UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE ENGENHARIA ELÉTRICA E ELETRÔNICA
CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Arthur Moreira de Deus

APLICATIVO PARA MONITORAMENTO DE ESTUFA AGRÍCOLA COM COMUNICAÇÃO LORAWAN

FLORIANÓPOLIS
2022

Arthur Moreira de Deus

APLICATIVO PARA MONITORAMENTO DE ESTUFA AGRÍCOLA COM COMUNICAÇÃO LORAWAN

Trabalho de Conclusão de Curso submetido à Universidade Federal de Santa Catarina, como requisito necessário para obtenção do grau de Bacharel em Engenharia Elétrica

Florianópolis, Julho de 2022

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Deus, Arthur Moreira de
Aplicativo para monitoramento de estufa agrícola com comunicação LoRaWAN / Arthur Moreira de Deus ; orientador, Richard Demo Souza, 2022.
63 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Engenharia Elétrica, Florianópolis, 2022.

Inclui referências.

1. Engenharia Elétrica. 2. LoRaWAN. 3. Monitoramento. 4. Django. 5. React Native. I. Demo Souza, Richard. II. Universidade Federal de Santa Catarina. Graduação em Engenharia Elétrica. III. Título.

Arthur Moreira de Deus

Aplicativo para monitoramento de estufa agrícola com comunicação LoRaWAN

Este Trabalho Conclusão de Curso foi julgado adequado para obtenção do Título de "Bacharel em Engenharia Elétrica" e aceito, em sua forma final, pelo Curso de Graduação em Engenharia Elétrica.

Florianópolis, 29 de julho de 2022

Documento assinado digitalmente
Miguel Moreto
Data: 01/08/2022 10:02:56-0300
CPF: 948.850.100-63
Verifique as assinaturas em https://v.ufsc.br

Prof. Miguel Moreto, Dr.
Coordenador do Curso de Graduação em Engenharia Elétrica

Banca Examinadora:

Documento assinado digitalmente
Richard Demo Souza
Data: 01/08/2022 09:59:46-0300
CPF: 004.267.379-89
Verifique as assinaturas em https://v.ufsc.br

Prof. Richard Demo Souza, Dr.
Orientador
Universidade Federal de Santa Catarina

Documento assinado digitalmente
BARTOLOMEU FERREIRA UCHOA FILHO
Data: 01/08/2022 16:17:06-0300
CPF: 476.362.114-91
Verifique as assinaturas em https://v.ufsc.br

Prof. Bartolomeu F. Uchoa Filho, Dr.
Universidade Federal de Santa Catarina

Documento assinado digitalmente
MARIO DE NORONHA NETO
Data: 01/08/2022 11:57:28-0300
CPF: 003.859.519-22
Verifique as assinaturas em https://v.ufsc.br

Prof. Mario de Noronha Neto, Dr.
Instituto Federal de Santa Catarina

Agradecimentos

Gostaria de agradecer primeiramente aos meus pais por todos os ensinamentos, incentivos e principalmente por todo o amor que sempre me deram. Serei eternamente grato por tudo.

Aos meus avós, que sempre estiveram lá quando precisamos e fizeram de tudo pelos filhos e netos. Se não fosse por vocês, nada disso seria possível.

À minha irmã Bruna por ter sido sempre um grande exemplo para mim e por ter me aguentado enquanto moramos juntos durante a pandemia.

À minha namorada Carla por ser a melhor companheira que eu poderia imaginar ter. Obrigado pelo carinho, pelos conselhos e pela paciência.

À gêmea Caroline e à família Bedin por me receber de braços e corações abertos e por terem me apresentado a capital do Oeste.

Ao professor Richard, que tornou tranquilo o desafio de desenvolver e escrever um TCC. Obrigado por todo o suporte, apoio e orientação.

Aos amigos que me acompanharam nesta caminhada: Douglas Nadai, Guilherme Hosoda, Julia Nunes, Lucas Fiamoncini, Marcello Ferreira, Matheus Hohmann e Nicolas Yamakoshi. Obrigado pela parceria e pelas risadas.

Ao meu primo Ricardo, parceiro de longa data, companheiro de estudos desde as épocas do IFSC e futuro colega de profissão.

Aos amigos que a UFSC me deu: Lucas Augusto, Rafael Graefling e Vitor Goulart. Obrigado pelos papos e pelos rolês.

Ao PET EEL, aos tutores Prof. Walter e Prof. André e aos colegas com quem tive o prazer de compartilhar os dois anos em que estive no grupo.

Aos amigos da Jungle Devs por terem me apadrinhado e mentorado nos meus primeiros passos no mundo do desenvolvimento de software. Aos amigos da Parade por estarem comigo no dia a dia, fazendo com que o trabalho seja um prazer e não uma obrigação.

Aos companheiros de time do Peñarohm e da Panela EEL pela parceria e pelos títulos.

Resumo

O cultivo de alimentos orgânicos é uma área da agricultura que vem crescendo na última década em um ritmo bastante rápido, de forma a contribuir cada vez mais para a oferta de alimentos, principalmente em entressafra. O uso de estufas é comum nesta modalidade de cultivo, uma vez que oferece ao agricultor maior controle sobre variáveis climáticas no ambiente. Com isso, este trabalho tem o objetivo de desenvolver um aplicativo para dispositivos móveis que possibilite o monitoramento de temperatura, umidade e pressão no interior de estufas, fazendo uso de um dispositivo LoRaWAN para fazer a leitura e transmissão dos dados. A metodologia adotada consiste em realizar a configuração do dispositivo LoRaWAN e conectá-lo à rede, desenvolver uma aplicação *backend* que receba e armazene os dados das leituras, bem como os expõe para consumo no aplicativo móvel. Como resultado, foi possível desenvolver todo o fluxo de dados desde as medições até o consumo nos *endpoints* da API, bem como o *design* e a implementação do aplicativo móvel, elencando e justificando as tecnologias empregadas em cada área.

Palavras-chave: LoRaWAN. Django. React Native. Internet das Coisas. Monitoramento.

Abstract

Organic food farming is an area of agriculture which has been growing rapidly in the last decade, contributing to food offer mainly during the off seasons. Greenhouses are common to be employed in this farming technique, given it provides to the farmer more control over the environment variables. This way, this work has the goal to develop a mobile application that allows monitoring temperature, humidity and pressure inside greenhouses, using a LoRaWAN device to measure and transmit the data. The methodology adopted consists in setting up the LoRaWAN device and connect it to the network, deveop a backend application to receive and store the measurement data as well as expose this data to the mobile app. As results, it was possible to develop the whole data flow from the measurements to the data access in the API endpoints, while developing the application according to the conceived design, pointing and justifying the technologies used in each step.

Keywords: LoRaWAN. Django. React Native. Internet of Things. Monitoring.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

LoRaWAN - Protocolo de comunicação que utiliza LoRa como técnica de acesso ao meio. Sigla para *Long Range Wide Area Networking.*

LPWAN - Sigla em inglês para redes de baixa potência e longo alcance - *Low Power Wide Area Networks*

IoT - Sigla em inglês para internet das coisas - *Internet of Things*

M2M - Sigla em inglês para interface máquina-máquina - *Machine to Machine*

CSS - Sigla em inglês - *Chirp Spread Spectrum*

GCP - Google Cloud Platform - Plataforma de serviços em nuvem da Google

DRF - Django REST Framework - Conjunto de ferramentas construídas sobre o *framework* Django

REST - Sigla em inglês para Transferência Representacional de Estados - Representational State Transfer

API - Sigla em inglês para Interface de Programação de Aplicação - *Application Programming Interface*

IP - Sigla em inglês para protocolo de internet - Internet Protocol

TTN - The Things Network - Ecossistema colaborativo global que utiliza LoRaWAN

UFSC - Universidade Federal de Santa Catarina

URL - Sigla em inglês para localizador uniforme de recursos - Uniform Resource Locator

HTTP - Sigla em inglês para protocolo de transferência de hipertexto - Hypertext Transfer Protocol

WSL2 - Sigla em inglês para subsistema Windows para Linux 2 - Windows Subsystem for Linux 2

Mapa - Ministério da Agricultura, Pecuária e Abastecimento

Sumário

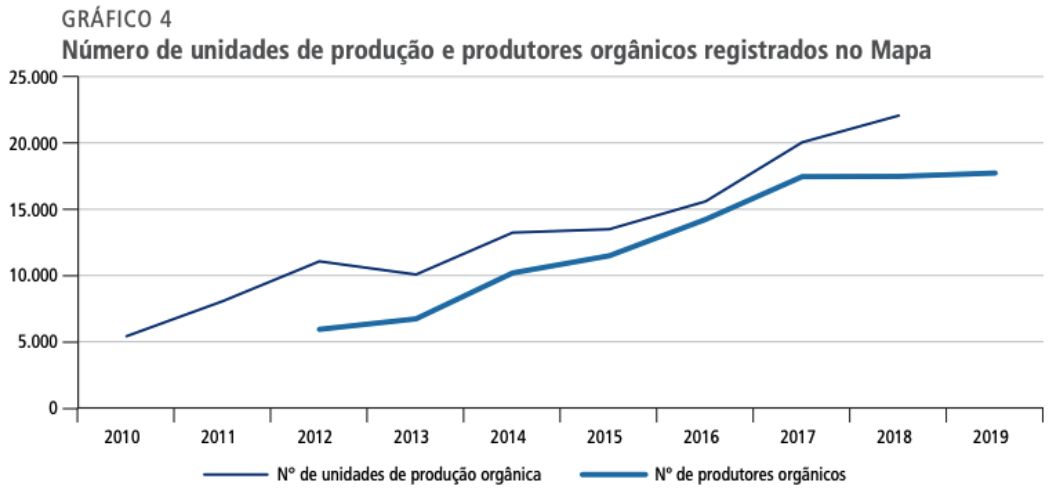
1 Introdução

O conteúdo do presente texto aborda o desenvolvimento de um aplicativo móvel para monitoramento de estufas agrícolas utilizando comunicação LoRaWAN.

1.1 Contextualização

Segundo estudo realizado pelo Instituto de Pesquisa Econômica Aplicada (IPEA), entre os anos 2000 e 2017, a área destinada ao cultivo de vegetais orgânicos cresceu em média 10% ao ano [Boiteux 2020]. Um gráfico com dados deste estudo é apresentado na Figura 1.

Figura 1 – Número de unidades de produção e produtores orgânicos registrados no Ministério da Agricultura, Pecuária e Abastecimento (Mapa)



Fonte: [Boiteux 2020]

Dentro desta área crescente da agricultura, o uso de estufas de cultivo é realizado principalmente quando há demanda pelos produtos em sua entressafra [Machado 2019]. As estufas proporcionam às plantas condições ambientais semelhantes às da época de sua safra, permitindo ao agricultor complementar seu calendário produtivo.

Para que seja feito o controle das variáveis de ambiente da estufa a fim de proporcionar condições ótimas de cultivo, são necessários equipamentos tais como aquecedores, nebulizadores, lâmpadas, entre outros. Estes equipamentos podem ser acionados manualmente ou automaticamente, fazendo uso de sensores e atuadores.

O emprego de sensores no ambiente interno da estufa abre espaço para que estas variáveis aferidas sejam monitoradas remotamente. Com o intuito de desenvolver uma

alternativa para monitorar convenientemente estas variáveis, chegou-se aos objetivos do presente trabalho.

1.2 Objetivo Geral

Desenvolver um aplicativo para dispositivos móveis para monitoramento de estufas agrícolas com comunicação LoRaWAN.

1.3 Objetivos Específicos

Visto o objetivo geral exposto e considerando o desenvolvimento do trabalho, os objetivos específicos são os apresentados a seguir:

1. Entender o processo de desenvolvimento de aplicativos para dispositivos móveis do começo ao fim;
2. Elencar as principais tecnologias e plataformas que permitem o desenvolvimento rápido de aplicações;
3. Compreender e aplicar o protocolo de comunicação LoRaWAN.

2 Revisão de Literatura

2.1 Protocolo LoRaWAN

2.1.1 LPWANs

As LPWANs são tecnologias de rede de baixa potência capazes de suprir grande parte das necessidades oriundas do nicho de IoT. As LPWANs complementam as redes tradicionais de celular e comunicação de baixo alcance ao proporcionar conectividade em grandes áreas para dispositivos de baixo consumo e baixa taxa de transmissão de dados. É esperado que aproximadamente um quarto de um mercado de 30 milhões de dispositivos IoT/M2M conectem-se à internet através de LPWANs [Raza, Kulkarni e Sooriyabandara 2017]. Alguns exemplos de LPWANs são a SigFox [Sigfox.com 2022] e a LoRaWAN [What is LoRaWAN 2021].

Neste capítulo, será abordado com maior profundidade as características da tecnologia LoRaWAN, que foi a escolhida para o desenvolvimento do trabalho.

2.1.2 LoRa

LoRa é um tipo de modulação *Chirp Spread Spectrum* (CSS) robusta, adquirida e comercializada pela Semtech. A camada física LoRa foi desenvolvida com foco no *uplink* de dados, proporcionando um baixo consumo de energia, baixa taxa de transmissão de dados e alcance na casa de quilômetros. É possível experimentar diferentes configurações nos parâmetros da modulação, havendo um compromisso entre largura de banda e robustez da transmissão [Boquet et al. 2021].

O que determina quais parâmetros da camada física podem ser usados é a especificação LoRaWAN, uma vez que LoRa é a tecnologia utilizada na camada física deste protocolo. Esses parâmetros impactam diretamente no alcance, na taxa de dados e consumo de energia.

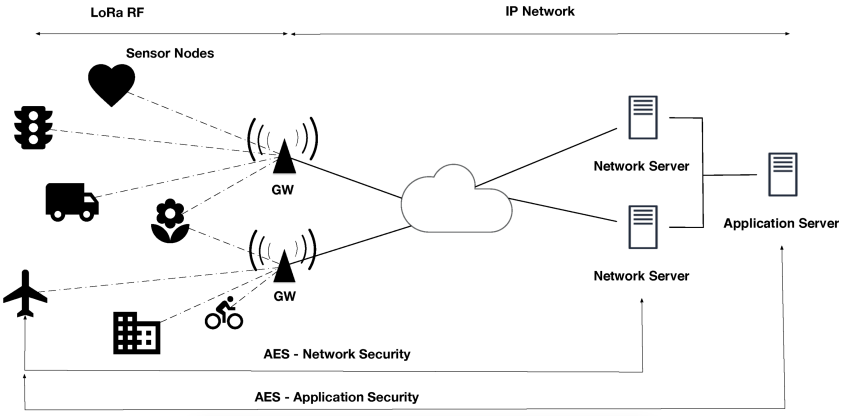
Uma abordagem mais detalhada da camada física não é o foco deste trabalho. No entanto, o leitor interessado pode ler mais a respeito em [Devalal e Karthikeyan 2018] e [What is LoRaWAN 2021].

2.1.3 Arquitetura de rede

Do ponto de vista de arquitetura, a topologia de rede LoRaWAN apresenta três principais componentes, conforme representado na Figura 2.

- Servidor de Rede
- *Gateways*
- Dispositivos (também chamados de nós)

Figura 2 – Arquitetura de rede LoRaWAN



Fonte: [Ertürk et al. 2019]

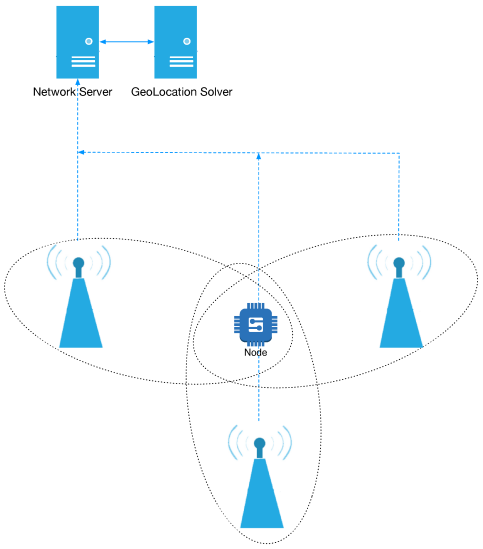
LoRaWAN consiste no protocolo de comunicação e arquitetura de rede, enquanto a camada física LoRa permite o enlace de comunicação de longo alcance. A LoRaWAN apresenta arquitetura em estrela, ou seja, os dispositivos, também chamados de nós, se comunicam diretamente com os *gateways*, não estando associados a nenhum *gateway* específico.

Ao contrário da arquitetura de rede em malha, na qual os dispositivos que a constituem são responsáveis pela transmissão das informações uns dos outros, a comunicação direta entre os dispositivos e os *gateways* resulta numa maior eficiência energética para os nós, uma vez que são responsáveis por transmitir somente os seus dados. [Haxhibeqiri et al. 2018]

Por não estarem associados a um *gateway* específico, os dados transmitidos podem ser recebidos por mais de um *gateway*. Isso torna possível a geolocalização de dispositivos através da triangulação do sinal. Para que isso ocorra, uma mensagem qualquer enviada por um dispositivo precisa ser recebida por pelo menos 3 *gateways*, conforme representado na Figura 3. Então, através de métodos como RSSI ou TDOA, é possível determinar a localização do nó com resolução de 200 à 20 metros. [Ertürk et al. 2019]

Os *gateways*, por sua vez, são responsáveis por transmitir os dados do dispositivo para o servidor de rede, podendo esta transmissão ocorrer via Wi-Fi, satélite, redes celulares ou internet via cabo. O servidor de rede é responsável por fazer o gerenciamento de toda a informação que chega até ele, absorvendo grande parte da complexidade da rede. Nele são realizadas checagens de segurança, filtragem de dados repetidos, entre outros procedimentos. [What is LoRaWAN 2021]

Figura 3 – Infraestrutura necessária para rastreamento de dispositivo



Fonte: [Ertürk et al. 2019]

2.1.4 Classes de dispositivo e aplicações

O protocolo LoRaWAN apresenta um vasto leque de aplicações. Essa versatilidade é possível através de três classes de dispositivos, que apresentam características e finalidades diferentes [Pukrongta e Kumkhet 2019].

- Classe A: É a classe com menor consumo de energia e taxa de transferência de dados e deve ser suportada por todos os dispositivos. A comunicação é iniciada sempre a partir do dispositivo e, após cada transmissão feita pelo dispositivo, são abertas duas janelas de recebimento de dados (*downlink*) a partir do *gateway*. Os dispositivos classe A são comumente alimentados à bateria [Cheong et al. 2017] e apresentam longos intervalos entre transmissões e alta latência para *downlink*. Podem ser aplicados em rastreamento, detecção de incêndios, medição de qualidade do ar [Miao et al. 2022], entre diversas outras aplicações não sensíveis ao tempo.

- Classe B: Dispositivos de classe B periodicamente abrem janelas de *downlink* através de sinais denominados *beacons* transmitidos pelo *gateway*. Esta classe apresenta uma menor latência de *downlink* do que a classe A, já que é possível acessar os dispositivos em tempos pré configurados. No entanto, há um compromisso entre a menor latência e o maior consumo de energia [Cheong et al. 2017], uma vez que os dispositivos desta Classe passam mais tempo ativos. Uma aplicação comum dos dispositivos de classe B são em medidores de energia elétrica e água [Lalle et al. 2020]. Estas aplicações se devem ao fato de não serem altamente sensíveis ao tempo, mas requererem alguma previsibilidade no intervalo de comunicação.

- Classe C: Os dispositivos ficam com a janela de *downlink* sempre aberta, exceto

quando estão fazendo transmissões. Isso possibilita aplicações sensíveis ao tempo, já que os dispositivos possuem baixa latência. Estes dispositivos comumente apresentam alimentação externa, uma vez que apresentam um maior consumo energético em relação à classe B [Cheong et al. 2017]. Os dispositivos desta classe podem ser aplicados em iluminação pública e medidores de água e energia com seccionamento. Nestas aplicações, é necessário que haja uma baixa latência de *downlink*, uma vez que é desejado poder enviar comandos para o dispositivo executar ações sensíveis ao tempo.

2.1.5 Vantagens e desvantagens

O protocolo LoRaWAN apresenta características positivas no que diz respeito à autonomia de bateria e alcance de transmissão, tendo dispositivos com vida útil de bateria em torno de 10 anos, alcance de 2 a 5 quilômetros em áreas urbanas e até 15 quilômetros em áreas rurais [Adelantado et al. 2017].

No entanto, tais benefícios só são possíveis graças ao compromisso da taxa de transferência de dados e latência dos dispositivos [Raza, Kulkarni e Sooriyabandara 2017]. Como apresentado anteriormente, ambas estas características variam de acordo com a classe do dispositivo em questão.

2.2 Desenvolvimento *Full Stack*

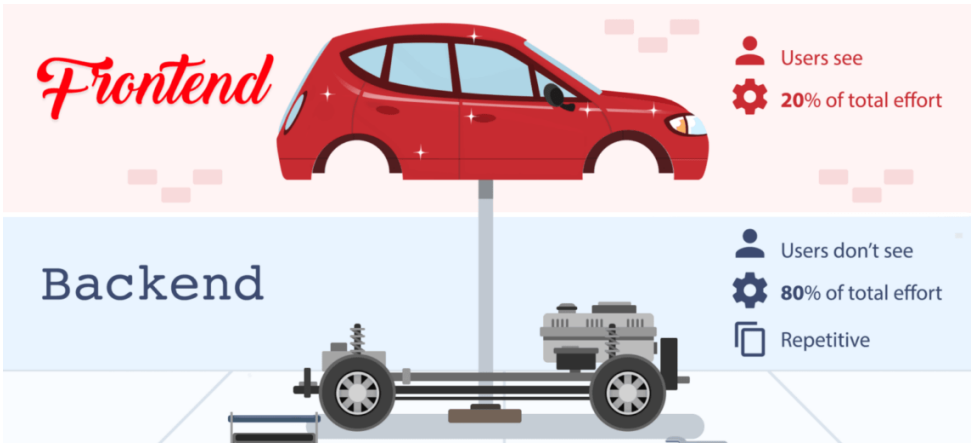
2.2.1 Conceitos Básicos

O desenvolvimento *web* moderno pode ser dividido em duas grandes áreas: *frontend* e *backend*. Uma analogia interessante para entender melhor o papel de cada uma dessas áreas é comparar um produto digital com um carro, conforme ilustado na Figura 4. Toda a parte exterior, com a qual os usuários do veículo usualmente interagem, tais como a lataria, vidros, interior, etc, pode ser relacionada ao *frontend*. Ou seja, tudo aquilo que o usuário pode ver e interagir [Gerasimov et al. 2020].

Já o *backend* pode ser comparado à toda a parte mecânica do carro: o motor, câmbio, transmissão, suspensão, etc. São partes imprescindíveis para o funcionamento do veículo, mas que não interagem diretamente com os usuários do carro. No *backend* é onde se encontram as lógicas de negócio e onde é feita a manipulação e armazenamento dos dados de uma aplicação [Gerasimov et al. 2020].

As interações entre *frontend* e *backend* podem ser feitas, no exemplo do carro, através dos pedais, do volante, da alavanca de câmbio, entre outros meios. No caso de *softwares*, essas interações podem se dar através de cliques em botões, envios de formulários,

Figura 4 – Comparativo de *frontend* e *backend*



Fonte: [Clark]

inserção de textos em caixas de texto, etc. Dessa forma, os dados podem trafegar do usuário para a aplicação e vice versa.

O desenvolvedor *Full Stack* é um indivíduo que possui entendimento de toda a *stack*, ou seja, de todas as áreas tecnológicas envolvidas em um projeto de *software*. Em geral, essas áreas são *frontend*, *backend*, gerenciamento de projetos, infraestrutura do sistema, bancos de dados e ferramentas de controle de versão [Vainikka 2018].

Um exemplo de *stack* bastante popular no mercado é a LAMP - Linux Apache, MySQL e PHP. Composta por um conjunto de *softwares* de código aberto, Linux é a tecnologia de sistema operacional, Apache de servidor de aplicação, MySQL de banco de dados e PHP de linguagem de programação de *backend* [Karanjit 2016].

Nesta seção, abordaremos com mais detalhes uma *stack* mais moderna, constituída por Python como linguagem e Django como *framework* para o *backend*, JavaScript como linguagem e React Native como *framework* para *frontend* e Google Cloud Platform como provedor de infraestrutura em nuvem.

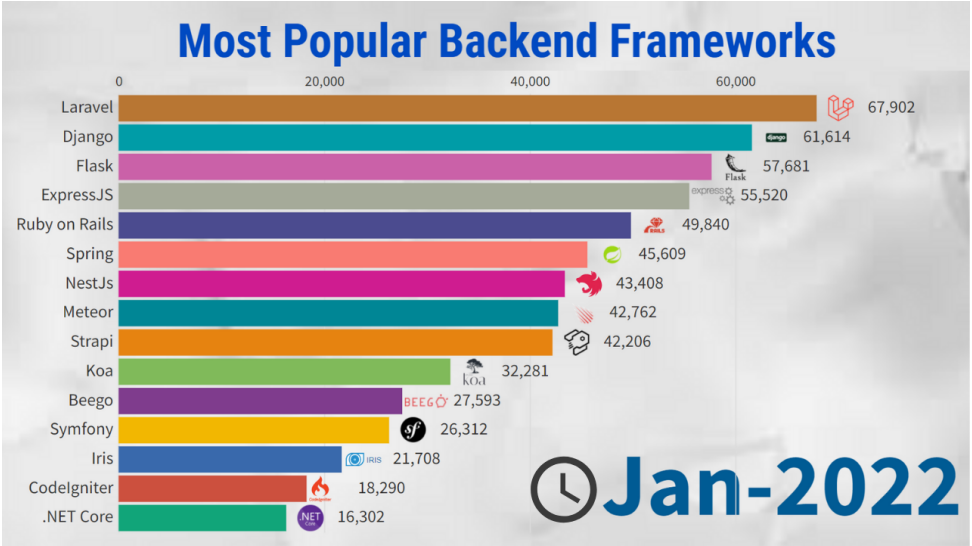
2.2.2 Desenvolvimento Backend com Python e Django

Python é uma linguagem de programação orientada a objetos, de alto nível, interpretada e que apresenta uma sintaxe bastante amigável e de fácil entendimento [Forcier, Bissex e Chun 2008]. Tornou-se bastante popular no desenvolvimento *web* por tornar o código fácil de ser lido e, assim, tornar mais ágil o desenvolvimento de aplicações. Além disso, Python é bastante utilizado em aplicações de aprendizado de máquina e computação científica, oferecendo diversas bibliotecas e APIs para tais finalidades [Raschka, Patterson e Nolet 2020].

Django é um *framework* construído em Python lançado em 2005, com foco no desenvolvimento *backend*. No entanto, é possível desenvolver interfaces de usuário utilizando

sua linguagem de *templates*. Atualmente, é o segundo *framework* de *backend* mais popular do mundo, conforme dados apresentados na Figura 5.

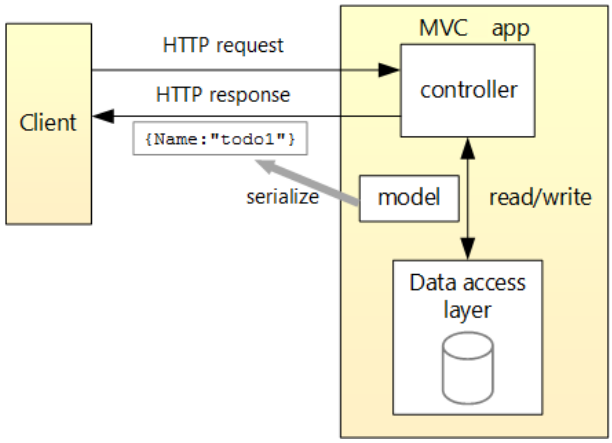
Figura 5 – Frameworks de *backend* mais utilizados



Fonte: [Statistics and Data 2022]

Apesar de ser possível desenvolver tanto o *backend* quanto as interfaces com Django, para aplicações profissionais costuma-se usá-lo em conjunto com o Django REST Framework (DRF), de modo a fornecer uma API REST de uma maneira mais conveniente. Desta maneira, a responsabilidade do Django torna-se somente lidar com as requisições do cliente (usuário), prover e modificar dados [Vainikka 2018], tornando sua arquitetura conforme representado na Figura 6.

Figura 6 – Arquitetura de API REST



Fonte: [Anderson e Larkin]

2.2.3 Desenvolvimento Multiplataforma com React Native

Inicialmente, para desenvolver aplicativos para dispositivos móveis era necessário desenvolvê-lo de forma nativa, demandando na prática um projeto para cada sistema

operacional diferente. Atualmente, existem alternativas para tal, tornando possível desenvolver um único serviço que seja acessado por dispositivos com diferentes sistemas operacionais, sem perder as funcionalidades nativas [Danielsson 2016]. Uma destas alternativas é desenvolver a aplicação utilizando um *framework* híbrido, como é o caso do React Native.

React Native foi lançado pela Meta (antiga Facebook) em 2015 com a premissa de revolucionar o desenvolvimento de aplicativos móveis, introduzindo o conceito de escrever um único código para múltiplas plataformas [Danielsson 2016]. O código é escrito em JavaScript e pode ter seu conteúdo renderizado em aplicativos iOS, Android e até mesmo *web*. Para entender como o React Native é capaz de funcionar desta maneira, o leitor interessado pode aprofundar-se em [React Native Internals].

2.2.4 Infraestrutura em Nuvem

Computação em nuvem é um termo bastante amplo. Neste trabalho, o termo infraestrutura em nuvem se refere ao tipo de técnica de em que serviços de tecnologia da informação são disponibilizados a baixos preços por potentes unidades de computação conectadas por redes IP [Qian et al. 2009].

Para que uma aplicação seja disponibilizada ao público, é necessário hospedá-la em algum tipo de servidor. A grosso modo, isso pode ser feito de duas formas: executando a aplicação em um servidor local, prática conhecida como *On Premises*, ou então utilizando uma infraestrutura em nuvem para tal, sem a necessidade de adquirir nenhum tipo de *hardware*. Uma vantagem deste último é a disponibilidade de diversos planos de cobrança sob demanda oferecidos pelas plataformas de computação em nuvem, tornando acessível o lançamento de aplicações [Gupta, Mittal e Mufti 2021].

No entanto, para o caso de computação de alto desempenho (HPC), pode ser mais viável economicamente adquirir ou montar um supercomputador do que utilizar serviços em nuvem para tal, conforme apresentado por [Carlyle, Harrell e Smith 2010].

Antes de implementar uma aplicação em nuvem, é necessário provisionar os serviços necessários para que ela seja executada conforme o esperado. Uma possível infraestrutura para um *backend* mais simples consiste em provisionar um serviço de banco de dados relacional, um serviço de armazenamento de objetos e um serviço de contêineres escalável para a execução da aplicação.

Em sistemas mais complexos, pode se fazer necessário um serviço de fila de tarefas, tal como Celery ou RabbitMQ, serviços de busca, como Elasticsearch, entre vários outros, de acordo com as demandas da aplicação.

Atualmente, segundo [Gupta, Mittal e Mufti 2021], os principais provedores de infraestrutura em nuvem são Amazon Web Services [About AWS 2018], Google Cloud

Platform [Google Cloud Platform] e Microsoft Azure [O que é o Azure].

3 Metodologia

3.1 Definição da área de aplicação do projeto

Conforme apresentado no Capítulo 1, o cultivo de alimentos orgânicos tornou-se uma tendência para a agricultura nos últimos anos, dispondo de um mercado crescente e com muitas oportunidades.

Levando isto em consideração, o desenvolvimento do projeto foi direcionado para a aplicação de tecnologia no monitoramento de estufas agrícolas com o intuito de democratizar a tecnologia na produção de orgânicos, visto que na agricultura de larga escala já existe um uso extensivo de insumos tecnológicos.

Por democratizar a tecnologia entende-se torná-la disponível para o maior número de usuários possível. Para isso, é imprescindível que o custo desta tecnologia seja acessível para o pequeno produtor, e assim, para os demais produtores de maior escala.

Com a intenção de proporcionar um projeto de qualidade e baixo custo para o usuário final, as tecnologias empregadas foram escolhidas cuidadosamente, de forma a encontrar um ponto ótimo entre custo, desempenho e facilidade de uso.

3.2 Definição das tecnologias empregadas e justificativas

3.2.1 LoRaWAN

Para a escolha do protocolo de comunicação foram levados em conta diversos requisitos. Por tratar-se de uma aplicação com baixa taxa de dados, optou-se por utilizar uma LPWAN frente à tecnologias celulares. O baixo consumo de energia desta tecnologia ainda reforça sua escolha, de forma a reduzir custos com manutenção e troca de bateria dos dispositivos. Além disso, por ter um alcance de quilômetros, é possível cobrir uma área com múltiplas estufas utilizando um único *gateway*.

A escolha do protocolo LoRaWAN se deu por dois principais motivos. O primeiro deles foi a opção de utilizar a rede de comunicação The Things Network (TTN). A TTN oferece o serviço de servidor de rede gratuitamente, o que contribui para a redução de custos do projeto. O segundo foi a disponibilidade do *gateway* e dos dispositivos responsáveis pela medição e transmissão dos dados, cedidos pelo professor orientador deste trabalho.

3.2.2 *Stack* de desenvolvimento

Para os *frameworks* que compõem a *stack* de desenvolvimento do projeto, os principais fatores que guiaram a escolha foram o uso do *framework* e de sua linguagem de programação na comunidade de desenvolvimento e se eram *frameworks* de código aberto ou não.

Quanto ao uso pela comunidade, este ponto é importante no que diz respeito à solução de problemas e dúvidas que venham a ocorrer durante o desenvolvimento da aplicação. Quanto maior a comunidade e mais estabelecido o *framework*, maiores são as chances de encontrar ajuda em fóruns na internet e também tutoriais bem consolidados e detalhados.

3.2.2.1 Django

A escolha de usar Python e Django para o *backend* pautou-se, além da ampla utilização e possuir código aberto, pela facilidade e agilidade no desenvolvimento de aplicações.

Django oferece abstrações para diversas interfaces que, caso necessitem ser desenvolvidas do zero, podem ser bastante trabalhosas. Um exemplo disso é a interface com banco de dados apresentada pelo *framework* como "Modelos". Além disso, o desenvolvimento com Python e sua sintaxe legível e elegante tornam mais fáceis o entendimento do código, bem como eventuais consertos dos erros que se apresentam.

3.2.2.2 React Native

Da mesma forma que o Django, o React Native também é um *framework* extensivamente utilizado pela comunidade de desenvolvedores e de código aberto.

A característica que fez com que este *framework* se destacasse foi o fato de proporcionar desenvolvimento multiplataformas. Assim, com um mesmo código, é possível portar a aplicação para dispositivos móveis com sistema operacional iOS, Android e até mesmo executá-lo em computadores, no navegador.

Esta característica possibilita que a aplicação seja acessível para um maior número de usuários, contribuindo para a democratização da tecnologia.

3.2.2.3 Google Cloud Platform

A plataforma de infraestrutura em nuvem escolhida foi a Google Cloud Platform. A GCP oferece um produto de hospedagem de contêineres bastante fácil de usar, chamado de Cloud Run. Este produto simplifica bastante o provisionamento de toda a infraestrutura necessária para a execução de aplicações *backend* utilizando frameworks como Django, Node.js e Flask.

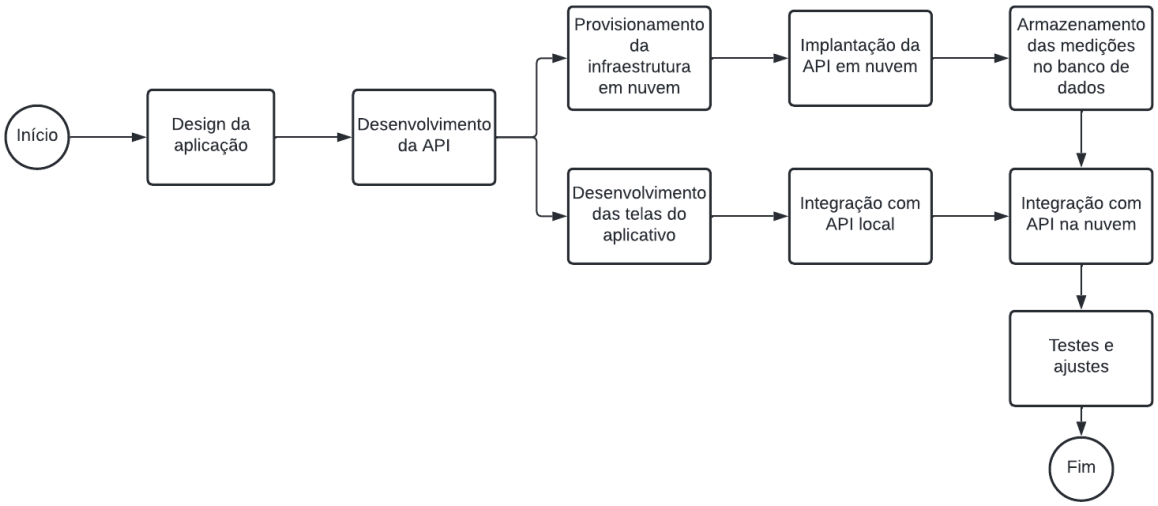
Além de facilitar a implementação da infraestrutura, a Google oferece um crédito de $300,00 (trezentos dólares americanos) para novos usuários testarem a plataforma por três meses, bem como uma categoria de serviços gratuita, destinada a aplicações simples, com baixo número de requisições e armazenamento.

Entretanto, ao aumentar o tráfego da aplicação, a GCP oferece uma escalabilidade automática dos serviços utilizados, provisionando mais máquinas executando a aplicação de forma a atender o aumento de carga.

3.3 Fluxograma do desenvolvimento

O desenvolvimento da aplicação foi dividido em diversos passos intermediários, com o objetivo de concentrar melhor os esforços em cada uma das etapas do projeto. As etapas são as apresentadas na Figura 7 e serão detalhadas ao longo desta seção.

Figura 7 – Fluxograma do desenvolvimento do aplicativo



Fonte: Elaborado pelo autor

A primeira etapa desenvolvida foi a concepção do *design* da aplicação, visando obter uma interface esteticamente agradável, que apresentasse as informações necessárias para o usuário e de fácil navegação.

Após desenvolvido o *design*, foi possível ter clareza de quais seriam os dados necessários para que a interface apresentasse o funcionamento esperado. Desta forma, deu-se início ao desenvolvimento da API. Nesta etapa, que será abordada com mais detalhes na Seção 3.7, foi feita toda a modelagem de dados, bem como desenvolvimento dos *endpoints* necessários para receber os dados do servidor da TTN e também expor os dados para posterior integração com o *frontend*.

Com a API desenvolvida em ambiente local, o desenvolvimento do projeto sofreu uma bifurcação. Para poder coletar e armazenar os dados medidos pelos sensores, era necessário implantar a API em uma infraestrutura em nuvem. Portanto, um dos caminhos seguidos nessa fase foi o de provisionar a infraestrutura na GCP, implantar a API em nuvem e iniciar o armazenamento dos dados no banco de dados da aplicação.

Em paralelo com isto, foi realizado o desenvolvimento das telas do aplicativo. Para desenvolver as telas foi possível utilizar dados fictícios, de forma a não demandar uma integração com a API nesta etapa.

Após desenvolver e integrar as telas do aplicativo com a API local e ter dados populados no banco de dados em nuvem, deu-se sequência com a integração do aplicativo com a API em nuvem, utilizando dados reais das medições. Como os dados reais eram diferentes dos dados de teste utilizados para o desenvolvimento das telas, pequenos ajustes foram realizados na formatação dos dados, de forma a ter um comportamento consistente por todo o aplicativo.

3.4 Equipamentos e infraestrutura necessária

3.4.1 Infraestrutura LoRaWAN

O *gateway* utilizado foi o ITG 200 Indoor [ITG 200 indoor], desenvolvido pela empresa Khomp. A utilização deste equipamento se deu por conta de sua disponibilidade junto ao professor orientador do trabalho. No entanto, para uma implementação prática, qualquer *gateway* LoRaWAN que seja capaz de conectar-se à TTN poderia ser usado, como por exemplo [LPS8].

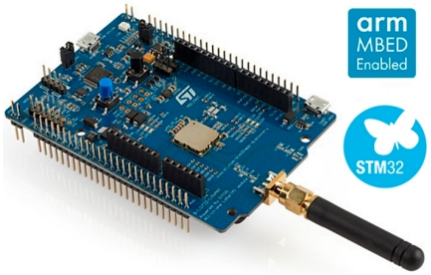
Apesar deste dispositivo apresentar um custo elevado quando comparado aos demais componentes do projeto, encontram-se no mercado alternativas mais acessíveis.

Os dispositivos de medição e transmissão dos dados consistem em uma junção do radiotransmissor B-L072Z-LRWAN1 [B-L072Z-LRWAN1], apresentado na Figura 9, com a placa de sensores X-NUCLEO-IKS01A2 [X-NUCLEO-IKS01A2], apresentada na Figura 10. Estes dispositivos apresentam um custo razoável para a aplicação do projeto. Entretanto, são dispositivos de uso exclusivo para desenvolvimento, disponibilizados pelo professor orientador deste trabalho. No mercado, encontram-se outras opções de transmissores e sensores próprios para uso comercial, até mesmo com menores custos, como por exemplo [SX1276]. Além disso, dependendo das medições a serem realizadas, podem ser utilizados sensores mais simples, que meçam apenas as grandezas relevantes para a aplicação.

Figura 8 – Gateway LoRaWAN ITG 200 Indoor

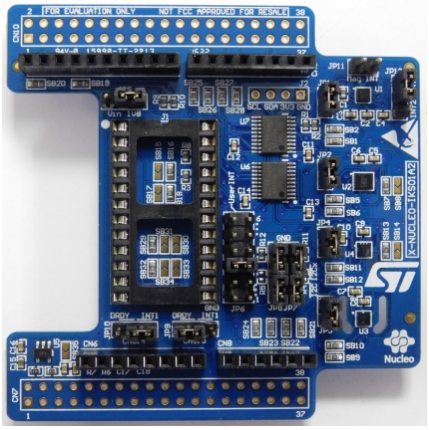
Fonte: [ITG 200 indoor]

Figura 9 – Radiotransmissor LoRaWAN



Fonte: [B-L072Z-LRWAN1]

Figura 10 – Placa de sensores adicional



Fonte: [X-NUCLEO-IKS01A2]

3.4.2 Infraestrutura para Desenvolvimento do Aplicativo

Para o desenvolvimento do aplicativo, foram utilizadas dois ambientes de desenvolvimento integrados diferentes. Para a linguagem Python e desenvolvimento do *backend* foi utilizado o *software* PyCharm [PyCharm], que oferece suporte para desenvolvimento de aplicações com Django. Para o desenvolvimento do *frontend* em JavaScript foi utilizado o *software* Visual Studio Code [Microsoft 2021], desenvolvido pela Microsoft. Para desenvolver o *design* da aplicação foi utilizado a plataforma Figma [Figma].

O desenvolvimento foi realizado em um computador com sistema operacional MacOS. Isto possibilitou simular o aplicativo em dispositivos móveis com sistema operacional iOS, mais especificamente iPhones. Para testar o aplicativo em um dispositivo real, foi utilizado um iPhone 7.

3.5 Design da Aplicação

Para o desenvolvimento do *design* da aplicação é necessário ter clareza de quais informações precisam ser mostradas e onde deve estar o foco de cada página. É importante tomar cuidado para não adicionar informações em excesso, tornando a leitura da página mais lenta e difícil.

Uma estrutura interessante de ser utilizada é ter uma página principal, chamada de *dashboard*, a qual contém informações gerais sobre uma determinada listagem de elementos. Se for necessário ter mais detalhes sobre algum elemento desta listagem, o usuário pode acessar uma página de detalhes deste elemento, aprofundando-se mais nos dados do mesmo.

Com estes conceitos e cuidados em mente, realiza-se o desenvolvimento das interfaces em si, posicionando os elementos de cada uma das páginas, ajustando cores, tamanhos e formas.

3.6 Aquisição dos dados através do kit

3.6.1 Gravação do *Firmware*

A configuração do kit de rádio com a placa de sensores à TTN pode ser realizada utilizando algum *firmmware* já existente. Um exemplo é o do projeto I-CUBE-LRWAN [I-CUBE-LRWAN], disponibilizado pela STMicroelectronics. Nesta seção, abordaremos os detalhes de como realizar a configuração do *firmware* que foi personalizado para as disciplinas de Tópico Avançado em Telecomunicações IV, ministradas pelo professor orientador do presente trabalho, Dr. Richard Demo Souza.

Inicia-se a configuração preparando a *toolchain*, que é o compilador do *framework*. Para isso, é recomendado a utilização de um computador com sistema operacional Ubuntu ou um subsistema Windows para Linux (WSL2). No terminal, deve-se o instalar os seguintes pacotes:

- Cutecom - `sudo apt-get install cutecom`
- GCC-ARM - `sudo apt-get install gcc-arm-node-eabi`
- Make - `sudo apt-get install make`
- Git - `sudo apt-get install git`
- *Firmware* a ser utilizado - `git clone https://github.com/esantosjr/EEL7x15`

Em seguida, deve-se navegar via terminal para o diretório EEL7x15 e abrir o arquivo *Commissioning.h*. Neste arquivo, deve-se certificar que a variável `OVER_THE_AIR_ACTIVATION`

possui valor 1. As demais variáveis são utilizadas para configurar as chaves de acesso ao dispositivo, que não se fazem necessárias para o uso de *Over The Air Activation* (OTAA). Caso seja utilizada uma ativação personalizada, é necessário que as demais variáveis possuam os mesmos valores presentes no servidor de rede.

No arquivo *Makefile*, é necessário definir a região na qual o dispositivo irá operar. Deve-se então incrementar a variável *DEFS* com a região AU915 conforme apresentado a seguir:

```
DEFS += -DREGION_AU915
```

No arquivo *main.c* deve-se configurar as seguintes variáveis:

- `APP_TX_DUTYCYCLE` - Intervalo, em minutos, entre as transmissões
- `LORAWAN_ADR_STATE` - Configuração do uso do ADR, habilitado por padrão
- `LORAWAN_DEFAULT_DATA_RATE` - Taxa de transmissão, utilizado se o ADR estiver desativado
- `LORAWAN_DEFAULT_CLASS` - Classe do dispositivo
- `LORAWAN_DEFAULT_CONFIRM_MSG_STATE` - Realização de *uplinks* sem confirmação
- `LORAWAN_FSB` - Sub-banda de operação. Deve-se utilizar 2.

Além destas configurações, neste arquivo o *payload* com os dados também é montado e enviado.

Para compilar o projeto, deve-se navegar via terminal para a pasta na qual o arquivo *Makefile* está localizado e executar o comando `make`. Após o término da compilação, o arquivo *end_ node.bin* será criado nesta mesma pasta. Para gravar o *firmware* na placa, deve-se copiar este arquivo para o disco removível correspondente ao kit, chamado *DIS_ L072Z*.

Para realizar a verificação local da configuração, deve-se utilizar o Cutecom ou outro *software* terminal de serial para observar o comportamento do rádio. Através desta verificação, é possível ver as mensagens iniciais enviadas pelo kit ao realizar o processo de conexão com a rede, nas quais encontram-se os parâmetros necessários para cadastro do dispositivo na TTN.

3.6.2 Configuração na TTN

Para conectar o dispositivo à TTN, é necessário primeiramente criar uma aplicação na rede. Para isso, é necessário criar uma conta na mesma região configurada na etapa de gravação do *firmware*.

Após criada a conta e feito o *login*, deve-se clicar em *Go to applications* e criar uma nova aplicação em *Add application*. Após preencher os campos necessários, clicar em *Create application*.

Com a aplicação criada, é necessário adicionar um novo dispositivo a ela. Para isso, deve-se clicar em *Add end device*. Na aba *Manually*, deve-se selecionar o mesmo modo de ativação configurado na etapa de gravação do *firmware*, selecionar a versão LoRaWAN e clicar em *Start*.

O próximo passo consiste em configurar os detalhes do dispositivo com os dados de *dev_ eui* e *app_ eui* obtidos através da verificação local do kit. Após preencher estes e os demais campos obrigatórios, a configuração segue para os ajustes da camada de rede.

Na parte da camada de rede, deve-se selecionar o *frequency plan* a ser utilizado, bem como a configuração de classes LoRaWAN a serem suportadas.

Por fim deve-se inserir a chave da aplicação obtida também na etapa de verificação local, denominada *AppKey*. Feitas estas configurações, o dispositivo está pronto para ser conectado à rede. Na aba *Live data* do *dashboard* é possível observar os dados que estão sendo transmitidos e recebidos pelo dispositivo.

3.7 Desenvolvimento da API

3.7.1 *Webhook*

O desenvolvimento da API iniciou-se com o desenvolvimento do *webhook* para receber os dado do servidor da TTN. Um *webhook* nada mais é do que uma URL que, neste caso, realiza a recepção de dados.

Inicialmente o *webhook* recebe os dados da requisição, transforma-os em um formato mais facilmente manipulável em Python e verifica se os dados recebidos são simulados. Caso sejam, a execução do *webhook* é interrompida, retornando uma resposta vazia para quem fez a requisição. Este recurso foi utilizado ao realizar a implantação da API na infraestrutura em nuvem, para verificar se o *webhook* estava funcionando conforme esperado.

Conhecendo a estrutura dos dados, foi possível desenvolver os modelos do Django, que são uma abstração da estrutura das tabelas do banco de dados oferecida pelo *framework*. O desenvolvimento dos modelos será abordado com mais detalhes na subseção 3.7.2.

Quando os dados recebidos são provenientes de um kit de sensores, o *webhook* continua sua execução, buscando o sensor ao qual os dados recebidos pertencem. Essa busca é feita utilizando o *dev_ eui*, que é um identificador único de cada dispositivo. Caso não haja nenhum dispositivo registrado no banco de dados com o *dev_ eui*, é criada uma nova instância de sensor.

Em seguida, são extraídos os dados decodificados do pacote de informações recebido e é criado uma nova instância de *log* de dados, contendo os valores das medidas, dados de data e hora e relacionando-os com o sensor que fez as leituras.

Caso aconteça algo de errado na execução do *webhook*, as exceções são tratadas, sempre retornando um registro no *console* da aplicação contendo mais detalhes sobre o erro para posterior investigação e correção.

3.7.2 Modelos de Dados

Para armazenar os dados no banco de dados, é necessário que sejam criados modelos no Django. Estes modelos contém informações a respeito dos dados que serão armazenados em suas instâncias, bem como relações entre diferentes modelos. Estas informações sobre os dados podem ser o tipo de dado (número, texto, booleano, etc), se o dado pode conter valores nulos, se o dado possui um valor predefinido de inicialização, entre outras configurações possíveis.

O primeiro modelo desenvolvido foi o modelo do sensor. Este modelo conta com três campos, sendo eles o nome do sensor, o *dev_ eui* e o *app_ eui*.

O segundo modelo foi o modelo do *log* de dados. Este modelo contém um campo para cada valor medido pelo sensor, conforme apresentado na Tabela 1, bem como um campo para registrar o momento em que a medida foi recebida e um campo responsável por relacionar o *log* com o sensor.

Tabela 1 – Campos do modelo de *log* relacionados às medições

Nome do Campo	**Grandeza Medida**	**Tipo de Campo**
pressure	Pressão atmosférica	Número com ponto flutuante
temperature	Temperatura ambiente	Número com ponto flutuante
humidity	Umidade	Número com ponto flutuante
accel_x_axis	Aceleração no eixo X	Número com ponto flutuante
accel_y_axis	Aceleração no eixo Y	Número com ponto flutuante
accel_z_axis	Aceleração no eixo Z	Número com ponto flutuante
gyro_x_axis	Giroscópio no eixo X	Número com ponto flutuante
gyro_y_axis	Giroscópio no eixo Y	Número com ponto flutuante
gyro_z_axis	Giroscópio no eixo Z	Número com ponto flutuante
magnometer	Campo magnético	Número com ponto flutuante
bat_voltage	Tensão da bateria	Número com ponto flutuante

Fonte: Elaborado pelo autor

3.7.3 *Endpoints* para consumo dos dados

Após receber, estruturar e armazenar os dados no banco de dados, é necessária uma forma para expô-los ao *frontend*. Para isso, deve-se construir diferentes *endpoints*,

estruturas com um URL que, ao receberem requisições, neste caso HTTP, realizam operações e retornam alguma resposta ao requisitante.

Como o objetivo destes *endpoints* é somente o de expor informações, os mesmos não possuem lógica para receber dados, funcionando somente como leitura.

Analisando as informações que necessitam ser expostas para o *frontend* da aplicação, chegou-se a uma estrutura com 3 *endpoints* com ênfases diferentes.

Um primeiro *endpoint* deve ser responsável por retornar os dados necessários para montar os gráficos presentes na tela inicial do aplicativo. Este *endpoint* precisa retornar os dados de apenas um sensor, logo, deve apresentar uma filtragem por sensor. Além disso, é interessante que estes dados sejam retornados em um formato facilmente manipulável pelo *frontend*, para que não haja reprocessamentos desnecessários sendo executados no dispositivo do usuário final.

O segundo *endpoint* é necessário para trazer dados mais específicos sobre as últimas medidas e em qual data e hora foram realizadas. Com o intuito de simplificar e deixar mais leve a aplicação no lado do cliente, uma abordagem a ser explorada é a de formatar os dados no *backend* e expô-los prontos para uso no *endpoint*.

Por fim, um terceiro *endpoint* é necessário para retornar os dados de métricas de cada uma das variáveis monitoradas. É interessante que as métricas sejam retornadas em um *endpoint* separado pois as mesmas são computadas em tempo de execução. Além disso, como estes dados serão usados somente ao acessar os detalhes de uma grandeza monitorada, não se faz necessário ter estes dados em mãos ao montar a página de *dashboard* para o usuário, por exemplo.

3.8 Desenvolvimento do App

O desenvolvimento do aplicativo inicia-se com a "tradução" do *design* da aplicação em código. Este primeiro momento consiste na estruturação as páginas, criação de divisões lógicas do conteúdo, como seção de título, seção de conteúdo, navegação, etc.

Após ter uma estrutura rudimentar da tela montada com dados meramente ilustrativos, inicia-se a estilização dos elementos que compõem a tela. Nesta etapa deve-se adicionar as cores desejadas, arredondar bordas, definir estilo e tamanho de tipografia, entre outros detalhes.

Os passos anteriores do desenvolvimento do aplicativo são realizados fazendo uso de dados fictícios ou simplesmente inserindo valores para ter os campos populados. Com a interface devidamente estilizada, parte-se para a etapa de integração do *frontend* com a API, a fim de testá-la com dados reais. Se faz necessário testá-la pois alguns dados podem vir de maneira ligeiramente diferentes do que se esperava, fazendo-se necessário

a adequação da interface. A integração pode ser realizada usando diversas estruturas e metodologias diferentes, mas para casos com baixa complexidade, pode-se utilizar a Context API [Context API] do React, que possibilita a persistência de dados através de diversos componentes distintos.

4 Resultados

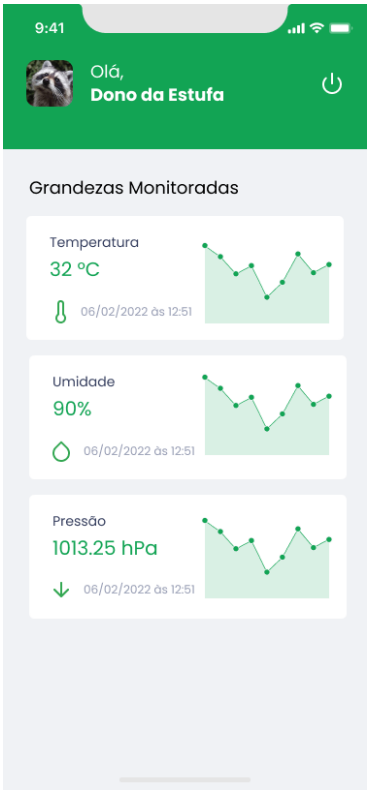
4.1 Design

Conforme descrito na Seção 3.5, a estrutura escolhida para o *design* da aplicação apresenta uma *dashboard* com informações gerais e telas de detalhes para cada uma das grandezas medidas.

O *design* da *dashboard* pode ser visto na Figura 11. A análise desta tela pode ser dividida em duas partes. A parte superior contém informações sobre o proprietário da estufa que está sendo monitorada, exibindo uma foto de perfil e uma saudação personalizada.

Logo abaixo, é apresentada uma lista com as grandezas monitoradas pelo kit. Caso seja necessário monitorar outras grandezas, a lista pode ser expandida sem que se perca sua funcionalidade.

Figura 11 – *Dashboard* da aplicação



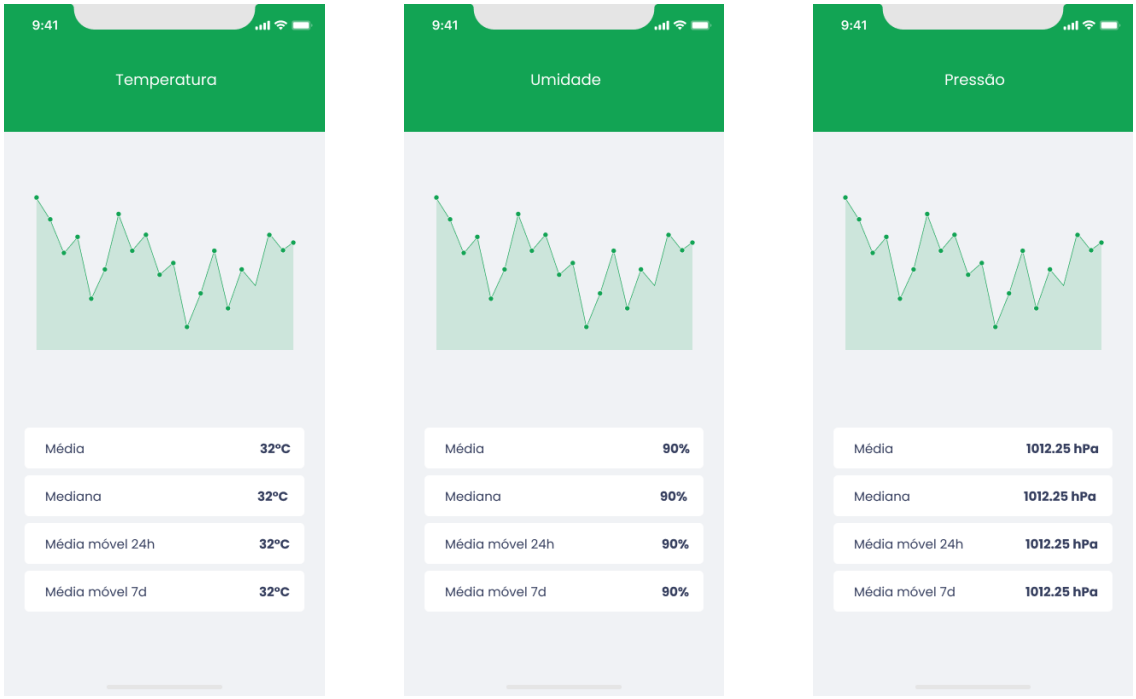
Fonte: Elaborado pelo autor

O cartão de cada grandeza apresenta o valor da última medição, bem como a data e o horário em que a medição foi recebida, juntamente com um ícone personalizável. Além disso, apresenta um gráfico com dados das últimas medições. Cada um destes cartões

é clicável e, ao clicar em um dos cartões, o usuário deve ser direcionado para a tela de detalhes daquela grandeza.

As telas de detalhes das grandezas apresentam uma estrutura bastante parecida, alterando-se apenas o título da página, conforme apresentado na Figura 12.

Figura 12 – Detalhes das grandezas monitoradas



Fonte: Elaborado pelo autor

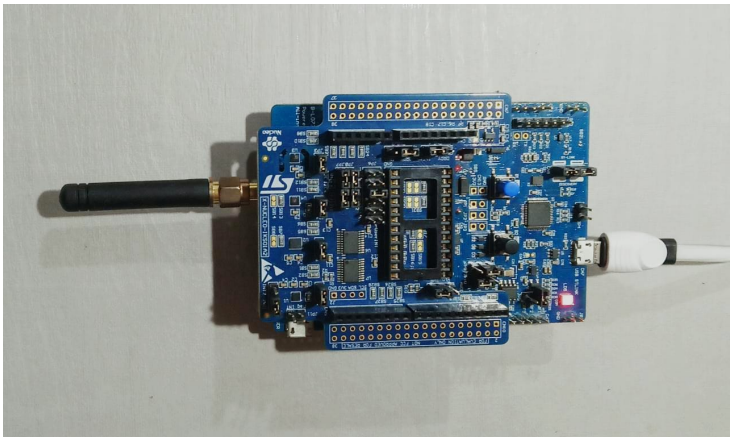
Nas telas de detalhes, o usuário tem acesso às métricas calculadas a partir dos dados coletados pelos sensores, bem como uma visualização ampliada do gráfico das últimas medições.

4.2 Aquisição dos dados através do kit

Após realizados os passos descritos na Seção 3.6.1, o dispositivo com os sensores foi conectado com sucesso à TTN. Na Figura 13 pode-se observar o dispositivo conectado ao computador para realizar a gravação do *firmware*.

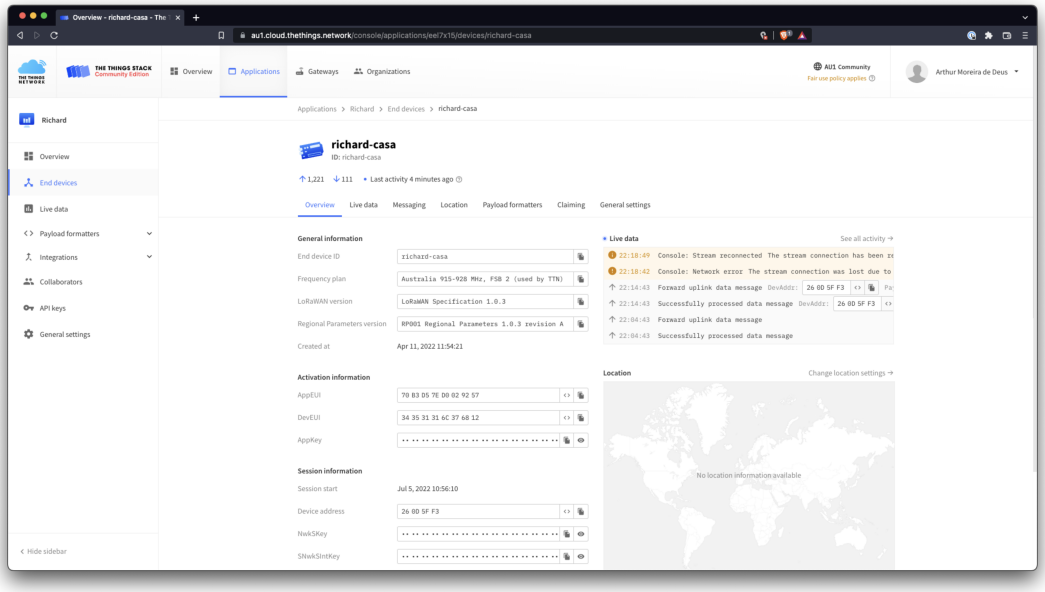
Após concluída a gravação do *firmware*, foi criada uma aplicação na TTN e inseridos os detalhes do dispositivo a ser conectado na mesma. Uma vez que o dispositivo realiza o processo de *join* na rede com sucesso, o mesmo passa a aprecer na tela de *dashboard* da TTN, conforme apresentado na Figura 14.

Figura 13 – Dispositivo LoRaWAN



Fonte: Elaborado pelo autor

Figura 14 – *Dashboard* da TTN



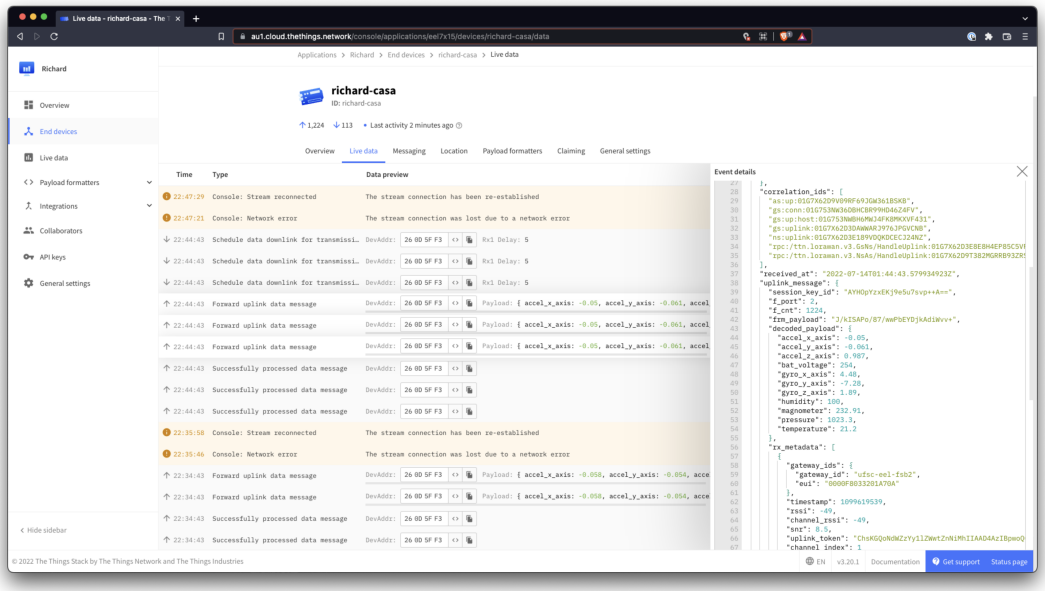
Fonte: Elaborado pelo autor

Além das informações que dizem respeito ao dispositivo, no *dashboard* da TTN também é possível monitorar os dados que estão sendo enviados, bem como se há problemas na conexão com o dispositivo. Estes dados estão disponíveis na aba *Live data* e são apresentados na Figura 15.

4.3 Desenvolvimento da API

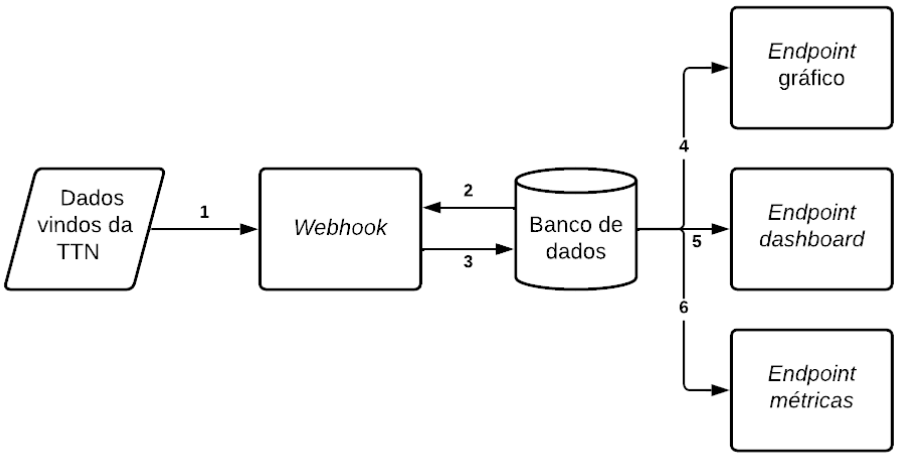
A API foi desenvolvida com a finalidade de receber os dados da TTN, armazená-los e retorná-los para o *frontend*. Na Figura 16 podemos observar um diagrama de blocos simplificado representando o fluxo dos dados na API.

Figura 15 – Dados do dispositivo na TTN



Fonte: Elaborado pelo autor

Figura 16 – Diagrama de blocos da API



Fonte: Elaborado pelo autor

Inicialmente, os dados são recebidos pelo *webhook*, representado pelo número 1. Em seguida, o *webhook* verifica no banco de dados se há alguma instância de sensor com os dados de *dev_ eui* recebidos, representado pelo número 2. Em seguida, o *webhook* escreve no banco de dados as informações recebidas através do *webhook*, representado pela etapa de número 3.

Por fim, os *endpoints* consumidos pelo *frontend* são apenas de leitura e estão representados no diagrama pelos números 4, 5 e 6. Os *endpoints* de leitura de dados retornam dados para a elaboração dos gráficos, da *dashboard* e das métricas de cada

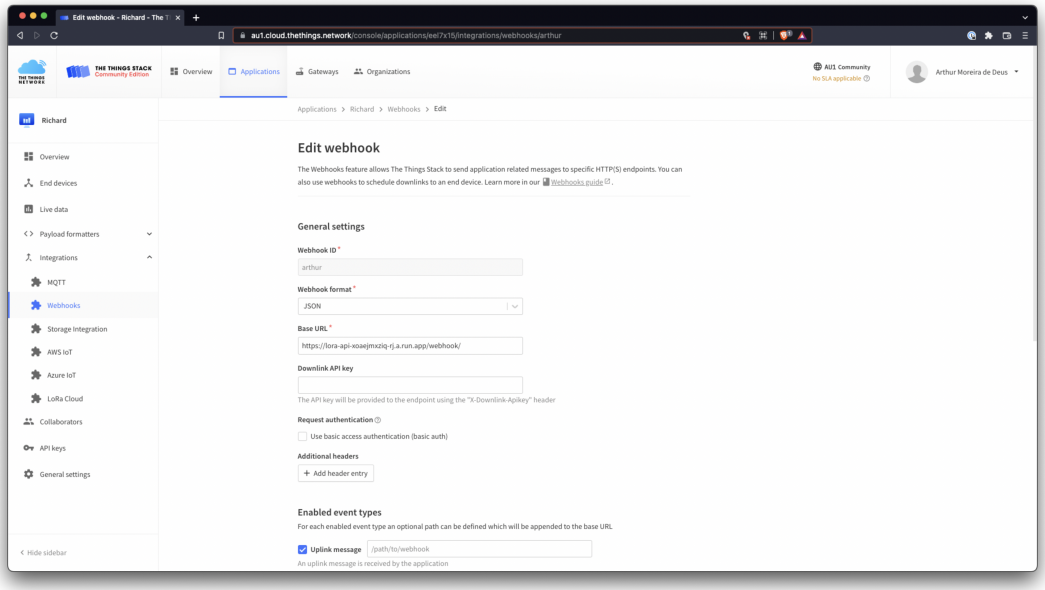
grandeza, respectivamente.

4.3.1 *Webhook*

O ponto de entrada dos dados na aplicação é o *webhook*, que recebe requisições periódicas da TTN com informações das medições. A rota escolhida para o *webhook* foi simplesmente `/webhook/` e aceita requisições HTTP do tipo POST.

Para que os dados do dispositivo fossem enviados para o *webhook* da aplicação, foi necessário registrá-lo na TTN, conforme apresentado na Figura 17. Para isso, foi configurado um novo *webhook* no formato JSON apontando para a URL do *webhook*, e o evento a ser transmitido selecionado foi o de mensagens de *uplink*.

Figura 17 – Configuração do *webhook* na TTN



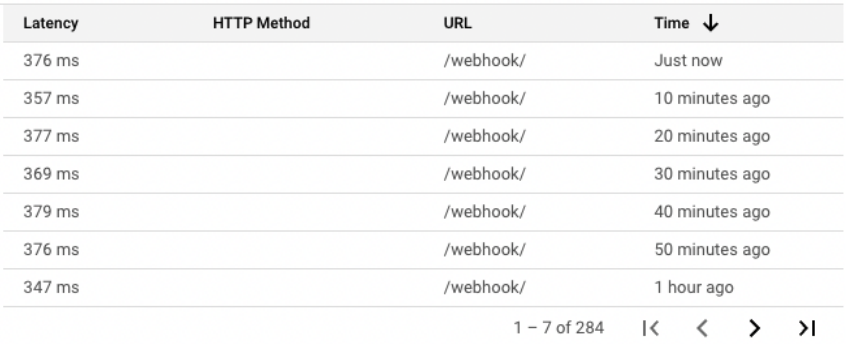
Fonte: Elaborado pelo autor

Na Figura 18, extraída do serviço Trace da GCP, é possível observar que o *webhook* recebe requisições em intervalos de dez minutos. Além disso, a latência de execução do *webhook* oscila em torno de 350 milissegundos, o que é um tempo de resposta bastante razoável, visto que não é uma aplicação em tempo real.

Um exemplo de *payload* recebido pelo *webhook* é apresentado no Código Fonte 1. Neste trecho de código é possível observar logo no início do *payload* os dados de *dev_eui* utilizados para identificação do dispositivo na API, bem como dados referentes ao dispositivo na TTN.

Na chave *decoded_payload* é possível observar os dados lidos pelo sensor já decodificados e prontos para uso. Estes são os dados efetivamente armazenados no banco de dados da aplicação.

Figura 18 – Requisições recebidas pelo *webhook*



Latency	HTTP Method	URL	Time ↓
376 ms		/webhook/	Just now
357 ms		/webhook/	10 minutes ago
377 ms		/webhook/	20 minutes ago
369 ms		/webhook/	30 minutes ago
379 ms		/webhook/	40 minutes ago
376 ms		/webhook/	50 minutes ago
347 ms		/webhook/	1 hour ago

1 – 7 of 284

Fonte: Elaborado pelo autor

Código Fonte 1 – *Payload* recebido pelo *webhook*


```
{
    "service": "sensor",
    "msg": "WEBHOOK_RECEIVED",
    "device_id": null,
    "received_at": "2022-07-12T23:14:54.149538286Z",
    "payload": {
        "end_device_ids": {
            "device_id": "richard-casa",
            "application_ids": {
                "application_id": "eel7x15"
            },
            "dev_eui": "343531316C376812",
            "join_eui": "70B3D57ED0029257",
            "dev_addr": "260D5FF3"
        },
        "correlation_ids": [
            "as:up:01G7TB3BJ5ZZ9JCQP3B12SF9TJ",
            "gs:conn:01G753NW36DBHCBR99HD46Z4FV",
            "gs:up:host:01G753NWBH6MWJ4FK8MKXVF431",
            "gs:uplink:01G7TB3BBQSFG6W6QYHV0GWR3C",
            "ns:uplink:01G7TB3BBR0G074N9KVG2RAQME",
            "rpc:/ttn.lorawan.v3.GsNs/HandleUplink:01G7TB3BBRTT5XG0BJETHP1080",
            "rpc:/ttn.lorawan.v3.NsAs/HandleUplink:01G7TB3BJ4W2C7ZMS9QFNVMRC0"
        ],
        "received_at": "2022-07-12T23:14:54.149538286Z",
        "uplink_message": {
            "session_key_id": "AYHOpYzxEKj9e5u7svp++A==",
            "f_port": 2,
            "f_cnt": 1065,
            "frm_payload": "J8gJQgOW/8r/xgPbDAjsUAdiWpP+",
            "decoded_payload": {
                "accel_x_axis": -0.054,
                "accel_y_axis": -0.058,
```

```
                "accel_z_axis": 0.987,
                "bat_voltage": 254,
                "gyro_x_axis": 3.08,
                "gyro_y_axis": -5.04,
                "gyro_z_axis": 1.89,
                "humidity": 91.8,
                "magnometer": 231.87,
                "pressure": 1018.4,
                "temperature": 23.7
            },
            "rx_metadata": [
                {
                    "gateway_ids": {
                        "gateway_id": "ufsc-eel-fsb2",
                        "eui": "0000F8033201A70A"
                    },
                    "timestamp": 199572172,
                    "rssi": -57,
                    "channel_rssi": -57,
                    "snr": 11.2,
                    "uplink_token": "ChsKGQoNdWZzYy1lZWwtZnNiMhIIAAD4AzIBpwoQzPWUXxoMCO2AuJ
                    YGEIas3MEDIODZtrvnk6IB",
                    "channel_index": 5
                }
            ],
            "settings": {
                "data_rate": {
                    "lora": {
                        "bandwidth": 125000,
                        "spreading_factor": 7
                    }
                },
                "coding_rate": "4/5",
                "frequency": "917800000",
                "timestamp": 199572172
            },
            "received_at": "2022-07-12T23:14:53.944349391Z",
            "consumed_airtime": "0.077056s",
            "network_ids": {
                "net_id": "000013",
                "tenant_id": "ttn",
                "cluster_id": "au1",
                "cluster_address": "au1.cloud.thethings.network"
            }
        }
    }
}
```

Fonte: Elaborado pelo autor

4.3.2 Modelos de Dados

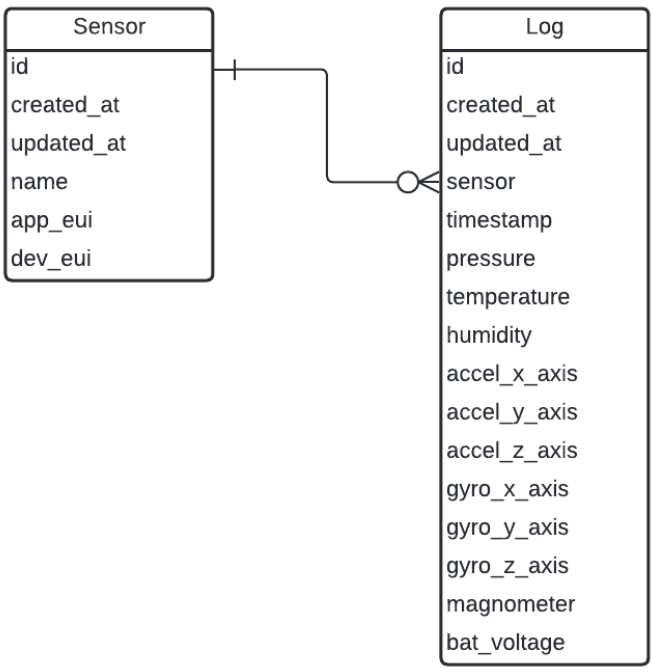
A modelagem dos dados da aplicação foi feita de modo a possibilitar a relação entre os dados de leitura recebidos e o dispositivo que a realizou.

Para isso, foi criado um modelo denominado *Sensor*, que guarda somente informações que dizem respeito ao kit que realizou as medições, tais como *dev_ eui*, *app_ eui* e nome do dispositivo.

Os dados da leituras foram salvos com a estrutura do modelo denominado *Log*, que salva os dados referentes às medições e também tem uma relação para com o *Sensor* que a realizou.

A relação presente entre um *Sensor* e um *Log* é de um para muitos opcional. Isto significa que um *Sensor* pode ter vários *Logs* relacionados a ele, mas um *Log* só pode ser relacionado a um único sensor. O termo opcional significa que o *Sensor* pode existir sem que hajam *Logs* relacionados a ele.

Figura 19 – Diagrama de Relacionamento de Entidades da aplicação



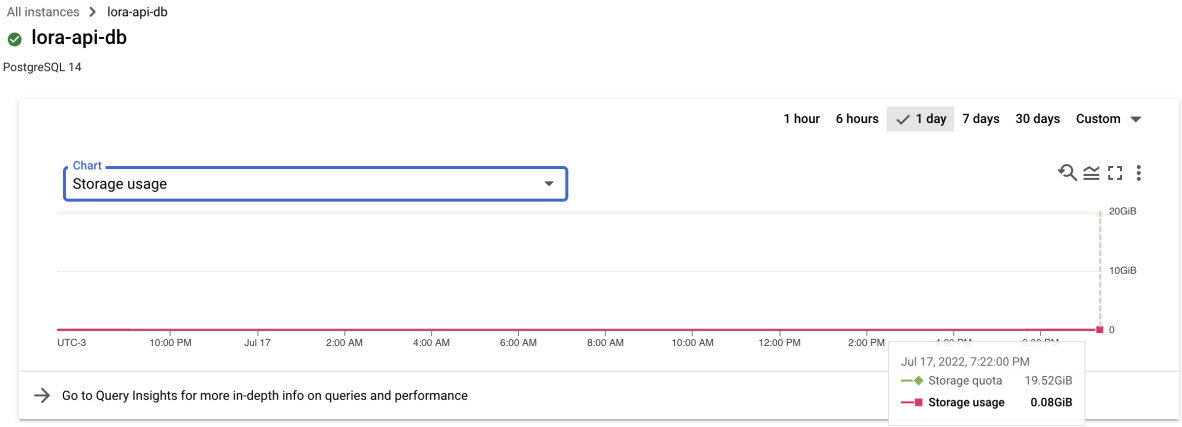
Fonte: Elaborado pelo autor

No Diagrama de Relacionamento de Entidades apresentado na Figura 19 é possível visualizar todos os campos presentes em cada um destes modelos. Os modelos podem ser entendidos como uma representação de alto nível das tabelas do banco de dados, sendo

assim, os campos presentes nos modelos são, na verdade, colunas homônimas presentes no banco de dados da aplicação.

Desde o primeiro *Log* criado no dia 11 de Maio de 2022, foram criados um total de 9193 entradas. Estes dados, somados aos dados da tabela de *Sensores*, com uma entrada para o *Sensor* real e outra para o *Sensor* utilizado no desenvolvimento da API, totalizaram 0,08 GiB, ou 80 MiB de armazenamento, conforme apresentado na Figura 20.

Figura 20 – Gráfico de utilização do armazenamento do banco de dados



Fonte: Elaborado pelo autor

4.3.3 *Endpoints* para consumo dos dados

Para expor os dados armazenados no banco para que o aplicativo pudesse utilizá-los foram construídos três *endpoints*.

O primeiro deles, denominado *LogViewSet*, é responsável por retornar os dados dos últimos 30 *Logs* para utilização nos gráficos da *dashboard* e das telas de detalhe. Um trecho da resposta deste *endpoint* é apresentado no Código Fonte 2. O exemplo apresentado foi truncado com dados das últimas duas leituras pois os demais 28 seguem a mesma estrutura, fazendo-se desnecessária sua exposição.

Código Fonte 2 – Resposta do *LogViewSet*


```
[
    {
        "id": 9194,
        "timestamp": "2022-07-17T22:34:07.158899Z",
        "pressure": 1022.6,
        "temperature": 22.4,
        "humidity": 89.9
    },
    {
        "id": 9193,
        "timestamp": "2022-07-17T22:24:07.193838Z",
```

```
        "pressure": 1022.4,
        "temperature": 22.5,
        "humidity": 90.7
    },
]
```

Fonte: Elaborado pelo autor

O segundo *endpoint* é responsável por retornar os dados para composição da *dashboard*. Seu nome é *DashboardDataAPIView*. Nele, são retornados os dados já formatados da última leitura de cada uma das grandezas. Um exemplo de resposta deste *endpoint* é apresentado no Código Fonte 3

Código Fonte 3 – Resposta do *DashboardDataAPIView*

```
{
    "temperature": {
        "id": 1,
        "type": "temperature",
        "title": "Temperatura",
        "measure": "22.3°C",
        "date": "17/07/2022",
        "time": "19:54",
        "icon": "thermometer"
    },
    "pressure": {
        "id": 3,
        "type": "pressure",
        "title": "Pressão",
        "measure": "1022.9hPa",
        "date": "17/07/2022",
        "time": "19:54",
        "icon": "arrow-down"
    },
    "humidity": {
        "id": 2,
        "type": "humidity",
        "title": "Umidade",
        "measure": "100.0%",
        "date": "17/07/2022",
        "time": "19:54",
        "icon": "droplet"
    }
}
```

Fonte: Elaborado pelo autor

O último *endpoint*, denominado *MetricsAPIView*, é responsável por expor os dados das métricas de cada uma das grandezas monitoradas, que são exibida nas telas de detalhe. As métricas são calculadas em tempo de execução, ou seja, quando a requisição é realizada, as métricas são calculadas e retornadas na resposta. A resposta do *endpoint* é apresentada no Código Fonte 4.

Código Fonte 4 – Resposta do *MetricsAPIView*


```
{
    "temperature": [
        {
            "name": "Média",
            "value": 21.882876727233164
        },
        {
            "name": "Mediana",
            "value": 21.5
        },
        {
            "name": "Média Móvel 24H",
            "value": 21.63873031358885
        },
        {
            "name": "Média Móvel 7D",
            "value": 21.801338874536672
        }
    ],
    "pressure": [
        {
            "name": "Média",
            "value": 1016.1468719399412
        },
        {
            "name": "Mediana",
            "value": 1016.6
        },
        {
            "name": "Média Móvel 24H",
            "value": 1016.3243484320558
        },
        {
            "name": "Média Móvel 7D",
            "value": 1016.2038077052679
        }
    ],
    "humidity": [
        {
```

```
            "name": "Média",
            "value": 96.58784680665869
        },
        {
            "name": "Mediana",
            "value": 100.0
        },
        {
            "name": "Média Móvel 24H",
            "value": 98.54872473867596
        },
        {
            "name": "Média Móvel 7D",
            "value": 96.6529484443446
        }
    ]
}
```


Fonte: Elaborado pelo autor

4.4 Desenvolvimento do App

O desenvolvimento do aplicativo, foi utilizada a plataforma Expo, que permite gerar gerar *builds* do aplicativo para dispositivos reais e simuladores. Desta forma, foi possível testar o aplicativo nestes dois ambientes e notar as diferenças da simulação para o dispositivo real.

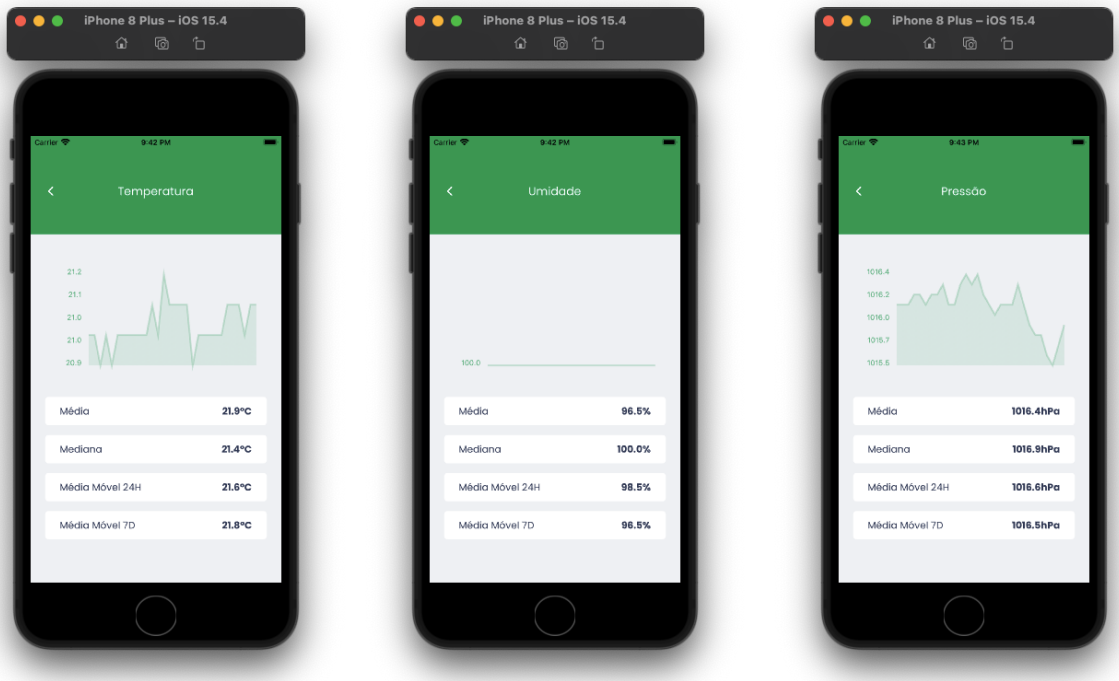
A Figura 21 corresponde à tela inicial do aplicativo sendo executada no simulador. O dispositivo utilizado para a simulação foi um iPhone 8 Plus, com sistema operacional iOS 15.4.

Figura 21 – *Dashboard* da aplicação no simulador

Fonte: Elaborado pelo autor

Na Figura 22 são apresentadas as telas de detalhes das grandezas, com o aplicativo sendo executado no dispositivo simulado.

Figura 22 – Detalhes das grandezas monitoradas no simulador



Fonte: Elaborado pelo autor

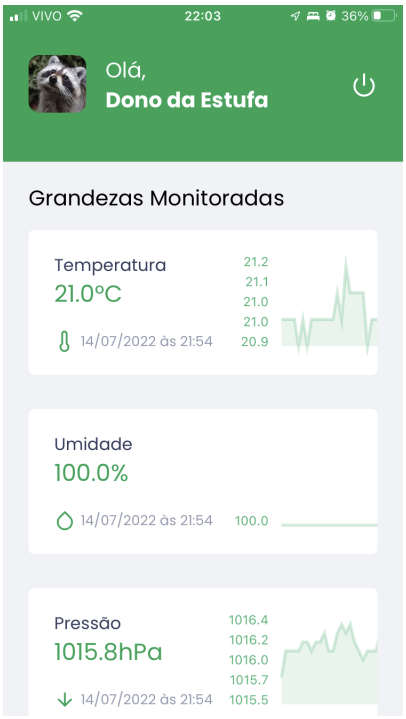
É possível observar uma ligeira diferença entre os gráficos oriundos do *design* e os que foram implementados no aplicativo. Isso acontece pois, para criar o *design* das telas, foi utilizada uma imagem vetorizada do gráfico que não necessariamente representa com fidelidade os gráficos possíveis de serem gerados através de bibliotecas do JavaScript. Por outro lado, os gráficos apresentados nas Figuras 21 e 22 foram gerados a partir dos dados armazenados através destas bibliotecas.

Na Figura 23 temos a *dashboard* do aplicativo sendo exibida em um dispositivo real. O dispositivo utilizado para os testes foi um iPhone 7, com sistema operacional iOS 15.5.

A Figura 24 mostra as telas de detalhes das grandezas sendo executadas pelo dispositivo real.

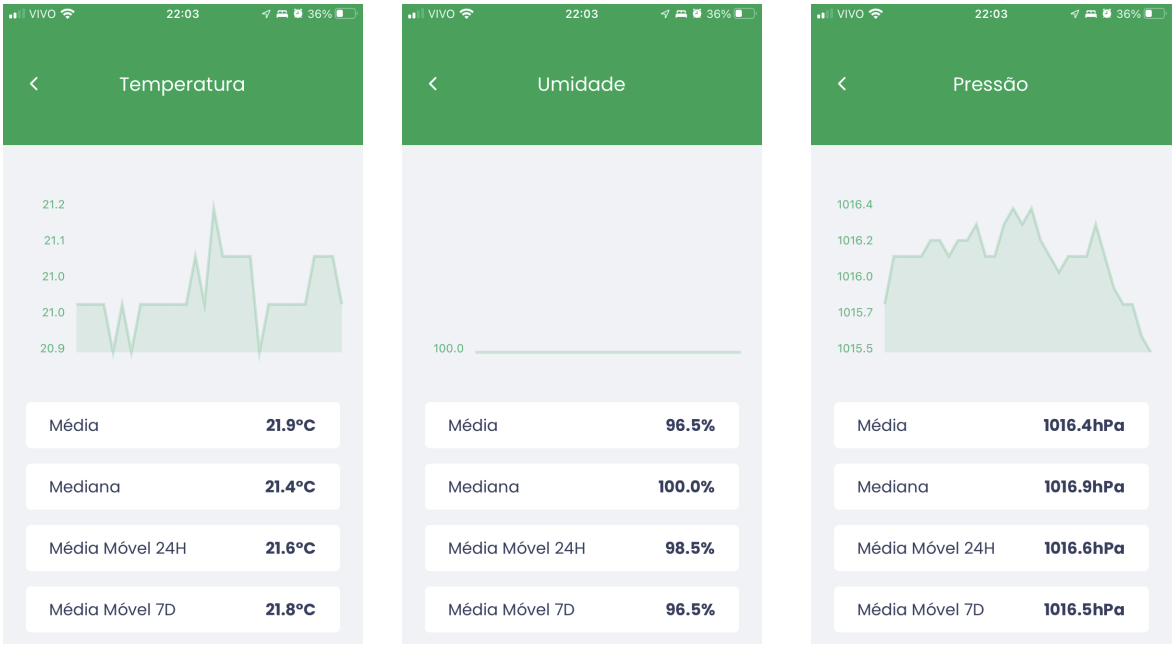
Observa-se que, na Figura 23, a parte mais à direita dos gráficos acaba ficando omitida da apresentação. Além disso, o cartão de pressão tem sua parte inferior também omitida. Estas pequenas diferenças ocorreram devido aos tamanhos de tela que foram utilizados. No dispositivo simulado, a tela é de 5,5 polegadas, enquanto que no dispositivo real é de 4,7 polegadas.

Figura 23 – *Dashboard* da aplicação no dispositivo real



Fonte: Elaborado pelo autor

Figura 24 – Detalhes das grandezas monitoradas no dispositivo real



Fonte: Elaborado pelo autor

5 Considerações finais

5.1 Alcance dos objetivos

5.1.1 Objetivo Geral

O objetivo geral do trabalho, que é o de desenvolver um aplicativo para dispositivos móveis para monitoramento de estufas com comunicação LoRaWAN, foi alcançado com sucesso.

A configuração dos sensores e do kit LoRaWAN foi realizada, os dados foram recebidos pela API e expostos para o aplicativo. O aplicativo oferece uma visualização destes dados de acordo com o *design* proposto para a aplicação.

Ainda que disponibilizado de forma experimental, o aplicativo foi executado com sucesso em dispositivos simulados e reais, de forma a validar seu desenvolvimento.

5.1.2 Objetivos Específicos

- Entender o processo de desenvolvimento de aplicativos para dispositivos móveis do começo ao fim.

Através da concepção e do desenvolvimento da aplicação, foi possível ter um melhor conhecimento do processo de desenvolvimento de aplicativos.

O processo inicia-se na fase de concepção e *design* da aplicação, passando pela escolha das tecnologias para cada *stack*, desenvolvimento da arquitetura do *backend*, da infraestrutura em nuvem e das estruturas de dados, desenvolvimento do aplicativo e integração com o *backend* e chega até a disponibilização e execução do aplicativo no dispositivo final.

No entanto, o ciclo de vida de um produto digital vai além disso, fazendo parte do mesmo os desafios de escala e manutenção da aplicação. Desafios estes que não foram contemplados no desenvolvimento deste projeto, mas que podem ser vistos como os passos seguintes aos que foram alcançados.

- Elencar as principais tecnologias e plataformas que permitem o desenvolvimento rápido de aplicações.

Este objetivo específico foi concluído em conjunto com o supracitado, uma vez que para ter um entendimento satisfatório do processo de desenvolvimento de aplicativos se

faz necessário conhecer as tecnologias e como utilizá-las.

A tecnologia utilizada para o desenvolvimento do *backend*, o *framework* Django, oferece um desenvolvimento bastante rápido, com abstrações de alto nível de diversas tarefas não tão simples, como acesso ao banco de dados.

Para o *frontend*, o React Native oferece uma grande vantagem ao permitir o desenvolvimento de aplicativos para dispositivos Android e iOS a partir de um mesmo código.

Ambas as tecnologias usadas no *frontend* e no *backend* dispõem de uma grande comunidade de desenvolvedores que faz uso delas, o que é um fator importantíssimo ao escolher as tecnologias a serem empregadas em um projeto.

Para a infraestrutura em nuvem, o serviço Cloud Run da GCP possibilita um *deploy* rápido da aplicação *backend* para a nuvem, ao simplificar e automatizar diversas configurações que, de outra forma, precisariam ser feitas manualmente.

- Compreender e aplicar o protocolo de comunicação LoRaWAN.

A aplicação do protocolo de comunicação LoRaWAN para a aquisição dos dados foi uma das etapas mais interessantes do desenvolvimento do trabalho.

Para que fosse possível aplicar a tecnologia, se fez necessário entender qual o *hardware* necessário e como configurá-lo para ter um funcionamento adequado.

Com o auxílio do professor orientador e fazendo uso de seus dispositivos, foi realizada a gravação do *firmware* no dispositivo, feita sua conexão à rede da TTN através do *gateway* e posterior envio dos dados à API via *webhook*, para que fossem utilizados pelo aplicativo.

5.2 Conclusão

As tecnologias de Internet das Coisas trouxeram novas possibilidades de usos e desenvolvimento de aplicações. Com um potencial enorme de crescimento, é importante ter em mente que a democratização da tecnologia pode ser tão importante quanto o desenvolvimento da mesma.

O desenvolvimento do aplicativo mostrou que, com recursos de código aberto, é possível desenvolver aplicações de qualidade com baixo, ou até mesmo nenhum custo financeiro.

O aplicativo e a arquitetura desenvolvidos cumprem com o papel de proporcionar a visualização dos dados medidos em um dispositivo móvel e pode servir como base para o desenvolvimento de aplicações mais complexas com finalidades semelhantes.

5.3 Sugestões para trabalhos futuros

Neste trabalho não foram realizados testes de carga da arquitetura. Estes testes tem como objetivo verificar o comportamento da aplicação quando submetida a uma grande taxa de requisições, de forma a desenvolver mecanismos de *autoscaling* para garantir a disponibilidade do serviço. Testes de carga se mostram importantes para uma eventual aplicação em larga escala do aplicativo, sendo assim, uma possibilidade de tema para o desenvolvimento de trabalhos futuros.

Referências

ABOUT AWS. Omnigraphics, 2018. Disponível em: <https://aws.amazon.com/about-aws/>. Citado na página 27.

ADELANTADO, F. et al. Understanding the limits of LoRaWAN. *IEEE Communications Magazine*, v. 55, n. 9, p. 34–40, 2017. Citado na página 24.

ANDERSON, R.; LARKIN, K. *Create a web API with ASP.NET CORE*. Disponível em: <https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-web-api?view=aspnetcore-2.1&tabs=visual-studio>. Citado na página 26.

B-L072Z-LRWAN1. Disponível em: <https://www.st.com/en/evaluation-tools/b-l072z-lrwan1.html>. Citado 2 vezes nas páginas 32 e 33.

BOITEUX, H. *Demanda crescente estimula a produção orgânica no Brasil e no mundo*. 2020. Disponível em: <https://www.ipea.gov.br/portal/index.php?option=com_content&view=article&id=35326>. Citado na página 19.

BOQUET, G. et al. LR-FHSS: Overview and performance analysis. *IEEE Communications Magazine*, v. 59, n. 3, p. 30–36, 2021. Citado na página 21.

CARLYLE, A. G.; HARRELL, S. L.; SMITH, P. M. Cost-effective hpc: The community or the cloud? In: IEEE. *2010 IEEE Second International Conference on Cloud Computing Technology and Science*. [S.l.], 2010. p. 169–176. Citado na página 27.

CHEONG, P. S. et al. Comparison of LoRaWAN classes and their power consumption. In: *2017 IEEE Symposium on Communications and Vehicular Technology (SCVT)*. [S.l.: s.n.], 2017. p. 1–6. Citado 2 vezes nas páginas 23 e 24.

CLARK, M. *Qual o significado de backend? - back4app blog*. Disponível em: <https://blog.back4app.com/pt/qual-o-significado-de-backend/>. Citado na página 25.

CONTEXT API. Disponível em: <https://pt-br.reactjs.org/docs/context.html>. Citado na página 39.

DANIELSSON, W. React native application development. *Linköpings universitet, Swedia*, v. 10, n. 4, 2016. Citado na página 27.

DEVALAL, S.; KARTHIKEYAN, A. LoRa technology - an overview. In: *2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA)*. [S.l.: s.n.], 2018. p. 284–290. Citado na página 21.

ERTÜRK, M. A. et al. A survey on LoRaWAN architecture, protocol and technologies. *Future Internet*, MDPI, v. 11, n. 10, p. 216, 2019. Citado 2 vezes nas páginas 22 e 23.

FIGMA. Disponível em: <https://www.figma.com/>. Citado na página 33.

FORCIER, J.; BISSEX, P.; CHUN, W. J. *Python web development with Django*. [S.l.]: Addison-Wesley Professional, 2008. Citado na página 25.

GERASIMOV, A. et al. Generated enterprise information systems: Mdse for maintainable co-development of frontend and backend. In: *Modellierung (Companion)*. [S.l.: s.n.], 2020. p. 22–30. Citado na página 24.

GOOGLE Cloud Platform. Google. Disponível em: <https://cloud.google.com/>. Citado na página 28.

GUPTA, B.; MITTAL, P.; MUFTI, T. A review on amazon web service (aws), microsoft azure & google cloud platform (gcp) services. 2021. Citado na página 27.

HAXHIBEQIRI, J. et al. A survey of LoRaWAN for IoT: From technology to application. *Sensors*, MDPI, v. 18, n. 11, p. 3995, 2018. Citado na página 22.

I-CUBE-LRWAN. Disponível em: <https://www.st.com/en/embedded-software/i-cube-lrwan.html>. Citado na página 34.

ITG 200 indoor. Disponível em: <https://www.khomp.com/pt/produto/itg-200/>. Citado 2 vezes nas páginas 32 e 33.

KARANJIT, A. Mean vs. lamp stack. 2016. Citado na página 25.

LALLE, Y. et al. LoRaWAN network capacity analysis for smart water grid. In: *2020 12th International Symposium on Communication Systems, Networks and Digital Signal Processing (CSNDSP)*. [S.l.: s.n.], 2020. p. 1–6. Citado na página 23.

LPS8. Disponível em: <https://www.dragino.com/products/lora-lorawan-gateway/item/148-lps8.html>. Citado na página 32.

MACHADO, C. R. P. Produção orgânica em estufa na comunidade rural de cacimbas, quarto distrito de mostardas-rs. 2019. Disponível em: <https://lume.ufrgs.br/handle/10183/197724>. Citado na página 19.

MIAO, H.-Y. et al. On construction of a campus outdoor air and water quality monitoring system using LoRaWAN. *Applied Sciences*, MDPI, v. 12, n. 10, p. 5018, 2022. Citado na página 23.

MICROSOFT. *Visual Studio Code*. Microsoft, 2021. Disponível em: <https://code.visualstudio.com/>. Citado na página 33.

O que é o Azure. Disponível em: <https://azure.microsoft.com/pt-br/overview/what-is-azure/>. Citado na página 28.

PUKRONGTA, N.; KUMKHET, B. The relation of LoRaWAN efficiency with energy consumption of sensor node. In: IEEE. *2019 International Conference on Power, Energy and Innovations (ICPEI)*. [S.l.], 2019. p. 90–93. Citado na página 23.

PYCHARM. Disponível em: <https://www.jetbrains.com/pt-br/pycharm/>. Citado na página 33.

QIAN, L. et al. Cloud computing: An overview. In: SPRINGER. *IEEE international conference on cloud computing*. [S.l.], 2009. p. 626–631. Citado na página 27.

RASCHKA, S.; PATTERSON, J.; NOLET, C. Machine learning in python: Main developments and technology trends in data science, machine learning, and artificial intelligence. *Information*, Multidisciplinary Digital Publishing Institute, v. 11, n. 4, p. 193, 2020. Citado na página 25.

RAZA, U.; KULKARNI, P.; SOORIYABANDARA, M. Low power wide area networks: An overview. *IEEE Communications Surveys Tutorials*, v. 19, n. 2, p. 855–873, 2017. Citado 2 vezes nas páginas 21 e 24.

REACT Native Internals. Disponível em: <https://www.reactnative.guide/3-react-native-internals/3.1-react-native-internals.html>. Citado na página 27.

SIGFOX.COM. 2022. Disponível em: <https://www.sigfox.com/en>. Citado na página 21.

STATISTICS and Data. 2022. Disponível em: <https://statisticsanddata.org/data/most-popular-backend-frameworks-2012-2022/>. Citado na página 26.

SX1276. Disponível em: <https://www.semtech.com/products/wireless-rf/lora-core/sx1276>. Citado na página 32.

VAINIKKA, J. Full-stack web development using django rest framework and react. Metropolia Ammattikorkeakoulu, 2018. Disponível em: <https://www.theseus.fi/bitstream/handle/10024/146578/joel_vainikka.pdf?sequence=1>. Citado 2 vezes nas páginas 25 e 26.

WHAT is LoRaWAN. 2021. Disponível em: <https://lora-alliance.org/resource_hub/what-is-lorawan/>. Citado 2 vezes nas páginas 21 e 22.

X-NUCLEO-IKS01A2. Disponível em: <https://www.st.com/en/ecosystems/x-nucleo-iks01a2.html>. Citado 2 vezes nas páginas 32 e 33.