas, inquéritos e outros métodos. Já os dados de laboratório incluem informações sobre o vírus SARS-CoV-2, como sua sequência genética, sua capacidade de mutação e sua resposta aos medicamentos. Eles são coletados por laboratórios de pesquisa e instituições de saúde. Os dados coletados na pandemia são essenciais para a compreensão da doença e para o desenvolvimento de políticas e intervenções eficazes. Esses dados podem ser utilizados para monitorar a evolução da pandemia, investigar a doença, desenvolver vacinas e tratamentos e educar a população. Existem várias maneiras de explorar esses dados, por exemplo, os dados epidemiológicos podem ser usados para criar mapas e gráficos que mostram a distribuição da doença ao longo do tempo e do espaço. Os dados clínicos podem ser usados para realizar análises estatísticas que identificam fatores de risco e prognósticos. Os dados comportamentais podem ser usados para realizar pesquisas que identificam as atitudes e crenças das pessoas sobre a doença. Os dados de laboratório podem ser usados para realizar simulações que ajudam a entender a propagação do vírus. Os dados sobre covid-19 são um tesouro de informações que podem ser usados para entender a doença ou para desenvolver políticas e intervenções eficazes. A coleta e principalmente a análise desses dados são essenciais para combater e entender a pandemia e para proteger a saúde da população.

2.3 Índices Econômicos

Os índices econômicos são medidas da situação econômica e da performance de uma nação. Através deles é possível prever a saúde e a direção de uma economia, geralmente um determinado índice foca em áreas ou problemas específicos. Mankiw (1998) nos diz que os índices econômicos são usados para acompanhar o desempenho da economia, como o crescimento do PIB, a inflação e a

taxa de desemprego. Neste trabalho conceituaremos alguns índices econômicos utilizados na construção das visualizações da nossa aplicação.

2.3.1 Índice de GINI

O índice de Gini, criado pelo matemático italiano Conrado Gini, é um instrumento para medir o grau de concentração de renda em determinado grupo. Ele aponta a diferença entre os rendimentos dos mais pobres e dos mais ricos. Numericamente, varia de 0 a 1 (alguns apresentam de 0 a 100). O valor zero representa a situação de igualdade, ou seja, todos têm a mesma renda. O valor 1 (ou 100) está no extremo oposto, isto é, uma só pessoa detém toda a riqueza. Na prática, o índice de Gini costuma comparar os 20% mais pobres com os 20% mais ricos (Wolffenbüttel, 2004).

2.3.2 Proporção de pessoas vivendo abaixo da mediana de renda

A proporção de pessoas vivendo abaixo da mediana de renda (ou consumo) é a porção da população de um país que vive com menos da metade do nível de renda/consumo da mediana da distribuição de renda/consumo nacional. A sua unidade de medida é a porcentagem. Esse indicador é medido utilizando a distribuição nacional per capita de consumo ou renda, com dados obtidos por pesquisas. A mediana é estimada da mesma distribuição do qual o indicador também é estimado, logo os 50% que a mediana limita vai variar através do tempo. A renda per capita ou o consumo são estimados usando a renda familiar total ou o consumo dividido pelo tamanho (número de integrantes) da unidade familiar (CEPALSTAT).

2.3.3 Índice de pobreza multidimensional (IPM)

O índice de pobreza multidimensional é uma medida de pobreza que considera vários aspectos da vida das pessoas, incluindo saúde, educação e acesso à bens e serviços básicos. O IPM é calculado como a proporção da população que vive em pobreza multidimensional, definida como a situação de uma pessoa que está privada de pelo menos um terço dos indicadores de pobreza multidimensional. É uma medida de pobreza que reflete a privação simultânea em múltiplos domínios de vida, incluindo saúde, educação e nível de vida. Por exemplo, se uma pessoa

estiver privada de acesso a cuidados de saúde, educação primária e saneamento, ela será considerada como vivendo em pobreza multidimensional. O IPM é calculado usando uma escala de 0 a 1, com zero representando nenhuma pobreza multidimensional e um representando pobreza multidimensional extrema. O cálculo também envolve um conjunto de indicadores que representam diferentes dimensões da pobreza. Os indicadores usados no cálculo do IPM variam de acordo com o país ou a região. Os indicadores mais comuns usados no cálculo do IPM incluem: saúde, educação e nível de vida. Saúde consiste no acesso a cuidados de saúde, saneamento e nutrição. A educação consiste no acesso à educação primária e secundária. E o nível de vida consiste no acesso à água potável, saneamento, habitação e energia.

2.3.4 Razão das pessoas abaixo das linhas de pobreza nacionais

De acordo com o Banco Mundial (World Bank) esse índice é uma medida de pobreza baseada na renda das pessoas. Essa medida é calculada como a proporção da população que vive com uma renda inferior a uma linha de pobreza definida pelo governo, ou seja, proporção de pessoas de baixa renda = (número de pessoas com renda inferior à linha de pobreza) / (população total). Por exemplo, se um país tem uma população de 100 milhões de pessoas e 50 milhões de pessoas vivem com uma renda inferior à linha de pobreza, a proporção de pessoas de baixa renda em relação à população total é de 50%. A linha de pobreza é o valor monetário que representa o nível mínimo de renda necessário para atender as necessidades básicas de uma pessoa. Ela é calculada de acordo com uma cesta de bens e serviços considerados essenciais para a sobrevivência e o bem-estar. O conceito deste índice foi desenvolvido pelo Banco Mundial que publica anualmente um relatório sobre o desenvolvimento mundial que inclui dados sobre a pobreza em países de todo o mundo.

2.3.5 Índice de Desenvolvimento Humano (IDH)

O índice de desenvolvimento humano é uma medida composta de três indicadores: expectativa de vida ao nascer, educação e renda. Ele foi desenvolvido pelo programa das Nações Unidas para o desenvolvimento (PNUD) e é publicado

anualmente no relatório de desenvolvimento humano. O IDH é calculado usando uma escala de 0 a 1, com zero representando o menor desenvolvimento humano e um representando o maior desenvolvimento humano. A expectativa de vida ao nascer mede o número médio de anos que uma pessoa pode esperar viver, dadas as condições de saúde e nutrição atuais. Educação mede a média de anos de escolaridade que as pessoas em uma população concluíram e a taxa de alfabetização de adultos. Renda mede a renda per capita de um país, ajustada pela paridade do poder de compra.

2.3.6 Índice de Rigidez (*Stringency Index*)

O índice de rigidez é uma medida da severidade das medidas de resposta a covid-19 adotadas por um governo. Ele é calculado usando uma escala de 0 a 100, com zero representando a menor rigidez e cem representando a maior rigidez. Este índice foi desenvolvido pelo *Oxford COVID-19 Government Response Tracker* (*OxCGRT*), um projeto de pesquisa que monitora respostas governamentais à covid-19 em todo o mundo. O seu cálculo se dá a partir de nove categorias, onde o resultado do índice é a soma das pontuações das nove categorias de medidas dividido por 9. As nove categorias de medidas usadas no cálculo do índice de rigidez são: fechamento de escolas, fechamento de locais de trabalho, cancelamento de eventos públicos, restrições aglomerações públicas, fechamento de transporte público, requisitos de permanência em casa, campanhas de informação pública, controle de movimento interno, controle de viagens internacionais.

2.4 Desenvolvimento de aplicações REST para Web

Aqui serão apresentadas as ferramentas, padrões de projeto, e os conceitos relevantes para uma aplicação RESTful Web que faz chamadas para uma API cuja responsabilidade é a comunicação com o banco de dados. Vale ressaltar que RESTful é como são chamadas as aplicações que seguem as especificações do REST, ainda que na atualidade as aplicações não precisam atender todos os pontos abordados para serem rotuladas RESTful. Também serão conceituados Java, Hibernate, *Spring Data* e WebMVC, PostgreSQL e bancos de dados relacionais.

2.4.1 REST

Arquitetura REST(Representational State Transfer) é um estilo arquitetural para o desenvolvimento de interfaces web. Na arquitetura REST interações entre o cliente e um servidor são tratadas como trocas sem estado, onde cada requisição do cliente é resolvida como uma interação auto-contida e independente. Os dados são enviados em formatos flexíveis como XML ou Json, que permitem que o servidor processe a requisição de forma modular e customizável. Essas trocas foram desenhadas para serem previsíveis, confiáveis, eficientes e fáceis de entender.

2.4.2 Java

Deitel (2017) nos diz que o Java é uma das linguagens de programação mais usadas no mundo. Ela trabalha principalmente com orientação a objetos que segundo o autor é uma metodologia-chave de programação. Devido a sua máquina virtual chamada de JVM (Java Virtual Machine) é possível trabalhar com um paradigma chamado “escreva uma vez e rode em qualquer lugar” nesta linguagem. Atualmente o Java já está na sua versão 21 o que o torna uma escolha robusta para qualquer projeto, uma vez que devido ao seu tempo de existência, tamanho de comunidade e grande variedade de bibliotecas, a possibilidade de implementação de um projeto do começo ao fim usando apenas o ecossistema Java é bastante alta.

2.4.3 Hibernate

É um *framework* baseado em Java de mapeamento objeto-relacional que é utilizado para persistir e gerenciar objetos java em um banco de dados. Frameworks ORM simplificam o processo de mapeamento de classes Java para o banco de dados. O Hibernate é um desses *frameworks*, é maduro e bem conhecido, utilizado para guardar e recuperar objetos java de um banco de dados, gerenciar mapeamentos de objetos do banco de dados, e automaticamente gerar queries SQL baseadas nas mudanças do objeto. O hibernate também pode lidar com algumas funcionalidades diretamente no java em vez de gerar SQL, como cacheamento de queries e validação de objetos.

2.4.4 Spring

Um *framework* de desenvolvimento de aplicações usado principalmente com a linguagem Java. Ele acelera o desenvolvimento e provê uma estrutura arquitetural extensível e flexível para aplicações. O Spring ajuda a simplificar e padronizar o processo de desenvolvimento oferecendo um número de ferramentas e bibliotecas, incluindo inversão de controle, injeção de dependência, e configuração de módulos. Spring também suporta testes, segurança, e gestão de dados. Adicionalmente Spring traz uma gama de módulos e extensões que cuidam de necessidades específicas, como o *SpringWebMVC*, Spring Data, Spring Batch e Spring Cloud. De modo geral o Spring pode simplificar e acelerar o processo de desenvolvimento, facilitando teste e depuração, e aumentando a flexibilidade da arquitetura.

Spring MVC (Modelo, Visão e Controlador) é um módulo dentro do Spring que oferece um Framework para a construção e organização de aplicações web baseadas na arquitetura MVC. Aplicações web usando o Spring MVC seguem uma arquitetura particular em que o modelo representa a lógica de negócios, a visão lida com apresentação de dados, e o controlador controla os inputs do usuário e o fluxo de dados entre o modelo e a visão. Nessa arquitetura o controlador recebe inputs de usuários e usa eles para atualizar modelo, que é então refletido na visão. Este módulo também provê mecanismos para gerenciar outros aspectos de desenvolvimento web, como roteamento e gestão de requisições.

Spring Data é um módulo do Spring que nos traz uma gama de abstrações e integrações que suportam o acesso aos dados. ele foi desenhado para facilitar o trabalho com vários tipos de armazenadores de dados, como banco de dados relacionais, banco de dados NoSQL, e até mesmo XML. Spring Data é uma fonte de recursos para trabalhar com dados de forma intuitiva e uniforme, não importando qual o tipo de banco de dados está sendo utilizado por baixo dos panos. Por abstrair as diferenças entre os bancos de dados permite que desenvolvedores foquem no dado e na sua lógica sem estarem confinados a uma especificação técnica dos mecanismos de guarda de dados utilizados.

2.4.5 React

É um *Framework* de *front-end* escrito em JavaScript que oferece um mecanismo simplificado para a construção e renderização de componentes baseados em web. Foi desenvolvido para criar interfaces de usuário responsivas e interativas, possibilitando que desenvolvedores criem componentes modulares e reutilizáveis para padronizar e simplificar o processo de desenvolvimento. React se apoia no conceito de componentes, que são seções isoladas de uma página Web que podem ser atualizadas individualmente sem afetar outras partes da página. Os componentes são implementados em HTML e JavaScript, e são conectados através de um mecanismo de fluxo de dados que leva em consideração as interações do usuário e mudanças de estado. React têm várias ferramentas que tornam fácil o trabalho com componentes e a construção de interfaces complexas.

2.4.6 PostgreSQL

É um SGBD (Sistema de Gestão de Banco de Dados) gratuito e *open-source* que é utilizado com bancos de dados relacionais e é conhecido por sua consistência, performance e precisão. É bastante popular entre desenvolvedores web, provendo uma plataforma flexível e poderosa para gerenciar e guardar dados. Ele suporta uma extensa gama de ferramentas e funções, incluindo a interpretação de queries SQL, segurança focada em dados, e replicação de dados. O postgresql também conta com um mecanismo de extensão que permite que usuários estendam suas funcionalidades com ferramentas personalizadas e integrações.

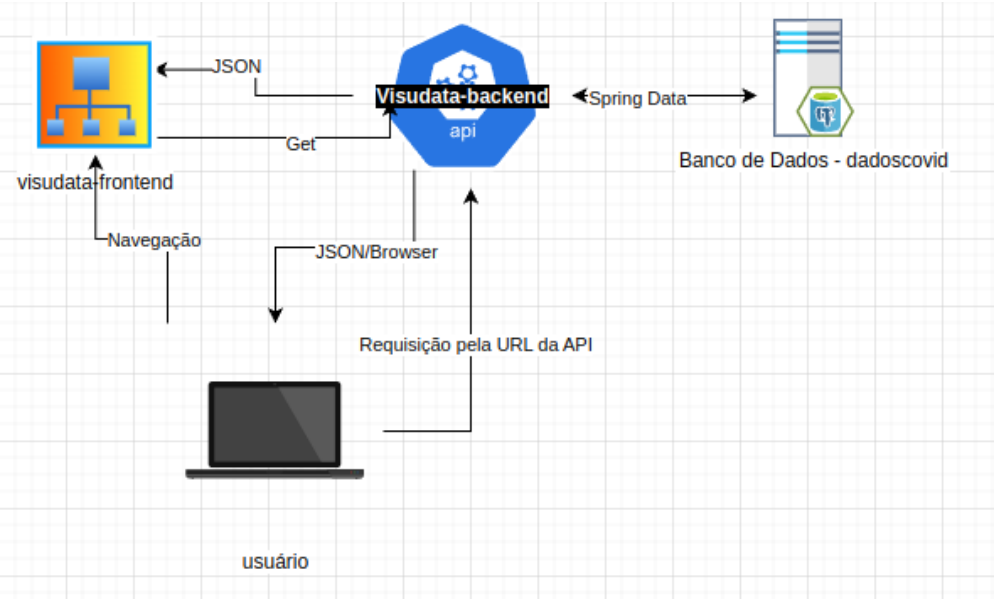
Os bancos de dados relacionais que este SGBD utiliza são bancos de dados que armazenam dados organizados em tabelas ou relações. Cada tabela no banco de dados relacional representa um conjunto de dados relacionados, que são estruturados em linhas e colunas. Os relacionamentos entre os dados são definidos pelos esquemas do banco de dados, que especificam a estrutura das tabelas e do dado dentro delas. Bancos de dados relacionais são ferramentas poderosas para armazenar e organizar quantidades grandes de dados, e eles fornecem um Framework para manipulação e gerenciamento de muitos *datasets* em conjunto. Eles também são utilizados em vários contextos, como em inteligência de negócios e ciência de dados, gestão de dados e pesquisa científica.

3 METODOLOGIA

Neste capítulo falaremos sobre o sistema de visualização de dados criado pelo autor e também sobre a metodologia utilizada na sua elaboração e implementação. O sistema em questão, que chamaremos a partir de agora de *visudata*, é uma aplicação web que permite que o usuário tenha acesso a diversas visões estruturadas com gráficos e dados sobre a covid-19 e dados de índices econômicos dos diversos países. O propósito desta aplicação é permitir que o usuário ao ver os dados estruturados em visualizações possa ter *insights* tanto científicos quanto de negócios sobre o relacionamento entre a covid e a economia dos países.

O *visudata* apesar de ser referenciado como apenas uma aplicação na verdade é dividido em três camadas: o front-end, o back-end e o banco de dados. Tradicionalmente considera-se o banco de dados como parte do back-end, no entanto como utilizamos uma aplicação à parte para o banco de dados e o caminho de comunicação entre a aplicação que funciona como back end de fato segue o mesmo princípio da comunicação entre ela e o front-end decidimos separá-lo em uma outra camada com o propósito de facilitar o entendimento (Figura 3.1).

Figura 3.1 - Esquema de relacionamento entre as camadas



Fonte: Autoral

As camadas citadas são desacopladas, ou seja elas não fazem parte de apenas um grande sistema monólito mas sim pequenos subsistemas que podem rodar independentemente uns dos outros e isso inclui o banco de dados que pode funcionar sem que seja necessário que se rode nenhuma das outras camadas da aplicação. O método de comunicação entre o front-end e o back-end é por meio de uma API RESTful, no caso o nosso back-end é essa API, ela expõe os *endpoints* que ao serem chamados devolvem os dados requisitados para o front-end no formato JSON.

O *front-end* foi feito utilizando *react* que é uma biblioteca do JavaScript que torna mais fácil a utilização de outras bibliotecas e criação e importação de módulos e componentes. A API restful foi feita em Java utilizando Spring que é uma biblioteca que modifica a forma como utilizamos a orientação a objetos no Java além de fornecer vários outros módulos para integração com o banco de dados e aplicações web. Para o banco de dados utilizamos um SGBD chamado PostgreSQL, esse é um dos melhores SGBDs gratuitos existentes na atualidade com suporte para vários tipos de bancos de dados relacionais incluindo plugins para trabalhar com GIS.

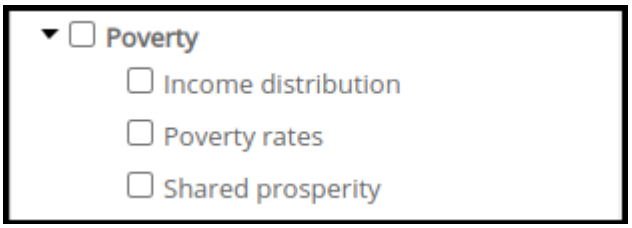
A seguir entraremos em detalhes sobre as etapas percorridas para a implementação do nosso sistema são elas: A extração dos dados de covid-19 e dos dados econômicos dos países utilizados no sistema, a montagem do banco de dados e a importação dos dados mencionados anteriormente para as devidas tabelas, a criação do backend e a integração com o banco de dados existente, a criação do front end e a montagem das páginas e por fim a integração de todas as camadas e o ajuste do funcionamento da aplicação.

3.1 Extração dos Dados

Os dados utilizados para alimentar o sistema foram retirados de duas fontes distintas. Os dados de covid-19 foram obtidos através do site da *Our World in Data* e os dados relativos aos índices econômicos foram obtidos através do site do Banco Mundial. O período de tempo escolhido a qual os dados fazem referência são os anos de 2020 e 2021. Este período foi escolhido devido a ser um período onde a pandemia já havia sido reconhecida pela maioria dos países como uma ameaça e onde a maior parte dos esforços já estavam estabelecidos em relação à pesquisa

sobre covid-19. O site do Banco Mundial possui uma plataforma onde podemos filtrar quais dados necessitamos, de quais países e de qual período. As visões que nós construímos para a aplicação focam nos índices econômicos relativos à pobreza e à saúde, portanto escolhemos pegar todos os dados dos países referentes a estes índices econômicos. Os dados de pobreza são divididos em três categorias distintas de índices: Distribuição de renda, Taxas de pobreza e Prosperidade relativa.

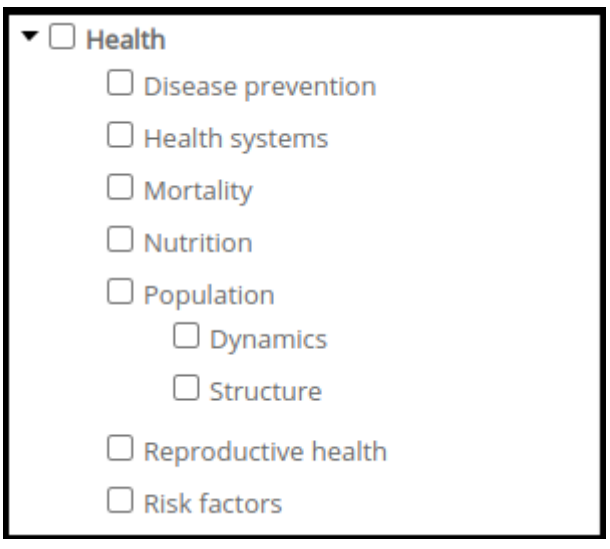
Figura 3.1.1 - Divisão dos Índices de Pobreza



Fonte: Autoral

Os dados de acesso à saúde obtidos pelo Banco Mundial são divididos em várias categorias diferentes, são elas: prevenção de doenças, sistemas de saúde, mortalidade, nutrição, dinâmicas populacionais e estrutura, saúde reprodutiva, fatores de risco.

Figura 3.1.2 - Divisão dos Índices de Acesso à Saúde



Fonte: Autoral

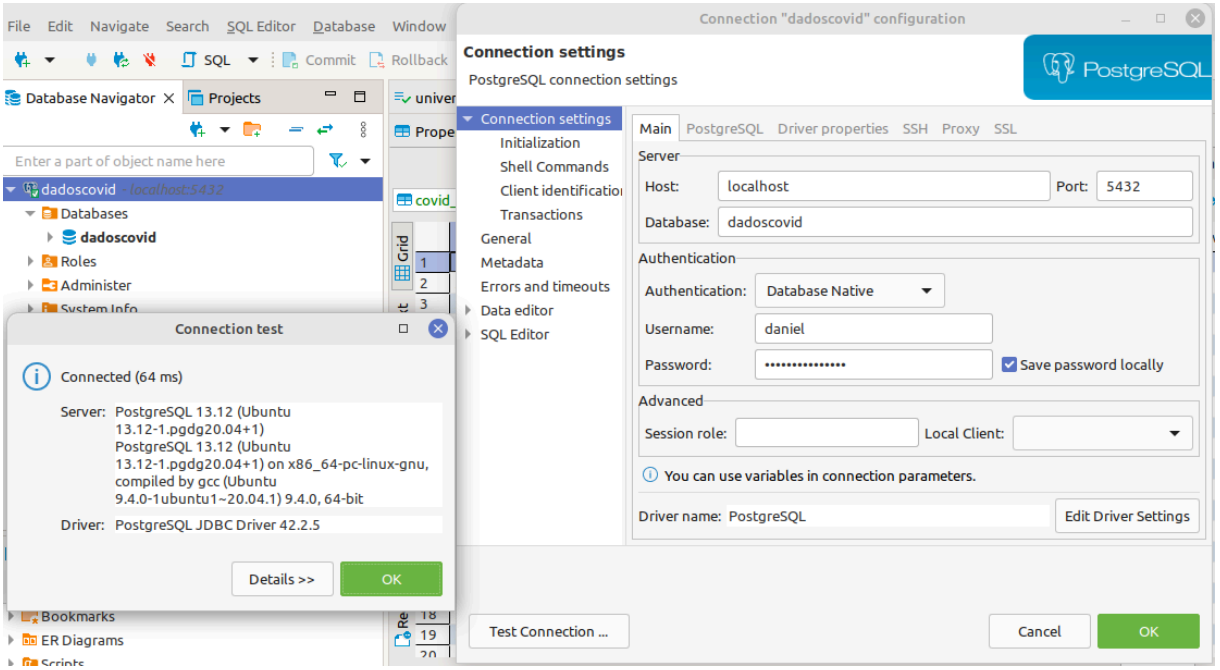
O site da Our World in Data também possui uma função similar ao do Banco Mundial onde podemos estar importando os dados relativos a covid-19 colhidos pelo órgão em uma determinada janela de tempo. Os dados obtidos vão do dia dezoito

de janeiro de 2020 até o último dia do ano de 2021. Os *datasets* foram ambos exportados no formato .csv que é um formato muito comum para importação e exportação de dados para a utilização em banco de dados.

3.2 Montando o Banco de Dados

Para a montagem do banco de dados foi utilizado um sgbd chamado postgreSQL. A versão utilizada foi a 13.6. Ele foi instalado em um sistema operacional Linux *Mint* através da linha de comando. Após a instalação foi criado o primeiro cluster para a utilização com os dados que seriam importados. O *cluster* assim como o primeiro banco de dados foram ambos chamados de *dadoscovid*, como é possível verificar na Figura 3.2.1. Para a gestão deste banco de dados utilizamos um outro software chamado *dbeaver* que é usado para a gestão de vários tipos de bancos de dados, ele funciona como um agregador de sgbds uma vez que a sua interface pode ser utilizada com uma gama enorme dos mesmos. Na interface do *dbeaver* conectamos com o banco de dados criado anteriormente que se encontrava vazio (Figura 3.2.1).

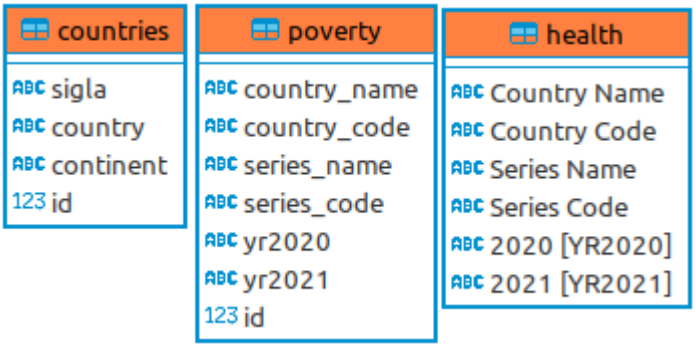
Figura 3.2.1 - Visão do *Cluster* e do Banco de Dados e sua Configuração



Fonte: Autoral

Após verificar que a conexão com o banco de dados *dadoscovid* foi bem sucedida iniciou-se a importação de dados através da ferramenta de importação do próprio dbeaver. Através dela foi localizado primeiramente o arquivo csv responsável pelos dados econômicos que havia sido baixado do site do Banco Mundial. Concluída essa etapa o processo foi repetido utilizando o arquivo csv com os dados de covid-19 que pegamos do site do *Our World in Data*. Constatou-se que no banco de dados dadoscovid haviam sido criadas duas tabelas cada uma com os dados brutos que foram importados anteriormente. No entanto, para poder utilizar esses dados em conjunto com uma aplicação foi necessária a organização em algumas tabelas (Figura 3.2.2). Primeiramente foi criada uma tabela com a lista de todos os países e suas características além de uma coluna de ID, gerada automaticamente pelo banco de dados através de uma query, que possibilitasse a busca quando solicitado pela aplicação.

Figura 3.2.2 - Tabelas criadas para organização dos dados importados



Fonte: Autoral

Os dados com o índice de pobreza e os dados com os índices de saúde obtidos pelo Banco Mundial foram separados cada uma em uma tabela própria, chamadas respectivamente de poverty e *health*. Essas tabelas ficaram com a sua estrutura parecida modificando apenas os dados que cada uma carrega. Elas também ficaram com 7 colunas sendo elas a coluna id, nome do país, código do país, nome da série ou índice econômico, código da série, dados de 2020 e dados de 2021. Cada linha dessas colunas representava um índice econômico de um determinado país, um exemplo disso está na Figura 3.2.3.

Figura 3.2.3 - Índices Econômicos de um País no Banco de Dados

	country_name	country_code	series_name	series_code	yr2020	yr2021	id
1	Indonesia	IDN	Annualized average growth rate in per capita r	SI.SPR.PC40.ZG	..	..	2,521
2	Indonesia	IDN	Annualized average growth rate in per capita r	SI.SPR.PCAP.ZG	..	..	2,522
3	Indonesia	IDN	Gini index	SI.POV.GINI	37	37.3	2,523
4	Indonesia	IDN	Income share held by fourth 20%	SI.DST.04TH.20	22	21.7	2,524
5	Indonesia	IDN	Income share held by highest 10%	SI.DST.10TH.10	29	29.6	2,525
6	Indonesia	IDN	Income share held by highest 20%	SI.DST.05TH.20	44.4	44.9	2,526
7	Indonesia	IDN	Income share held by lowest 10%	SI.DST.FRST.10	3	3	2,527
8	Indonesia	IDN	Income share held by lowest 20%	SI.DST.FRST.20	7.2	7.1	2,528
9	Indonesia	IDN	Income share held by second 20%	SI.DST.02ND.20	11	11	2,529
10	Indonesia	IDN	Income share held by third 20%	SI.DST.03RD.20	15.4	15.3	2,530
11	Indonesia	IDN	Multidimensional poverty headcount ratio (% o	SI.POV.MDIM	..	..	2,531
12	Indonesia	IDN	Multidimensional poverty headcount ratio, chil	SI.POV.MDIM.17	..	..	2,532
13	Indonesia	IDN	Multidimensional poverty headcount ratio, fen	SI.POV.MDIM.FE	..	..	2,533
14	Indonesia	IDN	Multidimensional poverty headcount ratio, hou	SI.POV.MDIM.HH	..	..	2,534
15	Indonesia	IDN	Multidimensional poverty headcount ratio, ma	SI.POV.MDIM.MA	..	..	2,535
16	Indonesia	IDN	Multidimensional poverty index (scale 0-1)	SI.POV.MDIM.XQ	..	..	2,536
17	Indonesia	IDN	Multidimensional poverty index, children (popu	SI.POV.MDIM.17.XQ	..	..	2,537
18	Indonesia	IDN	Multidimensional poverty intensity (average sh	SI.POV.MDIM.IT	..	..	2,538

Fonte: Autoral

Os dados de covid-19 obtidos do site da *Our World in Data* ficaram organizados em uma tabela própria da seguinte maneira: cada entrada representa um dia de um determinado país de forma que a pesquisa mais básica envolve sempre fornecer um país e um dia específico ou um intervalo de dias como data inicial e data final. A quantidade de dados fornecidos sobre covid-19 é enorme por isso o número elevado de colunas nesta tabela no entanto não foram utilizadas todas as informações nesta aplicação como é possível verificar na Figura 3.2.4.

Figura 3.2.4 - Exemplo de Dados da Covid-19

	iso_code	continent	location	date	total_cases	new_cases	new_cases_smoothed	total_deaths
1	ARG	South America	Argentina	2022-07-25	9,507,562	0	5,962.143066406	129,278
2	ARG	South America	Argentina	2022-07-26	9,507,562	0	5,962.143066406	129,278
3	ARG	South America	Argentina	2022-07-27	9,507,562	0	5,962.143066406	129,278
4	ARG	South America	Argentina	2022-07-28	9,507,562	0	5,962.143066406	129,278
5	ARG	South America	Argentina	2022-07-29	9,507,562	0	5,962.143066406	129,278
6	ARG	South America	Argentina	2022-07-30	9,507,562	0	5,962.143066406	129,278
7	ARG	South America	Argentina	2022-07-31	9,560,307	52,745	7,535	129,369
8	ARG	South America	Argentina	2022-08-01	9,560,307	0	7,535	129,369
9	ARG	South America	Argentina	2022-08-02	9,560,307	0	7,535	129,369
10	ARG	South America	Argentina	2022-08-03	9,560,307	0	7,535	129,369
11	ARG	South America	Argentina	2022-08-04	9,560,307	0	7,535	129,369
12	ARG	South America	Argentina	2022-08-05	9,560,307	0	7,535	129,369
13	ARG	South America	Argentina	2022-08-06	9,560,307	0	7,535	129,369
14	ARG	South America	Argentina	2022-08-07	9,602,534	42,227	6,032.429199219	129,440
15	ARG	South America	Argentina	2022-08-08	9,602,534	[NULL]	[NULL]	129,440

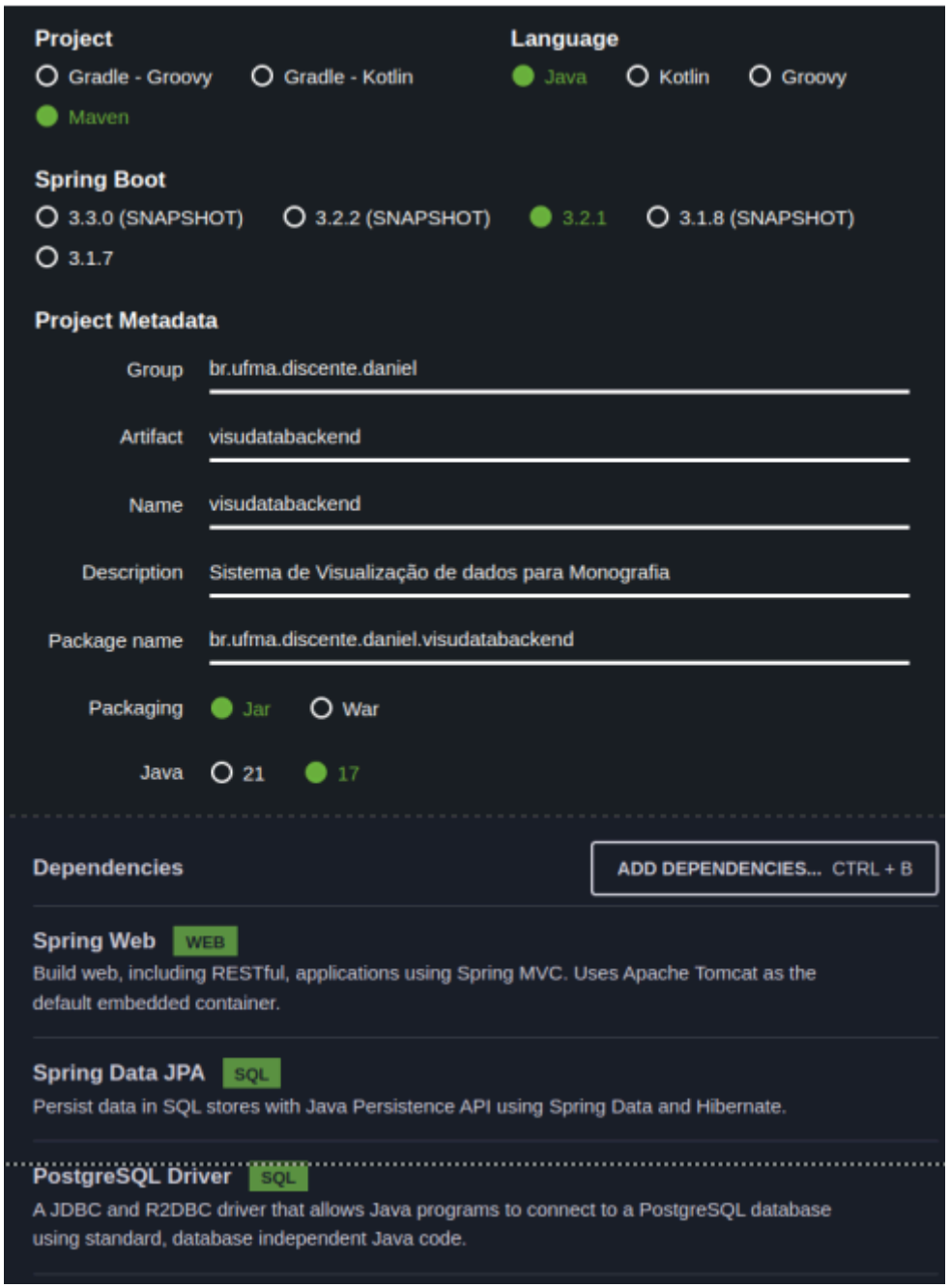
Fonte: Autoral

3.3 Criação do Back-end utilizando Spring e Java

O *back-end* da aplicação, ou seja, a API RESTful foi criada utilizando *Spring*. O *Spring* fornece um serviço para a construção de APIs chamado *Spring Boot*, este serviço facilita a criação do projeto inicial fazendo a configuração automática dos arquivos de configuração da aplicação através de uma interface acessada pelo site

https://start.spring.io/, o nome desse serviço é *Spring Initializer*. Neste site utilizamos a seguinte configuração para dar início ao nosso projeto (Figura 3.3.1).

Figura 3.3.1 - Configuração do Spring Initializer

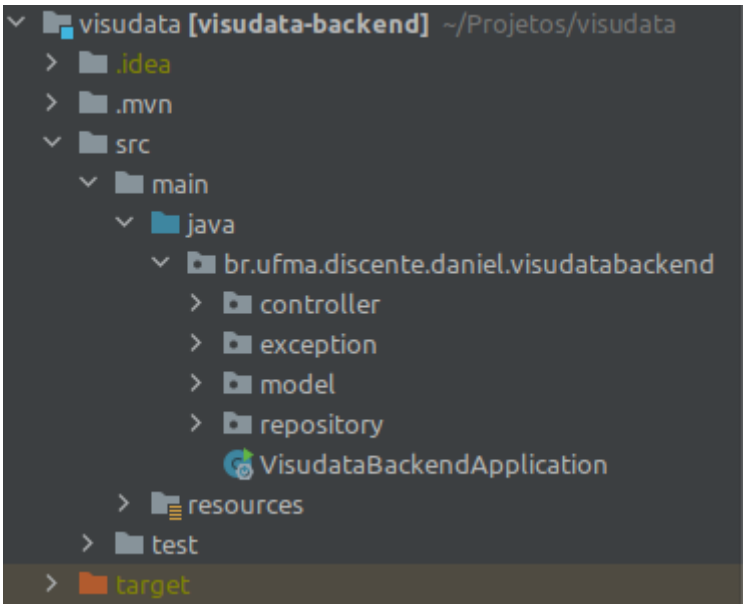


Fonte: Autoral

O projeto é gerado dentro de um arquivo .zip, que foi devidamente importado para o *IntelliJ*, a IDE utilizada durante todo o processo de desenvolvimento, nela foi possível tanto realizar modificações no código quanto rodar as diferentes instâncias da aplicação. O projeto é gerado apenas com o pacote original da aplicação e uma

classe gerada automaticamente pelo *Spring*, chamada *VisudataBackendApplication*, cuja função é a execução da aplicação (Figura 3.3.2).

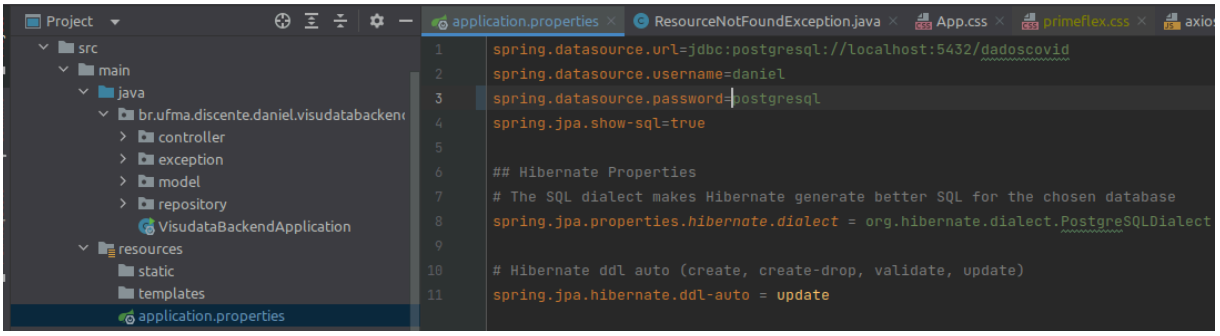
Figura 3.3.2 - Esquema de Pastas do Projeto



Fonte: Autoral

No desenvolvimento dessa aplicação foi utilizado o GitHub como plataforma de controle de versionamento. Após a subida do projeto pro github e feito o commit inicial, foi feita a conexão desta api com o banco de dados, para isso criou-se uma pasta chamada *repository* que armazenou todas as classes de repositório, ou seja, as classes utilizadas para representar e extrair os dados das tabelas do banco de dados. Também foi necessário criar um arquivo de configuração dentro da pasta *resources* com os dados de login do banco de dados, para que a aplicação pudesse se comunicar com o *dadoscovid* (o banco de dados criado anteriormente) com sucesso. O nome deste arquivo é *application.properties*, ele é reconhecido pelo módulo do *Spring* como um arquivo de configuração padrão e é utilizado pelo *Spring Data* para criar a comunicação com qualquer banco de dados que esteja apontado neste arquivo (Figura 3.3.3).

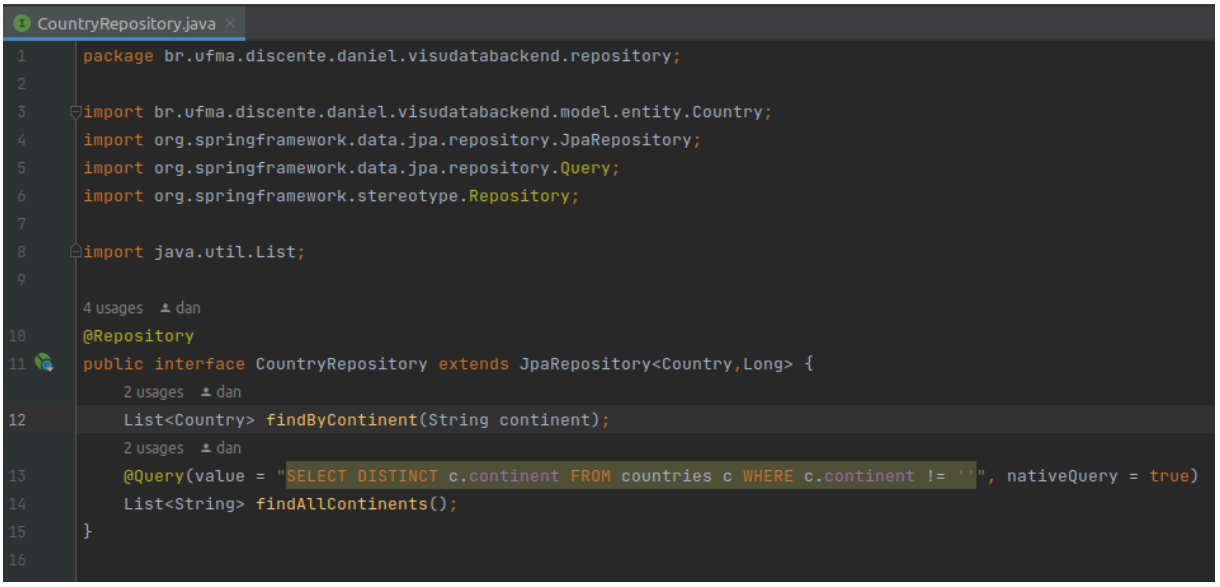
Figura 3.3.3 - Propriedades de Conexão com o Banco de Dados



Fonte: Autoral

A aplicação utilizou quatro classes de repositório, são elas: o repositório de países, o repositório de dados de saúde, o repositório de dados de pobreza e o repositório de dados de covid-19. Vale ressaltar que graças ao *Spring Data* não é necessário fazer as implementações de CRUD (*Create, Read, Update, Delete*) uma vez que as classes de repositório são interfaces que extendem a classe *JpaRepository* do *Spring Data* e ela já conta com uma pré-implementação desses métodos, bastando que seja informado o tipo de variável que será *Id* e o tipo do objeto. Existe também outra função muito interessante do *Spring* que são as funções *findBy*. Esse tipo de função permite que seja possível declarar uma função *find* (que é uma função de busca) apenas citando elementos que constam na classe de modelo da entidade em questão. No caso do repositório de países, a interface *CountryRepository* nós declaramos uma dessas funções chamada *findByContinent* que devolve uma lista de países formada por objetos do tipo *Country*. Na interface desse repositório nós também declaramos um método que não utiliza as funções automáticas do *Spring* em vez disso este método leva uma *query* montada de forma particular por meio da anotação @Query, a query traz a lista de todos os continentes, o nome desse método ficou *findAllContinents* e ele devolve uma lista de *string* com essas informações (Figura 3.3.4).

Figura 3.3.4 - Detalhe do Repositório de Países

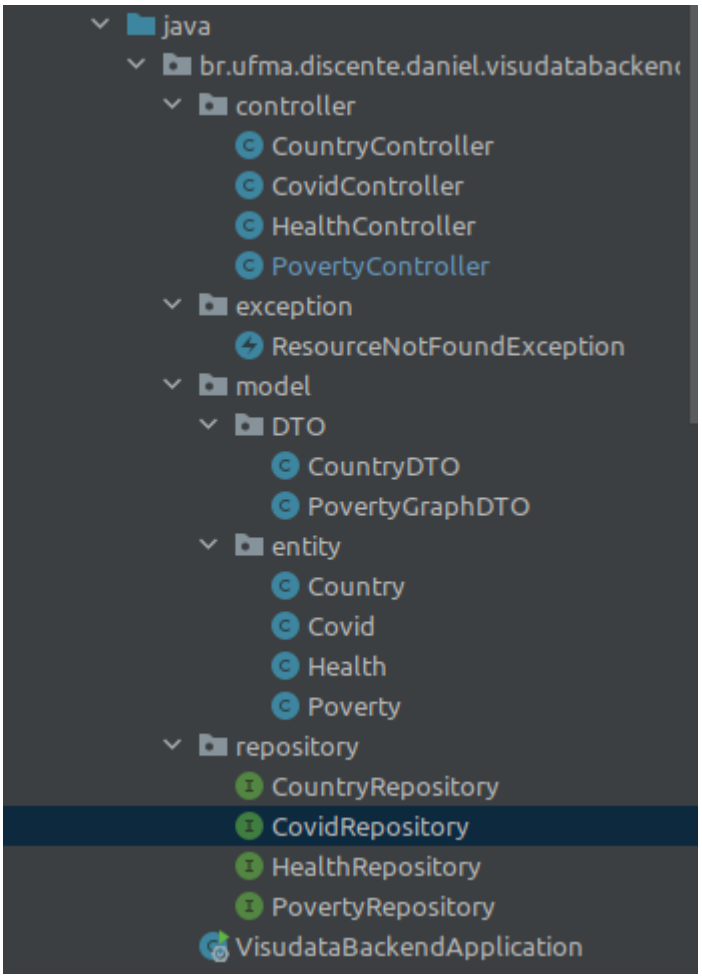


Fonte: Autoral

A classe de repositório de dados de pobreza e a classe de repositório de dados de saúde ficaram com código similar uma vez que elas representam tabelas parecidas e o método de acesso aos seus dados seguiu a mesma regra. Portanto, na interface implementamos dois métodos, ambos usando a função *findBy*. O primeiro método é o *findByCountryName* que pesquisa uma lista de índices de pobreza correspondentes ao país fornecido por meio de uma *string* como parâmetro do método. O segundo método é o *findByCountryNameAndSeriesCode*, esse método recebe duas variáveis uma string com o nome do país e uma string com o código da série, esse código é único para cada série (ou índice econômico) e foi gerado pelo Banco Mundial. Com essa busca é possível achar um único índice econômico correspondente a um determinado país. Esse tipo de busca é possível porque na classe modelo de entidade *Poverty,* que é o modelo do índice de pobreza, existe tanto o atributo *CountryName* quanto o atributo *SeriesCode* que representam colunas na tabela correspondente do banco de dados. O *HealthRepository* implementa os mesmos métodos que a classe anterior, também com os mesmos nomes que foram mencionados anteriormente uma vez que a forma de funcionamento é similar. O repositório com os dados de covid-19 cuja classe é a interface *CovidRepository* contém uma função *findByData*AndCountry recebendo uma string com a data e uma string com país como parâmetro. Há também outro método implementado neste repositório que utiliza a notação de *query* devido a necessidade de pesquisar um período entre datas. Esse método devolve uma lista

de entradas de dados de covid-19 que é o tipo de objeto Covid como está escrito nos modelos de entidades. Seguindo a organização da aplicação foi criada uma pasta para abrigar os modelos o nome dessa pasta foi chamada de *Model* dentro dela colocamos todas as classes de modelo de objeto, ou seja, as classes objetos do banco de dados, as classes objetos de transição e qualquer outro tipo de classe de objeto que seja modelo e utilizada em alguma parte da aplicação.

Figura 3.3.5 - Classes e seus diretórios

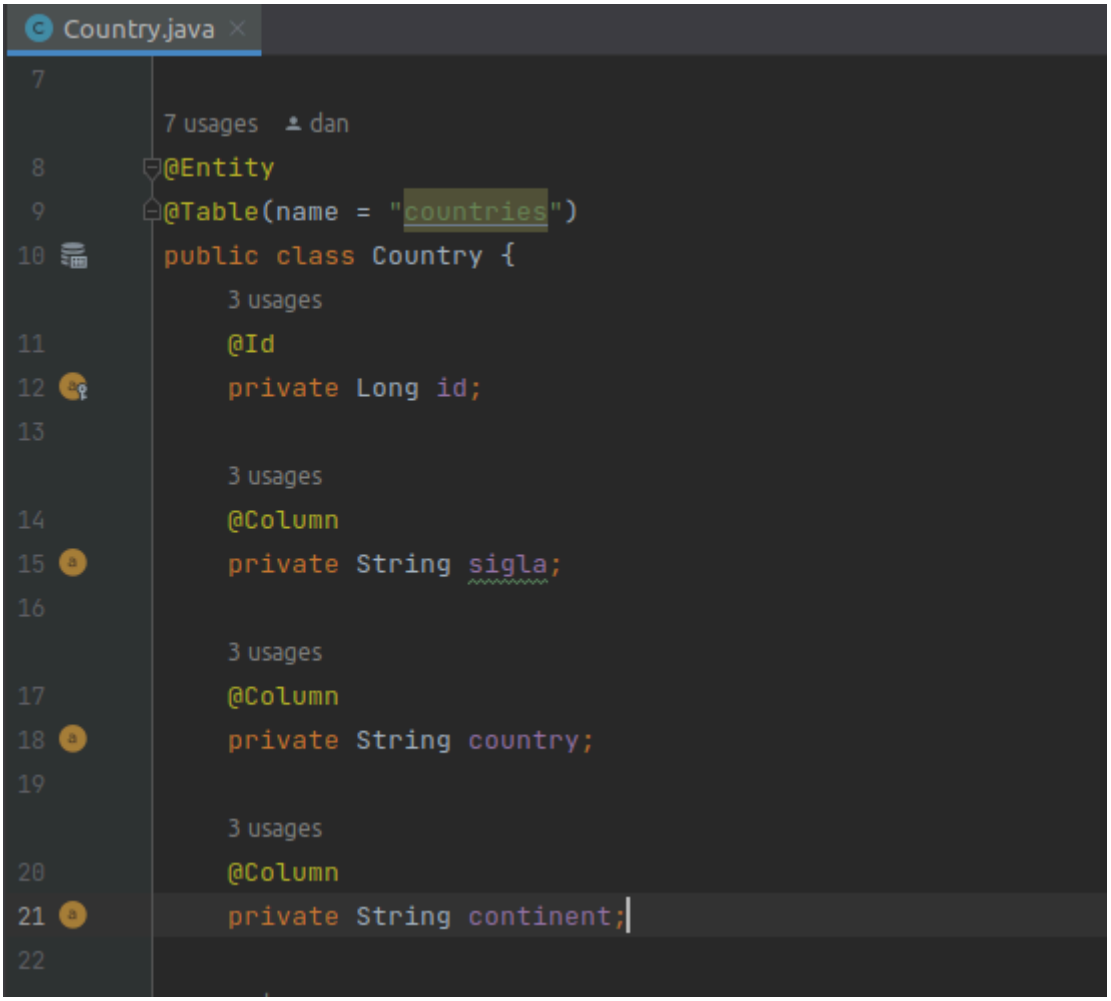


Fonte: Autoral

Para abrigar as classes de entidade criamos uma pasta dentro da pasta *Model* chamada *Entity* , como é possível ver na Figura 3.3.5. Dentro dessa pasta colocamos as quatro classes de objeto entidade utilizados nos repositórios, são elas: *Country*, *Covid*, *Health e* Poverty. As classes de entidade são objetos POJOs (*Plain Java Old Objects*) que são objetos que contém atributos e métodos de acesso como *getters* e *setters*, por exemplo, a classe de entidade *Country* possui atributos como

id, sigla, *country* e *continent* e os seus respectivos *getters* e *setters,* a única diferença de um POJO é que ela leva algumas anotações da biblioteca de persistência do Java, na assinatura da classe ela leva a anotação @Entity e @Table com o nome da tabela correspondente no banco de dados, e nos seus atributos ela leva a anotação @Id no atributo que é o identificador da entidade, em todos os outros atributos ela leva a anotação @Column para identificar que cada atributo representa uma coluna da tabela correspondente no banco de dados (Figura 3.3.6).

Figura 3.3.6 - Classe Country e suas anotações

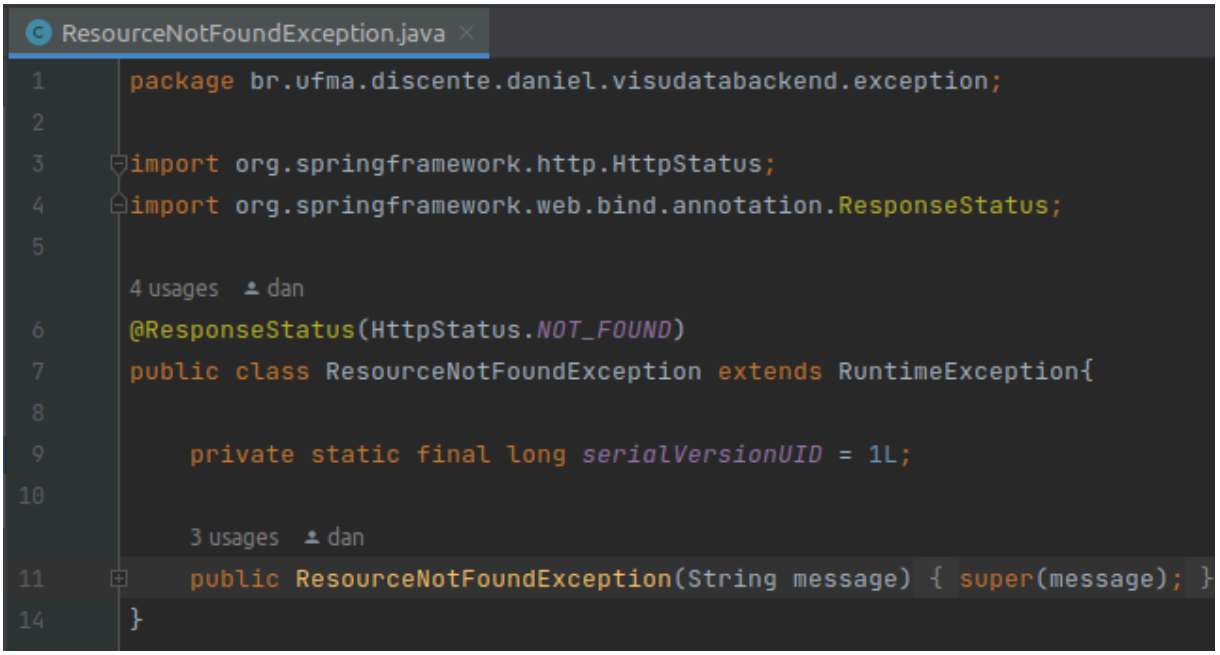


Fonte: Autoral

Os outros objetos dentro da pasta entidade também seguem a mesma lógica, com as mesmas anotações, mudando apenas os atributos de cada classe, por exemplo na classe *Poverty* temos além do id o nome do país o nome da série o código da série os dados dessa série de 2020 e os dados de 2021, além dos seus *getters* e *setters*. Ainda na pasta *Model* criamos outra pasta chamada DTO (*Data*

Transfer Object) para abrigar classes modelos com outras finalidades (Figura 3.3.5). Essas classes são utilizadas para coletar nos objetos as informações que serão necessárias enviar para o front no formato que é correto para ele consumir. A seguir iremos falar das classes mais importantes da API RESTful que são as classes de controle. Porém antes da implementação dessas classes tornou-se necessária a criação de uma pasta chamada exception para abrigar as classes de Exception, no caso da nossa aplicação foi criada apenas uma classe de exceção chamada *ResourceNotFoundException*. Esta classe estende a classe padrão do Java a *RunTimeException* com a diferença que ela levou uma anotação do framework do Spring chamada @ResponseStatus que devolve o status http 404 *Not Found* ou não encontrada, essa personalização numa classe de Exception é importante já que estamos lidando com a respostas web uma vez que por padrão não existe nenhuma exceção já programada no Java para os erros que acontecem por requisições web (Figura 3.3.7).

Figura 3.3.7 - Classe personalizada de exceção



Fonte: Autoral

O Spring possui outro módulo chamado *Spring-WebMVC* que é responsável pelas anotações que nós utilizamos nos controladores para dar a eles funções de comunicação web. A aplicação também tem quatro controladores, são eles: o *CountryController*, CovidController, *HealthController*, *PovertyController*. A classe

controladora é a classe responsável por expor os endpoints em que numa API padrão o usuário iria chamar para obter os dados, nesta API os *endpoints* são chamados pelo *front-end* para obter os dados necessários para que ele preencha as suas visões. Os *endpoints* expostos pelo Spring-WebMVC levam a seguinte nomenclatura: “a URL base / o nome que é colocado como mapeamento da classe / o nome que é colocado como mapeamento do método utilizado". Por exemplo, no controlador de país, a classe *CountryController* leva anotações na assinatura da sua classe são elas *@CrossOrigin*, *@RestController* e *@RequestMapping*. Esta última anotação é onde nós colocamos o endereço de acesso a essa classe no nosso caso colocamos “/countries" (Figura 3.3.8).

Figura 3.3.8 - Detalhe do Controlador de Países

```
@CrossOrigin
@RestController
@RequestMapping("/countries")
public class CountryController {

    5 usages
    @Autowired
    private CountryRepository countryRepository;

    dan
    @GetMapping("/all")
    public List<Country> findAllCountries() { return countryRepository.findAll(); }

    dan
    @GetMapping("/allcontinent")
    public List<String> findAllContinents() { return countryRepository.findAllContinents(); }
```

Fonte: Autoral

Nesta classe chamamos o repositório de países *CountryRepository* e anotamos ele com a anotação *@Auto Wired* que é uma anotação responsável por fazer todos os ajustes de implementação e inicialização do repositório dentro dessa classe controladora. O *CountryController* expõe quatro *endpoints* através da anotação @GetMapping que a nota quatro métodos distintos desta classe. O primeiro método é o *findAllCountries* que devolve uma lista de países, ele não recebe nenhum parâmetro e usa o *CountryRepository* para acessar a sua função *findAll* que é uma função implementada automaticamente pelo *Spring-Data* para os repositórios, onde ele devolve todos os resultados de uma busca similar a um “*select*

* *from…*” para aquela tabela. O segundo método deste controlador é a função *findAllContinents* que devolve uma lista de string e chama dentro dele a função findAllContinents do repositório de países, aquela que foi citada anteriormente, onde nós utilizamos uma *query* personalizada. O terceiro método é o *findAllFormatted* que é um método que devolve uma lista de countryDTO, esse método após obter o resultado do *findAllContinents* do *CountryRepository* também faz uma busca dos países por continentes e armazena tudo numa lista de countryDTO fazendo as devidas modificações para devolver os dados no formato que o *front-end* espera receber (Figura 3.3.9).

Figura 3.3.9 - Método *findAllFormated*

```
@GetMapping("/all/formatted")
public List<CountryDTO> findAllFormatted(){
    List<CountryDTO> countries = new ArrayList<>();
    List<String> continents = countryRepository.findAllContinents();
    AtomicReference<Integer> id = new AtomicReference<>( initialValue: 999);
    continents.forEach(c -> {
        List<Country> validCountries = countryRepository.findByContinent(c);
        List<CountryDTO> readyCountries = new ArrayList<>();
        validCountries.forEach(vc -> readyCountries.add(new CountryDTO(vc.getId().toString(), vc.getCountry())));
        CountryDTO readyContinent = new CountryDTO(id.toString(), c, readyCountries);
        id.getAndSet( newValue: id.get() - 1);
        countries.add(readyContinent);
    });
    return countries;
}
```

Fonte: Autoral

O último método expõe o *endpoint* que recebe uma variável, aqui assinalada como ID, mas que na verdade é uma string contendo o continente em que o usuário deseja fazer a pesquisa, dentro do método é chamado a função *findByContinent* do *CountryRepository* e é passado a *string* continente recebida, retornando o resultado como uma lista de países. Os demais controladores também tem o funcionamento similar ao controlador de países expondo *endpoints* diferenciados onde o usuário pode acessá-los para obter os dados do banco de dados através da API.

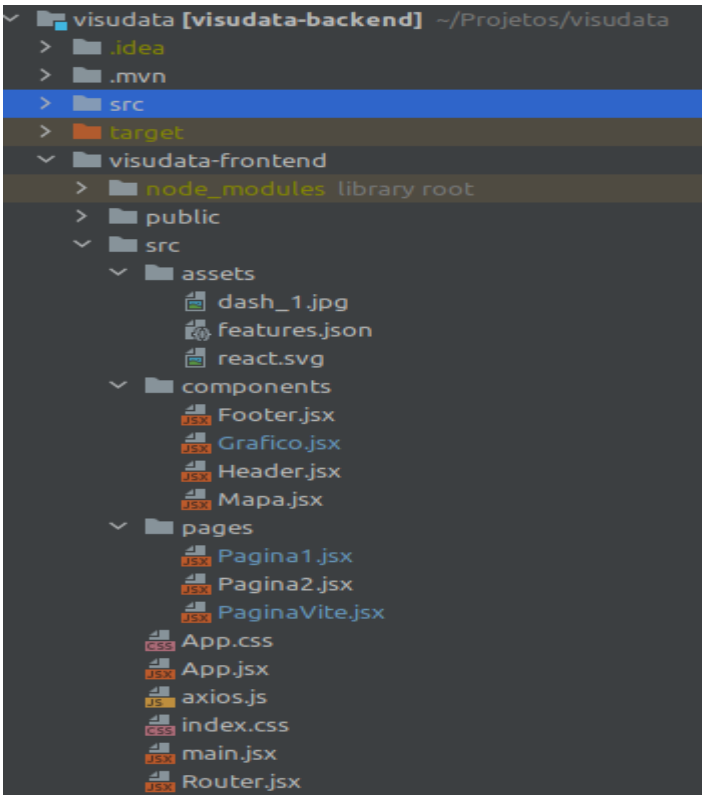
3.4 Front-end utilizando React

O *front-end* da nossa aplicação foi feito utilizando uma ferramenta chamada Vite, que além de criar o projeto inicial de *front-end* facilita a execução do mesmo fornecendo um servidor de desenvolvimento que pode funcionar localmente. Ele

também foi feito para funcionar com uma gama enorme de frameworks de programação em JavaScript. Para esse sistema foi utilizado o *React* em conjunto com o Vite, que é uma biblioteca de componentes de JavaScript que modifica a forma como nós construímos, implementamos e utilizamos os componentes de *front*.

A instalação Inicial é feita através do *Node Packages Manager* (NPM) que funciona por linha de comando. Na própria linha de comando foi utilizado o comando *vite* para criar o projeto inicial, no menu resultante é possível selecionar entre várias libs e opções como vue.js ou *EcmaSript* puro, entre outras, como mencionado anteriormente foi utilizado o *React*. O nome escolhido para o projeto foi visudata-frontend. Na concepção do projeto foi decidido que apesar desses dois sistemas serem desacoplados, com o objetivo de facilitar o trabalho neles, ambos deveriam ser criados dentro da mesma estrutura de projeto, por isso foi escolhido que o visudata-frontend ficasse dentro do projeto *visudata* (Figura 3.4.1).

Figura 3.4.1 - Estrutura do Front-end



Fonte: Autoral

As aplicações feitas com react são single page applications ou seja elas funcionam a partir de uma única página que é montada como a estrutura de HTML e todas as

suas modificações são feitas trocando os componentes da página, por isso na estrutura inicial, após o projeto ser criado nós temos dentro da pasta do projeto de *front-end* uma pasta que abriga os módulos do node e a pasta src (*source*) que vem com as classes padrão criada pelo projeto. Essas classes são a main e app ambas são .jsx que é uma extensão dos arquivos javascript (.js) utilizada no react, além delas temos duas classes de CSS a index e a app.css, elas são importadas respectivamente na main e na app.jsx. A classe main durante todo o projeto não sofre nenhuma alteração e sua única função é renderizar a classe app, logo, a função da classe de css index é abrigar os estilos globais da aplicação (Figura 3.4.1). A classe App é a primeira classe a ser renderizada no DOM, que é a árvore responsável pelos elementos de exibição da aplicação, e a partir dela são chamados os outros elementos (percorrendo a árvore) de acordo com a navegação do usuário. Em outras palavras não acontece a mudança de página no sentido clássico (por exemplo como acontece no HTML padrão) e sim uma re-renderização da classe App na Main seguindo o caminho de navegação que o usuário utilizou (no caso a classe App chama algum elemento filho que chama outro e outro e etc.). Continuando com o desenvolvimento da aplicação, a etapa seguinte foi adicionar uma classe Router (roteadora) cuja função é mapear as diferentes páginas da aplicação e direcionar a navegação para elas quando necessário. E assim foi feito, criou-se a classe Router.jsx que utilizou os componentes Routes e Router do react-router-dom (biblioteca de roteamento do react), essa classe devolve uma instância do componente Routes contendo os endereços das rotas e os elementos a serem carregados para cada rota (subentende-se que o caminho deve ser adicionado à url base), assim como consta na Figura 3.4.2 .

Figura 3.4.2 - Classe Router

```
import {Routes, Route} from "react-router-dom";
import PaginaVite from "./pages/PaginaVite.jsx";
import Pagina1 from "./pages/Pagina1.jsx";
import Pagina2 from "./pages/Pagina2.jsx";

export function Router(){
    return(
      <Routes>
          <Route path="/" element={<PaginaVite />}></Route>
          <Route path="/graph" element={<Pagina1 />}></Route>
          <Route path="/graph2" element={<Pagina2 />}></Route>
      </Routes>
    );
}
```

Fonte: Autoral

Na classe App foi necessário importar a classe Router criada e retornar ela como elemento envolvida no componente BrowserRouter também da biblioteca react-router-dom, (Figura 3.4.3).

Figura 3.4.3 - Classe App

App.jsx

```
import { BrowserRouter } from 'react-router-dom'
import './App.css'
import {Router} from "./Router.jsx";
import { PrimeReactProvider, PrimeReactContext } from 'primereact/api';
import "primeflex/primeflex.css"
import "primereact/resources/themes/lara-light-indigo/theme.css"
import 'primeicons/primeicons.css';

function App() {

  return (
  <div>
    <PrimeReactProvider>
        <BrowserRouter>
            <Router />
        </BrowserRouter>
    </PrimeReactProvider>
  </div>
  )
}

export default App
```

Fonte: Autoral

Nesta aplicação foi utilizado um pacote de componentes visuais pré-prontos chamado de PrimeReact, ele oferece vários componentes para a criação de sistemas e páginas web que vão desde elementos de página como dashboards e tabelas à gráficos. Após instalado o PrimeReact no projeto foi necessário importá-lo na classe App e envolver os elementos já existentes no componente PrimeReactProvider. Além desses componentes também foi utilizado o primeflex, que é um arquivo css com várias classes de organização de páginas da web, e o primeicons, que é uma classe css com ícones para serem utilizados em conjunto dos componentes do PrimeReact (Figura 3.4.3). Para facilitar a organização da aplicação a sua estrutura foi separada em três pastas dentro da pasta src: a pasta assets para abrigar os arquivos utilizados pelos componentes mas que não fazem parte do código, como figuras e arquivos; a pasta components que abriga os módulos personalizados da aplicação; a pasta pages que abriga as classes (ou componentes) que representam as páginas da aplicação. Antes de continuar a descrição das classes da aplicação é necessário explicar a estrutura básica de uma classe em *React*. Um arquivo .jsx é composto primeiramente pelos seus imports (importações) que podem ser tanto bibliotecas, arquivos css ou outros componentes da aplicação (classes), logo após é declarada uma função principal com o nome da classe do arquivo .jsx, essa função pode ou não receber parâmetros dependendo da utilização que o desenvolvedor pretende para ela, e depois no fim do arquivo um comando de exportação dessa classe (em alguns casos o export pode ficar na mesma linha de declaração da função, existe uma diferença mínima de funcionamento dessas duas formas, porém não é relevante para a aplicação em questão). Dentro da função existe um comando de retorno (*return*) que recebe um parâmetro, no *React* este parâmetro é a parte visual do componente onde vão as tags HTML que constroem aquilo que o componente vai exportar (ou seja, aquilo que ele representa), devido ao funcionamento interno do *React* esta exportação deve acontecer como um único elemento (para ser contado como um nó da árvore do DOM) e para isso tudo que estiver dentro do *return* deve ser envolvido com a tag <> </>. Antes do retorno, ainda no arquivo .jsx é possível declarar variáveis e funções, além de manipular os componentes importados para o arquivo, isso torna o funcionamento do *React* altamente flexível (Figura 3.4.4).

Figura 3.4.4 - Exemplo de retorno

```
return(
    <>
        <Toolbar start={startContent} center={centerContent} end={endContent} />
    </>
);
```

Fonte: Autoral

Na pasta *components* da aplicação foram criados quatro classes que representam os componentes que podem ser reaproveitados nas diferentes visualizações, são eles: Footer.jsx, Grafico.jsx, Header.jsx e Mapa.jsx. Os componentes Header e Footer são respectivamente os elementos de Cabeçalho e Rodapé da página e são reaproveitados em cada visualização construída, bastando apenas que sejam importados e montados na página. Os componentes Grafico e Mapa são classes que servem quase como uma interface para a implementação desses elementos, no caso dos gráficos o sistema utiliza os da biblioteca do PrimeReact e para facilitar a utilização dos diversos tipos de gráficos que a ela disponibiliza, a classe agrega todo o código dos diversos tipos de gráficos apenas escolhendo qual vai ser exportado dependendo das configurações que recebe quando é instanciada, este funcionamento será detalhado mais adiante. O componente Mapa funciona quase da mesma forma, agregando os componentes da biblioteca react-simple-maps necessários para a formação de uma mapa de acordo com as configurações recebidas por este componente.

A pasta pages contém as classes responsáveis pelas páginas que são exibidas pelo usuário, são elas que são chamadas e renderizadas na classe App quando a url correspondente (previamente cadastrada na classe Router) é chamada pelo browser ou por algum componente. Nesta aplicação foram feitas três classes de páginas, a classe responsável pela página principal foi nomeada PaginaVite.jsx e as outras duas responsáveis pelas visualizações da aplicação foram nomeadas respectivamente Pagina1.jsx e Pagina2.jsx (Figura 3.4.1).

A partir de agora será detalhado o funcionamento de cada uma destas classes, com o objetivo de entender melhor suas estruturas internas e seu funcionamento com o React. A primeira classe a ser destrinchada será a Footer.jsx. Essa classe tem um código simples, uma vez que é um módulo apenas com o rodapé das páginas. Nela

foi importada o componente Toolbar do PrimeReact, que é uma barra horizontal configurável, ele aceita três possíveis parâmetros dependendo de onde deseja-se que as alterações apareçam, como é possível ver na Figura 3.4.5 foi utilizado o parâmetro end para que a mensagem desejada aparece-se no fim da barra (alinhado o lado direito).

Figura 3.4.5 - Classe Footer

```
import {Toolbar} from "primereact/toolbar";
export function Footer() {

    const endContent = (
        <>
            <span>@Desenvolvido por Daniel</span>
        </>
    )
    return(
        <>
            <Toolbar end={endContent}/>
        </>
    );
}
```

Fonte: Autoral

Seguindo têm-se a apresentação detalhada da classe Grafico.jsx, aqui foram utilizados dois componentes do React essenciais para adicionar funcionalidades e interação nas páginas Web, o *useState* e o *useEffect*. O *useState* torna possível a atualização de variáveis durante a interação do usuário com a página ou componente, existe a possibilidade de colocar algum valor como estado inicial ao chamar o método, e a sua forma de utilização é declarando um vetor constante que leva sempre dois atributos, o primeiro guarda o valor desejado para a funcionalidade e o segundo serve de função de atualização (no seguinte formato: variável, setVariável), o vetor recebe a função *useState* e o valor inicial que se colocar de parâmetro (Figura 3.4.6).

Figura 3.4.6 - Declaração do *useState* em uma classe

```
const [getCountryValue, setCountryValue] = useState( initialState: null)
const [getYearValue, setYearValue] = useState( initialState: null)
const [giniIndex, setGiniIndex] = useState( initialState: null)
const [giniPanel, setGiniPanel] = useState( initialState: null)
```

Fonte: Autoral

O useEffect é um método que torna possível a aplicação de código (efeitos) quando há a ocorrência de algum gatilho, na sua assinatura ele leva dois parâmetros, o primeiro é o efeito (na forma de código javascript, podendo inclusive utilizar outros componentes já importados na classe) e o segundo é um vetor contendo todos os gatilhos. Esses gatilhos são variáveis que passam a ser observadas pelo registro global do useEffect, qualquer alteração em alguma variável de gatilho dispara a ativação daquele useEffect específico (Figura 3.4.7).

Figura 3.4.7 - Utilização do *useEffect*

```
useEffect( effect: ()=>{
    setNodes(countries)
    setSelectedNodeKey( value: '1')
}, deps: [countries])
```

Fonte: Autoral

Na classe Grafico.jsx são importados, além do useState e useEffect do React, componentes do PrimeFaces, ícones, um tema e um componente chamado Chart. Esse componente é responsável por criar um gráfico desde que seja passado três parâmetros para ele: o tipo do gráfico, um objeto contendo os dados e um objeto contendo opções que variam de acordo com o tipo de gráfico escolhido. Nesta classe foi utilizado o componente Chart chamando-o no comando de retorno e passando as opções necessárias para o funcionamento do mesmo (Figura 3.4.8).

Figura 3.4.8 - Comando de Retorno da classe Grafico

```
return (
    <Chart type={props.nomeGrafico} data={chartData} options={chartOptions} className={props.tam}/>
);
```

Fonte: Autoral

A função Grafico recebe como parâmetro a variável props que é um vetor com todos os parâmetros passados para o componente quando o mesmo é utilizado em outros

níveis. Dentro dela são definidos dois vetores constantes recebendo seus respectivos useStates, um para os dados e outro para as opções que serão enviados para o componente Chart, em seguida é implementado uma função useEffect cuja variável gatilho é a própria props e cujo efeito é dado por uma arrow function que cria uma estrutura de if elses que filtra através do nome do gráfico recebido e cria objetos diferentes que preencherão os vetores de dados e opções previamente definidos com o objetivo de utilizar os mais variados gráficos da biblioteca sem ter que implementá-los manualmente (bastando apenas chamar a classe Grafico e passar o nome do gráfico desejado e os dados) como consta na Figura 3.4.9.

Figura 3.4.9 - Exemplo de utilização da classe Grafico

```
<Card title="Participação da Receita Nacional" className="flex justify-content-center align-items-center p-0" style={{height:"100%"}}>
    <Grafico className="m-0 md:w-50rem" nomeGrafico="pie" dadosGrafico={graph1} anoGrafico={getYearValue} style={{height:"100%"}}/>
</Card>
```

Fonte: Autoral

A classe Header.jsx é a responsável por ser o componente de cabeçalho que a aplicação utiliza e é chamada em cada página. O elemento principal exportado por ela é o componente Toolbar do PrimeReact, nesse caso é utilizado os três lados da barra (start, center e end). como é possível verificar na Figura 3.4.10.

Figura 3.4.10 - Detalhe da Classe Header

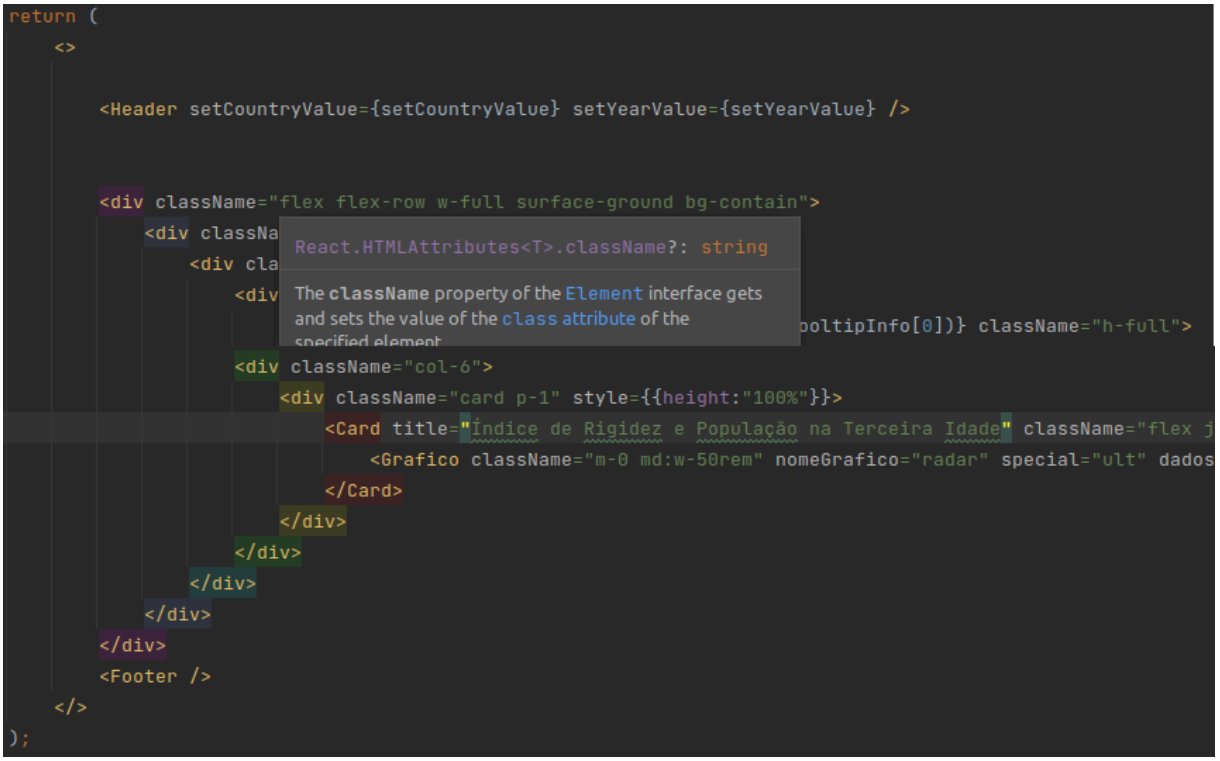
```
useMountEffect( effect: () => {
    api.get( url: '/countries/all/formatted').then((data :AxiosResponse<any> )=>setCountries(data.data))
    console.log('mounted')
})
useEffect( effect: ()=>{
    setNodes(countries)
    setSelectedNodeKey( value: '1')
}, deps: [countries])
useEffect( effect: ()=>{
    if(props.setCountryValue!=null&&props.setYearValue!=null){
        props.setCountryValue(selectedNodeKey)
        props.setYearValue(value)
    }
}, deps: [value, selectedNodeKey])
return(
    <>
        <Toolbar start={startContent} center={centerContent} end={endContent} />
    </>
```

Fonte: Autoral

Nos seus imports é possível notar vários componentes de formulário do PrimeReact, como botões e menus, no entanto um componente de destaque é o useMountEffect. Este componente do PrimeReact é na verdade uma personalização de um dos Hooks do React, cuja função é executar código apenas uma vez quando o componente que o utiliza é montado. Ainda no Header é possível notar a declaração de várias constantes que fazem uso do useState e a definição das três constantes mais importantes para esse componente, as que vão definir o conteúdo que aparece no Toolbar, elas foram definidas como: startContent, centerContent e endContent. A primeira leva o componente Sidebar, uma barra lateral com as informações e links de navegação para as outras páginas da aplicação, e um botão responsável por fazer esta barra aparecer. A segunda constante leva o um componente de menu chamado TreeSelect, que carrega os valores dos países e possibilita ao usuário selecionar de qual país ele deseja ver as informações. A terceira constante é um botão que alterna o ano a qual os dados da página vão se referir, as opções sendo 2020 e 2021. Ainda nesta classe de cabeçalho temos a utilização do useMountEffect que aqui é responsável por carregar os dados dos países da API assim que este componente é montado e também a utilização de dois useEffect, o primeiro tem como gatilho a variável que guarda os países, essa ao mudar do seu valor original (que é null) faz com que a barra selecione automaticamente o país de id 1 (isso é para que no primeiro carregamento a página não fique sem carregar dados, uma vez que ela depende de um país ter sido selecionado nessa barra), o segundo useEffect atualiza os valores do país e do ano dentro da variável props, isso tem o propósito de passar esses valores um nível acima para serem utilizados nas páginas que utilizam este componente (Figura 3.4.10).

Das páginas, a landing page da aplicação é dada pela classe PaginaVite.jsx. Os imports dessa classe contém Toolbar, Button, Card e Image que são componentes do PrimeReact e um componente Footer criado pelo autor. Uma vez que essa é uma página o seu retorno leva muito mais elementos do que uma classe de componente. E nesse caso específico o retorno estrutura o Toolbar e divs utilizando instâncias de Cards e no final o componente Footer (Figura 3.4.11).

Figura 3.4.11 - Detalhe do retorno da Pagina 1



Fonte: Autoral

A classe Pagina1.jsx representa a primeira visão do sistema onde são exibidas informações sobre covid-19 e índices econômicos. Ela faz uso de vários componentes já citados como useEffect, useState, Grafico, Header, Footer e um componente Tooltip do PrimeReact que serve para adicionar uma janela com informações acerca de algum card ou elemento da página. As constantes que utilizam useState definidas nesta classe armazenam os valores dos índices econômicos que são obtidos através da API (Figura 3.4.12).

Figura 3.4.12 - useStates utilizados na Pagina 1

```
function Pagina1() {
    const [getCountryValue, setCountryValue] = useState( initialState: null)
    const [getYearValue, setYearValue] = useState( initialState: null)
    const [giniIndex, setGiniIndex] = useState( initialState: null)
    const [giniPanel, setGiniPanel] = useState( initialState: null)
    const [below50Index, setBelow50Index] = useState( initialState: null)
    const [below50Panel, setBelow50Panel] = useState( initialState: null)
    const [mphrIndex, setMphrIndex] = useState( initialState: null)
    const [mphrPanel, setMphrPanel] = useState( initialState: null)
    const [mphrFemaleIndex, setMphrFemaleIndex] = useState( initialState: null)
    const [mphrFemalePanel, setMphrFemalePanel] = useState( initialState: null)
    const [mphrMaleIndex, setMphrMaleIndex] = useState( initialState: null)
    const [mphrMalePanel, setMphrMalePanel] = useState( initialState: null)
    const [nationalPLinesIndex, setNationalPLinesIndex] = useState( initialState: null)
    const [nationalPLinesPanel, setNationalPLinesPanel] = useState( initialState: null)
```

Fonte: Autoral

Esta classe utiliza dois useEffects, o primeiro é responsável por pegar o valor de cada índice da API e preencher suas respectivas variáveis, o gatilho desta função é a variável getCountryValue que é atualizada devido a utilização da função setCountryValue como parâmetro do componente Header, como explicado anteriormente, dentro deste componente a função é utilizada para atualizar o getCountryValue quando o usuário seleciona outro país na barra de exibição dos países. O segundo useEffect é responsável por atualizar o valor dos painéis quando o usuário seleciona o ano de exibição dos dados, além disso ele também faz a atualização dos painéis quando qualquer uma das variáveis que guardam os índices é atualizada (Figura 3.4.13).

Figura 3.4.13 - Detalhe da primeira função useEffect utilizada

```
useEffect( effect: () => {
    if(getCountryValue!=null){
        if(getCountryValue.valueOf()<900) {
            api.get( url: '/poverty/' + getCountryValue + '/SI.POV.GINI').then((ret : AxiosResponse<any> ) => setGiniIndex(ret.data))
            api.get( url: '/poverty/' + getCountryValue + '/SI.DST.50MD').then((ret : AxiosResponse<any> ) => setBelow50Index(ret.data))
            api.get( url: '/poverty/' + getCountryValue + '/SI.POV.MDIM').then((ret : AxiosResponse<any> ) => setMphrIndex(ret.data))
            api.get( url: '/poverty/' + getCountryValue + '/SI.POV.MDIM.FE').then((ret : AxiosResponse<any> ) => setMphrFemaleIndex(ret.data))
            api.get( url: '/poverty/' + getCountryValue + '/SI.POV.MDIM.MA').then((ret : AxiosResponse<any> ) => setMphrMaleIndex(ret.data))
            api.get( url: '/poverty/' + getCountryValue + '/SI.POV.NAHC').then((ret : AxiosResponse<any> ) => setNationalPLinesIndex(ret.data))

            api.get( url: '/poverty/graph/' + getCountryValue).then((ret : AxiosResponse<any> ) => setGraph1(ret.data))
        }
    }
}, deps: [getCountryValue]);
```

Fonte: Autoral

Os Tooltips foram utilizados aproveitando uma função do componente Card, onde é possível separar o mesmo em cabeçalho, conteúdo e rodapé, aqui foi utilizado a função de cabeçalho. Criou-se uma função chamada headerInfo que recebe uma string chamada text, dentro desta função há um retorno com as configurações do Tooltip onde sua mensagem é o conteúdo da variável text (Figura 3.4.14).

Figura 3.4.14 - Exemplo da implementação dos Tooltips

```
function headerInfo(text){
    return (
    <>
        <Tooltip target=".custom-target-icon" />
        <i className="custom-target-icon pi pi-info-circle p-text-secondary p-overlay-badge"
            data-pr-tooltip={text}
            data-pr-position="right"
            data-pr-at="right+5 top"
            data-pr-my="left center-2"
            style={{ fontSize: '1rem', cursor: 'pointer' , left: '45%' }}></i>
    </>)
}
const tooltipInfo = ["O índice GINI mede o tanto que a distribuição de renda ou o consumo entre indivíduos ou lares des
"Esse indicador é uma medida da pobreza relativa a nível nacional. Mede o quão longe os indivíduos estão do padrão medi
"como desemprego, moradia precária, saúde inadequada e barreiras no acesso à educação.",
"A proporção de pessoas (dentro de uma dada população) que experimentam múltiplas deprivações financeiras e sociais.",
"A proporção de mulheres (dentro de uma dada população) que experimentam múltiplas deprivações financeiras e sociais.",
"A proporção de homens (dentro de uma dada população) que experimentam múltiplas deprivações financeiras e sociais.",
"Razão de pessoas cuja renda está inserida abaixo da linha de pobreza nacional"]
```

Fonte: Autoral

Fora desta função foi criado um vetor constante de strings com todos os textos para serem utilizados nos tooltips. No retorno da classe, onde o componente Card é utilizado, em seu parâmetro header é utilizada a função headerInfo e o parâmetro passado para ela é o tooltipInfo em sua posição correspondente à mensagem desejada (Figura 3.4.15).

Figura 3.4.15 - Utilização dos Tooltips

```
<Card title="Índice GINI" header={headerInfo(tooltipInfo[0])} className="h-full">
    <p>{giniPanel}</p>
</Card>
```

Fonte: Autoral

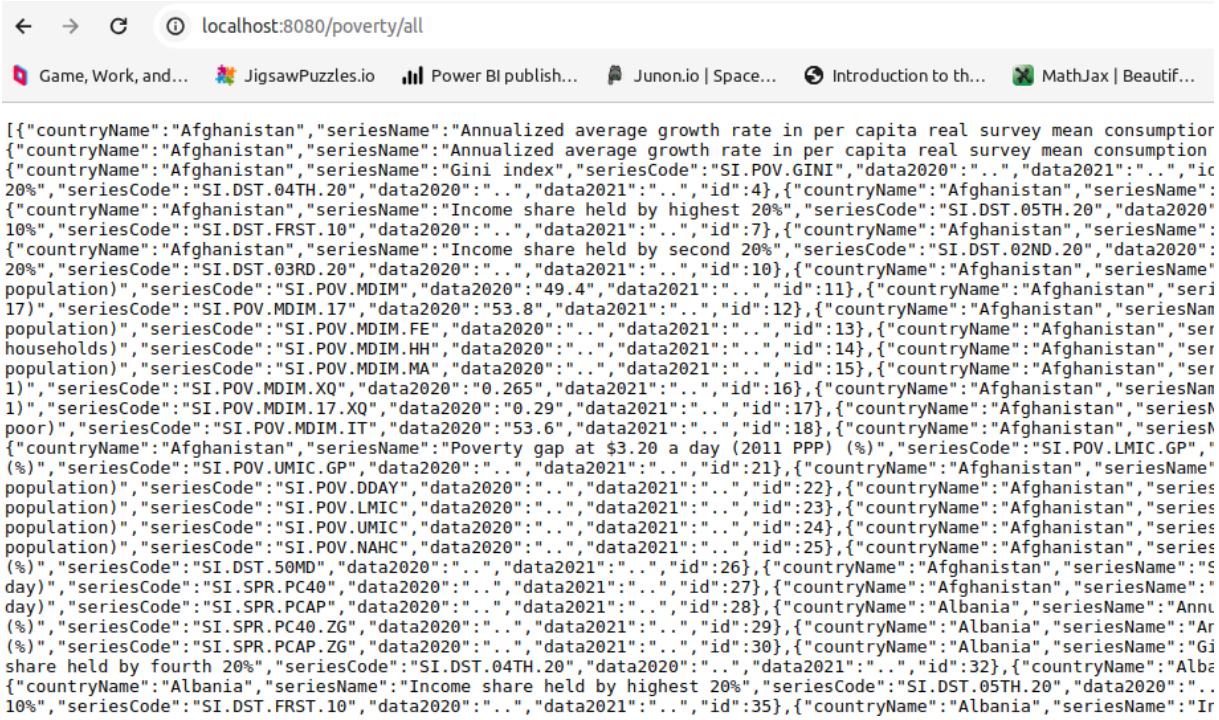
O retorno desta classe monta a página correspondente à primeira visualização da aplicação fazendo uso dos painéis e do componente Gráfico nos seus elementos, e o componente Footer no final (Figura 3.4.11). A classe Pagina2.jsx corresponde a segunda página com uma visualização de dados utilizando painéis e gráficos com

dados de covid e índices econômicos. Sua implementação foi similar à primeira página com o uso de dois useEffects com o mesmo propósito da primeira página, a principal diferença se dá na disposição dos elementos visuais da página, os tipos de gráficos escolhidos e os dados consumidos da API.

3.5 Integração dos módulos e conclusão da aplicação

Como mencionado anteriormente a aplicação foi feita em camadas que se comunicam mas podem ser hospedadas em servidores diferentes. A API (backend) se comunica com o banco de dados através do Spring-Data (que por sua vez utiliza o Hibernate por debaixo dos panos). Já a aplicação Web (o front feito em React) obtém os dados necessários para preencher seus elementos através de chamadas para os endpoints da API. Os dados da API são disponibilizados em texto simples no formato JSON (Figura 3.5.1).

Figura 3.5.1 - Exemplo de retorno da API

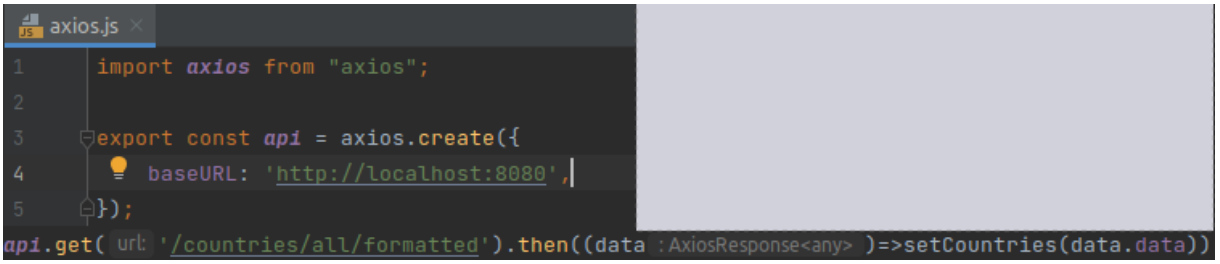


Fonte: Autoral

Para chamar esses endpoints e utilizar estes dados corretamente, a aplicação Web utiliza um componente chamado axios. O axios é uma biblioteca de requisições HTTP baseada em promessas (promises) que funciona tanto com requisições feitas

ao browser (através de URL) quanto com NodeJs, no caso desta aplicação seu uso se dá por URLs de endpoints. Para utilizá-lo foi necessário primeiramente instalar no projeto por meio do NPM. Em seguida criou-se uma classe chamada axios.js que funciona como uma espécie de factory (padrão de projeto), nela foi importado o axios da biblioteca de mesmo nome e exportado um componente chamado api onde cria-se uma instância do axios com a url base da API que se deseja enviar requisições. Esse componente api vem com as requisições básicas HTTP como get, post, put, patch e delete. Na aplicação Web ele foi principalmente utilizado para obter informações (através do get).

Figura 3.5.2 - Implementação do axios e exemplo de sua utilização

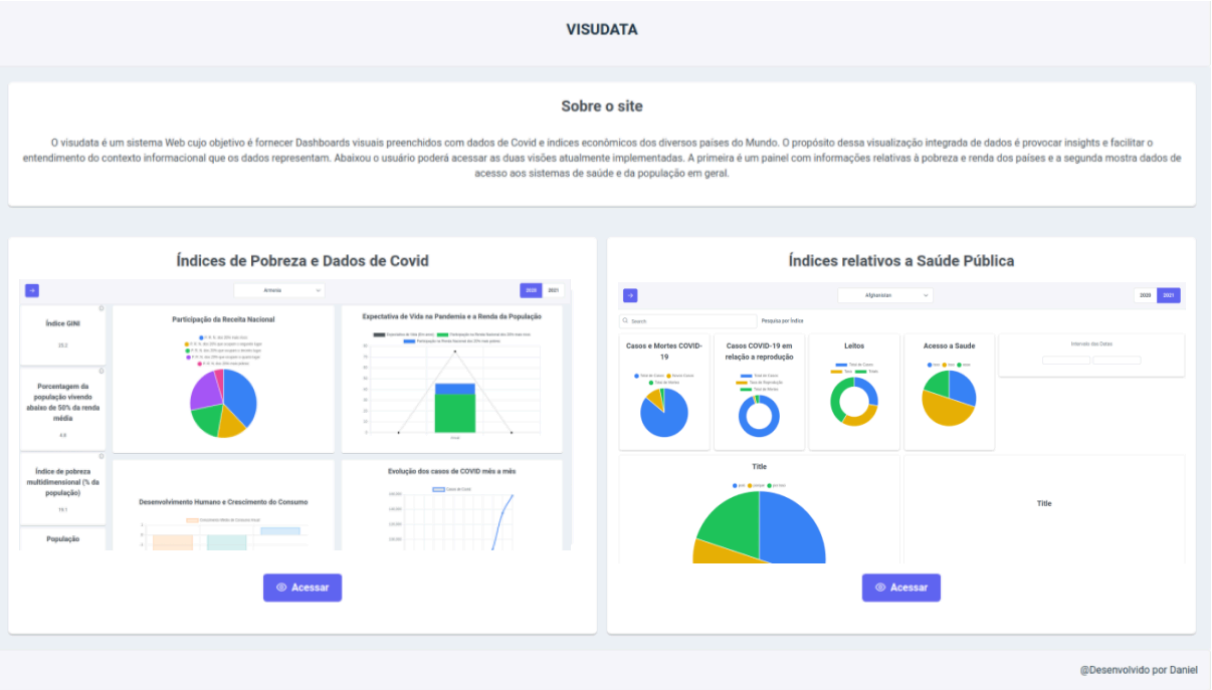


Fonte: Autoral

4 Resultados e Discussões

Para o usuário a aplicação consiste na primeira página (landing page) que apresenta a aplicação com o nome da mesma e a proposta que traz (apresentar dados de covid-19 e índices econômicos com o objetivo de facilitar *insights* através da visualização de dados). Os elementos desta página são o título, o texto de apresentação, abaixo têm-se dois painéis com uma pequena visualização das duas páginas com visões de dados, que se forem clicadas levarão à renderização dessas respectivas páginas, e mais abaixo há o elemento de rodapé (Figura 4.1).

Figura 4.1 - Primeira Página da Aplicação

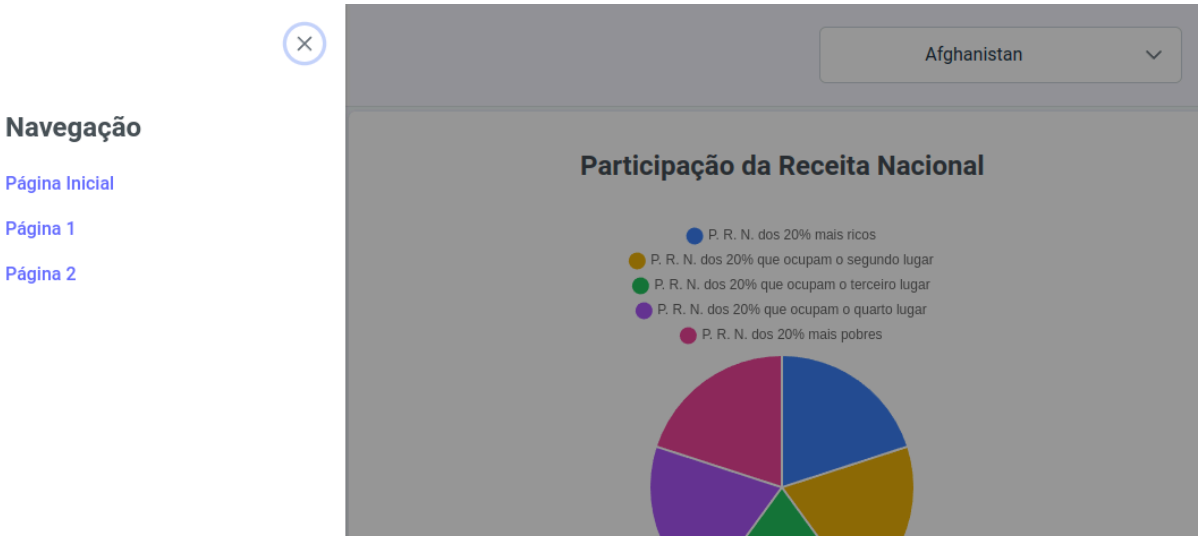


Fonte: Autoral

Se o usuário clicar no primeiro painel (o que se encontra no lado esquerdo na Figura 4.1) ele será direcionado para a página com a primeira visão de dados. Esta nova página contém a visualização de dados de covid em conjunto com os dados de índices econômicos. No topo da página é possível notar o elemento de cabeçalho com um botão de seta em seu lado esquerdo, uma barra de menu no meio e dois botões que se alternam com os anos 2020 e 2021. A página é carregada com um país já selecionado no menu do meio do cabeçalho, isso faz com que os elementos da visualização (painéis e gráficos) sejam preenchidos com os dados relacionados àquele país. Se o usuário clicar em um dos anos (2020 e 2021) os dados mudarão

para mostrar as informações daquele país naquele determinado ano. Se o botão de seta for pressionado, um menu lateral com links para navegação entre as páginas irá aparecer, através dele o usuário poderá retornar a página principal ou seguir para a outra página de visualização (Figura 4.2).

Figura 4.2 - Visualização do menu lateral

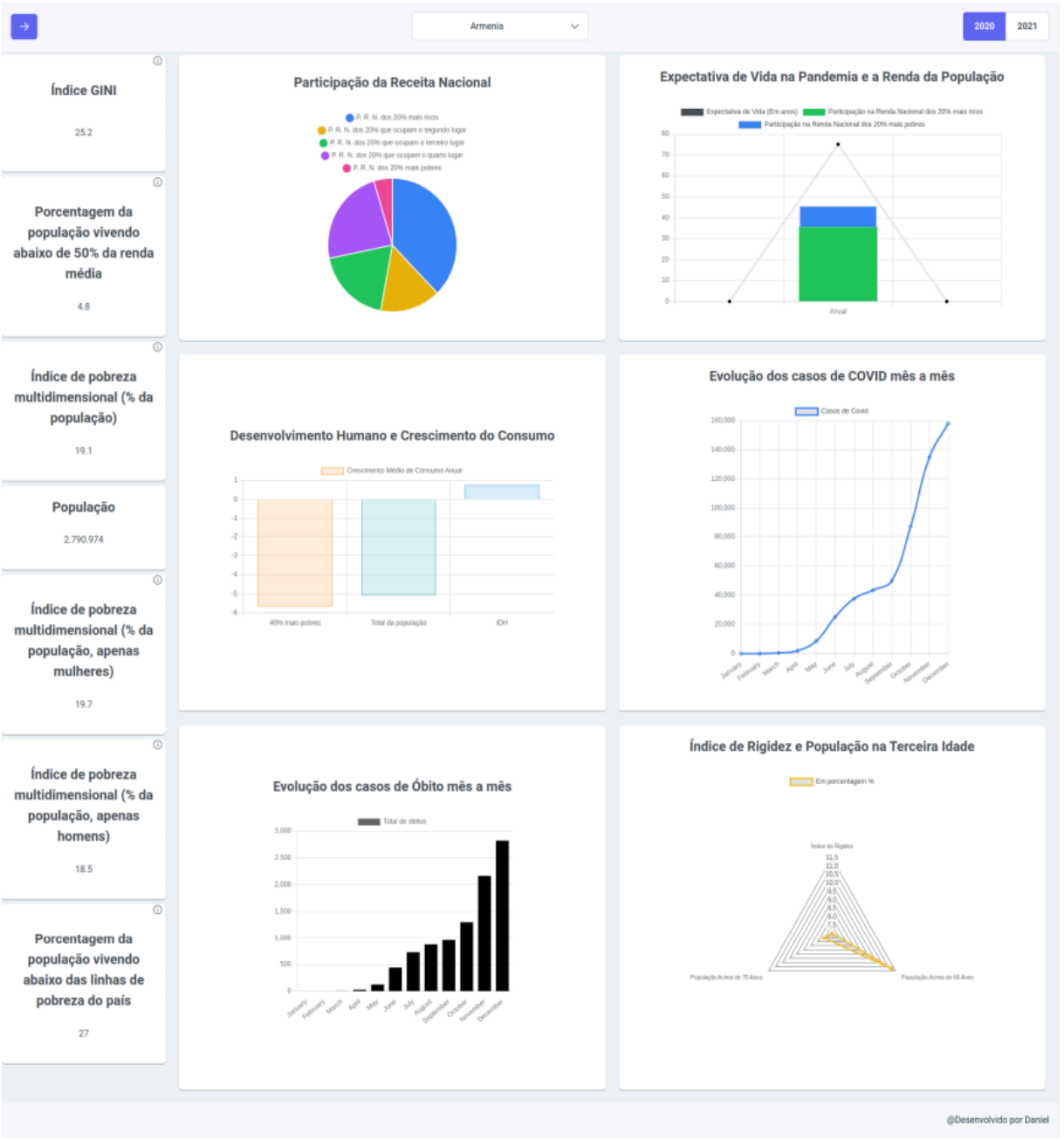


Fonte: Autoral

Essa visualização é composta de uma coluna de painéis alinhados do lado esquerdo, estes painéis trazem informações como Índice de Gini do país, população, percentual da população abaixo da linha de pobreza entre outros índices relacionados à pobreza do país. Vale ressaltar que as visualizações desta aplicação trazem mais de um tipo de dado em seus dashboards, como no caso desta primeira visualização que traz índices de pobreza e informações de covid-19, e alguns gráficos com dados relacionados, este comportamento é intencional e tem o propósito de facilitar a obtenção de *insights* e o entendimento dos dados relacionados. Estes painéis possuem um pequeno ícone de informação onde ao passar-se o mouse sobre o mesmo automaticamente aparece uma janela com informações acerca do índice econômico deste determinado painel. No centro da página existem seis gráficos com informações de covid-19 e sua relação com algum índice de pobreza. O primeiro é um gráfico de pizza com os dados de renda de cada camada social divididas a cada 20% (começando pelos 20% mais ricos até os 20% mais pobres totalizando 5 camadas). O segundo gráfico traz a renda dos 10% mais

ricos e dos 10% mais pobres e mostra a expectativa de vida neste país. O terceiro gráfico mostra a média de crescimento anual do país e o índice de desenvolvimento humano. O quarto gráfico mostra o total de casos de covid-19 evoluindo mês a mês. O quinto mostra o total de mortes evoluindo mês a mês. O sexto gráfico traz o índice de rigidez (stringency index) e a taxa de cidadão com 65 anos ou mais e com 70 anos ou mais (Figura 4.3).

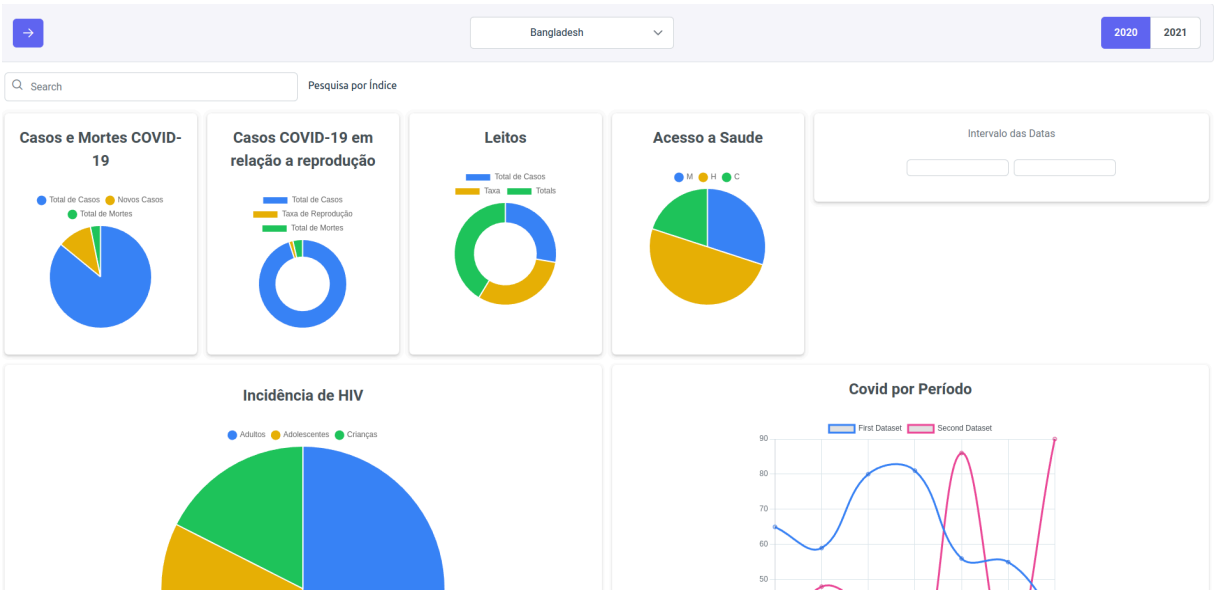
Figura 4.3 - Visualização dos índices de Pobreza e Covid



Fonte: Autoral

Caso o usuário chame o menu lateral e clique no link para a página 2, será carregada uma página com um layout ligeiramente diferente da primeira página. O cabeçalho continua o mesmo exibindo o mesmo comportamento de controlar a que país e ano se referem os dados exibidos pelos painéis e gráficos. No entanto A disposição dos mesmos segue outro sentido de layout, e os dados exibidos em seus painéis são referentes aos dados de covid-19 em conjunto com dados de condições de saúde dos países, como expectativa de vida, taxa de mortalidade, prevalência de doenças crônicas como diabetes e infecciosas como AIDS, densidade populacional e quantidade de leitos de hospitais para cada mil pessoas.

Figura 4.4 - Visualização dos índices de Saúde e Covid



Fonte: Autoral

5 Conclusões

O trabalho em questão apresentou uma aplicação para visualização de dados e propôs que a mesma é de suma importância para a obtenção de insights para os pesquisadores e o público em geral que se interessem pela covid-19 e a economia dos países e estejam buscando achar padrões no relacionamento desses dados através da visualização dos mesmos, também é possível que a aplicação seja utilizada como uma forma de facilitar o entendimento desses dados após a utilização de algoritmos de big data.

Neste trabalho foi conceituado visão e visualização de dados, índices econômicos, Covid-19 e o seu impacto global. Além disso, também foram apresentados conceitos referentes a aplicações Web, APIs, banco de dados, padrões de projetos e as ferramentas relevantes para a elaboração da aplicação *visudata*.

Na metodologia apresentou-se o passo a passo na elaboração da aplicação, a partir do banco de dados, passando pela API que alimentou a aplicação Web e suas visualizações. Mostrou-se as funcionalidades da aplicação e suas visões, bem como a forma como o usuário interage com a mesma. A partir daí depreende-se que a aplicação *visudata* cumpre com o seu objetivo de relacionar os dados de covid-19 dos países e seus respectivos dados econômicos. Claro que ela não é uma solução completa, no entanto, mostra-se como alternativa à utilização de softwares pré-prontos e inchados que muitas vezes custam valores que nem todas as empresas (e pessoas) podem ou querem pagar.

A sua implementação mostra que é possível até mesmo para uma equipe pequena de desenvolvedores, fazer uma aplicação de visualização de dados satisfatória e que pertence 100% à equipe (ou à empresa) que a construir. Esta aplicação também serve de modelo inicial para aqueles que estiverem interessados em construir suas próprias aplicações de visualização de dados.

Para o autor, a elaboração desta aplicação e a sua posterior implementação serviu para a aplicação de muitos dos conceitos aprendidos no curso de ciência da computação, principalmente aqueles relacionados a aplicações Web, mas não encerrando-se à eles. Também foi importante para mostrar as funcionalidades do

React que é um ecossistema de front-end muito versátil e gratuito com várias aplicações no mercado global.

É possível que no futuro essa aplicação possa ser refeita utilizando-se outras tecnologias, ou padrões de projetos mais modernos. Do jeito que está ela também pode ser adaptada para a exibição de outros tipos de dados que não seja covid ou dados econômicos, desde que se tenha um dataset com dados numéricos.

.

Referências

BALDONADO, M. Q. W.; WOODRUFF, A.; KUCHINSKY, A. **Guidelines for Using Multiple Views in Information Visualization.** Proceedings of the working conference on Advanced visual interfaces. Palermo, Italy: ACM Press. 2000. p. 110-119. Disponível em: https://dl.acm.org/doi/pdf/10.1145/345513.345271

BIBLIOTECA VIRTUAL EM SAÚDE MS. Covid 19. 2021. Disponível em: https://bvsms.saude.gov.br/covid-19-2/

BRAGA, T. E.; ALVES, L. A.; LEITE, N. C. Ferramentas de visualização de dados e informações e suas características. , . Disponível em: http://hdl.handle.net/20.500.11959/brapci/195019. Acesso em: 25 out. 2023.

BUSCHMANN , et al. **Pattern-Oriented Software Architecture Volume 1**: A System of Patterns. [S.l.]: Wiley, v. 1, 1996.

CARD, S. K.; MACKINLAY, J. D.; SCHNEIDERMAN, B. Readings in **Information Visualization: Using Vision to Think**. San Francisco (California), Morgan Kaufmann Publishers, Inc. 1999.

CEPALSTAT. *Statistical Databases and Publications*. Sustainable Development Goals >> Indicator 10.2.1 Proportion of people living below 50 per cent of median income, by sex, age and persons with disabilities. Disponível em: https://statistics.cepal.org/portal/cepalstat/technical-sheet.html?lang=en&indicator_id=4213

CORRÊA, F. C. **Visualização de dados:** passado, presente e futuro | data vizualization: past, present and future. Liinc em revista, v. 15, n. 2, 2019. Disponível em: http://revista.ibict.br/liinc/article/view/4812

DAVENPORT, T., PRUSAK, L. **Ecologia da informação:** por que só a tecnologia não basta para o sucesso na era da informação. São Paulo: Futura, 1998. Disponível em: https://ppgic.files.wordpress.com/2018/07/davenport-t-h-2002.pdf.

DEITEL, PAUL. Java: como programar / Paul Deitel. Harvey Deitel; tradução Edson Furmankiewicz; revisão técnica Fabio Lucchini. – São Paulo: Pearson Education do Brasil, 2017.

ILIINSKY, N.; STEELE, J. Designing data visualizations. Intentional communication from data to display. 2011.

KNAFLIC, C. N. **Storytelling com dados:** Um guia sobre visualização de dados para profissionais de negócios. 2nd Edition. Rio de Janeiro: Alta Books, 2019.

MAGAZZINO, C.; MELE, M.; MORELLI, G. **The relationship between renewable energy and economic growth in a time of covid-19**: A machine learning experiment on the brazilian economy. Sustainability, v. 13, n. 3, 2021. ISSN 2071-1050. Disponível em: https://www.mdpi.com/2071-1050/13/3/1285.

MANKIW, N. GREGORY. (1998). *Principles of economics*. Forth Worth, TX: Dryden Press. ISBN 0-03-098238-3. OCLC 37611615

RIBEIRO, D. M. **Visualização de dados na Internet**. 2009.132 f. Dissertação (Mestrado em Tecnologias da Inteligência e Design Digital) -Pontifícia Universidade Católica de São Paulo, São Paulo, 2009. Disponível em: http://www.danielmelo.net/wp-content/uploads/2009/03/dissertacao_final.pdf .Acesso em:

RODRIGUES, A. A.; DIAS, G. A. Estudos sobre visualização de dados científicos no contexto da data science e do big data. , . DOI: 10.22478/ufpb.1981-0695.2017v12n1.34774 Acesso em: 25 nov. 2023.

SANTOS, H. M. D.; FLORES, D. A obsolescência do conhecimento em preservação digital. , p. 41-58, . 2018. Disponível em: http://hdl.handle.net/20.500.11959/brapci/36461. Acesso em: 20 dez. 2023.

SETZER, V. W. **Dado, informação, conhecimento e competência. DataGramaZero**, v. 0, n. 0, 1999. Disponível em: https://www.ime.usp.br/~vwsetzer/datagrama.html. Acesso em:

SILVA, William dos Santos da. **Visualização de dados:** análise dos procedimentos para proporcionar visualização de dados para agências de fomento à ciência e tecnologia. 2022.

SILVA JUNIOR, Jairo de Jesus Nascimento da. **Uma arquitetura orientada a serviços para visualização de dados em dispositivos inteligentes**. 2014. 77 f. Dissertação (Mestrado) - Universidade Federal do Pará, Instituto de Ciências Exatas

e Naturais, Belém, 2014. Programa de Pós-Graduação em Ciência da Computação. Disponível em: https://repositorio.ufpa.br/jspui/handle/2011/9015.

SOUZA, Lucelia Lima. Descoberta de conhecimento nas bases de dados da pandemia da COVID- 19 e de indicadores socioeconômicos e ambientais. 2023. 83 f. Dissertação (Programa de Pós-Graduação em Ciência da Computação/CCET) - Universidade Federal do Maranhão, São Luís, 2023. Disponível em: https://tedebc.ufma.br/jspui/handle/tede/4731

WOLFFENBÜTTEL, ANDRÉA. (2004). O que é? - Índice de Gini. Revista Desafios do Desenvolvimento, 2004, Ano 1, Edição 4 - 1/11/2004. Disponível em: https://www.ipea.gov.br/desafios/index.php?option=com_content&id=2048:catid=28#:~:text=O%20que%20%C3%A9%3F%20%2D%20%C3%8Dndice%20de%20Gini&text=O%20%C3%8Dndice%20de%20Gini%2C%20criado,apresentam%20de%20zero%20a%20cem).

WORLD HEALTH ORGANIZATION. Coronavirus Disease (COVID-19). 2023. Disponível em: https://www.who.int/emergencies/diseases/novel-coronavirus-2019/question-and-answers-hub/q-a-detail/coronavirus-disease-covid-19