UNIVERSIDADE ESTADUAL DA PARAÍBA

CAMPUS CAMPINA GRANDE

CENTRO DE CIÊNCIAS E TECNOLOGIA

DEPARTAMENTO DE COMPUTAÇÃO

CURSO DE CIÊNCIA DA COMPUTAÇÃO

MATEUS CAVALCANTE HONORATO DA SILVA

ARQUITETURA DE SOFTWARE PARA BIG DATA EM OBESIDADE INFANTIL

CAMPINA GRANDE

2022

MATEUS CAVALCANTE HONORATO DA SILVA

ARQUITETURA DE SOFTWARE PARA BIG DATA EM OBESIDADE INFANTIL

Trabalho de Conclusão de Curso ou apresentado ao Programa de Graduação em Ciência da Computação da Universidade Estadual da Paraíba, como requisito parcial à obtenção do título de Bacharel em Computação.

Orientador: Prof. Dr. Paulo Eduardo e Silva Barbosa

CAMPINA GRANDE

2022

S586a Silva, Mateus Cavalcante Honorato da.
Arquitetura de software para big data em obesidade infantil [manuscrito] / Mateus Cavalcante Honorato da Silva. - 2022.
60 p. : il. colorido.

Digitado.
Trabalho de Conclusão de Curso (Graduação em Computação) - Universidade Estadual da Paraíba, Centro de Ciências e Tecnologia , 2022.
"Orientação : Prof. Dr. Paulo Eduardo e Silva Barbosa , Coordenação do Curso de Computação - CCT."

1. Arquitetura de software. 2. Obesidade infantil. 3. Big data. I. Título

21. ed. CDD 005.1

Elaborada por Deivid A. M. Neves - CRB - 15/899 BC/UEPB

MATEUS CAVALCANTE HONORATO DA SILVA

ARQUITETURA DE SOFTWARE PARA BIG DATA EM OBESIDADE INFANTIL

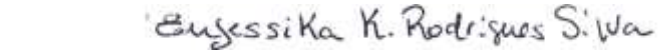
Trabalho de Conclusão de Curso ou apresentado ao Programa de Graduação em Ciência da Computação da Universidade Estadual da Paraíba, como requisito parcial à obtenção do título de Bacharel em Computação.

Aprovado em: 22 de Abril de 2022.

BANCA EXAMINADORA

Prof. Dr. Paulo Eduardo e Silva Barbosa (Orientador)

Universidade Estadual da Paraíba (UEPB)



Prof. Me. Eujessika Katielly Rodrigues Silva

Universidade Estadual da Paraíba (UEPB)

Prof. Dr. Fábio Luiz Leite Jr

Universidade Estadual da Paraíba (UEPB)

Dedico este trabalho a meus pais, Lina e Gersonilson, que sempre me incentivaram a estudar e me dedicar em tudo que me propusesse a fazer e a todos os professores que me auxiliaram nessa jornada.

AGRADECIMENTOS

Quero agradecer

Aos meus pais, Lina Cavalcante e Gersonilson Honorato que sempre me incentivaram nos estudos e a quem devo tudo que sou.

A todos do departamento de computação da UEPB que me auxiliaram diretamente a conquistar esse primeiro sonho.

A todos os professores que contribuíram para que pudesse chegar até aqui.

A meu primo Pedro Cavalcante que sempre me auxiliou e me incentivou muito a seguir para área de estudo que eu queria.

A Estêvão Arruda que foi um grande amigo que ganhei ao longo do curso e que esteve junto comigo em muitos estudos e desenvolvimento de projetos e trabalhos.

A todos os colegas e amigos da UEPB que estiveram junto comigo em cada período.

A todos os amigos da UFCG que estiveram comigo tanto em farras quanto me auxiliando de uma forma ou de outra no meu curso.

Ao Prof. Fabiano Cavalcante, Prof. John Kennedy e todos do Laboratório de Engenharia e Pavimentos da UFCG que me deram a oportunidade de adquirir muita experiência e conhecimento em meu estágio.

A todos que participaram desta minha jornada, agradeço.

RESUMO

O uso de tecnologias vem impactando diversas áreas atualmente, trazendo soluções e propondo melhorias diariamente para médicos, professores, engenheiros e diversas profissões presentes na sociedade. Tem-se então como proposta deste trabalho, apresentar a construção de uma arquitetura de um sistema web necessária no desenvolvimento do sistema "Brasil2030" que é uma ferramenta que tem como objetivo promover o engajamento das famílias e de diferentes setores e atores da sociedade a proteção da criança e promoção de ambientes favoráveis ao seu desenvolvimento e bem estar. Este trabalho apresenta um sistema com arquitetura escalável para diversas cidades e grandes quantidades de dados, buscando também o uso de novas tecnologias (MongoDB, Node, React) que sejam bem adaptadas em uso de big data. Como uma amostra do que a ferramenta pode oferecer, ao final do trabalho é apresentado resultados dos testes na cidade de Tupã-SP e também resultados da pesquisa de satisfação feita com usuários após o teste. O "Brasil2030", nome dado ao sistema, foi participante e co-vencedor do Hackathon Global de Obesidade Infantil promovido pela SWELife.

Palavras-Chave: Arquitetura de software, Tecnologia, Obesidade infantil, Big Data.

ABSTRACT

The use of technologies is currently impacting several areas, bringing solutions and proposing improvements daily for doctors, teachers, engineers and various professions present in society. Therefore, the proposal of this work is to present the construction of an architecture of a web system necessary in the development of the "Brasil2030" system, which is a tool that aims to promote the engagement of families and different sectors and actors of society to protect of the child and the promotion of favorable environments for their development and well-being. This work presents a system with a scalable architecture for several cities and large amounts of data, also seeking the use of new technologies (MongoDB, Node, React) that are well adapted to use big data. As a sample of what the tool can offer, at the end of the work, test results are presented in the city of Tupã-SP and also results of the satisfaction survey carried out with users after the test. "Brasil2030", the name given to the system, was a participant and co-winner of the Global Hackathon on Childhood Obesity promoted by SWELife.

Keywords: Software Architecture, Technology, Childhood Obesity, Big Data.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
CADÚNICO	Cadastro Único
CSS	Cascading Style Sheets
ERP	Enterprise Resource Planning
ETL	Extract, Transform, Load
HTML	HyperText Markup Language
IMC	Índice de massa corporal
INTERSSAN	Centro de Ciência, Tecnologia e Inovação para Soberania e Segurança Alimentar e Nutricional
IOT	Internet of Things
JSON	JavaScript Object Notation
JSX	JavaScript XML
MVC	Model-View-Controller
NoSQL	No Structured Query Language
NPM	Node Package Manager
OMS	Organização Mundial da Saúde
REST	Representation State Transfer
SGBDR	Sistema de Gerenciamento de Bancos de Dados Relacionais
SISVAN	Sistema de Vigilância Alimentar Nutricional
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SUS	Sistema Único de Saúde
URI	Unified Resource Identifier
W3C	World Wide Web Consortium
WSDL	Web Services Description Language
XML	eXtensible Markup Language

SUMÁRIO

1 INTRODUÇÃO

A evolução das tecnologias da informação e forte migração de diversas áreas do mercado de trabalho para o mundo digital, vem trazendo cada vez mais demanda e complexidade para o âmbito. Desde a apresentação dos nove pilares da Quarta Revolução Industrial (termo criado em 2011 durante Feira de Hannover na Alemanha) e a demonstração de melhorias que o uso desses pilares poderiam tornar o processo de produção das indústrias mais eficiente, cada vez mais softwares são criados buscando interação com algum ou mais destes pilares: Big Data e Análise de Dados, Robótica, Simulação, Internet das Coisas (IoT), Cibersegurança, Cloud Computing, Manufatura Aditiva, Sistemas de Integração Horizontal e Vertical e Realidade Aumentada.

Com o uso de princípios básicos dessas novas tecnologias foi-se percebendo que é possível conectar máquinas, criar redes inteligentes com eficiência e autonomia, permitindo que os processos de produção de uma fábrica ou principais assuntos do momento no mundo, sejam monitorados de forma remota por uma única pessoa de qualquer parte do mundo em tempo real em um simples smartphone por exemplo (Equipe ESSS. 2017).

Destes nove pilares, uma parte fundamental, e que permite estimar desde a produção futura de uma fábrica até o estado nutricional de uma criança em sua cidade, é a Análise de Dados e o Big Data.

O Big Data é apresentado com grande volume e variedade de dados, que por sua vez podem ser coletamos desde sensores ERPs a comentários nas mídias sociais. Esta variedade de dados muda a maneira de se analisar dados de forma radical. Além da variedade e volume, Taurion (2013) ainda adiciona outros três quesitos que complementam e que são relevantes para desenvolvimento de qualquer aplicação com Big Data: a veracidade dos dados, o valor para o negócio - uma vez que analisar dados que não tem significado ou é sujeira deixa de agregar valor - e a velocidade, que tem sido cada vez mais um critério importante para suprir o grande volume de dados que cresce constantemente.

> Estima-se que dos 1,8 zetabytes (1021 bytes) gerados em 2012 pularemos para 7,9 zetabytes em 2015. Na verdade, cerca de 90% dos dados que existem hoje foram criados nos últimos dois anos. [...] Em 2000, apenas 25% dos dados do mundo estavam armazenados em formato digital e em 2007 já eram 94%. Hoje deve estar bem próximo dos 99,9%. Não significa que não existam mais dados em formatos analógicos, como em papel, mas o volume de dados digitalizados cresce de forma assombrosa. (Taurion. 2013. p. 26)

Neste aspecto, o Big Data começou a constatar na prática que o imenso volume de dados gerados a cada dia, excedeu a capacidade das tecnologias que os tratavam adequadamente. É fato que isso acabou impulsionando a surgir softwares de banco de dados NoSQL como o Big

Table da Google, DynamoDB da Amazon e MongoDB, fazendo também com que novos frameworks surjam se adequando a essas novas tecnologias, buscando otimizar o tempo de coleta, análise e tratamento dados (Taurion. 2013. p. 27).

Cloud Computing é também um impulsionador para Big Data, pois podem-se usar nuvens públicas para suportar imensos volumes de dados e as características de elasticidade das nuvens permitem que acionemos servidores virtuais sob demanda, apenas no momento de tratar estes dados. Na prática, o modelo de Cloud Computing faz muito sentido para as atividades de Big Data, dada a imprevisibilidade do montante de dados que poderá ser tratado.

Uma vez que se pode coletar dados a partir de várias fontes, é preciso garantir que um sistema possa utilizar dados com diferentes padrões de dados e consiga eliminar ruídos e incoerência com esses diversos volumes de dados. A arquitetura de um sistema necessita que seja pensada para que se atenda essa demanda, onde não apenas deva tratar a grande quantidade de dados mas também a velocidade e variedade de dados com que serão tratadas e levadas ao usuário final, assim como integrar sistemas legados com sistemas que usam tecnologias com dados relacionais a sistemas que usarão tecnologias diferentes como o Hadoop (plataforma de hardware voltada para processamento de grandes volumes de dados). Uso de computação paralela em grandes volumes de dados também devem ser considerados na arquitetura uma vez que é necessário definir que dados serão processados localmente e quais serão enviados para rodarem em algum algoritmo na nuvem.

Desse modo, utilizando Big Data em uma arquitetura que possa suprir a grande quantidade de processamento e análise de dados necessário, diferentes áreas da sociedade podem ser analisadas, adicionando a essa análise o auxílio de mentores e especialistas, pode ser construído um sistema mais precisão no seus objetivos.

1.1 PROBLEMA

O aumento da obesidade infantil está alarmando o mundo. Já é consenso que estamos diante de uma sindemia global de causas complexas e multidimensionais. Pais, educadores, cuidadores e gestores, mesmo com iniciativas de mercado para esse segmento da população não têm à disposição, um ambiente capaz de conectar diferentes dimensões do saber e dos atores desse processo, de forma a frear o avanço da obesidade infantil, principalmente em crianças com menos de seis anos (SOUZA, 2021, traduzido).

A partir de uma iniciativa sueca “SWELIFE: zero obesidade infantil na escola começa em 2030” tem-se como objetivo acabar com a obesidade infantil nas escolas até o ano de 2030, um grupo de pesquisadores do Centro de Ciência e Tecnologia em Soberania e Segurança Alimentar e Nutricional (INTERSSAN) aceitou um desafio de, apontando as necessidades para gerar a transformação da sociedade para a prevenção da obesidade infantil, através de uma solução tecnológica, avaliar o estado nutricional de uma criança e propor melhorias aos pais ou responsáveis pela criança (SOUZA, 2021, traduzido).

Desse modo, UEPB se juntou a esta causa e entramos nesse desafio com o objetivo de desenvolver essa solução tecnológica que possa ajudar a sociedade com a obesidade infantil, tendo como pergunta: “Como elaborar uma arquitetura de software que possa utilizar de Big Data na análise de Obesidade Infantil em determinadas cidades?”

1.2 OBJETIVOS

1.2.1 Objetivo Geral

Propor uma arquitetura de software adaptável a grandes volumes de dados (escalabilidade), a velocidade da coleta, extração e tratamento dos dados, a veracidade do dado e o valor que os dados das crianças de determinada cidade podem nos gerar e ainda a variedade de dados que poderão vir de diversas fontes.

1.2.2 Objetivos Específicos

Visando atingir o objetivo principal, alguns objetivos específicos são requeridos, entre eles:

- Analisar e decidir as tecnologias que serão usadas para a base de dados e desenvolvimento do sistema.
- Decidir a linguagem de programação, framework e estratégias de comunicação com a base de dados, necessários para desenvolvimento de um protótipo.
- Desenvolver um protótipo que possa se comunicar com a base de dados e informar a um usuário final dados nutricionais de uma criança, escola ou cidade buscando passar dicas de como melhorar o estado da criança, quando não estiver adequado.
- Realizar uma discussão dos resultados apresentados no protótipo e pontos que poderão ser melhorados.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 ARQUITETURA BÁSICA DE UMA APLICAÇÃO WEB

Buscando entender um pouco sobre como funciona uma arquitetura web no geral, podemos definir que esta é composta por dois atores principais: o cliente e o servidor. O cliente normalmente, mas não necessariamente, é um navegador como o Google Chrome ou Firefox onde contém as páginas web com arquivos HTML, JavaScript e CSS. O servidor é uma aplicação, na forma de um serviço, normalmente hospedado remotamente.

O servidor aguarda uma comunicação vinda do navegador, que chega através de uma requisição, esta requisição é processada pelo servidor e uma resposta é gerada, podendo ser composta por HTML, XML, JSON e outros elementos dependendo da tecnologia utilizada. O navegador recebe esta resposta, a interpreta e interage com o usuário através do navegador.

2.2 ARQUITETURA DO SERVIDOR

Antes de começar a falar sobre a arquitetura em si utilizada com o big data, iremos entender um pouco sobre a arquitetura utilizada em sistemas web, que é onde grande parte do nosso sistema vai estar.

2.2.1 Arquitetura de Web Services

No final do ano 2000, após a criação do grupo W3C formado por empresas de tecnologias como Microsoft, IBM e HP, o grupo W3C teve como propósito desenvolver uma arquitetura que permitisse a interoperabilidade entre aplicações e sistemas de plataformas, ambientes e arquiteturas diferentes que eram presentes na web. Visando atender a essas necessidades, foi proposta uma arquitetura computacional voltada para serviços Web, que tornou possível a interação entre diferentes tipos de aplicações mesmo rodando em plataformas e sistemas operacionais diferentes.

A Arquitetura de Web Services surgiu para permitir a interoperabilidade entre aplicações rodando em diferentes plataformas. Foi especificada com base em um protocolo que encapsula as mensagens SOAP em uma linguagem que descreve as interfaces dos serviços, conhecida como WSDL. Na prática, em sistemas que seguem essa visão de Web Services, as mensagens de requisição e resposta são “envelopadas” pelo protocolo SOAP e os seus serviços são “descritos” em interfaces WSDL. Porém, um grande problema passou a ser observado por usuários desse tipo de arquitetura: a perda de desempenho das aplicações. Ao utilizar o

protocolo SOAP, é gerada uma camada de abstração que envolve o protocolo HTTP – quando na verdade ela não precisaria existir. Veremos mais adiante que a comunicação entre serviços web pode ser feita de maneira mais simples e eficiente.

2.2.2 Arquitetura REST

Criado a partir de um capítulo da tese de doutorado de Roy Fielding no ano de 2000, Fielding nos diz que REST é um estilo arquitetural híbrido para sistemas distribuídos derivado da combinação de alguns estilos arquiteturais com algumas características adicionais.
As principais características arquiteturais incorporadas por Fielding foram:

- Separação de responsabilidades entre as camadas cliente e servidor;
- Comunicações independentes (stateless);
- Uso de cache (para eliminar algumas interações desnecessárias entre cliente e servidor);
- Utilização de uma interface uniforme entre os componentes.

Estas características, quando aplicadas ao desenvolvimento de Web Services, são capazes de melhorar o desempenho geral do sistema – se comparadas a utilização do protocolo SOAP, pois diminui o tempo de resposta das aplicações. Além disso, seu uso gera uma maior flexibilidade e simplicidade.

Como qualquer outro modelo arquitetural, REST também é composto por um conjunto de elementos arquiteturais e seus inter-relacionamentos. Fielding especificou em sua tese, que deu origem à arquitetura, as três classes de elementos arquiteturais: Elementos de Dados, Conectores e Componentes.

Elementos de Dados: São os elementos que contêm a informação a ser usada e transformada. Os principais elementos de dados de REST estão definidos e exemplificados na Tabela 1.

Elemento	Definição/Função	Exemplo
Recursos	Conceito-chave da arquitetura REST. Um recurso é qualquer informação que possa receber um nome. Na	Um filme. Por exemplo, Forrest Gump

	prática, é tudo que pode ser armazenado em uma base de dados.	
Identificadores de Recursos	Identifica um recurso específico envolvido em uma interação entre clientes e servidores. REST usa o URI como identificador de recursos.	Uma URI contendo o código específico e único de um filme. Por exemplo, http://localhost:8080/REST/filme/1
Representações de Recursos	As representações de um recurso são usadas para capturar o estado atual e o estado desejado de um recurso solicitado. Na prática, são sequências de bytes acrescidas de metadados que descrevem estes bytes.	Uma página da Web contendo o estado atual solicitado do recurso (ano, título, gênero). Por exemplo, Ano: 1994, Título: Forrest Gump – O Contador de Histórias, Gênero: Drama

Tabela 1: Elementos de dados da Arquitetura REST

Componentes: São os elementos que usam ou transformam a informação. Em REST, os componentes são caracterizados de acordo com o papel que exercem. São eles:

- Servidor de Origem (trata as requisições de um cliente). Por exemplo, o Servidor Web Apache.
- Proxy e Gateway (utilizados para permitir o encaminhamento das requisições e respostas).
- Agente do Usuário (inicia a requisição e, logo em seguida, se torna o destino final de uma resposta – o Web Browser).

Conectores: São os elementos que interligam outros elementos. REST usa conectores para encapsular a atividade de acesso aos recursos e para transferir as suas representações. De acordo com Fielding, "A utilização de conectores provê uma interface abstrata para a comunicação entre os componentes, melhorando a simplicidade do sistema através da

separação de responsabilidades e ocultando a implementação de recursos e os mecanismos de comunicação". Os principais conectores de REST são:

- Cliente e Servidor (responsáveis pelo acesso aos recursos). Exemplos: Libwww e Apache API, respectivamente.
- Cache (armazena as respostas passíveis de cache). Por exemplo, a cache de um Web Browser;
- Resolver (responsável pela conversão de um URI em um endereço de rede). Exemplo: Bind (DNS lookup library);
- Túnel (cria um caminho virtual para o tráfego dos recursos). Exemplo: SOCKS ou um SSL (Secure Sockets Layer) após um HTTP CONNECT.

Em REST os componentes se comunicam através de transferências de representações de um determinado recurso. Para padronizar o formato destas representações, REST utiliza uma interface uniforme de componentes.

Na prática, essa interface uniforme é aplicada através da utilização dos métodos nativos do protocolo HTTP. A Tabela 2 exibe os principais métodos HTTP utilizados em aplicações RESTFul (Web Services construídos de acordo com os conceitos REST).

Método	**Função**
GET	Obter uma representação de um recurso
POST	Cria um novo recurso
PUT	Criar um novo recurso / Modificar um já existente
DELETE	Remover um recurso existente

Tabela 2: Principais métodos HTTP utilizados em aplicações RESTFul

2.3 ARQUITETURA DO CLIENTE

Entendendo um pouco sobre a arquitetura de um servidor web e a forma como podem ser utilizadas para consumo das páginas web, partiremos então nesta seção para arquiteturas no lado do cliente.

Após o surgimento do termo micro serviços, usado em 2011 em uma conferência de arquitetos de software como proposta de arquiteturas que fossem mais flexíveis, escaláveis e

com manutenção mais simples do que as arquiteturas de sistemas monolíticos, que normalmente são utilizadas. As arquiteturas de micro serviços começam a ser criadas surgindo então o desacoplamento de uma estrutura monolítica, que continha tanto os serviços da aplicação, quanto a renderização de páginas web com HTML, CSS e Javascript num mesmo ambiente compartilhando os mesmos recursos.

Figura 1: Exemplo de arquitetura web monolítica. Fonte: (OPUS Software, 2021)

Uma vez que a interface do cliente é separada da arquitetura de serviços é possível ter mais foco na experiência do usuário, mais escalabilidade e mais atualizações suaves sem precisar alterar muitas partes do sistema. Tendo um ambiente e uma estrutura só voltada para o cliente e a parte front-end de uma aplicação, frameworks também começam a surgir e arquiteturas que uma vez eram usadas no servidor começaram a mudar de lugar para acompanhar a evolução tecnológica. React, Angular, Vue são exemplos de frameworks que auxiliam na construção de uma aplicação voltada para o cliente.

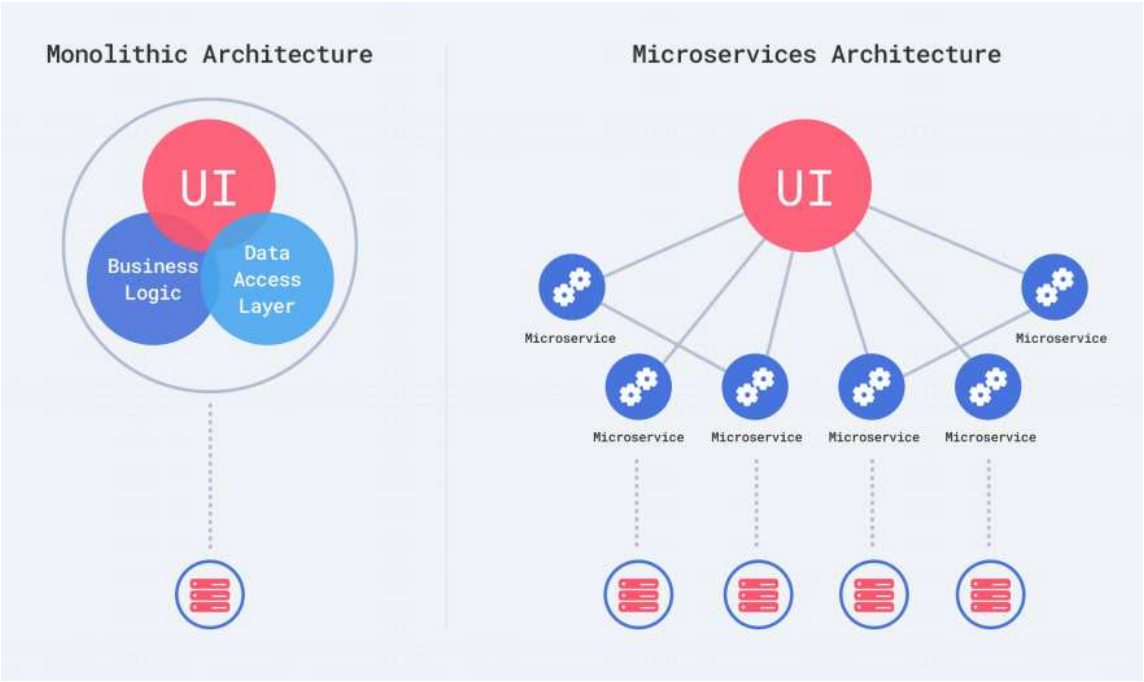


Figura 2: Exemplo de arquitetura web monolítica vs arquitetura de micro serviços. Fonte: (FELIX, 2019)

O padrão MVC (Model-View-Controller), muito utilizado tradicionalmente no lado do servidor, passou a ser usado mais no lado do cliente quando se trata de arquitetura de micro serviços, permitindo assim que o usuário tenha uma melhor experiência. Como o browser não precisa fazer requisições, download e parser de páginas completas a cada interação do usuário, o resultado é uma aplicação que responde muito mais rápido aos comandos. Aplicações construídas com a arquitetura client-side funcionam de uma forma tão natural quanto uma aplicação desktop instalada na máquina do usuário. A transferência de dados também ganha um desempenho considerável pois ao invés de trafegar o conteúdo HTML completo a cada interação do usuário, na arquitetura client-side o aplicativo completo é transferido na primeira requisição e as requisições seguintes são responsáveis por trafegar apenas os dados brutos entre o cliente-servidor, normalmente no formato JSON.

2.4 CONTEXTOS E ARQUITETURAS USADOS EM BIG DATA

Como sendo uma área em constante crescimento e relativamente nova, arquiteturas de software usadas para análise de big data têm-se mostrado diferentes umas das outras de acordo com o foco e o objetivo que deseja alcançar. Buscando suprir os diversos tipos de dados ou velocidade de análise que deva ser necessário na construção de uma arquitetura, veremos a

seguir algumas arquiteturas que têm focos diferentes, porém algumas semelhanças que podem auxiliar no desenvolvimento de nossa arquitetura.

Um dos focos na análise e processamento de dados que atualmente temos bastante é o de streaming, onde neste há uma necessidade de realizar processamento de dados em tempo real. Com o Twitter, por exemplo, é possível prever eventos de crises, analisando o comportamento social durante desastres. Pelo tipo dos dados e a urgência na resposta, as consultas precisam ser executadas em tempo real, o que demanda uma arquitetura de software robusta, confiável e adaptável, uma vez que os usuários precisam executar consultas para isto. Em um cenário de desastres da natureza, como terremotos, o principal benefício da plataforma é oferecer respostas em tempo real. Os requisitos do projeto envolvem a análise de big data em tempo real por meio de um mecanismo de consulta flexível, que disponha de respostas rápidas e de alta disponibilidade. Assim, o resultado desta pesquisa apresenta uma nova plataforma para análise de big data por meio de serviços REST, que pode ser vista na Figura 3. Para avaliar os resultados, os autores criaram um protótipo chamado EPIC Real-Time que combina o processamento streaming e batch, e usa como base a Arquitetura Lambda (REIS, 2018, p. 33).

Figura 3: Arquitetura usada no EPIC Real-Time. Fonte: (REIS, 2018, p. 34)

Já em certos frameworks, tem-se um SaaS para consultas em diferentes fontes de dados, incluindo SGBDR e armazenamento de big data, implantado em nuvem privada. Os dados das fontes são consolidados em um armazenamento big data que funciona como o ponto central para a integração e a análise de dados. A arquitetura desenvolvida é extensível e permite a utilização de outras soluções de armazenamento e de análise de dados, por exemplo, Apach e

HBase ou linguagem R. Essa flexibilidade é possível porque as conexões entre as camadas da arquitetura são fracamente acopladas, de forma que as atualizações de software em pontos específicos não impactam toda a solução. A validação é feita por meio de uma prova de conceito, na qual é entregue uma interface de usuário simplificada para consultar os dados sem necessidade de conhecimento de programação. Neste contexto, a Figura 4 mostra a proposta de arquitetura de Eftekhari, na qual são identificadas três camadas: interface de usuário, aplicação e recursos. O usuário final interage por meio da camada de interface, que aciona a camada de aplicação, a qual contém os adaptadores apropriados de acordo com a demanda. Assim, estão disponíveis adaptadores para ETL, SGBDR e big data. A última camada é responsável pelos recursos de armazenamento e de big data. As ferramentas usadas no protótipo foram o MySQL e Hadoop. Contudo, usa ferramentas diferentes e, com isso, não é possível separar o cluster dos dados, o que gera acoplamento, uma característica que se pretende evitar em sistemas de big data modernos. Assim, sua dissertação propõe que dados e cluster sejam independentes com o uso de data lakes e object stores (REIS, 2018, p. 35).

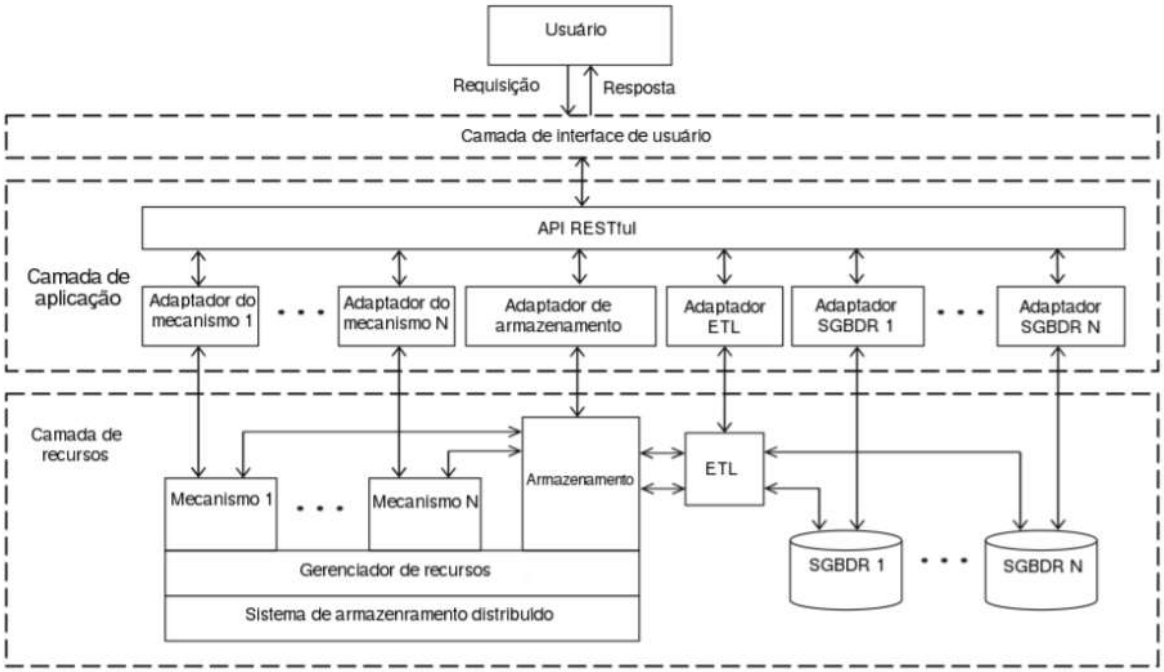


Figura 4: Arquitetura proposta para consultas com big data. Fonte: (REIS, 2018, p. 35)

Ainda em arquiteturas que se adequem mais ao Big Data, nos é apresentado uma extensa revisão bibliográfica da área de big data e uma arquitetura para sistemas de big data que é dividida em três camadas: camada de aplicação, camada de computação e camada de

infraestrutura respectivamente. A camada de aplicação, como visto na Figura 5, é composta por um mecanismo de consulta, algoritmos de clustering, classificação e recomendação. A camada de computação inclui diversos componentes, incluindo os frameworks para big data, NoSQL/SQL, sistema de arquivos e integração. A última camada gerencia a infraestrutura de servidores, rede e armazenamento. Esta arquitetura mostra seus elementos em nível bastante conceitual, na forma de uma revisão da literatura na área, tendo como base inclusive arquiteturas de grandes empresas como Walmart, Amazon e Facebook (REIS, 2018, p. 38).

Figura 5: Arquitetura em camadas para sistemas de big data. Fonte: (REIS, 2018, p. 39)

É possível encontrar também uma arquitetura de referência para soluções de big data que pode ser vista na Figura 6. O objetivo é criar um modelo conceitual de arquitetura para arquitetura big data, sem referência a tecnologias ou ferramentas específicas. Os dois eixos mostrados representam a Informação (horizontal) e a Tecnologia da Informação (vertical). No horizontal, o valor é criado pelas tarefas de coleta de dados, integração e análise. No eixo da vertical, o valor é criado pelo provisionamento de rede, infraestrutura, plataformas e aplicações,

bem como outros serviços de suporte. Neste modelo são definidos os seguintes componentes lógicos funcionais:

- Orquestrador do sistema: define e integra as atividades em um sistema operacional vertical (pessoa, software).
- Provedor de dados: inclui novos dados ou fontes de informação no sistema;
- Provedor de aplicação big data: encapsula a lógica de negócio e a funcionalidade para ser executada pela arquitetura. Inclui atividades como a coleção, a preparação, a análise, a visualização e o acesso aos dados;
- Provedor de framework big data: consiste de uma ou mais tecnologias para garantir flexibilidade e atender aos requisitos que são definidos pelo provedor de aplicação big data. É o componente que recebe mais atenção da indústria e o que tem mais informação disponível.
- Consumidor dos dados: são os usuários finais e outros sistemas que usam o resultado produzido pelo provedor de aplicação big data.

Buscando atender as funcionalidades, avanços na área de big data são refletidos através de frameworks. Dessa forma, para atender aos diversos requisitos, uma típica implementação big data conta com múltiplos frameworks. Assim, na Figura 6 é possível verificar que o componente provedor de framework big data é composto de três subcomponentes:

- Frameworks de infraestrutura: suportam recursos de rede, computação, armazenamento e recursos físicos (energia, resfriamento, segurança etc);
- Frameworks para plataforma de dados: responsáveis pela organização lógica de dados. Aplica-se a situações variadas, desde arquivos de texto delimitados até data stores distribuídas. O método de acesso ao dados inclui APIs para consulta ou SQL;
- Frameworks para processamento: definem como o processamento dos dados é organizado e executado. Eles são tipicamente focados na manipulação de dados e podem ser orientados a processamento batch ou streaming.

A arquitetura a seguir apesar de ser abstrata e não ter um caso real de uso para demonstrar performance e tecnologia essa é digamos o mínimo padrão usado para novas arquiteturas de sistemas big data

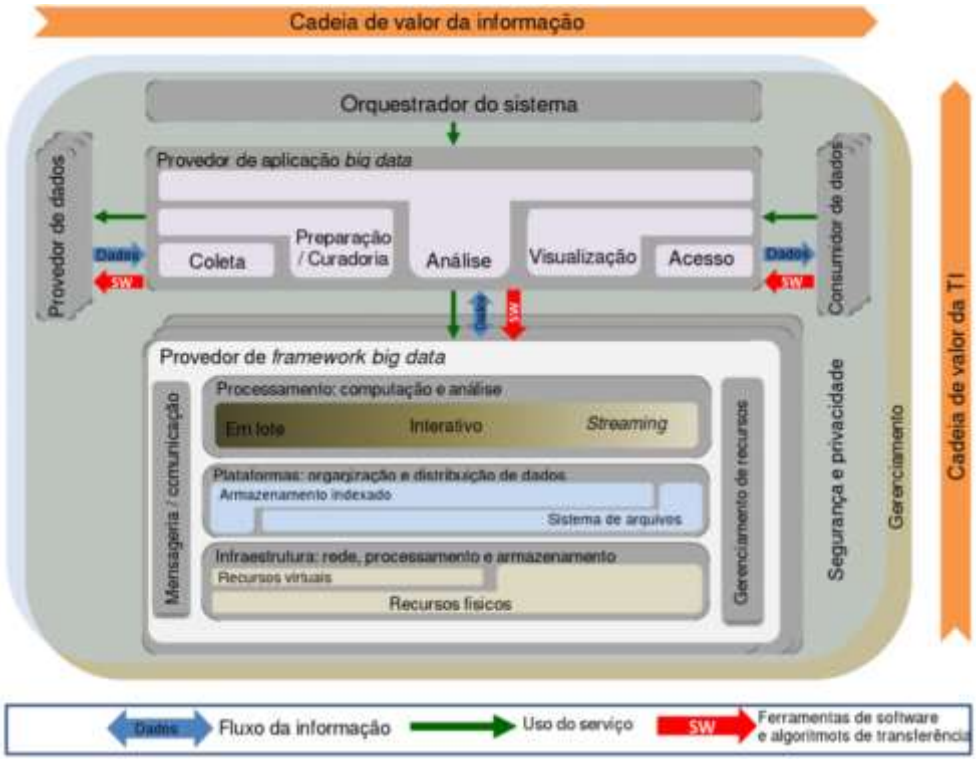


Figura 6: Arquitetura de referência para big data. Fonte: (REIS, 2018, p. 41)

3 METODOLOGIA

Neste capítulo abordaremos a tecnologia, requisitos, arquitetura, processos de extração, transformação e coleta de dados, usados para desenvolvimento do sistema, assim como a modelagem da arquitetura e também ferramentas, linguagens e bibliotecas utilizadas.

3.1 REGRA DE NEGÓCIO

Como regra de negócio para a aplicação tem-se:

Regra de Negócio	Descrição
RN-01	O sistema deverá ter usuários do tipo Gestor, Responsável, Agente-cuidador e Educador.
RN-02	Somente usuários do tipo Responsável não devem poder visualizar os dados gerais referentes a cidade e demais crianças presentes no sistema
RN-03	Somente usuários do tipo Responsável não devem poder visualizar os dados gerais referentes as escolas e demais crianças presentes no sistema
RN-04	Somente usuários do tipo Responsável não devem poder visualizar os dados gerais referentes a bairros e demais crianças presentes no sistema

Tabela 3: Regras de negócio

3.2 REQUISITOS

3.2.1 Funcionais

Como requisitos funcionais para a aplicação tem-se:

Requisito funcional	Descrição
RF-01	O sistema permitirá o acesso dos usuários através do login e senha.

RF-02	O sistema deverá apresentar índices percentuais de peso e altura das crianças da cidade.
RF-03	O sistema deverá apresentar índices percentuais de peso e altura das crianças que frequentam a escola.
RF-04	O sistema deverá apresentar índices percentuais de peso e altura das crianças que não frequentam a escola.
RF-05	O sistema deverá apresentar índices percentuais de altura e IMC de cada escola presente na cidade.
RF-06	O sistema deverá apresentar índices percentuais de altura e IMC de cada bairro presente na cidade.
RF-07	O sistema deverá apresentar um mapa da cidade contendo marcadores para Escolas, Feiras Livres e Supermercados
RF-08	O sistema deverá apresentar nome, data de nascimento, responsável, gênero, escola, endereço, altura e peso da criança
RF-09	O sistema deverá exibir junto as informações da criança uma classificação para os índices de altura, peso, alimentação e vulnerabilidade social e exibir recomendações para o usuário de cada índice.

Tabela 4: Requisitos funcionais

3.2.2 Não Funcionais

Como requisitos não funcionais para a aplicação tem-se:

Requisito não funcional	Descrição
RNF-01	Disponibilidade para smartphones
RNF-02	Bom processamento de dados em larga escala

RNF-03	Design responsivo
RNF-04	Bom desempenho
RNF-05	Boa acessibilidade
RNF-06	Escalabilidade

Tabela 5: Requisitos não funcionais

3.3 TECNOLOGIAS E LINGUAGENS UTILIZADAS

Para desenvolvimento da aplicação de forma a se obter uma solução de navegador e também nativa para smartphones foi-se utilizado novas tecnologias e frameworks que abrangem todos os requisitos necessários para desenvolvimento da solução. Tendo como base o uso de linguagens como Typescript e Kotlin veremos a seguir um pouco das tecnologias utilizadas no processo de desenvolvimento.

Linguagem	**Framework**	**Versão**	**Descrição**
Typescript	Node	8.0.0+	Tecnologia utilizada para o desenvolvimento do servidor.
Typescript	React	16.9.9+	Tecnologia utilizada para o desenvolvimento do cliente
Kotlin	N/A	1.4.31	Tecnologia utilizada para desenvolvimento do mobile
Python	Jupyter notebook	N/A	Tecnologia utilizada para extração, transformação e

			adição de dados no MongoDB
MongoDB	MongoDB Compass	1.26.0	Tecnologia utilizada para camada de persistência dos dados

Tabela 6: Tecnologias e linguagens utilizadas

3.4 BIBLIOTECAS UTILIZADAS

Utilização de bibliotecas é uma estratégia de reuso de código bem utilizada em todas as linguagens de programação, pois além de ser fácil sua implantação no projeto, garante que o desenvolvedor consiga implementar funções complexas em seu projeto sem precisar fazer desde o início (FILHO, 2013).

Cito abaixo algumas das principais bibliotecas utilizadas no desenvolvimento do sistema como um todo:

Nome	**Versão**	**Descrição**
Redux	4.0.4+	Biblioteca que permite o compartilhamento de estados entre vários componentes diferentes do React.
Material-ui	3.2.10+	Biblioteca com padrões de design e layout definidos que ajudam na estilização de páginas web.
Styled Components	4.4.1+	Biblioteca que nos permite escrever códigos CSS dentro do Javascript.

Axios	0.21.1+	Axios é um cliente HTTP baseado em Promises para fazer requisições.
Retrofit	2.9.0	Biblioteca que fornece um padrão simples de implementação para transmissão de dados entre aplicação e servidor.
sklearn	N/A	Biblioteca de aprendizado de máquina de código aberto para a linguagem de programação Python.
matplotlib	N/A	Biblioteca de software para criação de gráficos e visualizações de dados em geral, feita para e da linguagem de programação Python.
numpy	N/A	Biblioteca para a linguagem de programação Python, que suporta o processamento de grandes, multidimensionais arranjos e matrizes, juntamente com uma grande coleção de funções matemáticas de alto nível para operar sobre estas matrizes.

pandas	N/A	Biblioteca de software criada para a linguagem Python para manipulação e análise de dados. Em particular, oferece estruturas e operações para manipular tabelas numéricas e séries temporais.

Tabela 7: Bibliotecas utilizadas

3.5 CAMADA DE PERSISTÊNCIA

Com o objetivo de suprir uma escalabilidade de dados e uma volatilidade para melhor transmissão dos dados da base para o cliente, o MongoDB: banco de dados não relacional, foi o escolhido para suprir nossa necessidade na camada de persistência.

O banco de dados NoSQL surgiu motivado pela necessidade de trabalhar com grandes volumes de dados. Seus defensores afirmam que os sistemas desenvolvidos com banco de dados não relacionais são mais flexíveis do que os bancos de dados relacionais, já que são livres de esquemas rígidos, são distribuídos, escaláveis, possuem melhor desempenho e não necessitam de uma infraestrutura robusta para armazenar seus dados (FOWLER; SADALAGE, 2013, p. 18).

A seguir é apresentado uma modelagem das coleções base que são utilizadas para a exibição de informações na tela do cliente.

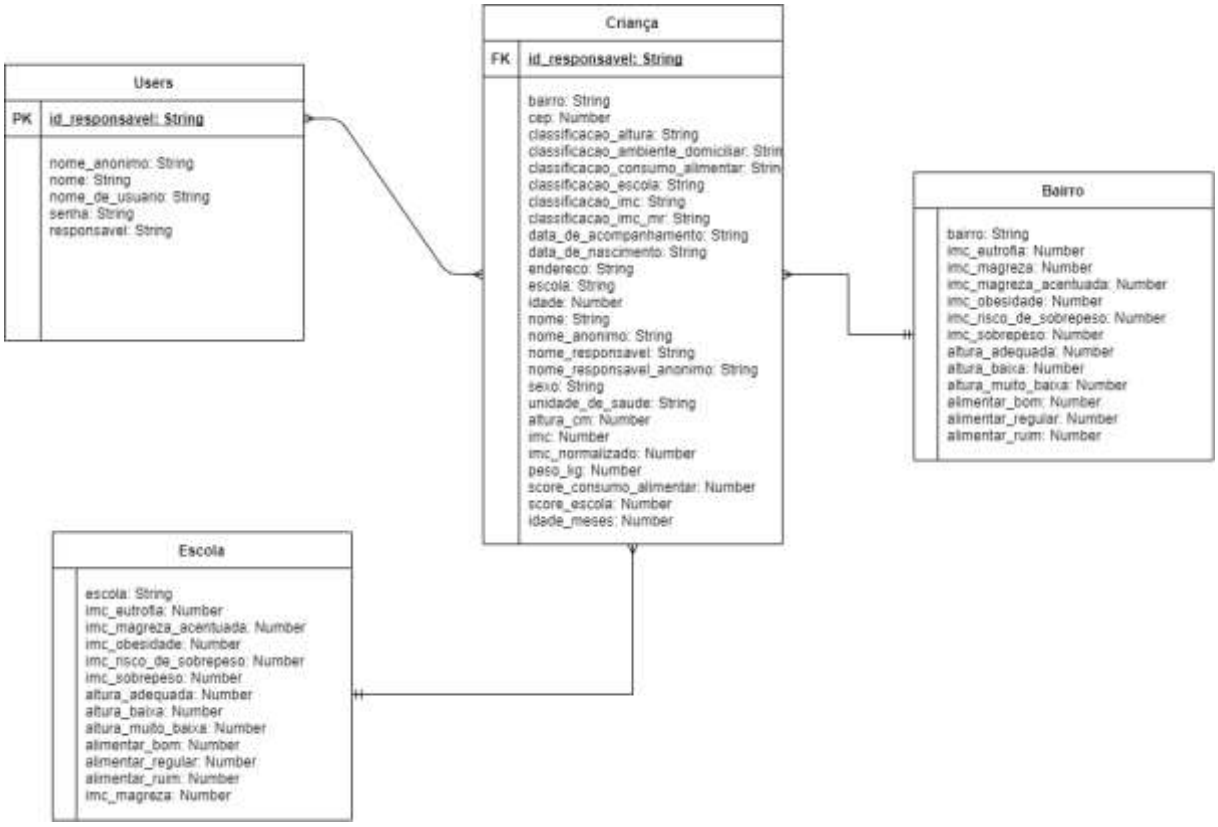


Figura 7: Modelagem de dados Brasil2030

Como fonte principal de dados para apresentação de informações em uma interface que possa mostrar ao usuário final um possível risco de obesidade e recomendar pontos a serem observados para melhor saúde a criança, o relacionamento existente que ainda é encontrado na

aplicação, é onde um usuário ao se logar possa ser responsável por uma ou mais crianças e que essa criança também possa ter como acompanhamento também um ou mais usuários, usuários esses que diferem em alguns tipos como mostrado nas regras de negócio (RN-01). A partir deste relacionamento, há presente também a relação de informações da criança quanto a sua escola e seu bairro, para que a título de informação mais ampla se possa passar a ter para o usuário final um indicador médio de uma comunidade ou instituição. Buscando atingir os requisitos funcionais como falado anteriormente, além das coleções presentes na nossa base de dados como mostrado na figura 7, há também a adição de algumas coleções necessárias para processos de ETL e classificação de indicadores como: Acompanhamento alimentar e Ambiente domiciliar.

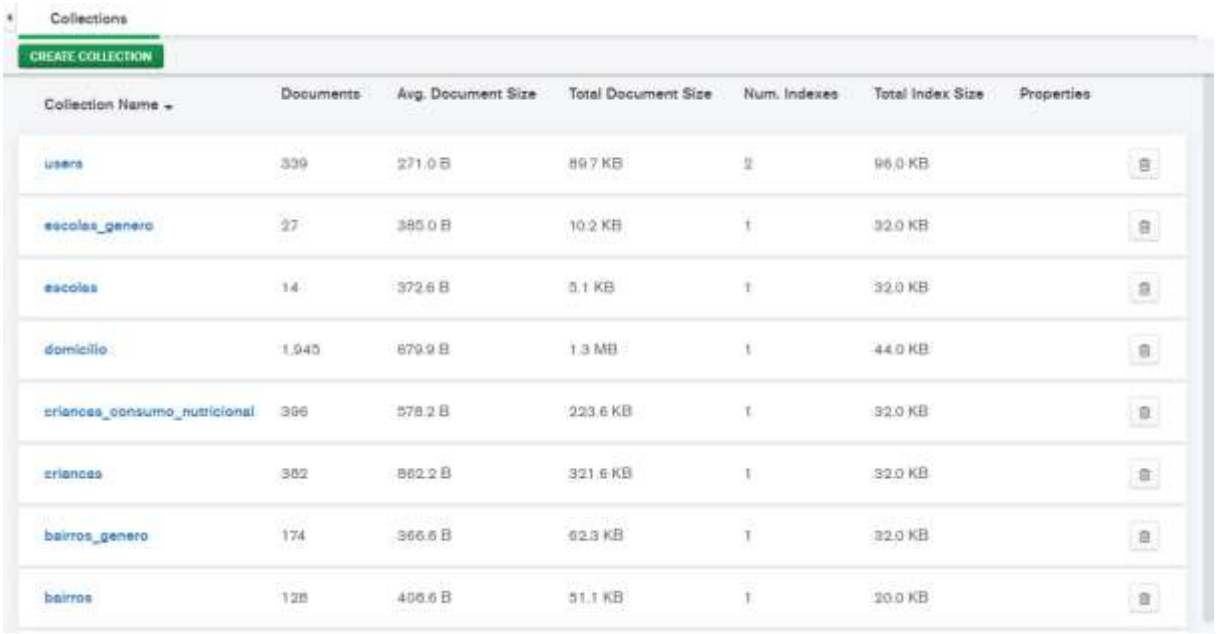


Figura 8: Lista de collections criadas no MongoDB

3.6 EXTRAÇÃO, TRANSFORMAÇÃO E LEITURA DOS DADOS

Um processo que é presente em quase todo sistema que utiliza Big Data é o processo de ETL (Extração, Transformação e Leitura de dados). Esse processo se dá devido à grande quantidade de dados e informações coletadas de um ambiente ou cenário onde esses dados nem sempre serão completamente usados ou serão usados após realizado um tratamento.

Como nosso objetivo proposto é o desenvolvimento de uma arquitetura que possa suprir as necessidades do Big Data, esse processo de ETL foi realizado pela equipe do INTERSSAN e disponibilizado o resultado desses dados já analisados através de arquivos, ficando a nosso

cargo lê-los e alimentar nossa base. Contudo a título de contextualização, trago um pouco sobre a fonte dos dados utilizados pela equipe INTERSSAN e a lógica em base utilizada por eles para extração, transformação e leitura dos dados.

Tendo como objetivo geral do projeto mostrar indicadores de obesidade infantil sob alguns ambientes como: escola, bairro, domicílio e cidade, o processo de ETL foi realizado com dados inicialmente trazidos pelo SISVAN (Sistema de Vigilância Alimentar Nutricional) e OMS (Organização Mundial da Saúde).

No SISVAN contém todos os dados de crianças de cada município através do registro do SUS (Sistema Único de Saúde), nele é possível obter informações como: nome, peso, altura, data de nascimento, etc. Uma vez obtido esses dados, também é realizada uma busca de dados referente a classificação de estado nutricional na OMS, que disponibiliza dados quanto a classificação por peso, estatura e IMC. Por sua vez, tendo sido realizada a união de dados do SISVAN e OMS é possível obter uma boa classificação quanto a peso, altura e IMC para cada criança. É realizada também a união com os dados do Cadastro Único, que disponibiliza uma série de dados não só sobre a criança mas também informações que a envolvem como: características do domicílio, escolaridade da mãe, situação de trabalho e renda, entre outras.

Tendo então unidos dados do SISVAN, OMS e Cadastro Único, foi-se utilizando então a lógica fuzzy para transformação e classificação de dados de acordo com cada objetivo. Sendo estes a classificação quanto a peso, altura e IMC e também foi possível a classificação quanto a ambiente doméstico e escolar de cada criança.

A lógica fuzzy (em português também pode ser chamada de lógica difusa) já é um conceito bem antigo, que surgiu na década de 60 e é aplicada em diversas áreas, desde engenharia até inteligência artificial. Basicamente, ela consiste em modelar um problema de modo aproximado ao invés de preciso, assim como é o raciocínio humano em diversas situações (YAMASSAKI, 2020).

Portanto, como uma classificação binária desviaria demais da realidade do estado nutricional de uma criança, a lógica fuzzy auxilia nessa questão para criar uma classificação que melhor reflete a sociedade. Fazendo analogia, uma escala likert é um exemplo de lógica fuzzy, visto que, diferente de uma pesquisa binária, nessa escala as respostas variam conforme o grau de concordância de uma pessoa respondendo a uma determinada pergunta.

Figura 9: Exemplo de lógica booleana e lógica fuzzy. Fonte: (YAMASSAKI, 2020)

4 RESULTADOS E DISCUSSÕES

4.1 ARQUITETURA DO SISTEMA

Como mostrado anteriormente, uma vez que a arquitetura do sistema é construída a base de micro serviços é passado para o lado do cliente o controle de como e quando realizar requisições ao seu serviço que irá trazer dados vindos de uma base ou outros micro serviços necessários para funcionamento de suas páginas.

Desenvolvendo então o nosso cliente com React e auxílio de algumas bibliotecas, é chegado a um padrão arquitetural MVC. Usando o HTML com auxílio do JSX foi possível obter todo ambiente da View responsável pela exibição de dados e layouts necessários. Já representando a parte do Controller e Model, o Typescript ficou responsável por todo o controle das requisições, alteração dos elementos na página e por fim determinação de quais e como os dados serão mostrados para o usuário.

Seguindo a mesma ideia da aplicação em desktop, no mobile a arquitetura padrão utilizada também é o MVC, porém buscando o desenvolvimento de uma aplicação totalmente nativa a linguagem usada em toda aplicação é o Kotlin consumindo a mesma API desenvolvida para o desktop.

Partindo para o servidor, ou API, sendo melhor referenciado em um sistema de micro serviços, este é quem realiza toda a comunicação com nossa base de dados e resposta as requisições solicitadas pelos clientes, seja Mobile ou Desktop. Solicitação como login, recuperação de dados de crianças e todos os indicadores que serão representados na tela do usuário são buscadas no MongoDB e devolvidas ao cliente seja ele tendo sido chamado pelo Typescript na aplicação em React ou Kotlin na aplicação nativa. Essa API foi desenvolvida em Node seguindo o padrão de arquitetura REST como vimos anteriormente.

Os processos de extração, transformação e leitura dos dados (ETL) atualizam a base de dados, quando necessário, com novas informações das crianças e complementam toda a arquitetura do sistema.

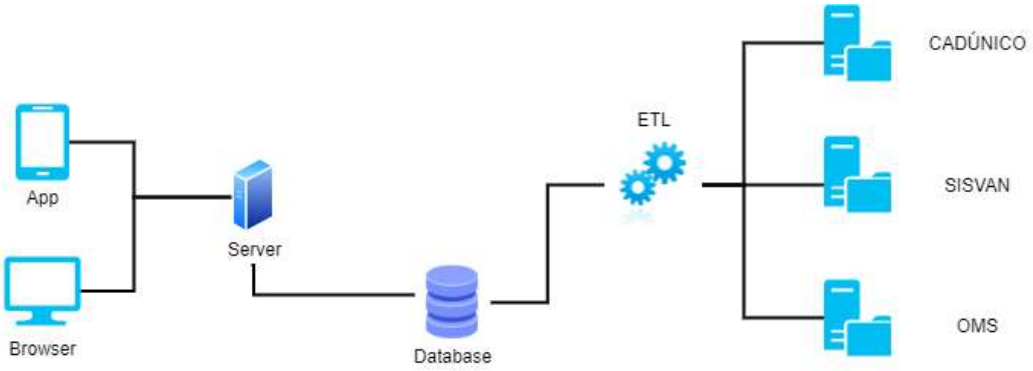


Figura 10: Arquitetura do sistema Brasil2030

Tendo uma visão mais ampla do processo, o cliente criado consome a API construída enviando requisições quando necessário para informar os dados na tela do usuário. A API por sua vez realiza uma comunicação com o banco de dados que aqui já está atualizado com todo o processo de ETL que é realizado com dados do SISVAN, OMS e CADÚNICO. O processo de ETL é executado via script sempre que uma nova criança é adicionada ou o sistema muda de cidade.

Na visão da arquitetura em camadas, temos como um todo então 4 camadas. A primeira camada é a do cliente com toda a apresentação de interface e interação do usuário, a segunda camada é a do serviço contendo a API e regras de negócio, a terceira camada é a base de dados com as informações necessárias e conteúdo que será apresentado, e a quarta camada é o processo de ETL que é realizado para atualizar a base de dados a partir dos sistemas do SISVAN, da OMS e do CADÚNICO.

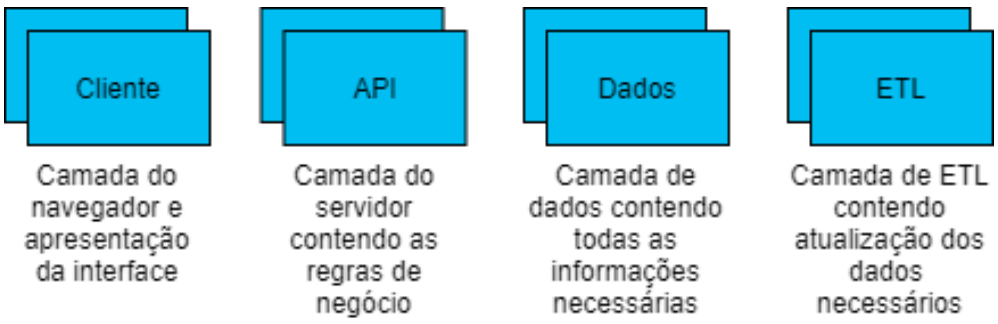


Figura 11: Camadas da arquitetura do sistema Brasil2030

4.2 APRESENTAÇÃO DA APLICAÇÃO

A aplicação desenvolvida durante o primeiro semestre de 2021 teve seu nome Brasil2030 e como falado anteriormente foi desenvolvida tanto para ambiente nativo, como o android, quanto para ambiente web. Apresentamos a seguir a interface da aplicação web e interface da aplicação nativa assim como algumas funcionalidades.

4.2.1 Interface web

Como tela inicial temos uma tela de login apresentado um formulário para que um usuário já cadastrado possa ter acesso inicial ao sistema.

Figura 12: Tela de Login Brasil2030

Aba Tupã

Uma vez realizado login no sistema, dependendo do usuário, este terá acesso a diversas abas na aplicação. As figuras 13 e 14 a seguir representam a aba Tupã que é a tela inicial mostrada após um usuário do tipo Gestor acessar o sistema Brasil2030. Nesta aba serão apresentados os dados gerais e os índices de obesidade infantil da cidade de Tupã, índices quanto a peso, altura, ambiente domiciliar e ambiente escolar. Além disso, também é possível visualizar os indicadores dos índices de Acompanhado de Peso e Altura classificados como: Extremo, Muito Baixo, Adequado, Acima, Muito Acima e indicadores de Ambiente Escolar e Domiciliar classificados como: Ruim, Regular e Bom.

Figura 13: Tela Tupã Brasil2030 - todas as crianças nas escolas

Figura 14: Tela Tupã Brasil2030 - todas as crianças

Ao clicar no botão “saiba mais” de algum indicador, é possível também visualizar detalhadamente os índices desse indicador, por exemplo: caso queiramos ver qual o índice de todas as crianças de Tupã em um ambiente escolar que não se encontram em estado “bom”, podemos clicar no “saiba mais” do indicador Ambiente Escolar e ver mais detalhadamente os índices como mostrado na figura 15 a seguir.

Figura 15: Tela Tupã Brasil2030 - ambiente escolar

Aba Escolas

Indo para aba de Escolas é possível ver separadamente os dados referente somente as escolas. Na figura 16 é possível visualizar o nome das escolas presentes na cidade e uma tabela com porcentagens referentes a classificação da altura e IMC médio das crianças de determinada escola.

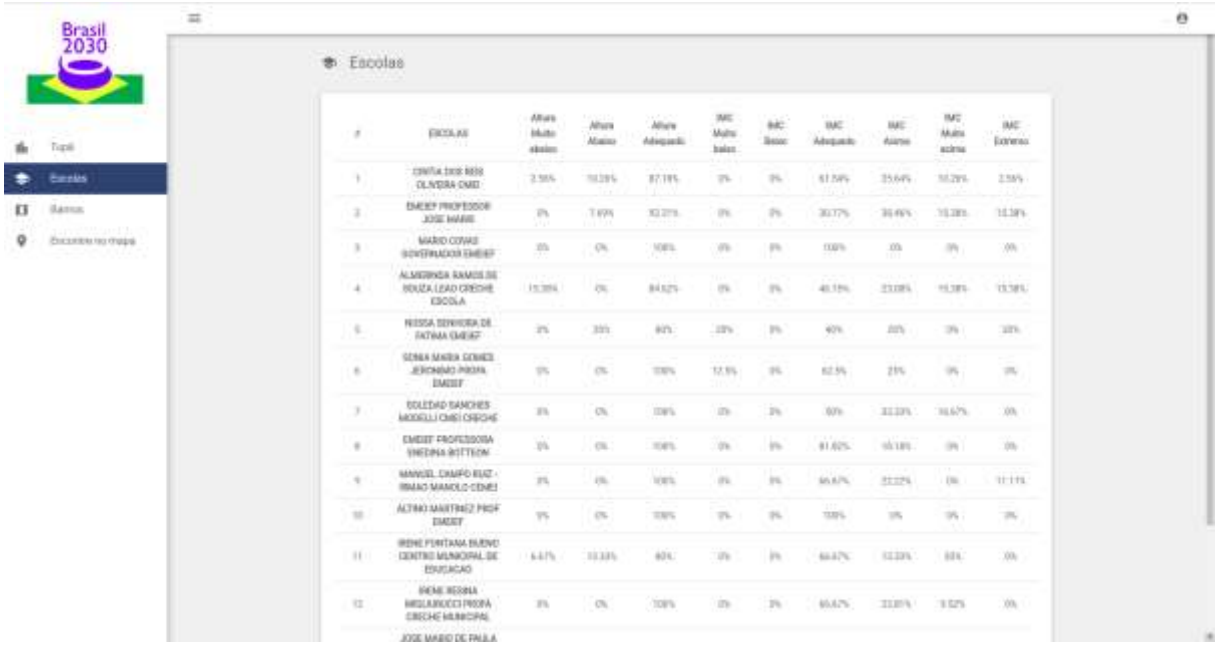


Figura 16: Tela escolas Brasil2030 - dados em tabela

Ainda, clicando no nome da escola também é possível ver mais detalhadamente em gráfico as porcentagens dos indicadores de altura, alimentação e peso das crianças da escola escolhida como mostrado na figura 17 a seguir.

Figura 17: Tela escolas Brasil2030 - escola específica

Aba Bairros

Por conseguinte, indo para aba Bairros, que assim com as escolas, também é visto em tabela, o nome, a classificação de altura e IMC das crianças dos bairros de Tupã. Tem-se também, a possibilidade de ver em gráfico de pizza as porcentagens dos indicadores de altura, alimentação e peso das crianças de um determinado bairro escolhido.

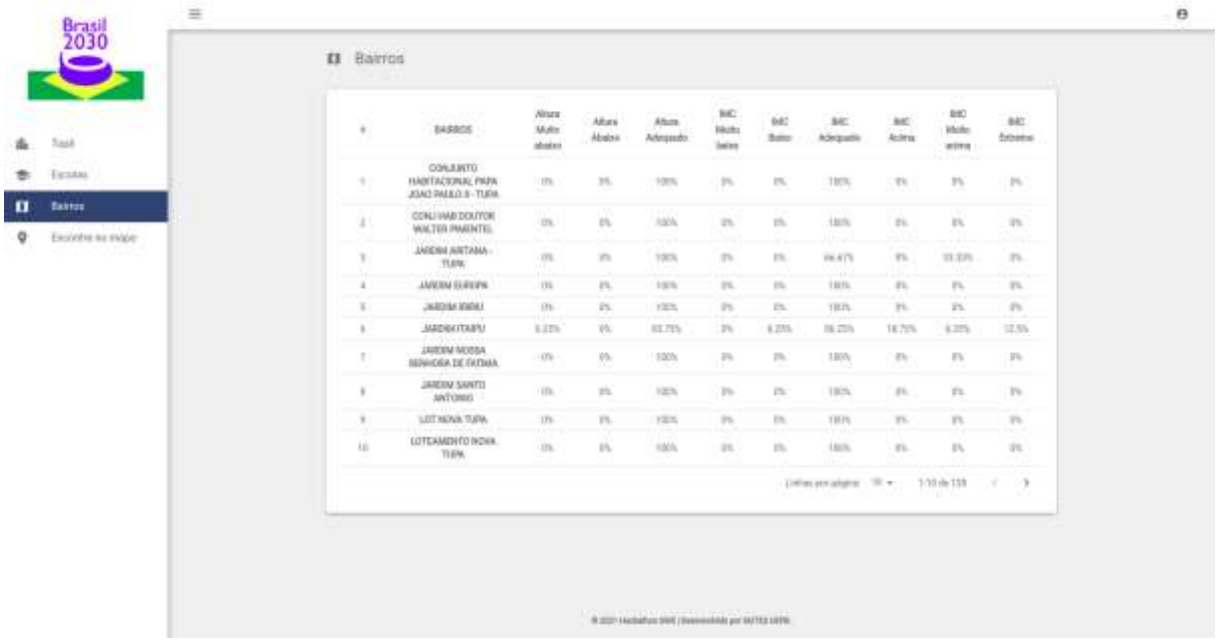


Figura 18: Tela bairros Brasil2030 - bairros em tabela

Figura 19: Tela bairros Brasil2030 - bairro específico

Aba Minhas Crianças

Como falado anteriormente ao realizar o login, o usuário terá um tipo de login atrelado a ele, com isso, sendo um usuário do tipo, “Educador”, “Responsável” ou “Agente Cuidador” a aba de “Minhas Crianças” é habilitada, assim esse usuário terá acesso a informações das crianças que estão vinculadas a ele. Nessa tela é possível ver nome, data de nascimento, gênero,

nome do responsável, escola, data de acompanhado, endereço, bairro, classificações, peso, altura, alimentação e indicadores de vulnerabilidade social.

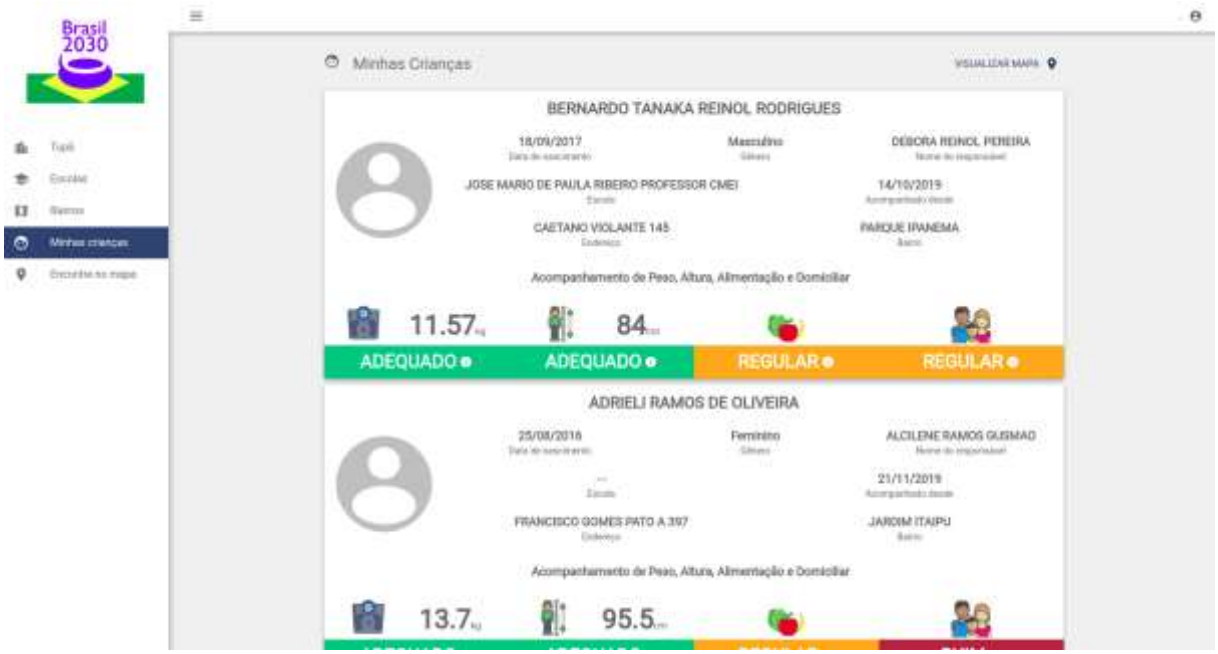


Figura 20: Tela Minhas Crianças Brasil2030

Recomendações também podem ser vistas para essa criança de acordo com a classificação que é apresentada para a mesma.

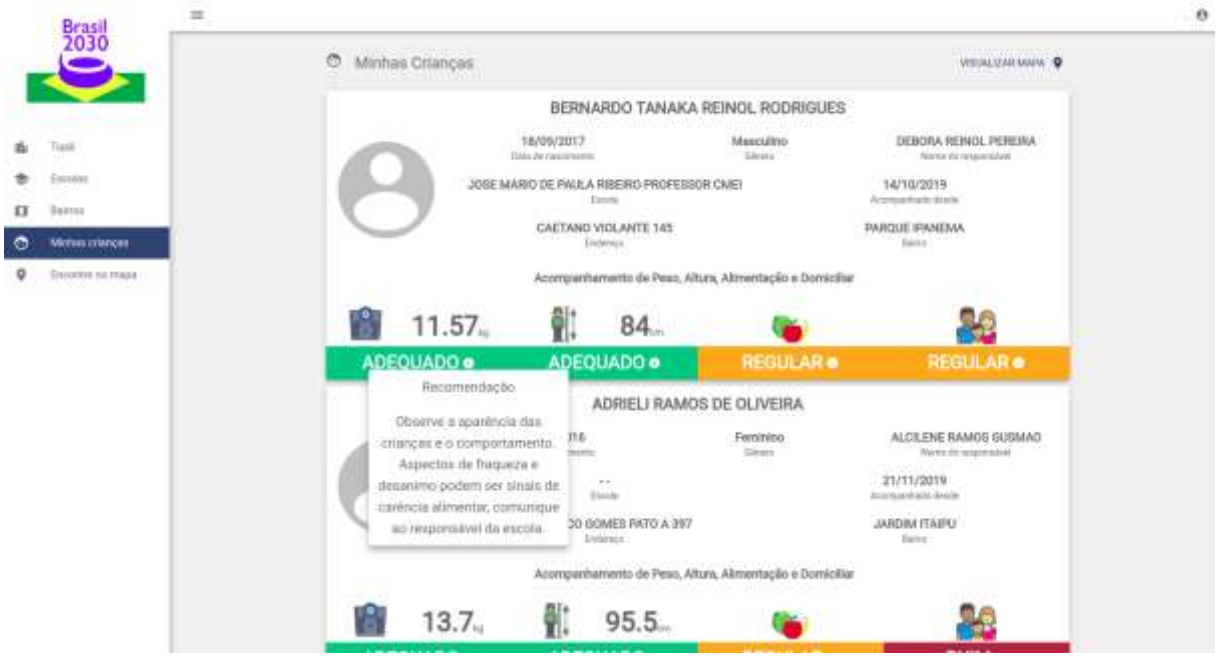


Figura 21: Tela Minhas Crianças com recomendação Brasil2030

Aba Encontre no mapa

Com objetivo de facilitar a recomendação e acesso de locais que possam ajudar no desenvolvimento saudável da criança, na aba “Encontre no mapa”, disponível para todos os tipos de usuários, é possível ver um mapa de Tupã apresentando escolas, feiras livres e supermercados com ícones específicos para cada um desses estabelecimentos, onde ao ser clicado também é apresentado o endereço completo do estabelecimento.

Figura 22: Tela Encontre no mapa Brasil2030

Figura 23: Tela Encontre no mapa Brasil2030 - detalhes de supermercado

4.2.2 Interface mobile

Seguindo para a apresentação da versão mobile do sistema, tem-se basicamente o mesmo objetivo em apresentação de informações e dados, porém com algumas diferenças em layout e nomenclaturas. Apresento a seguir a versão mobile do sistema.

Inicialmente assim como a versão web temos a tela de login, basicamente não há diferença da versão web, contudo deixo aqui a figura 24 para mostrar como ficou a título de curiosidade.

Figura 24: Tela Login versão Mobile Brasil2030

Listagem de Crianças

Na versão mobile ao se logar no aplicativo o usuário encontra como primeira tela uma lista com todas as crianças que estão atreladas a ele. Nessa tela é possível visualizar uma breve informação sobre cada criança como nome, idade e indicadores de peso e altura, etc.

Figura 25: Listagem das crianças Brasil2030

Saúde

Ao selecionar uma criança o usuário é passado para a tela saúde do aplicativo onde nesta, é apresentado com mais detalhes as informações e indicadores da criança escolhida, como: acompanhamento do peso em Kg, a altura da criança em cm, assim como indicadores de alimentação e ambiente domiciliar. Pressionando em algum desses indicadores é possível ver recomendações de acordo com a classificação que é mostrada para cada indicador, e ainda dá opções ao usuário de lugares que podem ajudar ou manter a saúde da criança adequada. Ver figura 26.

Figura 26: Tela saúde Brasil 2030

Pressionado em alguma das opções presentes nas recomendações dos indicadores da criança, é apresentado para o usuário uma tela com o mapa de Tupã onde é mostrado a localização de locais de acordo com o tipo da opção selecionada anteriormente, assim como o nome e endereço dos locais para que o usuário possa escolher o que mais lhe agradar. Ver figura 27.

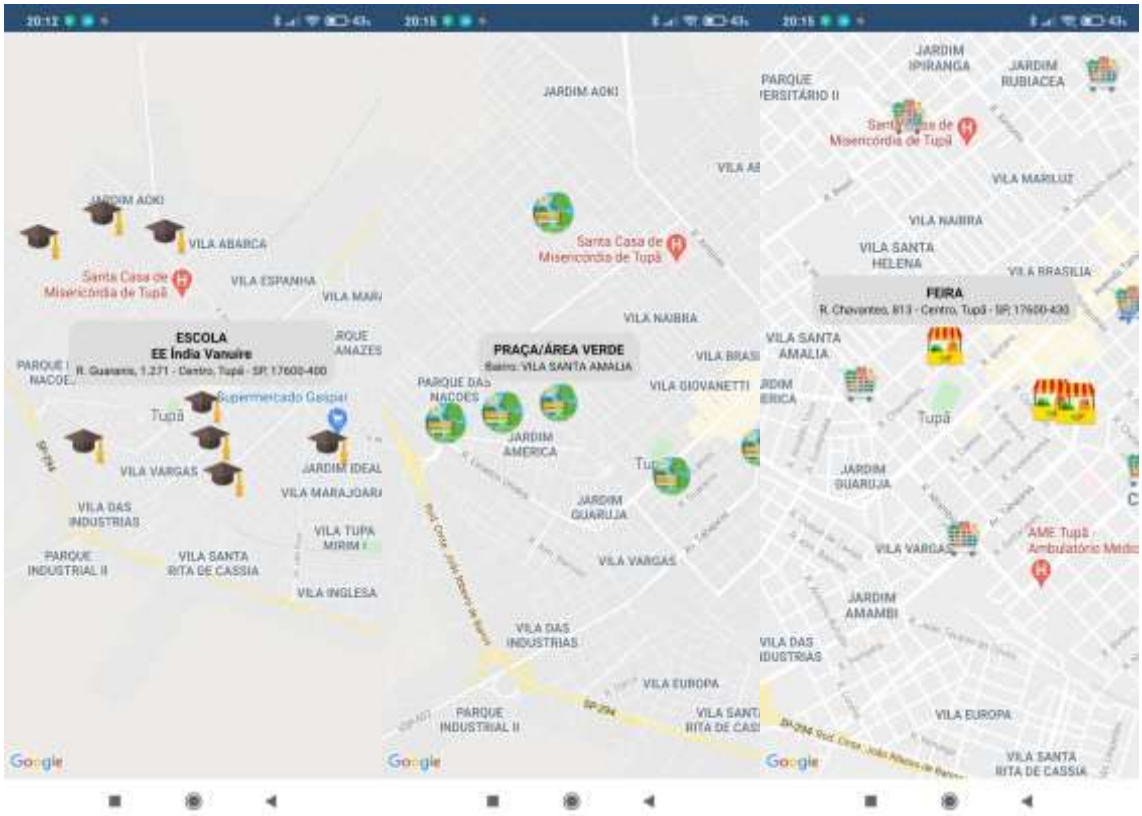


Figura 27: Tela mapa de Tupã Brasil2030

Nossa cidade

Paralelo a aba de “Saúde” temos a aba de “Nossa cidade” nessa aba é possível ver os dados gerais da cidade de Tupã. Caso um usuário do tipo Gestor se logue na aplicação, esta será a primeira tela que será apresentada ao mesmo. Na figura 28 é possível visualizar a porcentagem, a quantidade de crianças e suas respectivas classificações nos indicadores de peso, altura, ambiente domiciliar e ambiente escolar. Nos indicadores de ambiente domiciliar e ambiente escolar é apresentado também para cada classificação o motivo pelo qual as crianças daquele indicador se encontram naquela classificação.

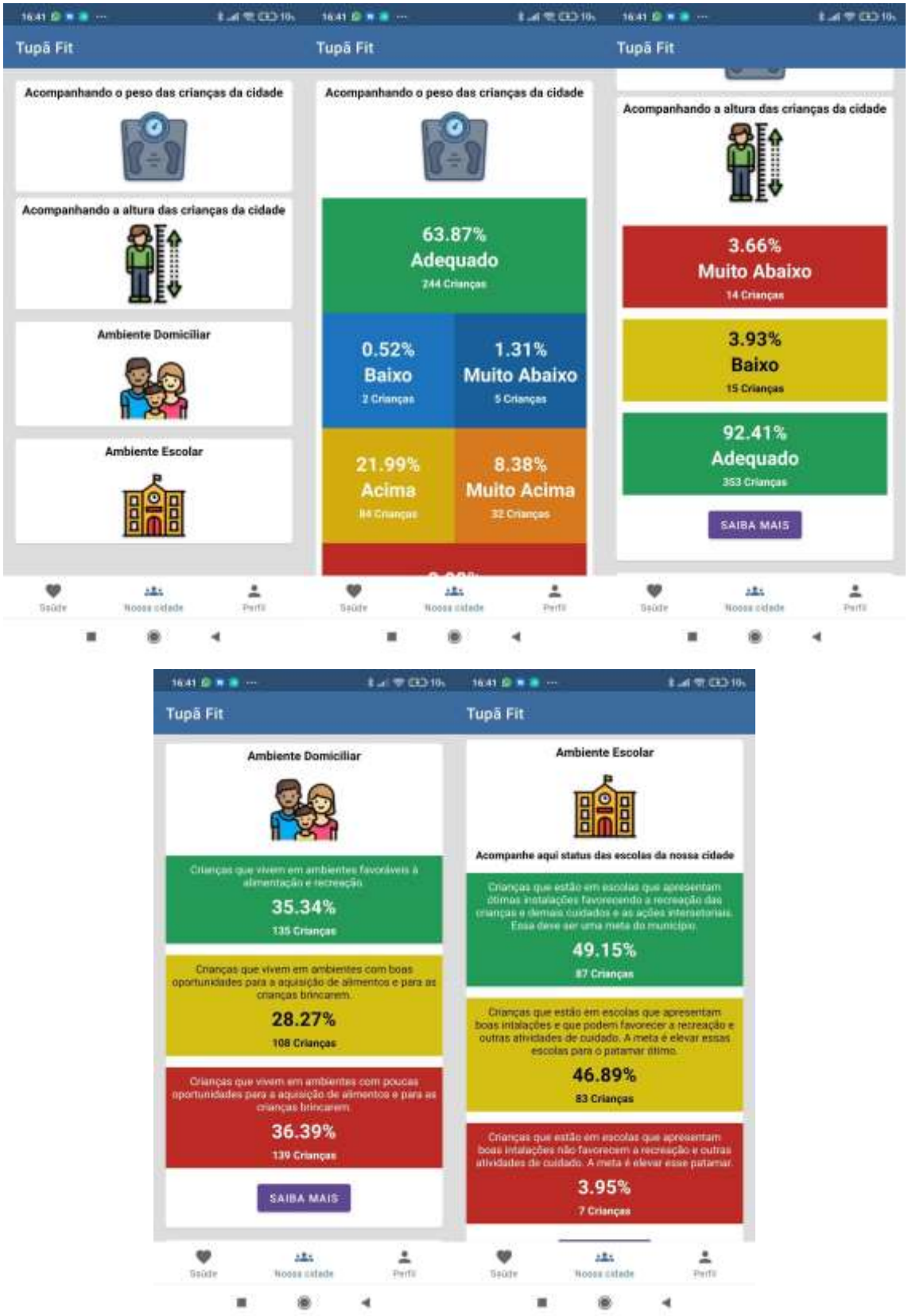


Figura 28: Tela Nossa cidade Brasil2030

Perfil

Por fim, como última aba, temos o perfil da criança selecionada na lista inicial, contendo agora todos os dados da criança, dados como: nome, data de nascimento, idade, gênero, endereço, escola matriculada, início da data de acompanhamento por agentes de saúde e a unidade de saúde que acompanha a criança. Ver figura 29.

Figura 29: Tela perfil Brasil2030

Como um dos requisitos fundamentais para integração de arquitetura de um sistema para big data é a necessidade de um bom processamento e variedade de dados e boa escalabilidade, verificamos então os tempos de respostas para consultas feitas a nossa API que é quem de fato traz os dados necessários para serem exibidos em tela.

4.3 PERFORMANCE E ESCALABILIDADE

Tendo boa parte do relacionamento dos dados de cada criança, responsável, escola, cidade e bairro sido realizada no processo de ETL para atualização do nosso banco de dados, a interação entre cliente-servidor-base ficou mais leve, o que tornou possível uma resposta mais rápida ao usuário final uma vez comparado a um sistema que utilizam ETL a cada requisição.

Em três testes feitos na tela inicial, que traz todos os dados da cidade de Tupã, e é onde mais é realizado requisições simultâneas, foram necessários um tempo médio de 4,55 segundos para carregamento de todos os dados necessários.

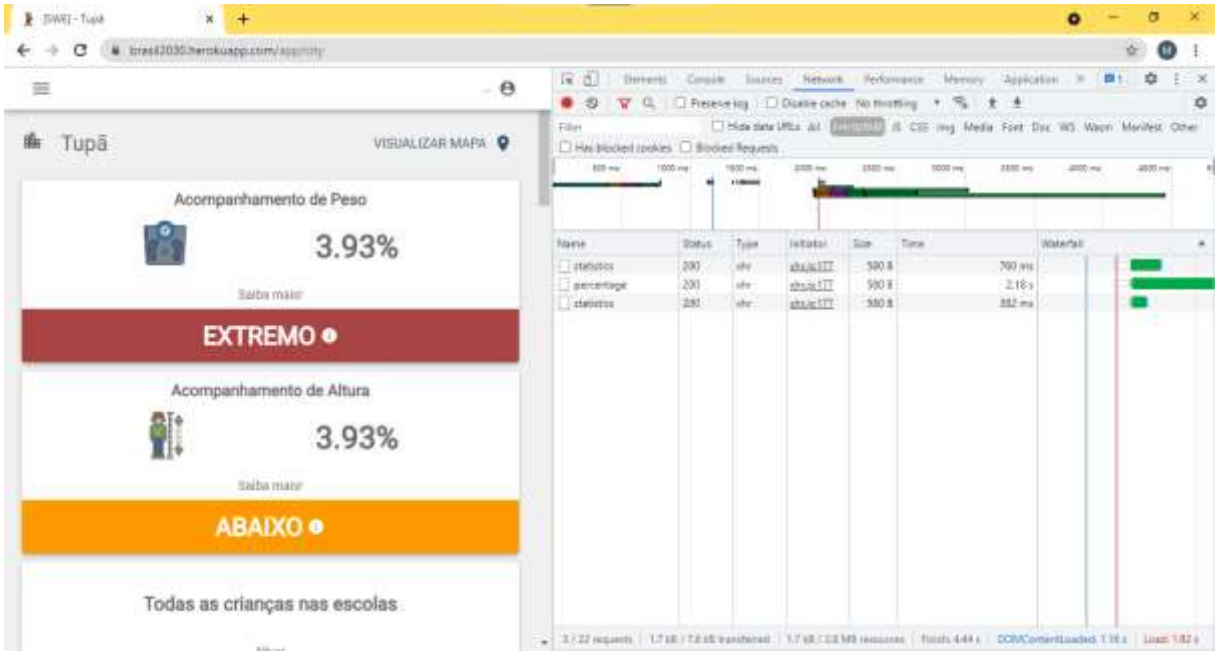


Figura 30: Tempo de requisição de tela inicial Brasil2030

Seguindo com testes em outras telas, que realizam o processamento de boa quantidade de dados como: busca de todos os bairros, crianças ou todas a escolas, foi utilizado a técnica de lazy loading que ao invés de termos que carregar 500 bairros de uma vez do banco de dados é utilizado uma paginação a nível de serviço onde uma vez escolhido quantos itens o usuário deseja ver na tela, a requisição é enviada tendo o retorno somente da quantidade de itens desejada. Na figura 31 é possível ver que ao realizar o carregamento da tela de bairros a resposta da requisição enviada tem somente os 10 dos 128 itens que são exibidos na contagem da tabela de bairros para o usuário, evitando assim um alto processamento de dados e uma escalabilidade quanto a um aumento de dados se necessário.

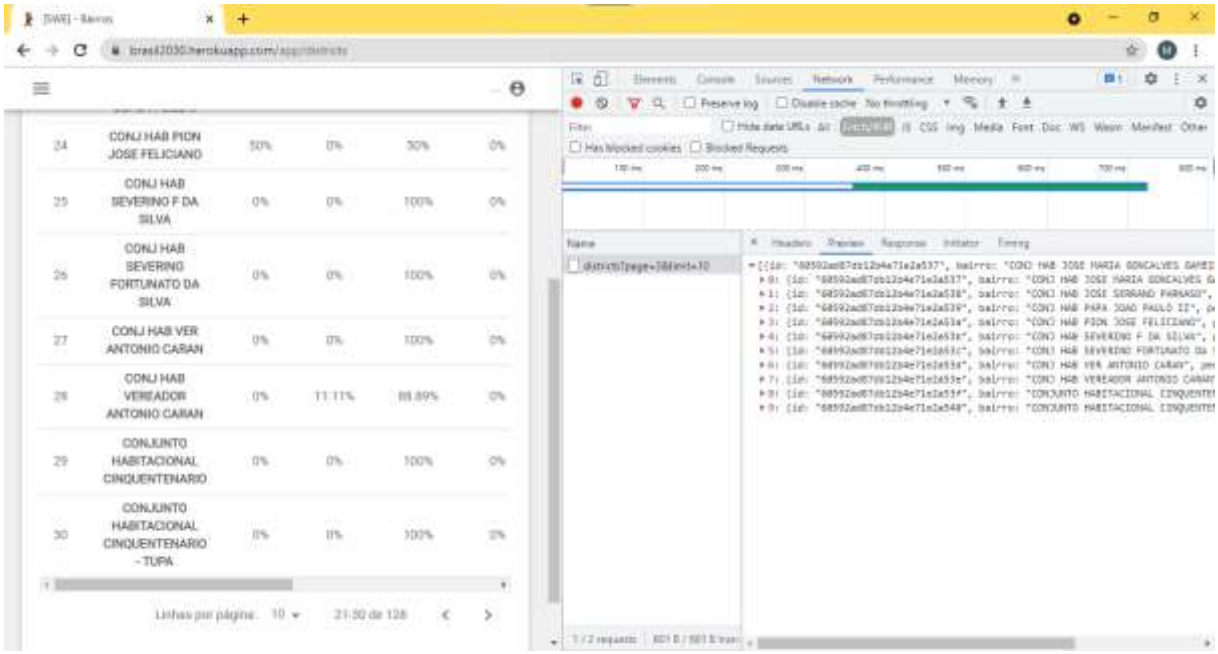


Figura 31: Requisições lazy loading Brasil2030

Com essa estratégia o tempo de resposta da requisição enviada a cada passagem de página ficou em uma de média 0,369 segundos, o que para o usuário é quase irrelevante comparado com os quase 2 segundos para retornar os 128 bairros e mais de 3 segundos para retornar dados das crianças em testes realizados no Postman.

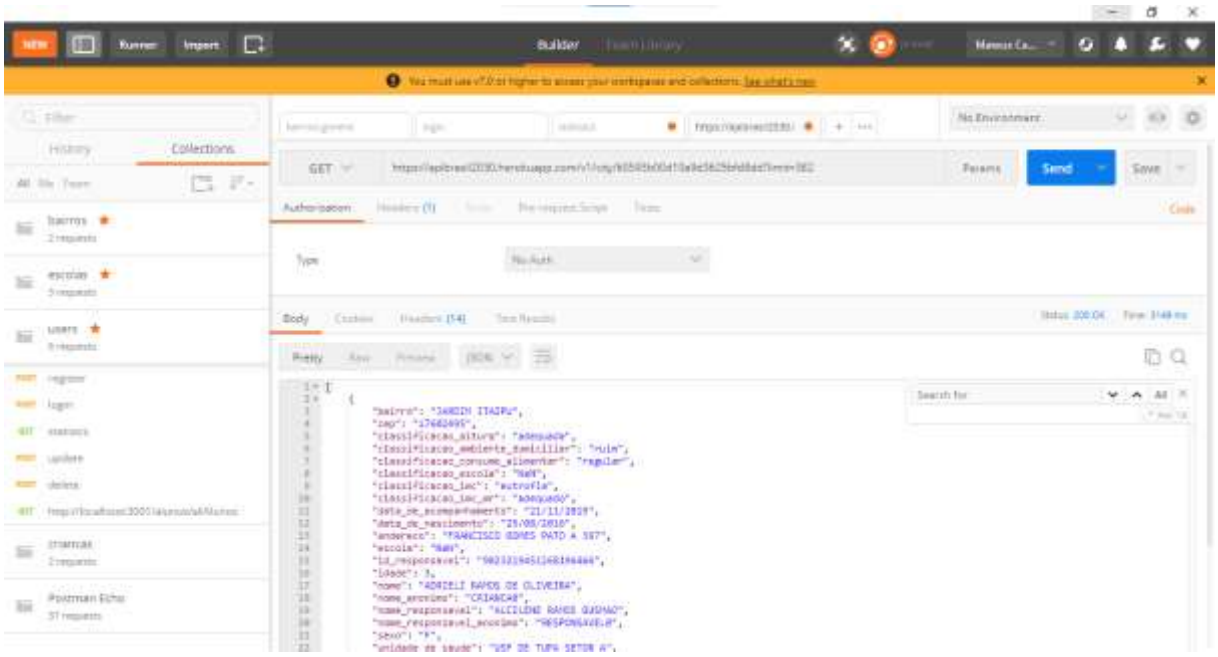


Figura 32: Teste de tempo requisição endpoint crianças Brasil2030

4.4 HACKATHON

Com o objetivo de ter uma arquitetura escalável para grande quantidade e variedade de dados, além de proporcionar uma bom tráfego e boa usabilidade para o usuário, a arquitetura construída foi utilizada na participação de um hackathon global promovido através de uma iniciativa SWELife da empresa SAS Analytics na Hackathon Prevent Childhood Obesity (Hackathon Global de Obesidade Infantil) que ocorreu na Suécia.

Fazendo parte do time Tupã-Fit e utilizando a cidade de Tupã-SP como cenário para o projeto, foi possível apresentar no Hackathon um breve funcionamento da aplicação e disponibilizar uma primeira versão para que pais da cidade pudessem usar e analisar a aplicação. Entre os dias 26 e 31 de março, 180 pais da cidade de Tupã utilizaram a ferramenta Brasil2030 em um piloto controlado gerenciado por gestores públicos. Em pesquisa realizada com 61 pais sobre a ferramenta, todas as 8 questões respondidas tiveram resultados positivos, o que pode dar uma melhor noção do funcionamento da ferramenta em um cenário real. Na figura 33 é possível visualizar o resultado da pesquisa com mais detalhes.

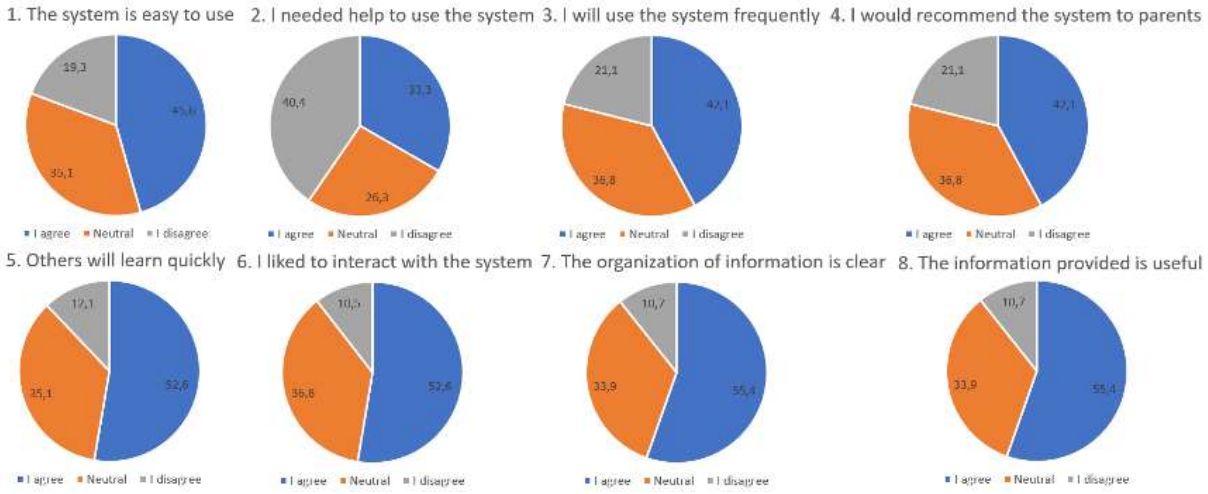


Figura 33: Pesquisa Sistema Brasil2030

Obtendo altíssimos graus de satisfação por parte dos usuários na apresentação do Hackathon, em meio a diversas categorias como: saúde, indústria, telecomunicações, entre outros, no dia 11 de março o time Tupã Fit com o projeto "Brasil2030", foi o vencedor do Hackathon Global de Obesidade Infantil possibilitando uma nova etapa de testes de viabilidade da solução que se desenrolará durante o resto do ano de 2021 e que possivelmente será experimentada também em uma cidade na Suécia.

Figura 34: Vencedor Hackathon Global de Obesidade Infantil. Fonte: SAS Software

5 CONCLUSÃO

Buscando o objetivo do projeto, uma arquitetura geral que suprisse as necessidades básicas dos requisitos propostos foi construída. Com embasamento de arquiteturas e tecnologias usadas recentemente por outros sistemas em big data, foi possível desenvolver o projeto com uma arquitetura de micro serviços onde tanto a parte do cliente quando do servidor da aplicação poderá escalar e se comunicar com outros serviços futuramente se necessário. Nesta etapa inicial o sistema realiza comunicação com um banco de dados alimentado com informações de crianças da cidade de Tupã-SP junto a uma lógica fuzzy aplicada as informações trazidas em conjunto pelo SISVAN, OMS e CADÚNICO.

Tendo a escalabilidade como um dos requisitos, o uso de um sistema em micro serviços permite com que a arquitetura futuramente possa utilizar diretamente um banco de dados do SISVAN por exemplo ou até mesmo um serviço de mensageria para que os pais das crianças recebam um e-mail ou notificação no seu smartphone quando alterado algum indicador de obesidade de seu filho. A arquitetura também permite um acoplamento com qualquer serviço de mensageria que pode ser usado no sistema sem comprometer qualquer parte do Brasil2030. O sistema Brasil2030 poderá se integrar também com qualquer outro sistema como o Portal da Transparência e buscar mais informações sobre recursos de uma cidade melhorando ainda mais seus indicadores gerais. É possível também se integrar com diversos sistemas de saúde que também podem passar mais informações sobre saúde de crianças de uma determinada cidade ou até mesmo de um outro país no geral. Portanto, nesse âmbito, a arquitetura tem a possibilidade de se integrar a qualquer outro sistema e cada vez se tornar mais robusta e completa.

No âmbito da velocidade e adaptabilidade, necessários para comunicação e integração com outros sistemas e informações, a tecnologia usada no desenvolvimento do sistema Brasil2030 é uma das mais recentes e que tem suporte ativo. O Node.js, tecnologia utilizada para o desenvolvimento do Brasil2030 é uma plataforma muito versátil e que pode ser usada em inúmeros cenários. Seu gerenciador de pacotes NPM (Node Package Manager) é classificado como o maior repositório de softwares disponível. Um grande diferencial também do Node no quesito velocidade mais exatamente é a execução do Node single thread, quando apenas uma thread executa o código da aplicação. Dessa maneira, menos recursos computacionais são exigidos, pois não é necessário criar uma nova thread para cada requisição recebida, suprindo assim a necessidade de boa velocidade de resposta para diversas requisições.

As tecnologias utilizadas para desenvolvimento do cliente por serem uma das mais recentes, também tem suporte ativo e constante. Em lista divulgada pelo site onebitcode.com em 2020, cerca de 12 bibliotecas foram listadas que tem suporte para React que auxiliam na agilidade do desenvolvimento da aplicação. Já o Kotlin, utilizado no desenvolvimento mobile, segundo site oficial é compatível com o desenvolvimento do Google para Android, o que significa que a documentação e as ferramentas do Android foram desenvolvidas com o Kotlin em mente, o que permitiu construir cliente completamente nativo na versão mobile.

6 REFERÊNCIAS

TAURON, Cesar. **Big Data**. Rio de Janeiro: Brasport, 2013.

SHAW, D. Garlan; Software Architecture. **Perspectives on an Emerging Discipline, Prentice Hall**. 1996.

SHAW, Garlan e Mary. **An introduction to software architecture. Technical Report- CMU-CS-94166**. Carnegie Mellon University, January 1994.

Shaw, D. Garlan; **Software Architecture. Perspectives on an Emerging Discipline**, Prentice Hall, 1996.

Equipe ESSS. **Conheça os pilares da indústria 4.0**. 2017. Disponível em: <https://www.esss.co/blog/os-pilares-da-industria-4-0/>. Acesso em: 17 mai. 2021.

Intel IT Center. **Guia de Planejamento Introdução à Big Data.** 2014. Disponível em: <https://www.intel.com.br/content/dam/www/public/lar/br/pt/documents/articles/e7-big-data-planning-guide-webready-por.pdf>. Acesso em: 07 mai. 2021.

Azadeh Eftekhari, Farhana Zulkernine, and Patrick Martin. **Binary: A framework for big data integration for ad-hoc querying.** IEEE International Conference. 2016.

Wo L. Chang. **NIST Big Data Interoperability Framework: Volume 1, Definitions.** 2019. Disponível em: <https://doi.org/10.6028/NIST.SP.1500-1r2>. Acesso em: 20 mai. 2021.

Sahar Hussain Jambi. **Engineering Scalable Distributed Services for Real-Time Big Data Analytics.** IEEE. 2017.

Han Hu, Yonggang Wen, Tat-Seng Chua, and Xuelong Li. **Toward scalable systems for big data analytics: A technology tutorial.** 2014. Disponível em: <https://ieeecs-media.computer.org/assets/pdf/T-06842585.pdf>. Acesso em: 12 jun. 2021.

Rodrigues. Lívia, **Arquitetura REST.** 2009. Disponível em: <http://monografias.ice.ufjf.br/tcc-web/exibePdf?id=17>. Acesso em: 17 jul. 2021.

FIELDING, Roy. **Architectural Styles and the Design of Network-based Software Architectures.** 2000. Disponível em: <https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>. Acesso em: 17 jul. 2021.

RUBACK, Lívia. **Arquitetura REST: Uma alternativa para construção de Serviços Web.** 2009. Disponível em: <https://www.devmedia.com.br/arquitetura-rest-uma-alternativa-para-construcao-de-servicos-web/15150>. Acesso em: 02 ago. 2021.

FILHO, Cezar Bastos. **Reuso de Software**. 2013. Disponível em: <http://www.uel.br/cce/dc/wp-content/uploads/VersaoPreliminarTCC-CezarBastos.pdf>. Acesso em: 25 nov. 2020.

SADALAGE, P.; FOWLER, M. **NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence.** 2013. Disponível em: <https://bigdata-ir.com/wp-content/uploads/2017/04/NoSQL-Distilled.pdf>. Acesso em: 20 mar. 2021.

KULESZA, SOUSA, et. al. **Evolução das arquiteturas de software rumo à Web 3.0.** 2018. Disponível em: <https://sol.sbc.org.br/livros/index.php/sbc/catalog/view/8/15/54-1>. Acesso em: 08 set. 2021.

YAMASSAKI, Vivian Mayumi. **Combinando modelos de Machine Learning com Lógica Fuzzy — Parte 1.** 2020. Disponível em: <https://medium.com/creditas-tech/combinando-modelos-de-machine-learning-com-l%C3%B3gica-fuzzy-parte-1-b5a9f0761a5d>. Acesso em: 08 set. 2021.

SOUZA, Thiago de. **Tupã Fit: Brasil2030 - an intelligent system for multisectoral use in promoting the nutritional health of children**. 2021. Disponível em: <https://communities.sas.com/t5/Hacker-s-Hub-library/Tup%C3%A3-Fit-Brasil2030-an-intelligent-system-for-multisectoral-use/ta-p/731435>. Acesso em: 05 set. 2021.

CONTINI, Guilherme. **Brasil 2030**. 2021. Disponível em: <https://www.guilhermecontini.com.br/brasil-2030>. Acesso em: 05 set. 2021.

Rede nacional de ensino e pesquisa. **RNP articula consórcio brasileiro vencedor do Hackathon Global de Obesidade Infantil**. 2021. Disponível em: <https://www.rnp.br/noticias/rnp-articula-consorcio-brasileiro-vencedor-do-hackathon-global-de-obesidade-infantil>. Acesso em: 05 set. 2021.

SAS Software. **SAS Global Hackathon Regional Awards Ceremony**. 2021. Disponível em: <https://www.youtube.com/watch?v=PilC-nAn3Lg&t=1349s>. Acesso em: 05 set. 2021.

Equipe TOTVS. **Node.js: O que é, quais as características e vantagens?.** 2020. Disponível em: <https://www.totvs.com/blog/developers/node-js/>. Acesso em: 09 set. 2021.

VASKEVITCH, David. **Estratégia Cliente/Servidor: um guia para a reengenharia da empresa.** São Paulo: Berkeley, 1995.

KRUPPA, Hugo. **Demystify BIG DATA Definitions and Architecture according to NIST.** 2017. Disponível em: <https://www.linkedin.com/pulse/demystify-big-date-definitions-architecture-according-hugo-kruppa/?published=t >. Acesso em: 21 fev. 2022.

REIS, Marco. **Uma Arquitetura de Big Data as a Service Baseada no Modelo de Nuvem Privada.** 2018. Disponível em: < https://repositorio.unb.br/bitstream/10482/33759/1/2018_MarcoAnt%C3%B4niodeSousaReis.pdf>. Acesso em: 21 fev. 2022.

FELIX, Waldyr. **Como escolher entre arquitetura de microsserviços e monolítica.** 2019. Disponível em: <https://waldyrfelix.com.br/os-crit%C3%A9rios-de-decis%C3%A3o-para-escolher-entre-os-estilos-de-arquitetura-monol%C3%ADtica-e-microsservi%C3%A7o-f2534be6a575>. Acesso em: 21 fev. 2022.

OPUS Software. **Micro Serviços: Qual a diferença para a Arquitetura Monolítica?** 2021. Disponível em: <https://www.opus-software.com.br/micro-servicos-arquietura-monolitica/>. Acesso em: 21 fev. 2022.