Universidade Estadual de Londrina
Centro de Tecnologia e Urbanismo
Departamento de Engenharia Elétrica

Mateus Perrut de Souza

Desenvolvimento de um sistema de IOT para monitoramento de grandezas agrometeorológicas

Londrina
2023

Universidade Estadual de Londrina

Centro de Tecnologia e Urbanismo
Departamento de Engenharia Elétrica

Mateus Perrut de Souza

Desenvolvimento de um sistema de IOT para monitoramento de grandezas agrometeorológicas

Trabalho de Conclusão de Curso orientado pelo Prof. Maria Bernadete de Morais França intitulado "Desenvolvimento de um sistema de IOT para monitoramento de grandezas agrometeorológicas" e apresentado à Universidade Estadual de Londrina, como parte dos requisitos necessários para a obtenção do Título de Bacharel em Engenharia Elétrica.

Orientador: Prof. Maria Bernadete de Morais França

Londrina
2023

Ficha Catalográfica

Mateus Perrut de Souza
Desenvolvimento de um sistema de IOT para monitoramento de grandezas agrometeorológicas - Londrina, 2023 - 128 p., 30 cm.
Orientador: Prof. Maria Bernadete de Morais França
1. ESP32. 2. Firebase. 3. Aplicação Web. 4. API.
I. Universidade Estadual de Londrina. Curso de Engenharia Elétrica. II. Desenvolvimento de um sistema de IOT para monitoramento de grandezas agrometeorológicas.

Mateus Perrut de Souza

Desenvolvimento de um sistema de IOT para monitoramento de grandezas agrometeorológicas

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia Elétrica da Universidade Estadual de Londrina, como requisito parcial para a obtenção do título de Bacharel em Engenharia Elétrica.

Comissão Examinadora

Prof. Maria Bernadete de Morais França
Universidade Estadual de Londrina
Orientador

Prof. Aziz Elias Demian Junior
Universidade Estadual de Londrina

Prof. José Alexandre de França
Universidade Estadual de Londrina

Londrina, 6 de junho de 2023

Agradecimentos

A todos aqueles que contribuíram, de alguma forma, para a realização deste trabalho, expresso minha mais profunda gratidão. Não há palavras suficientes para expressar o quanto sou grato à minha família por todo o amor, apoio e encorajamento que eles me deram durante a minha jornada acadêmica. Sem a paciência, o incentivo e o suporte financeiro deles, eu não teria conseguido chegar até aqui.

Também desejo agradecer aos amigos que fiz durante essa jornada. Vocês foram os pilares da minha lucidez, sempre presentes para me encorajar, me motivar e me lembrar que eu era capaz de conquistar qualquer coisa que eu me propusesse a fazer. Desde os momentos de estudo juntos até os momentos de lazer, cada momento compartilhado com vocês foi inestimável e inesquecível.

Além disso, minha gratidão se estende aos professores por sua dedicação, orientação e conhecimento compartilhado ao longo dessa graduação. Seus ensinamentos contribuíram diretamente para a minha formação profissional e pessoal, e foram fundamentais para o desenvolvimento deste trabalho.

Não são as ações grandiosas que definem uma pessoa,
mas as pequenas e constantes escolhas
que fazemos todos os dias.
(The Good Place - Michael Schur)

Mateus Perrut de Souza. **Desenvolvimento de um sistema de IOT para monitoramento de grandezas agrometeorológicas**. 2023. 128 p. Trabalho de Conclusão de Curso em Engenharia Elétrica - Universidade Estadual de Londrina, Londrina.

Resumo

O presente trabalho teve como objetivo desenvolver um sistema IoT para o monitoramento de grandezas agrometeorológicas. Frente ao avanço no desenvolvimento de tecnologias específicas para IoT e a relevância da aplicação dessa tecnologias em sistemas de produção agrícola. Portanto um sistema IoT foi implementado para coletar e armazenar em tempo real dados agrometeorológicos, como temperatura, umidade do ar e intensidade luminosa, utilizando sensores especializados. Uma aplicação web foi desenvolvida para permitir o acesso remoto e visualização dos dados coletados. A integração de API's possibilitou a combinação desses dados com outras fontes de informação, enriquecendo sua contextualização. O estudo revelou os desafios, devido a complexidade das ferramentas integradas, e benefícios por meio da implementação de novas tecnologias, abrindo perspectivas futuras de melhoria desse sistema com a adaptação a diferentes cultivos, inclusão de sensores adicionais, desenvolvimento de modelos preditivos e controle ambiental.

Palavras-Chave: 1. ESP32. 2. Firebase. 3. Aplicação Web. 4. API.

Mateus Perrut de Souza. **Development of an IoT system for monitoring agrometeorological magnitudes**. 2023. 128 p. Monograph in Electrical Engineering - Londrina State University, Londrina.

Abstract

The present work aimed to develop an IoT system for monitoring agrometeorological variables. Given the advancements in IoT-specific technologies and their relevance in agricultural production systems. Therefore an IoT system was implemented to collect and store agrometeorological data in real-time, such as temperature, air humidity, and light intensity, using specialized sensors. A web application was developed to enable remote access and visualization of the collected data. The integration of APIs allowed the combination of this data with other sources of information, enriching its contextualization. The study revealed the challenges, due to the complexity of the integrated tools, and the benefits of implementing new technologies, opening future prospects for system improvement through adaptation to different crops, inclusion of additional sensors, development of predictive models, and environmental control.

Key-words: 1. ESP32. 2. Firebase. 3. Web Application. 4. API.

Lista de ilustrações

Lista de tabelas

Lista de quadros

Lista de Siglas e Abreviaturas

IoT	*Internet of Things (Internet das Coisas)*
3G	*Third Generation (Terceira Geração de Redes de Comunicação Móvel)*
4G	*Fourth Generation (Quarta Geração de Redes de Comunicação Móvel)*
5G	*Fifth Generation (Quinta Geração de Redes de Comunicação Móvel)*
ADC	*Analog-to-Digital Converters*
AMQP	*Advanced Message Queuing Protocol (Protocolo de Filas de Mensagens Avançado)*
API	*Application Programming Interface (Interface de Programação de Aplicativos)*
CLI	*Command Line Interface (Interface de Linha de Comando)*
CoAP	*Constrained Application Protocol (Protocolo de Aplicação Restrita)*
CSS	*Cascading Style Sheets (Folhas de Estilo em Cascata)*
CSV	*Comma-Separated Values (Valores Separados por Vírgula)*
epcoch	*Época, em tempo Unix*
GMT	*Greenwich Mean Time (Tempo Médio de Greenwich)*
GPS	*Global Positioning System (Sistema de Posicionamento Global)*
HTML	*Hypertext Markup Language (Linguagem de Marcação de Hipertexto)*
HTTP	*Hypertext Transfer Protocol (Protocolo de Transferência de Hipertexto)*
IDE	*Integrated Development Environment (Ambiente de Desenvolvimento Integrado)*
I2C	*Inter-Integrated Circuit (Circuito Integrado Interno)*
JSON	*JavaScript Object Notation (Notação de Objeto JavaScript)*
LGPD	*Lei Geral de Proteção de Dados Pessoais*
M2M	*Machine-to-Machine (Comunicação de Máquina para Máquina)*
MIME	*Multipurpose Internet Mail Extensions (Correio de Internet de Uso Múltiplo)*
MongoDB	*Humongous Database (Banco de Dados Gigantesco)*
MQTT	*Message Queuing Telemetry Transport*
NTP	*Network Time Protocol (Protocolo de Tempo de Rede)*
NoSQL	*Not Only SQL (Não Apenas SQL)*
NodeMCU	*Node Microcontroller Unit*
REST	*Representational State Transfer (Transferência de Estado Representacional)*
RNA	*Ácido Ribonucleico*
RTDB	*Real-Time Database (Banco de Dados em Tempo Real)*
RX	*Receiver (Receptor)*
SDK	*Software Development Kit (Kit de Desenvolvimento de Software)*
SSL	*Secure Sockets Layer (Camada de Soquetes Seguros)*
TCP	*Transmission Control Protocol (Protocolo de Controle de Transmissão)*
TLS	*Transport Layer Security (Segurança da Camada de Transporte)*

TX	*Transmitter (Transmissor)*
UDP	*User Datagram Protocol (Protocolo de Datagrama de Usuário)*
UID	*User Id (identidade de usuário)*
URI	*Uniform Resource Identifier (Identificador Uniforme de Recursos)*
UV	*Ultraviolet (Ultravioleta)*
UVA	*Ultraviolet A (Ultravioleta A)*
URL	*Uniform Resource Locator (Localizador Uniforme de Recursos)*
UTMS	*Universal Mobile Telecommunications System (Sistema de Telecomunicações Móveis)*
Web	*World Wide Web (Rede de Alcance Mundial)*
Wi-Fi	*Wireless Fidelity (Rede sem Fio)*
XML	*eXtensible Markup Language (Linguagem de Marcação Extensível)*

Sumário

1 Introdução

Desde o final do século XX a internet vem se consolidando como uma das principais ferramentas de comunicação, sendo essa comunicação de pessoa para pessoa (*human-human*) ou pessoa para máquina (*human-machine*), e ainda outra forma de comunicação vêm se destacando recentemente como apresentado em (FAROOQ, 2015), a máquina para máquina (machine-machine, M2M), onde os dispositivos se comunicam, geram dados, tomam decisões e ações sem um ser humano como intermediário, é nessa ultima categoria de comunicação que o desenvolvimento IoT (*Internet of Things*) é baseada.

A internet das coisas, é a rede de em que objetos (coisas) se comunicam e estabelecem um fluxo de dados. Sendo esses objetos dispositivos, instrumentos, sistemas embarcados, circuitos, softweres e computadores, que permitem que dados sejam coletados e trocados entre si. Em (GOKHALE, 2018), é mostrado algumas áreas onde já está em desenvolvimento diversas aplicações IoT: Engenharia hospitalar, construção civil, automação de carros, automação industrial, agricultura, entre outros.

Com os recentes avanços em telecomunicações, como tecnologias 5G, trazem uma nova perspectiva para o desenvolvimento IoT, com uma maior disponibilidade de rede espera-se uma maior disseminação dessa tecnologia. O desenvolvimento de sistemas IoT é heterogêneo, ou seja, utiliza uma grande variedade de tecnologias na sua infraestrutura. Desde os protocolos de comunicação até o armazenamento dos dados em banco de dados, por esse motivo é necessário estudos validando essas tecnologias individualmente e a combinação delas, para a construção assertiva de arquiteturas IoT.

Para a concepção do projeto, a seleção dos dispositivos e sensores adequados é um processo crucial para o sistema, pois são esses que irão coletar a informação do mundo real, se esse processo não for adequado todo o resto do sistema não funcionará. No contexto de grandezas agrometeorológicas alguns fatores na escolha desses dispositivos devem ser considerados (ROSELIN, 2017), como o tipo de cultivo, o ambiente, a conectividade, a alimentação e eficiência de energia, a integração e o custo. Assim, a escolha se torna mais assertiva. Ainda para a operação desses dispositivos é necessária a escolha da plataforma de hardware e software, que deve garantir a compatibilidade com os dispositivos e sensores selecionados e a adequação ao ambiente em que o sistema será implantado, a escalabilidade do sistema, e a eficiência energética do sistema.

Os protocolos de comunicação são responsáveis pela transmissão dos dados, utilizados extensivamente desde os primórdios da internet, entretanto como em (AL-MASRI, 2017) para sistemas IOT novos desafios são encontrados, como um alto nível de interação entre dispositivos distribuídos, uma alta performance devido ao grande fluxo de dados, e maior eficiência energética para uma maior durabilidade do sistema. Para atender esses tópicos novos protocolos foram definidos como o MQTT, CoAP e o AMQP, e protocolos já exis-

tentes foram adaptados para essa realidade, como o HTTP e o Websocet. Cada protocolo apresenta características únicas diante dos problemas enfrentados, a escolha de um em detrimento dos outros deve-se as especificidades do sistema a ser desenvolvido.

A segurança desses sistemas também é um tópico fundamental, muitos dispositivos IoT são projetados com baixa potência de processamento e limitações de armazenamento, o que pode dificultar a implementação de medidas de segurança adequadas. A fim de garantir a segurança dos sistemas IoT, é necessário empregar protocolos de segurança robustos, como autenticação e criptografia de dados, bem como monitoramento constante e atualizações de segurança regulares para mitigar vulnerabilidades potenciais. Recentemente no Brasil entrou em vigor a Lei Geral de Proteção de Dados Pessoais (LGPD), Lei n° 13.709/2018, que tem como objetivo regulamentar a proteção, tratamento e armazenamento de dados pessoais no país, garantindo a privacidade e a transparência no uso desses dados. Portanto, a exigência de um maior nível de segurança nos sistemas não é mais uma boa prática mas um dever jurídico.

Outra tecnologia relevante para a concepção de um sistema IoT, são os bancos de dados, que são os responsáveis pelo armazenamento e gerenciamento dos dados coletados. Na internet das coisas, os dados são gerados em grande quantidade em tempo real, variando em termos de volume, variedade de formatos e velocidade. Nesse sentido, o banco de dados precisa ser capaz de lidar com esses dados suportando a escalabilidade e a disponibilidade para operar em tempo real (ASIMINIDIS, 2018). No contexto de banco de dados eles podem ser divididos em duas principais categorias, os bancos relacionais e os não relacionais. Aqui também a escolha do banco depende da especificações do projeto, entretanto os bancos não relacionais por apresentarem uma maior flexibilidade são uma escolha mais comum para sistemas IoT (RAUTMARE, 2016).

Além da construção do sistema IOT, um processo muito relevante atualmente é a integração desse sistema com outros. Como em (FERREIRA, 2013), um meio comum de realizar essa integração é através das APIs (*Application Programming Interface*) que permitem a comunicação entre diferentes sistemas na internet. Inclusive os sistemas IoT podem atuar como API's, e compartilhar os dados coletados e processados para outras aplicações. Outra possibilidade é usar outras API's para o processamento dos dados coletados e plotar os resultados obtidos na interface. Portanto o uso das APIs com sistemas IoT é fundamental para solução mais completas.

Por fim, os sistemas IoT lidam com um grande volume de dados, e para extrair informações relevantes desses dados, são aplicadas técnicas de ciência de dados, com as quais é possível identificar padrões, tendências e anomalias nos dados coletados, o que pode levar a percepções valiosas para a tomada de decisões e melhoria dos processos. Além disso, a ciência de dados pode ser utilizada para a criação de modelos preditivos que auxiliam na previsão de eventos futuros e no planejamento de ações preventivas.

Como demonstrado, o desenvolvimento de um sistema IoT pode ser um processo bas-

tante complexo e envolver diversas especificações e tecnologias. Para garantir que o projeto seja bem sucedido, é necessário realizar pesquisas e estudos que abranjam o sistema como um todo. Isso inclui a compreensão de como aplicar os requisitos conceituais na prática, a criação de um sistema robusto e, ao mesmo tempo, com uma boa usabilidade, e a elaboração de modelos de planejamento para o projeto. Dessa forma, é possível garantir que o sistema IoT seja desenvolvido de maneira eficiente e atenda às necessidades dos usuários e do mercado.

Ainda é necessário métodos de avaliação e classificação desses sistemas. E por meio de parâmetros bem definidos realizar um julgamento objetivo do sistema, compreendendo as condições do projeto e o contexto em que ele será inserido. Esses parâmetros são abordados em (THOMA, 2012) e (KRAIJAK, 2013), incluindo a confiabilidade, escalabilidade, segurança, eficiência energética, usabilidade e manutenibilidade. A confiabilidade refere-se à capacidade de transmitir dados com precisão e consistência. A escalabilidade está relacionada com a capacidade do sistema de lidar com um grande número de dispositivos e grandes volumes de dados. A segurança diz respeito ao quão seguro o sistema é. A eficiência energética se relaciona com a capacidade de usar a energia de forma sustentável e eficiente. A usabilidade está ligada ao quão fácil é o acesso e as ferramentas de interação com os usuários. Já a manutenibilidade refere-se à facilidade de manutenção e atualização do sistema.

Como discutido esses sistemas são usados em uma variedade de aplicações. Entretanto para o desenvolvimento do projeto é preciso ter um campo de estudo como objetivo. Como em (ROY, 2008), o uso de tecnologias IoT no setor agrícola vem se mostrado uma importante solução para aumentar a eficiência na produção e otimizar os processos, com a utilização de sensores conectados à internet, é possível monitorar e controlar diversas variáveis do ambiente agrícola, como a umidade do solo, a temperatura e a luminosidade. Isso permite um melhor gerenciamento dos recursos disponíveis e a aplicação mais precisa de insumos, como água, fertilizantes e pesticidas. Além disso, sistemas de análise de dados podem fornecer informações valiosas sobre o desempenho da produção e ajudar na tomada de decisões estratégicas.

Portanto, nesse trabalho pretende-se construir um sistema IoT para o monitoramento de grandezas agrometeorológicas, e validar as tecnologias desenvolvidas em todos os processos necessários para a sua construção. Posteriormente, serão implementadas essas tecnologias no sistema em questão. E com base nos parâmetros de análise, será feita uma avaliação da qualidade do sistema desenvolvido, contribuindo assim para a construção de um planejamento mais eficaz de projetos na área e para a criação de métodos mais precisos de avaliação desses sistemas.

1.1 Motivação

A agricultura é um setor vital para a economia brasileira, e o uso de tecnologias avançadas para melhorar a produtividade e a eficiência é cada vez mais importante, tanto para os grandes, médios e pequenos produtores. Nesse sentido, os sistemas IoT têm se destacado como uma solução promissora para o monitoramento de grandezas agrometeorológicas, permitindo uma análise mais precisa do ambiente e das condições climáticas. Além disso, o recente avanço nos estudos de tecnologias IoT tem mostrado um potencial significativo para a criação de sistemas cada vez mais robustos e eficientes. Diante disso, a presente pesquisa tem como principal motivação contribuir para o desenvolvimento de um sistema IoT capaz de monitorar grandezas agrometeorológicas de forma eficiente e precisa, visando melhorar a produtividade e a qualidade dos cultivos.

1.2 Justificativa

A Internet das Coisas tem sido uma das tecnologias mais promissoras e com maior potencial de transformação na atualidade (DACHYAR, 2019). O desenvolvimento da IoT tem sido impulsionado por diversos fatores, como a queda no custo dos sensores e dispositivos conectados, a evolução das tecnologias de comunicação sem fio, como o 5G, e a crescente demanda por soluções tecnológicas que possam aumentar a eficiência e a produtividade em diversos setores, como a indústria, a agricultura e o transporte.

A perspectiva para os próximos anos é que o desenvolvimento da IoT continue crescendo de forma acelerada, impulsionado pelo aumento da adoção de tecnologias como o 5G, que permitirá a conexão de um número ainda maior de dispositivos e objetos, além de possibilitar a transmissão de dados em tempo real com baixa latência.

Entretanto, o desenvolvimento de sistemas IoT se da a partir da integração de diversas tecnologias, o que abrange diversos temas complexos, trazendo oportunidade de pesquisas e avanços científicos nessas áreas. Tanto nos processos individuais do sistema quanto no conjunto dessas tecnologias.

Pelo crescente avanço na área de Internet das Coisas e pela complexidade do desenvolvimento desses sistemas, o trabalho justifica-se. Visando entender e analisar os processos de desenvolvimento de um sistema aplicado a um contexto específico, o monitoramento de grandezas agrometeorológicas.

1.3 Objetivos

Validar conceitos IoT que já vem sendo aplicados e desenvolver um sistema para o monitoramento de grandezas agrometeorológicas. Através de um microcontrolador, coletar dados de grandezas agrometeorológicas utilizando sensores. Formatar e transmitir

esses dados via internet por uma comunicação confiável. Armazenar os dados utilizando um sistema gerenciador de banco de dados próprio para a natureza do projeto. Construir uma interface de usuário em uma aplicação Web, e hospedar essa aplicação publicamente. Utilizar técnicas de autentificação e credenciamento para garantir um maior nível de segurança do sistema. Conectar a aplicação ao banco de dados, e expor esses dados de maneira clara e objetiva. Usar API's relevantes e que contribuam para o sistema. Construir uma API própria para manipulação dos dados.

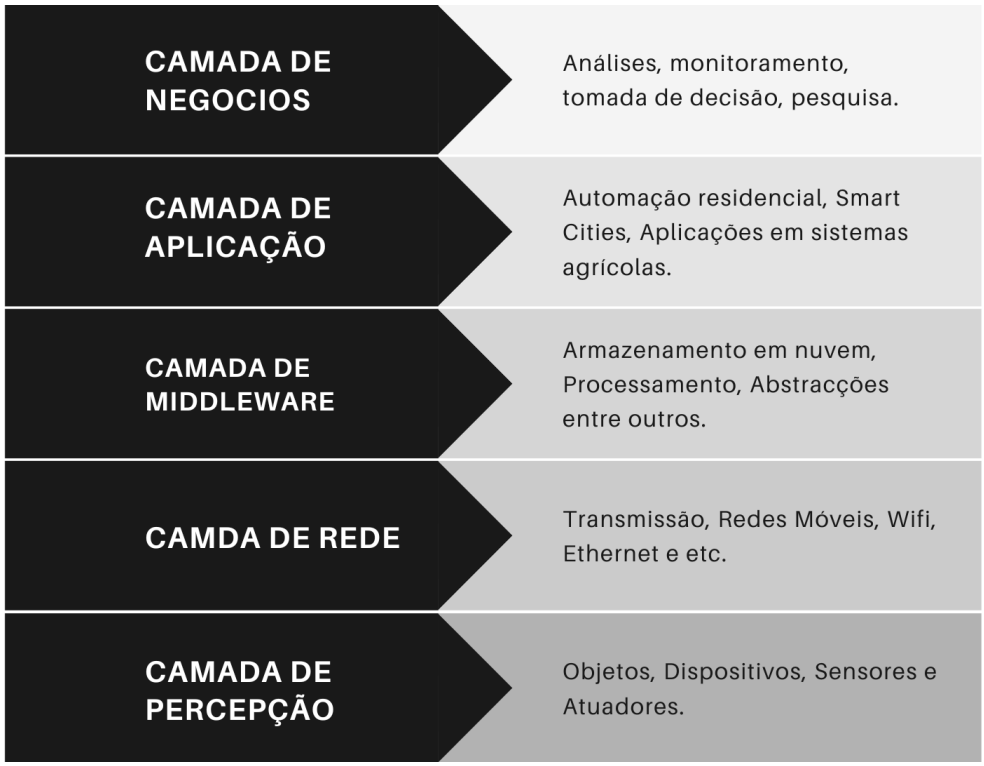
2 Fundamentação Teórica

Será abordado neste tópico os principais conceitos como a arquitetura de um sistema IoT, as tecnologias de comunicação utilizadas para a transmissão de dados, além de uma análise sobre os protocolos de comunicação e os dispositivos que serão utilizados. Será apresentado também as tecnologias específicas que serão utilizadas neste trabalho, como, o microcontrolador que será utilizado no desenvolvimento, os sensores e suas características relevantes para este projeto. Será abordado ainda a infraestrutura necessária para a implementação do sistema e as tecnologias de armazenamento de dados.

2.1 Arquitetura IoT

A arquitetura IoT é um conjunto de camadas que fundamental um sistema IoT, garantindo sua operação e segurança (GOKHALE, 2018). Essa arquitetura é responsável por conectar dispositivos físicos, dispositivos virtuais e aplicações de forma coesa e segura, para assim permitir o fluxo de dados.
O desenvolvimento da arquitetura envolve alguns fatores fundamentais como a rede, comunicação e processamentos, e ainda deve ser adaptativa para permitir que os dispositivos comuniquem entre sim de forma dinâmica e suportem a comunicação entre eles.

Figura 1 – Arquitetura IOT baseada em camadas



Fonte: Próprio autor

2.1.1 Camada de percepção

Também chamada de camada de sensoreamento, é camada onde se encontram os dispositivos físicos utilizados no sistemas como sensores, atuadores, tags e controladores. A principal função dessa camada é coletar as informações do ambiente converte-las em sinais digitais e transmitir esses sinais para a camada de rede (FAROOQ, 2015).
Em (GOKHALE, 2018) é exposto como cada dispositivo pode possuir uma identificação única e o ambiente ao entorno ser monitorado para diversos propósitos e aplicações. Todo objeto em um sistema IoT possui uma identidade digital e pode ser facilmente rastreado com seu domínio digital, essa característica é importante para sistemas de médio e grande complexidade com um número elevado de objetos, para manutenção do sistema, segurança.
Alguns aspectos são considerados no desenvolvimento dessa camada, como o custo, tamanho e consumo de energia, uma estratégia é minimizar os recursos utilizados assim como seu custo para tornar o sistema mais economicamente viável e sustentável. Outro fator determinante é a comunicação entre os dispositivos, estabelecer quais serão os protocolos de comunicação utilizados para permitir a interação entre diferentes dispositivos.

2.1.2 Camada de rede

O propósito dessa camada é receber as informações da camada de percepção na forma digital e transmiti-las para os sistemas de processamento na camada de middleware (FAROOQ, 2015), usando tecnologias de rede como 3G, 4G, UTMS, Wi-FI, infravermelho, etc.
A rede deve mapear automaticamente os objetos na arquitetura. É necessário estabelecer regras para implantar, gerenciar, e planejar as ações dos objetos, assim a camada de rede permite o compartilhamento de dados do sistema.
Nessa camada algumas propriedades devem ser entendiadas como: o gerenciamento de redes como, redes sem fio, redes móveis, ethernet entre outras; Tecnologias de busca e processamento de dados; Parâmetros de segurança e privacidade também devem ser implementados.
Como em (GOKHALE, 2018) confidencialidade e privacidade são aspectos criticos em sistemas IoT, pois esse conecta diversos objetos pessoais, e são implementados em ambientes como residencias, empresas e outros espaços privados, com potencial risco de ataques, portanto diversos estudos em segurança de redes estão sendo produzidos com a finalidade de atender sistemas IOT e evitar esse problema.

2.1.3 Camada de middleware

Nessa camada ocorre o processamento das informações recebidas dos objetos. Dado a dificuldade de padronização dos dispositivos em um sistema IoT, a camada de milddleware tem como função abstrair os requerimentos específicos de cada objeto afim de permitir que todos estejam em uma mesma aplicação. Como em (SETHI, 2017), essa camada pode ser definida como softweres que fazem pontes entre os objetos e as aplicações. Alguns desafios que essa camada é responsável:

- Abstrações: abstrair os hardwares e fornecer uma interface de programação de aplicações (API) para comunicações, gerenciamento de dados, segurança e privacidade

- Escalabilidade: conforme um sistema IOT for expandido e mais dispositivos forem coneta-dos as aplicações devem escalar aumentando os requerimentos, assim a camada de middleware deve gerenciar as mudanças na infraestrutura.

- Análise de dados: IOT trabalha com uma grande quantidade de dados, é necessário então que através de algoritmos esses dados sejam analisados, agrupados e manipulados a fim de facilitar a sua leitura pelas aplicações

- Segurança e privacidade: É necessário o uso de mecanismos que reforcem a segurança do sistema, como autentificação de usuários, controle de acessos e outras ferramentas.

- Armazenamento em nuvem: Além de tratar os dados coletados, esses dados precisam ser armazenados e centralizados, no contexto de internet uma opção interessante é utilizar o armazenamento em nuvem.

2.1.4 Camada de aplicação

Depois de os dados serem processados eles serão utilizados para construir aplicações, estudos indicam o potencial de melhoria de vida na sociedade através do uso de aplicações IoT (SETHI, 2017). Alguns de seus exemplos são automação residencial, monitoramento de saúde, proteção ambiental, cidades inteligentes, aplicações industriais, e em sistemas agrícolas.
Um fator importante no desenvolvimento de um aplicação IoT, é entender quem irá utilizar-la de forma a otimizar a aplicação para tornar a experiencia dos usuários mais assertiva como leitura clara dos dados através de gráficos e tabelas, ferramentas de fácil acesso e uso (sites, aplicativos mobile, e etc), garantir a segurança do sistema e respeitar as orientações quanto a privacidade dos usuários.

2.1.5 Camada de negócios

Essa camada valida o objetivo da implementação de um sistema IoT, pois as informações geradas pelas camadas anteriores só terão valor se resultarem nas soluções de problemas ou monitorarem um fenômeno desejado (LUCERO, 2016). Aqui através dos resultados que as informações do sistema serão propostas decisões com o objetivo de melhorar um processo, em um contexto comercial essas propostas são formuladas como modelos de negócios, já em um contexto acadêmico pode-se utiliza-la para confirmar ou confrontar uma hipótese, uma ferramenta de pesquisa científica.

2.2 Protocolos de comunicação

Avanços em tecnologias de rede tem contribuído em como dispositivos IoT produzem, compartilham e coletam dados. Sendo evidente no aumento de magnitude e da taxa dos dados gerados por sistemas IoT. E ainda, está ocorrendo um avanço no desenvolvimento e implementação de vários protocolos de comunicação que permitem dispositivos IOT trocarem dados mais eficientemente.
Sistemas IoT possuem um alto nível de interação entre um grande número de dispositivos distribuídos, portanto é necessário um protocolo de comunicação que atenda as demandas desses sistemas. Como em (AL-MASRI, 2017) demandas como:

- Coesão do sistema: capacidade de comunicar de maneira transparente dado um sistema heterogêneo, ou seja, com dispositivos diversos com especificidades individuais.
- Provisionamento de serviços: aplicações IoT consomem um ou mais serviços para realizar funcionalidades especificas, neste contexto serviços se referem a funcionalidades do protocolo para desenvolvimento, como depuração, métodos de manuseio de dados, criptografias entre outros serviços.
- Microsserviços e Rastreamento Distribuído: micro-serviços consistem em distribuir as funcionalidades de um softwere em diversos outros softweres menores e responsáveis por uma única ou poucas funções dentro daquele sistema, assim ao invés de executar um único serviço que executa todo o sistema é possível executar os micro-serviços de maneira paralela e assim permitir um ganho de velocidade do sistema. Já o rastreamento distribuído permite o acompanhamento individual das requisições de cada micro-serviço. Atualmente somente o protocolo HTTP atende completamente a essas funcionalidades.
- Escalabilidade: propriedade em que o sistema é capaz de manter suas funcionalidades dentro de uma margem de qualidade conforme o número de elementos do sistema cresce ou o número de usuários.

- Performance: a performance de um protocolo pode ser avaliada através da latência, tempo de resposta frente a uma requisição de serviço. Outra maneira de avaliar a performance é sobre sua taxa de transmissão, a velocidade em que os dados serão transmitidos, fator relevante no desenvolvimento do sistema, pois se a taxa de transmissão for menor que a velocidade em que os sensores estão realizando a leitura dos dados, ocorrerá o fenômeno conhecido como *overflow* (perda de dados devido ao transbordo). Além desses outros fatores influenciam de maneira relevante no sistema como processamento, memoria, e consumo de energia.

- Confiabilidade: Alguns protocolos são baseados em UDP (*User Datagram Protocol*) na camada de transporte, o UDP tem um cabeçalho significativamente menor que o do TCP (*Transmission Control Protocol*), porém o TCP prove controle de erros ou de fluxo, portanto afirma-se que o TCP possui confiabilidade e por isso alguns protocolos optam por ele.

- Segurança: é necessário atender alguns parâmetros de privacidade e segurança no sistema, como serviços de autentificação, controle de acesso, criptografia, registro de atividades (*logging*) entre outras ferramentas.

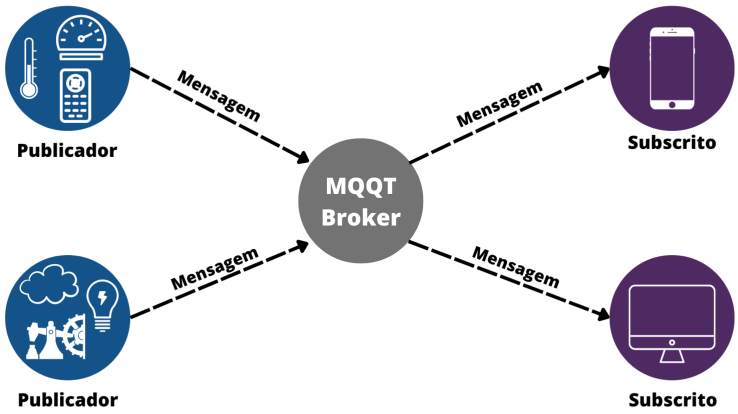
- Adaptabilidade: conforme tecnologias vão sendo desenvolvidos, surge a necessidade de estender componentes existentes de algoritmos, e mudanças e implementação de novas ferramentas, portanto o protocolo deve ser adaptável a essas transformações.

- Implementação do protocolos: importante levar em consideração o uso a disseminação do protocolo escolhido para o uso, quais dispositivos aceitam o seu uso, assim como os processos para sua implementação, documentação, comunidade e suporte.

2.2.1 Message Queuing Telemetry Transport (MQTT)

É um protocolo de comunicação construído para dispositivos restritos em redes não confiáveis, ou seja, com um baixo consumo de rede, banda e recursos de softwere (CORAK, 2018). O formato utilizado é de Cliente/Servidor e roda sobre TCP/IP, que oferece conexões bidirecionais entre nós.
O MQTT consistem em três principais componentes: subscrito, publicador e o *broker* (MORAES, 2019). O subscrito pode requirir e receber mensagens do publicador, ou seja terá o papel de receptor. O publicador comunica com o *broker* para enviar mensagens sobre um tópico específico para os subscritos, ou seja o emissor. Por fim o *broker* será o intermediário para fazer uma ponte de comunicação entre o publicador e o subscrito, se tornando responsável por fazer o recebimento, enfileiramento e envio das mensagens para os nós que estão subscritos naquele tópico (endereço pelo qual os dados serão enviados.

Figura 2 – Diagrama representativo do protocolo MQTT



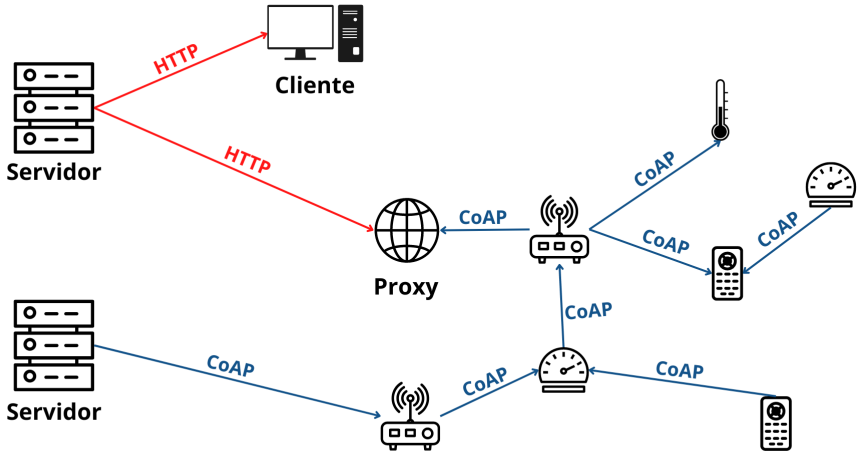
Fonte: Próprio autor

Em (AL-MASRI, 2017) são destacadas algumas vantagens do protocolo MQTT: tem um ciclo transitório de transmissão, é adequado para aplicações em nuvem, um protocolo leve, que funciona razoavelmente bem mesmo em conexões de rede com menos recursos, suporta mensagens assíncronas, é orientado a eventos, pode ser usado para comunicação M2M (*machine to machine*). Entre as principais desvantagens destacam-se a falta de suporte de cargas úteis, inadequado para dispositivos IoT que enviam conteúdos de multimídia (áudio, vídeos, imagens), falta de criptografia, clientes MQTT precisam de suporte TCP e o *broker* tem capacidade limitada de comunicação (CORAK, 2018).

2.2.2 Constrained Application Protocol (CoAP)

Assim como o MQTT é um protocolo para dispositivos e redes com recursos mais limitados. É baseado no protocolo de transporte UDP (CORAK, 2018), para nós e redes com uma conectividade mais baixa, acelera as comunicações ao não estabelecer formalmente uma conexão antes que os dados sejam transferidos. Isso permite que os dados sejam transferidos muito rapidamente, mas também pode fazer com que pacotes se percam em trânsito, portanto um protocolo de rede não confiável, então o CoAP prover um mecanismo interno de confiabilidade (MORAES, 2019).
CoAP possui uma arquitetura *REST*, ou seja a comunicação é feita através dos métodos *GET*, *PUT*, *PUT* e *DELETE*, respectivamente para ler, escrever, publicar e deletar dados entres os dispositivos.

Figura 3 – Diagrama representativo do protocolo CoAP



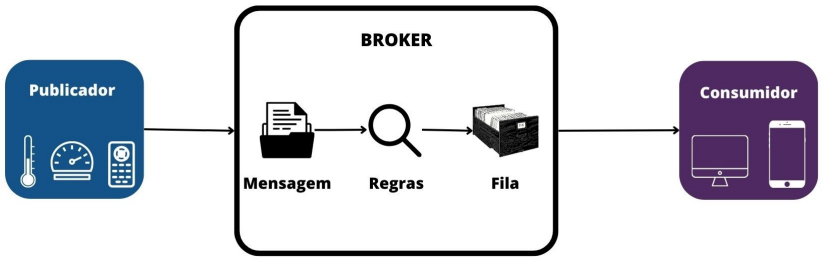
Fonte: Próprio autor

Como ilustrado na figura acima esse protocolo utiliza um servidor *proxy* intermediário entre a origem da requisição, o servidor e o destino da requisição, com a funcionalidade de gerenciar as requisições e respostas da arquitetura *REST*, assim quando o cliente requisita uma informação, essa requisição será enviada para o proxy, e ser direcionada para seu destino. Além disso é possível uma comunicação direta entre servidores através do protocolo CoAP.
Entre as vantagens (CORAK, 2018), desse protocolo destaca-se: a rápida velocidade de comunicação no envio de pacotes pequenos de dados; Suporte de comunicação assíncrona; pode ser utilizado em comunicações M2M. Entre as desvantagens (AL-MASRI, 2017) destacas-se: Protocolo não suporta comunicação entre dispositivo e nuvem, protocolo funciona bem para aplicações menores, falta ferramentas de escalabilidade e adaptabilidade.

2.2.3 Advanced Message Queuing Protocol (AMQP)

AMQP é um protocolo M2M que suporta tanto a arquitetura publicador/subscrito e requisição/resposta, tendo dois modos de interação: modo de pesquisa e modo de consumo (NAIK, 2017). No modo de pesquisa o protocolo funciona similarmente ao protocolo MQTT, o cliente através de um *broker* busca uma informação de um tópico especifico. Já o modo consumo, essas informações são buscadas e deletadas da fila.
A principal característica desse protocolo são os encaminhamentos das mensagens para as filas de acordo com as regras estabelecidas no *broker*, essas regras possuem alta flexibilidade e podem ser alteradas de acordo com a aplicação do protocolo, tornando o protocolo customizável para aplicações mais especificas.

Figura 4 – Diagrama representativo do protocolo AMQP



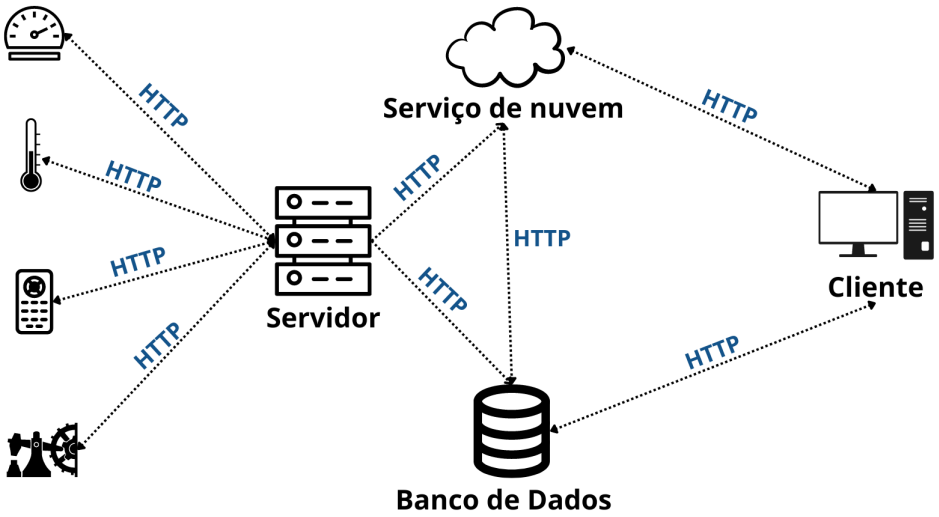
Fonte: Próprio autor

Devido a esse gerenciamento de dados através de estruturas de filas e pela comunicação multiplexada, o AMQP possui um alto nível de escalabilidade, ou seja uma garante a comunicação entre um alto número de elementos dentro de um sistema IoT (CORAK, 2018). Além disso permite a publicação de dados offline, e quando a rede for fornecida esses dados serão devidamente publicados.
Entretanto o protocolo possui alguns problemas, como um fluxo de mensagens lento e muitas vezes complexo, sem mecanismos de segurança adicionais, sem extensões ou comunicação com outros protocolos.

2.2.4 Hyper Text Transfer Protocol (HTTP)

HTTP é predominantemente o protocolo de comunicação web, é operado com a arquitetura *REST*, assim como o CoAP através de requisições e respostas. Diferente de outros protocolos HTTP usa o URI (*Universal Resource Identifier*) ao invés de tópicos, o servidor envia e o cliente recebe os dados através de um URI particular (NAIK, 2017).
O protocolo é construído sobre o protocolo de transporte TCP, e portanto possui uma conexão orientada e confiável. As mensagens desse protocolo são baseadas em texto e não em bits como outros protocolos e o tamanho de cabeçalho é indefinido e depende das técnicas e tecnologias utilizadas.

Figura 5 – Diagrama representativo do protocolo HTTP



Fonte: Próprio autor

Como HTTP é o protocolo padrão da internet, um sistema IoT utilizando-o pode estabelecer a comunicação sobre todos os elementos utilizando somente um protocolo. É também o único que permite o rastreamento distribuído, ou seja identificar de maneira simples os objetos e dispositivos e o acompanhamento individual das requisições. Além dessas características destaca-se o provisionamento de serviços, o protocolo permite a transmissão de estruturas de dados que podem ser implementadas de maneira direta em banco de dados e serviços de nuvem, e a confiabilidade, é estabelecido uma conexão persistente entre servidor e cliente.

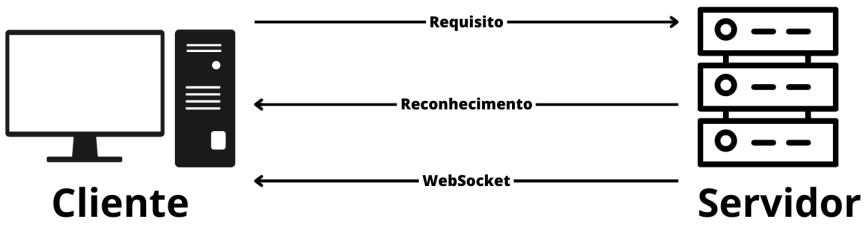
Devido a essa estruturação do protocolo, há uma perda de performance de rede, o que pode gerar uma latência, deixando o protocolo impróprio para aplicações em tempo real. Em geral o cabeçalho das mensagens HTTP é grande comparado com outros protocolos devido aos serviços de transmissão, segurança, detecção de erros.

Com o avanço e melhoria das tecnologias de rede, o protocolo HTTP se mostra uma opção atrativa para aplicações IoT, devido ao seu suporte e uso extensivo na internet e por já possuir funcionalidades de segurança, detecção de erros e privacidade que outros protocolos não. Entretanto quanto a redes mais restritas, ou aplicações que exigem respostas em tempo real, outros protocolos são uma opção melhor devido a uma maior eficiência de performance (YOKOTANI, 2016).

2.2.5 WebSocket

WebSocket é um protocolo que provê eficiência, comunicação confiável entre servidores e clientes. Reduz a sobrecarga da comunicação é foi desenvolvido para comunicação full-duplex em tempo real. O funcionamento do protocolo estabelece uma conexão persistente entre o cliente e o servidor, permitindo que ambas as partes enviem dados um ao outro a qualquer momento, sem a necessidade de repetidas solicitações HTTP. A conexão WebSocket começa com um reconhecimento HTTP, onde o cliente envia uma solicitação HTTP GET com o cabeçalho *"Upgrade: websocket"* para o servidor. Se o servidor for capaz de atender a uma conexão WebSocket, ele responde com um código de status HTTP 101, indicando que a conexão foi atualizada para o protocolo WebSocket. Uma vez que a conexão WebSocket foi estabelecida, tanto o cliente quanto o servidor podem enviar quadros de dados quando desejarem. Cada quadro de dados consiste em um pequeno cabeçalho (2 a 14 bytes) que contém um **opcode**, tamanho da carga útil e bit de mascaramento, seguido por uma carga útil de comprimento arbitrário. O **opcode** indica se o quadro contém dados de texto ou binários, enquanto o bit de mascaramento é usado para evitar certos tipos de ataques (PIMENTEL, 2012).

Figura 6 – Diagrama representativo do protocolo WebSocket



Fonte: Próprio autor

Uma das principais vantagens do WebSocket em relação às soluções baseadas em HTTP tradicionais é que ele permite comunicação bidirecional em tempo real entre clientes e servidores sem exigir solicitações ou *polling* repetidos. Isso o torna ideal para aplicativos como salas de chat, jogos online e plataformas de negociação financeira, onde atualizações em tempo real são críticas (MURLEY, 2021).

O protocolo é uma excelente opção para dispositivos IoT (MURLEY, 2021), pois frequentemente precisam enviar e receber dados em tempo real, como leituras de sensores ou sinais de controle. A baixa sobrecarga e a comunicação full duplex do WebSocket tornam-no um protocolo ideal para esses tipos de aplicativos. Além disso, a capacidade do WebSocket de manter uma conexão persistente entre o cliente e o servidor reduz a necessidade de solicitações repetidas ou polling, o que pode ajudar a preservar a vida útil da bateria em dispositivos IoT.

2.3 Segurança em sistemas IoT

A segurança de sistemas IoT é de importância essencial devido ao potencial de ataques maliciosos que podem se espalhar e serem realizados do mundo virtual para o mundo físico. O escopo da segurança inclui tarefas como sensoriamento confiável, computação, comunicação, privacidade e exclusão digital. (XU, 2014).

A segurança dos sistemas IoT apresenta diversos desafios devido ao amplo escopo de requisitos de segurança. Alguns dos desafios segundo (XU, 2014), incluem:

- Desafios únicos de segurança apresentados por sensores e atuadores: Sensores e atuadores são componentes comuns de dispositivos IoT, mas apresentam vários desafios únicos de segurança. Por exemplo, um atacante pode tentar manipular leituras de sensores ou saídas de atuadores para causar danos.

- Proteção do hardware, software e dados que considera a possibilidade de acesso físico aos dispositivos IoT: O acesso físico a um dispositivo IoT pode permitir que um atacante extraia informações sensíveis ou modifique seu comportamento. Portanto, é importante projetar sistemas IoT com segurança desde o nível do hardware.

- Ataques sistemáticos durante o processamento de dados coletados: Depois que os dados são coletados por um dispositivo IoT, eles precisam ser processados para extrair informações úteis. No entanto, os atacantes podem tentar manipular esses dados para causar danos ou obter acesso não autorizado.

Para aumentar o nível de segurança frente a esses desafios algumas novas técnicas podem ser implementadas (XU, 2014). Para garantir o sensoriamento confiável, dispositivos IoT podem usar mecanismos de inicialização segura para verificar a integridade do firmware e software. Protocolos de comunicação seguros, como SSL/TLS, podem ser usados para proteger os dados em trânsito. A privacidade pode ser protegida usando técnicas de criptografia e anonimização. Técnicas como fusão de sensores podem ser usadas para combinar dados de vários sensores para detectar anomalias ou ataques. Detecção de anomalias e algoritmos de aprendizado de máquina podem ser usadas para detectar ataques sistemáticos em dados coletados.

Em (BAGAA, 2021) é proposto um framework de segurança de IA unificado para sistemas IoT que possa monitorar, detectar e prevenir ameaças de segurança cibernética de forma automatizada, autônoma e harmonizada. O framework explora técnicas de aprendizado de máquina para lidar com detecção de intrusão por meio de reconhecimento de padrões e assinaturas de rede e detecção de intrusão baseada em anomalias, com base em desvios do comportamento normal dos dispositivos. Demonstrando como técnicas de inteligência artificial têm sido usadas para aumentar a segurança dos sistemas em geral, e especificamente nos de internet das coisas.

2.4 Uso de API em sistemas IoT

API (*Application Programming Interface*) são interfaces de programação de aplicativos que permitem que diferentes sistemas e aplicativos se comuniquem e compartilhem dados entre si. Uma API define um conjunto de regras e protocolos para a comunicação entre as aplicações. As APIs são amplamente usadas para integrar dispositivos IoT (Internet of Things) com outros sistemas e aplicativos. Elas permitem que os dispositivos IoT sejam controlados e monitorados remotamente, bem como que eles compartilhem dados com outros sistemas e aplicativos.
Dentro do universo de API, existem muitas classes com diferentes objetivos, entretanto para a comunicação WEB, utiliza-se as API's REST *(Representational State Transfer)*, onde os dados são normalmente transmitidos e recebidos através de solicitações HTTP, como GET, POST, PUT e DELETE, e representados em formato de mensagem, como JSON ou XML. O objetivo é permitir que diferentes aplicativos e sistemas possam acessar e manipular dados de maneira fácil, eficiente e escalável. As APIs REST podem lidar com grandes quantidades de dados, tornando-as ideais para a coleta e integração de dados de dispositivos IoT.
Em (GARG, 2019) é demonstrado o uso de API REST como autentificador de aplicações IOT, e operando como um dispositivo de segurança para o sistema. Já em (FERREIRA, 2013) uma API REST é usada para gerenciamento de eventos e controle de dispositivos elétricos. Em (PASHA, 2016) a API fornece a conexão entre os dados coletados para o softwere MatLab, que irá tratar e criar representações gráficas para esses dados. Há ainda outros tantos exemplos de uso dessas API's em sistemas IOT, demonstrando a capacidade de desenvolver variádas aplicações utilizando as ferramentas de IOT integradas com REST API's.

2.5 Banco de dados para sistemas IoT

Bancos de dados são estruturas de coleções de dados. O sistema que opera as transições de dados como coletar, agrupar, filtrar e deletar esses dados (AZAD, 2019). Os bancos de dados são classificado de duas principais categorias, SQL (*Structured Query Language*) e NoSQL (*No Structured Query Language*), na primeira o armazenamento de dados segue um modelo relacional, os dados são armazenados de maneira tabular, em linhas e colunas, já a segunda não segue um modelo relacional, e suporta outros esquemas de armazenamento de dados, e outras formas de estruturas, como objetos, árvores, filas e etc.

Sob a perspectiva de desenvolvimento IOT (ASIMINIDIS, 2018), a escolha do banco de dados a ser utilizado, afeta de diversas maneiras o sistema, como a escalabilidade, SQL suporta o aumento de performance em um único nó, já o NoSQL não suporta esse

mesmo aumento de performance em um único nó, mas escala quanto ao número de nós sendo utilizado. Quanto ao tempo de resposta, no SQL quando novos dados são coletados, eles terão que passar um processo de relacionar eles com os dados dentro do banco, já o NoSQL os dados são armazenados em objetos e não precisar estabelecer relações com os novos dados, tornando o tempo de resposta mais curto que o SQL (RAUTMARE, 2016). Outros fatores são considerados na escolha de um banco de dados para IoT, como segurança, facilidade de uso, integração com outros sistemas, desempenho e preço.

2.5.1 Firebase Realtime Database

Firebase é uma plataforma desenvolvida pela Google para desenvolvimento de aplicações, com ferramentas para construir, melhorar e expandir as aplicações (MORONEY, 2017a). Algumas das tecnologias disponíveis dentro do firebase são:

- Autentificação Firebase: Ferramenta que permite a aplicação saber a inidentidade do usuário, e a partir dessa informação, prover dados distintos para o usuário, formas seguras de *login* são necessárias. o Firebase disponibiliza uma API que permite utilizar provedores credenciados, como esquemas de email/senha ou integrar com outros meios de autentificação.

- Banco de dados em tempo real: Um banco de dados No-SQL hospedado na nuvem, prove uma conexão síncrona entre os dispositivos. Sempre que um dado sofre uma alteração dentro do banco de dados, eventos são acionados nos códigos dos clientes permitindo que as aplicações lidem com esse novo dado.

- Armazenamento de nuvem: Aplicações tem requerimentos de armazenar fotos e vídeos, e outros formatos de mídia maiores. O firebase fornece a API para armazenar e controlar os processos de upload e download.

- Hospedagem Firebase: o Firebase fornece um espaço de hospedagem para rodar aplicações web de front-end, como arquivos HTML, CSS e Javascripts.

Existem muitas outras funcionalidades dentro do Firebase, entretanto para sistemas IoT, as citadas são elencadas como as principais. O banco de dados em tempo real (*Firebase Realtime Database*) apresenta vantagens para aplicações IoT, pois o sistema utiliza processamento em tempo real, ou seja lida com as constantes mudanças de estados e gera eventos a partir delas. Diferente dos bancos de dados tradicionais, onde sua funcionalidade é conter dados não afetados pelo tempo, e não aciona eventos para as suas aplicações conectadas (MORONEY, 2017a). Essa maneira de operar é semelhante ao MQTT Broker, que reage a novas mensagens recebidas de um publicador, e as envia aos subscritos, a diferença é que no MQTT Broker é preciso direcionar a mensagens através de protocolos e aplicações até o cliente, já no banco de dados a conexão é direta com o cliente e a troca

de dados é de maneira continua.
A comunicação com o banco de dados é baseada na arquitetura REST (*Representational State Transfer*), ou seja, é feita através de métodos de requisições e respostas, os dados são estruturados em um formato JSON (*JavaScript Object Notation*), um formato de dados leve para transitar na internet, os dados são escritos em pares de nomes e valores, e podem ser estruturados como objetos.

O protocolo utilizado é o Websocket, de comunicação bidirecional full-duplex que permite a comunicação em tempo real entre um cliente e um servidor. Ele foi projetado para fornecer uma alternativa mais eficiente e flexível ao protocolo HTTP para aplicações que requerem atualizações de dados em tempo real (MORONEY, 2017b), como jogos online, aplicativos de mensagens instantâneas, aplicativos de IoT, etc.

2.6 Ciência de dados para sistemas IoT

A ciência de dados é um campo que combina diferentes ciências para extrair conhecimentos a partir de dados. Envolve técnicas como mineração de dados, aprendizado de máquina e análise estatística para encontrar padrões e fazer previsões. No contexto de sistemas IoT, a ciência de dados pode ser usada para analisar as vastas quantidades de dados gerados pelos sensores desses sistemas (HELAL, 2021).

Existem vários desafios envolvidos na implementação da ciência de dados em sistemas IoT. Um dos principais desafios é o volume e a variedade de dados gerados pelos dispositivos IoT, o que pode tornar difícil processar e analisar esses dados de maneira rápida e eficiente.

A variedade de dados é a sua heterogeneidade, ou seja, os diferentes formatos que eles são armazenados ou os diferentes protocolos. O que dificulta a integração e análise dos dados pois diferentes tipos de dados podem exigir diferentes técnicas ou ferramentas de processamento (RANJAN, 2018). As soluções para esse desafio são a padronização no armazenamento e processamento dos dados IoT, pois o uso de protocolos comuns evita uma maior variedade. Outra solução são as técnicas de aprendizado de máquina, como a transferência de aprendizado, podem ser usadas para aproveitar o conhecimento obtido de um tipo de dispositivo ou aplicativo de IoT e aplicá-lo a outros tipos de dispositivos ou aplicativos (RANJAN, 2018).

O volume de dados se refere a grande quantidade de dados gerados por diversas fontes, dispositivos, sensores, rede e etc. Os dados comumente são não estruturado, ou seja, sem um padrão definido, o que requer processamento em tempo real para extração das informações relevantes. Técnicas tradicionais de ciência de dados podem não ser capazes de lidar com essa quantidade de dados, o que pode levar problemas de processamento lento, aumento nos requisitos de armazenamento e redução de precisão dos resultados. Para superar esse desafio, são necessários novos algoritmos distribuídos baseados em aprendi-

zagem leve, online e incremental. Esses algoritmos podem lidar com o grande volume de dados gerados pelos sistemas IoT, processando-os em tempo real e extraindo conhecimentos valiosos. Além disso, técnicas como compressão e filtragem de dados podem ser usadas para reduzir a quantidade de dados que precisam ser processados (HELAL, 2021).

É importante utilizar ciência de dados em sistema IoT, pois esse sistema lida com uma grande quantidade de dados, e esses dados podem ser usados para extrair conhecimentos valiosos que podem ser utilizados para melhorar a funcionalidade dos sistemas IoT e torná-los mais adaptativos (HELAL, 2021). Ajudando a desbloquear o potencial desses sistemas e perceber seus benefícios para a sociedade.

2.7 Sensores e atuadores de grandezas agrometeorológicas

Agricultura de precisão é a abordagem de gerenciamento agrícola que utiliza tecnologias que permitem maior eficiência e precisão no cultivo de culturas. Esse gerenciamento visa uma resposta mais rápida para condições climáticas adversas, um melhor controle de qualidade e um maior nível de automação de maquinários agrícolas. Parar alcançar tais objetivos é necessário um sensoreamento intenso de grandezas agrometeorológicas. Que junto com atuadores permitem ações como identificação de pragas, e o aumento e diminuição da umidade do solo utilizando um sistema de irrigação automático (ROY, 2008).

Para a construção de uma agricultura de precisão alguns fatores devem ser atendidos (ROY, 2008), como a coleta de dados relevantes para o cultivo. A identificação da localização do sensoriamento, como campos de plantio possuem muitas vezes largas extensões a acurácia da localização se torna um fator relevante. A transferência dos dados para estações de controle e decisão. E por fim um sistema de ação e controle baseado nos dados coletados.

A escolha de sensores e atuadores é crucial na construção de um sistema de agricultura de precisão, para tanto deve-se considerar alguns fatores (ROSELIN, 2017):

- Tipo de cultivo: cada tipo de cultivo tem necessidades especificas e por tanto um determinado tipo de sensoriamento.
- Ambiente: condições ambientais, como temperatura, umidade e exposição ao sol, podem afetar a precisão e a eficácia dos sensores. É importante escolher sensores que possam operar de maneira confiável em condições específicas de cultivo.
- Conectividade: os sensores precisam se conectar à rede para enviar e receber dados. É necessário escolher sensores que possam se conectar à rede disponível na área de cultivo.

- Alimentação e eficiência de energia: importante escolher sensores e as suas respectivas fontes de alimentação, como baterias, células solares e outras fontes de energia, sempre visando manter um nível adequado de eficiência energética.
- Custo: escolher sensores que sejam acessíveis e ofereçam um bom custo-benefício para o uso em escala na agricultura, importante ter em mente a manutenção desse sistema, caso os sensores tiverem um custo mais elevado, consequentemente a manutenção também será encarecida.
- Integração: os sensores devem ser facilmente integrados com outras tecnologias, como sistemas de irrigação ou softwares de análise de dados, para obter o máximo de benefício da coleta de dados em tempo real.

2.7.1 Sensores de temperatura e umidade do ar

Os sensores de temperatura e umidade do ar são componentes importantes para medir as condições agrometeorológicas, pois coletam informações ambientais relevantes, esses sensores podem ser instalados em diversas outras aplicações, como em casas inteligentes, fábricas, hospitais, agronegócio. Eles são capazes de medir as condições ambientais em tempo real e transmitir essas informações para dispositivos conectados à internet, permitindo que esses dispositivos tomem decisões mais precisas e eficientes.

O SHT75 é um sensor digital de temperatura e umidade do ar. Ele utiliza um sensor capacitivo de umidade e um termistor de platina para medir a umidade relativa do ar e a temperatura ambiente. possui uma precisão elevada, com uma faixa de medição de umidade de 0 a 100 com uma precisão de ±2 e uma faixa de medição de temperatura de -40°C a 125°C com uma precisão de ±0,3°C. Além disso, ele tem uma resolução de 12 bits para ambas as medições de umidade e temperatura, o que significa que ele pode detectar mudanças muito pequenas na temperatura e umidade. A sua comunicação é por meio do protocolo serial I2C, o que o torna compatível com a maioria dos microcontroladores (DATASHEET SENSORION, 2011).

Figura 7 – Sensor SHT75

Fonte: SENSORION

2.7.2 Sensores de luminosidade

Os sensores de luminosidade desempenham um papel importante na IoT, permitindo que os sistemas inteligentes monitorem e controlem a iluminação de ambientes internos e externos, melhorando a eficiência energética e o conforto do usuário, além de ter diversas outras aplicações em áreas como agricultura, segurança e monitoramento ambiental. Especificamente para a agricultura eles podem ser usados para monitorar a quantidade de luz solar que as plantas recebem, ajudando a determinar a melhor hora para plantar, colher e regar as safras.

O BH1750 é um sensor de luz digital que pode ser usado no contexto da IoT (Internet das Coisas) para medir a intensidade da luz em uma determinada área. Ele é capaz de detectar luz na faixa de 0 a 65535 lux, tornando-o ideal para uma ampla gama de aplicações (ROHM Datasheet).

O BH1750 possui um barramento de comunicação I2C que permite a sua conexão a uma ampla variedade de microcontroladores e sistemas embarcados. Além disso, o sensor também possui uma ampla faixa de temperatura de operação, o que o torna adequado para uso em diferentes condições climáticas (ROHM Datasheet).

Figura 8 – Sensor SHT75

Fonte: ROHM

2.7.3 Módulos GPS

Os geolocalizadores desempenham um papel fundamental nos sistemas agrícolas no contexto de IoT, para otimizar a produção, reduzir custos e minimizar o impacto ambiental. Nesse sentido, os geolocalizadores fornecem informações geoespaciais precisas que são essenciais para melhorar a eficiência e a produtividade nas atividades agrícolas.

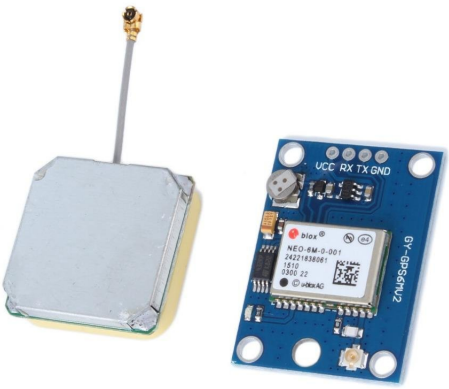
A principal vantagem dos geolocalizadores é a capacidade de rastrear e monitorar com precisão a localização de ativos, equipamentos e recursos agrícolas em tempo real. Isso permite que os agricultores otimizem o uso de máquinas, veículos e implementos agrícolas, garantindo que eles sejam alocados de maneira eficiente nos campos. Além

disso, os geolocalizadores ajudam no planejamento e na execução de tarefas agrícolas, como plantio, irrigação, pulverização de defensivos agrícolas, colheita e transporte.

Nesse contexto os módulos GPS desempenham um papel importante, o módulo GPS NEO-6 é um receptor GPS autônomo que apresenta o motor de posicionamento de alta performance u-blox 6. Ele é compacto, com dimensões de 16 x 12,2 x 2,4 mm, e oferece diversas opções de conectividade. O NEO-6 é ideal para dispositivos móveis alimentados por bateria com restrições rigorosas de custo e espaço (u-blox AG. 2009).

O módulo GPS NEO-6 oferece diversas opções de comunicação, incluindo UART, USB e SPI. Ele pode ser facilmente integrado a sistemas embarcados ou dispositivos móveis para fornecer informações precisas sobre a localização geográfica do usuário ou objeto em questão. (u-blox AG. 2009).

Figura 9 – Módulo GPS NEO-6



Fonte: u-blox AG

2.7.4 NodeMCU

O firmware NodeMCU é uma implementação do Lua, uma linguagem de programação de script leve, que é executada no microcontrolador da placa NodeMCU. É projetado para fornecer uma camada de abstração para o hardware da placa, tornando-a mais fácil de ser programada e utilizada para aplicações IoT. O firmware NodeMCU é código aberto (*open source*). Ele inclui uma série de bibliotecas que permitem acessar e controlar facilmente recursos da placa, como Wi-Fi, entradas e saídas digitais e analógicas, além de interfaces para sensores e outros dispositivos. Além disso, o firmware NodeMCU oferece suporte a uma ampla gama de protocolos e tecnologias IoT, tornando-o uma opção popular para desenvolvedores que desejam criar aplicativos IoT de maneira fácil e eficiente (MAIER, 2017).

O microcontrolador ESP-32 é baseado na tecnologia NodeMCU, um microcontrolador de baixo custo e baixo consumo de energia desenvolvido pela Espressif Systems. Ele é utilizado especialmente para aplicações IoT e vem com uma série de recursos integrados, incluindo Wi-Fi, Bluetooth, memória flash, entradas e saídas digitais e analógicas, além de interfaces para sensores e outros dispositivos, é baseado em um processador dual-core e oferece suporte a várias linguagens de programação, incluindo C, Lua e MicroPython.

Figura 10 – ESP-32

Fonte: Próprio autor

Algumas especificações de fábricante:

- **Processador**: Tensilica Xtensa 32-bit LX6 microprocessor.
- **Frequência de clock**: até 240 MHz.
- **Wifi**: 802.11 (802.11n @ 2.4 GHz até 150 Mbit/s).
- **Bluetooth**: v4.2 BR/EDR and Bluetooth Low Energy (BLE).
- **Memória**: ROM de 448kB; SRAM de 520kB; Embedded flash de até 4Mb e suporta até 16MB de memória externa

Algumas características do ESP-32 fazem dele uma escolha para projetos IoT, dentre elas elenca-se a sua conectividade, com recursos de Wi-Fi e Bluetooth integrados, o que permite conexão rápida e fácil com outros dispositivos IoT e gateways. Seu baixo consumo de energia, importante para sistemas IoT que precisam ser executados por longos períodos de tempo com recursos limitados. Recursos de hardwere, incluindo processadores dual-core, entradas e saídas digitais e analógicas, memória flash, além de interfaces para sensores e outros dispositivos. E seu preço acessível, que o faz adequado para economizar recursos durante o projeto.

3 Desenvolvimento

O desenvolvimento desse sistema foi segmentado em módulos independentes de códigos, e todos foram integrados a partir do banco de dados. Primeiramente foi estruturado o banco de dado Realtime Firebase, depois conectado ao ESP. Então desenvolveu-se o processo de coleta de dados a partir dos sensores conectados ao ESP. Depois a aplicação WEB foi criada e hospedada pelo Firebase, e foi feito a interface do usuário. Por fim uma API de análise de dados foi criada para rodar sobre o mesmo banco de dados. Assim cada projeto é independente do outro, e o sistema IoT se dá a partir da integração de todos.

O sistema ao todo possui mais de 7 algoritmos em três linguagens de programação, C, Javascript e Python. Alguns trechos de códigos, importantes para o entendimento do algoritmo, foram colocados ao longo dessa seção, mas o projeto completo encontra-se no repositório: <https://github.com/mateusperrut/Projeto-de-conclus-o-de-curso>

3.1 Planejamento do projeto

Como abordado, o desenvolvimento de um sistema IoT envolve a integração de diversas tecnologias. Portanto é necessária a escolha adequada dos componentes desse projeto, visando a compatibilidade entre as ferramentas. Outro ponto é a adequação do sistema aos objetivos e requisitos, assim a escolha das tecnologias do sistema também devem atender a esses critérios de projeto.

Portanto a fase de planejamento é crucial no processo de desenvolvimento de um sistema IoT pois nela se define toda a estratégia do projeto, desde a escolha dos componentes, passando pelo desenho da arquitetura do sistema, até a definição das metodologias de desenvolvimento e testes. Essa fase permite antecipar possíveis problemas técnicos e aprimorar a eficiência do sistema, além de garantir que os recursos disponíveis sejam utilizados de forma otimizada.

A primeira definição do projeto são quais os dado serão utilizados no sistema. Como o sistema é para grandezas agrometeorológicas, definiu-se que os dados seriam a temperatura do ambiente, a umidade do ar, as coordenadas de localização, a luminosidade do ambiente, a presença de chuva e umidade do solo. Com a definição dos dados de interesse do sistema a próxima etapa é encontrar os sensores que permitam colher esses dados.

Para escolha dos sensores seguiu-se como critérios fabricantes conhecidos, as documentações técnicas e a adesão do uso desses sensores no mercado. Assim os sensores e os seus respectivos dados de coleta são, o SHT75 que trabalha com a temperatura e umidade do ambiente, o BH1750 que trabalha com a luminosidade do ambiente, o neo-6m que é um módulo GPS operante com sinais de satélite, e por fim sensores resistivos de presença de chuva e umidade.

Com a escolha dos sensores, o próximo passo foi selecionar o microcontrolador mais adequado para o projeto, levando em consideração fatores como capacidade de processamento, memória, conectividade e energia. Nesse caso, optou-se pelo ESP32, que atendeu a todos os requisitos necessários. Além do seu uso intensivo em projetos IoT, o que facilita o desenvolvimento do código devido aos exemplos de projetos e bibliotecas de componentes encontrados online.

O banco de dados utilizado foi o Firebase Realtime Database, que permite que aplicativos IoT armazenem e sincronizem dados com facilidade. Ele foi projetado para permitir que aplicativos atualizem dados em tempo real, de modo que todas as mudanças sejam refletidas imediatamente em todos os dispositivos conectados ao banco de dados. Por se tratar de um banco de dados NoSQL, possui uma maior flexibilidade de armazenado quanto a estrutura de dados e pela característica de operar em tempo real se destaca entre os bancos NoSQL, sendo ideal para o projeto. O Firebase prioriza a disponibilidade do banco em detrimento da consistência, já os bancos como MongoDB priorizam a consistência dos dados em detrimento da disponibilidade. Assim, em aplicações com interação em tempo real com o usuário o Firebase se destaca, já em aplicações onde grandes volumes de dados serão armazenados e exigem processamentos mais longos o MongoDB é a opção mais viável.

Por fim, a escolha da linguagem de programação também foi um ponto importante. Para o desenvolvimento do front end, foi escolhido JavaScript, que é uma linguagem bastante utilizada em aplicações web e que possui diversas bibliotecas e frameworks disponíveis. Já para o back end, optou-se pela utilização do Flask em Python, devido à sua simplicidade e eficiência no desenvolvimento de APIs.

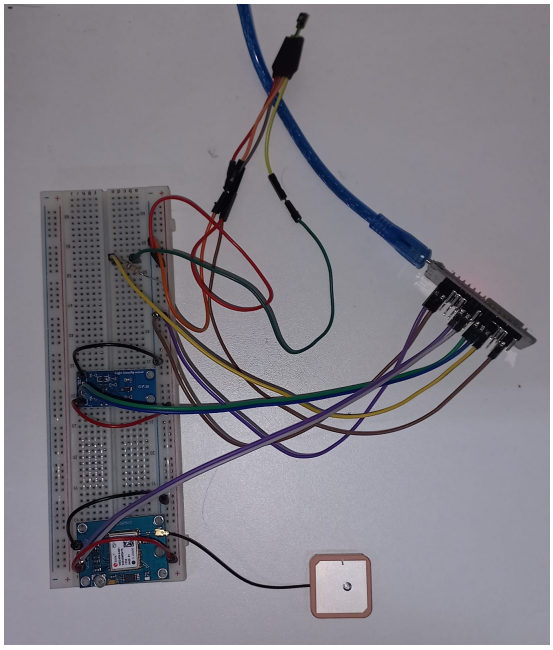
Todas essas decisões foram tomadas durante a fase de planejamento, o que permitiu que o projeto fosse conduzido de forma mais estruturada e com um menor risco de problemas técnicos ao longo do desenvolvimento.

3.2 Sensores e Hardware

Os sensores utilizados nesse trabalho foram conectados ao microcontrolador ESP-32. O sensor de temperatura e umidade SHT75, conectado por 2 pinos, um de dados e um de clock, ambos definidos como pinos digitais 21 e 22, respectivamente mais os pinos de alimentação (VCC e GND), assim a comunicação I2C é estabelecida com o microcontrolador. O sensor de luminosidade BH1750, conecta-se ao ESP32 também por meio do protocolo I2C, com o pino de dados (pino 18) e com o de clock (pino 19). Já módulo GPS, conectado ao ESP32 por meio de uma comunicação serial, utilizando os pinos RX e TX, definidos como pinos digitais 16 e 17, respectivamente.

A seguir a imagem da montagem dos sensores utilizando uma protoboard:

Figura 11 – Montagem sensores



Fonte: Próprio autor

3.3 Firebase Realtime Database

Para a conexão com o Firebase foi criado um projeto na plataforma do firebase.

Foi acessado o console do firebase através do site: *https://console.firebase.google.com*, e adicionado um novo projeto, vinculado com a conta google. O projeto então foi nomeado e criado.

Depois, foi preciso definir os métodos de autentificação, que são usados para verificar a identidade do usuário e fornecer acesso personalizado e seguro ao conteúdo e recursos do aplicativo. Para controlar o acesso a determinadas funcionalidades, personalizar experiências do usuário, proteger dados sensíveis. Dentre os diversos métodos disponíveis o escolhido para esse projeto foi o email/senha, assim toda aplicação que desejar conectar ao projeto terá que ser autentificada por meio desse email e senha definidas nesse processo.

O próximo passo foi a criação de um banco de dados (*Realtime Database*) que será utilizado para armazenar os dados do sistema. Foi escolhido a localização em que o banco será hospedado fisicamente, que para esse projeto é a América Latina.

As regras de segurança do realtime database são definidas de acordo com o nível de acesso aos dados, elas permitem controlar quem tem acesso aos dados e como eles podem ser acessados e garantir que seus dados sejam protegidos e que apenas usuários autorizados possam ler ou escrever neles. Essas regras são baseadas em uma sintaxe de estilo JSON e permitem definir permissões de acesso a diferentes partes do banco de dados. Dentre as regras principais, estão: ***Read***: define quem pode ler dados em uma

determinada localização do banco de dados; ***Write***: define quem pode gravar dados em uma determinada localização do banco de dados; ***Validate***: define as regras de validação para garantir que os dados gravados no banco de dados estejam no formato correto e sejam válidos; ***Auth***: define as regras de autenticação para controlar quem pode acessar o banco de dados e como eles podem autenticar. Para esse projeto foi definida a regra:

```
"rules": {
  ".read": "auth != null",
  ".write": "auth != null" }
```

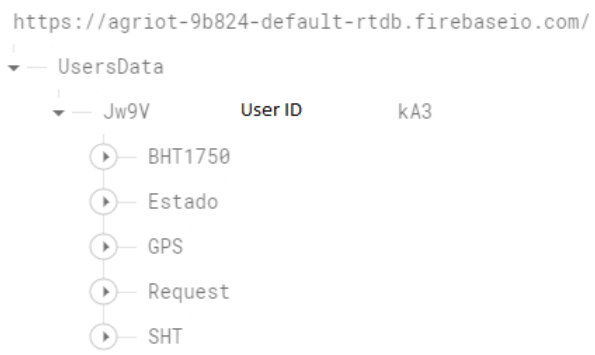
Assim, somente usuários com autentificação podem ler e escrever dados no banco de dados. As regras de validação e autentificação foram as padrões do Firebase.

Para a comunicação com o banco de dados é necessária a URL base, que identifica o banco de dados em tempo real. Fornecendo um ponto de acesso para o banco de dados em tempo real e é usada para se conectar uma aplicação. Seu formato é `https://<project-name>.firebaseio.com/`, esse URL é único para cada projeto do Firebase e deve ser usado em todas as suas interações com o Realtime Database.

Outra informação importante é API Key do Realtime Database, que é uma chave de acesso que identifica e autentica as solicitações feitas ao banco de dados. Quando uma solicitação é feita ao Realtime Database, a API Key é enviada juntamente com a solicitação para autenticar e autorizar a solicitação. Sem a API Key correta, as solicitações não serão processadas pelo banco de dados. Com a URL base e a chave API é possível conectar as aplicações ao banco de dados, para realizar as operações de escrita e leitura nele.

Segue, um exemplo de como o banco de dados ficou estruturado: Após a conclusão da inicialização o diretório de arquivos irá ser gerado na raiz do projeto:

Figura 12 – Estrutura do banco de dados



Fonte: Próprio autor

3.3.1 Conexão Firebase com o ESP32

A conexão do ESP32 ao banco de dados foi feita através da biblioteca aberta e comunitária `Firebase_ESP_Client.h`, que foi criada por um grupo de desenvolvedores independentes e é mantida pela comunidade. Ela tem como objetivo fornecer uma maneira fácil e prática de conectar o ESP ao Realtime Database do Firebase, permitindo o armazenamento e recuperação de dados em tempo real. O repositório git da biblioteca é o https://github.com/mobizt/Firebase-ESP-Client

Para usar-lá, primeiro é feito a instalação da biblioteca na IDE, nesse projeto ela é instalada na extensão PlataformIO, e depois adicionada ao projeto em questão. No arquivo .c devem ser realizadas as seguintes inclusões:

```
#include <Firebase_ESP_Client.h>
#include "addons/TokenHelper.h"
#include "addons/RTDBHelper.h"
```

O primeiro include é a biblioteca em si, com suas funções de autentificação e comunicação, e possui funções que permitem inserir, atualizar e ler dados do banco de dados em tempo real. Já os seguintes são funcionalidades adicionais. A `TokenHelper.h` fornece informações sobre o processo de geração de token, que é um processo importante para a autenticação de um usuário e autorização de acesso. Ele contém funções que ajudam a gerar e verificar tokens de autenticação, que são necessários para acessar o banco de dados. Já o `RTDBHelper.h` fornece informações úteis para imprimir dados do RTDB (`Realtime Database`) de forma organizada e legível, além de outras funções de ajuda para trabalhar com o banco de dados em geral, como definir regras de segurança e gerenciar conexões.

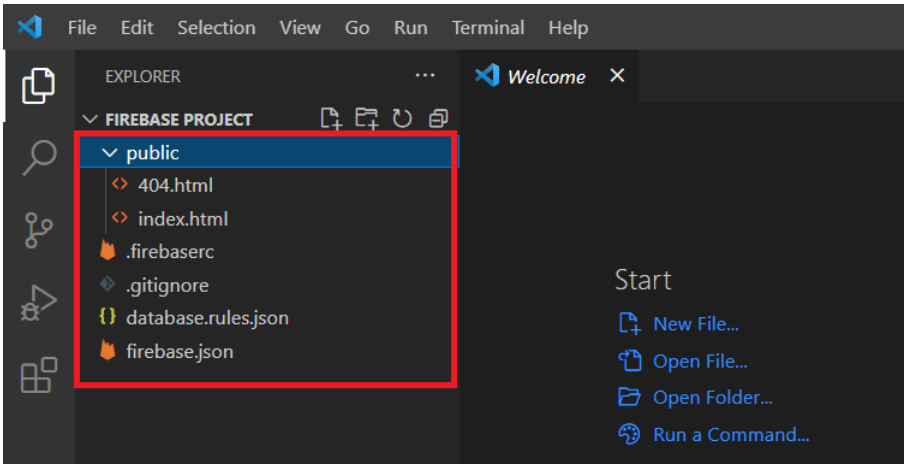
A operação básica consiste em definir os objetos referentes ao banco de dados. No projeto o objeto **`FirebaseData fbdo`** é um objeto usado para armazenar as respostas e resultados das operações do banco de dados, e será utilizado pelos sensores no processo de envio de dados, um documento no formato Json é definido com as novas leituras e transmitido para o banco por meio desse objeto. O objeto **`FirebaseAuth auth`** é usado para autenticar o usuário ao banco de dados. Já o **`FirebaseConfig config`** para configurar a conexão com o banco de dados. Primeiro a chave da API, o URL base e as credenciais de usuário são defenidas no objeto de configuração. Em seguida, a função de login é chamada para registrar um novo usuário no banco de dados. Após o registro do usuário, a função de inicialização é chamada para iniciar a conexão com o banco de dados. Por fim um token de usuário é gerado em seguida, armazena o UID do usuário em uma variável chamada **`uid`**, que remete a *user id*, uma identificação única para separar os dados de acordo com o usuário e isolar o bando de dados, assim os dados de cada usuário é acessível somente à aquele usuário específico.

Com a variável **`uid`** e os objetos definidos anteriormente, é possível ler e escrever no banco de dados utilizando o ESP e os sensores de interesse.

3.3.2 Conexão Firebase com Aplicação WEB

A hospedagem de sites através do Firebase é um serviço oferecido pela plataforma que permite hospedar aplicações web com facilidade e segurança. O processo de hospedagem envolve a configuração da aplicação e sua conexão com o banco de dados. Inicialmente foi instalado o Firebase CLI, uma interface de linha de comando usada para implantar e gerenciar recursos do Firebase. Nesse projeto a CLI foi instalada no Visual Studio, executa-se o comando de login, para entrar na conta Google, depois o comando de inicialização da hospedagem, de configuração e a inicialização é concluída, mais informações desse projeto na documentação oficial do Firebase em https://firebase.google.com/docs/hosting. Após a conclusão da inicialização o diretório de arquivos irá ser gerado na raiz do projeto:

Figura 13 – Diretório projeto Aplicação Web



Fonte: Próprio autor

Aqui a aplicação web está criada, é então possível desenvolver a aplicação e rodar-la localmente, e somente quando validada ser transmitida para o servidor de hospedagem. Essa característica é importante no processo de desenvolvimento pois permite que a aplicação seja testada e validade antes que ela esteja disponível para o público em geral. Essa abordagem também permite que não seja necessário uma conexão para acessar a aplicação, podendo ser acessada no computador local.

A conexão entre a aplicação web e o banco de dados Realtime Database do Firebase foi estabelecida por meio de uma API disponibilizada pelo Firebase. Para conectar a aplicação web ao Realtime Database, é necessário obter as credenciais do Firebase e incluir a API do Firebase na página HTML da aplicação. Com essas informações, a aplicação web pode é autenticada e acessa o banco de dados. Segue o objeto dessa configuração das credenciais:

```
  <script>
const firebaseConfig = {
```

```
    apiKey: "api-key",
    authDomain: "auth-domain",
    databaseURL: "database-url",
    projectId: "project-id",
    storageBucket: "storage-bucket",
    messagingSenderId: "messaging-sender-id",
    appId: "app-id"
  };
```

E por fim, o Firebase SDK é inicializado:

```
        firebase.initializeApp(firebaseConfig);
        const auth = firebase.auth()
        const db = firebase.database();
    </script>
```

Com o SDK do Firebase, quando os dados são alterados em qualquer dispositivo ou no servidor do Firebase, a biblioteca do Firebase SDK detecta a alteração e emite um evento de atualização para a aplicação, assim a aplicação é sincronizada em tempo real com o banco de dados.

3.3.3 Conexão Firebase com API em Python

Para estabelecer a conexão com a API em Python é preciso um certificado de credenciamento do firebase, esse certificado é um arquivo Json, que contém as informações necessárias para autenticar a conexão do aplicativo com o Firebase. Esse arquivo é gerado a partir do console do Firebase e possui uma chave privada criptografada que é usada para autenticar o aplicativo junto ao servidor do Firebase. O uso desse arquivo JSON é uma das formas de garantir a segurança da conexão, uma vez que somente é possível comunicar com o banco de dados com esse certificado, dificultando que aplicações acessem o banco de forma indevida.

Parra obter o certificado é preciso acessar o console do Firebase, escolher o projeto especificado e gerar uma chave privada, baixar esse arquivo e salva-lo no diretório da aplicação. Esse arquivo é passado para a função `credentials.Certificate()`, da biblioteca `firebase_admin`. Por fim o email de usuário é autentificado, e a conexão inicializada como segue:

```
1 cred_obj = credentials.Certificate("C:/caminho para o arquivo/certificado.
    json")
2 firebase_admin.initialize_app(cred_obj, {
3   'databaseURL': 'https://agriot-9b824-default-rtdb.firebaseio.com/'
4 })
5
```

```
email = "email@usuario.com"
user = auth.get_user_by_email(email)
```

3.4 Programação do ESP32

A programação do ESP32 foi desenvolvida utilizando a IDE Visual Studio com a extensão PlatformIO, uma plataforma colaborativa de desenvolvimento embarcado, que oferece um ecossistema unificado de ferramentas para desenvolvimento de firmware, gerenciamento de bibliotecas, integração com plataformas de hardware, depuração e testes. O PlatformIO é uma ferramenta muito útil para projetos IoT, sua documentação oficial encontra-se em <https://platformio.org/>. As configurações do projeto definidas dentro do arquivo `platformio.ini` foram:

```
[env:upstream_develop]
platform = https://github.com/platformio/platform-espressif32.git
board = ttgo-t-beam
framework = arduino
monitor_speed = 115200
```

O campo `{platform}` define a arquitetura do processador e o conjunto de ferramentas de desenvolvimento disponíveis. Nesse caso, está sendo usado o ESP32, e a plataforma escolhida é a **platform-espressif32**. O `{board}` é a placa de desenvolvimento selecionada. Já o `{framework}` é o framework de desenvolvimento escolhido, que define as bibliotecas e as APIs disponíveis para o desenvolvimento. E `{monitor_speed}` é a velocidade de comunicação usada pelo monitor serial.

Como mostrado acima, o framework escolhido foi o arduino, que possui suporte ao desenvolvimento no ESP-32. A maior vantagem de seu uso, é seu suporte de comunidade, o que possibilita o uso de milhares de bibliotecas e módulos já desenvolvidas, acelerando o processo de desenvolvimento. Já ESP-IDF possui um nível de abstração menor, o que permite um código mais personalizado e eficiente, entretanto como o ESP irá somente fazer a leitura dos sensores e enviar os dados para o banco de dados, sem nenhum processamento adjacente, o esforço do uso desse framework não se justifica pelo seu impacto. Ao longo desse projeto, diversas bibliotecas open-source da comunidade arduino foram utilizadas.

Dentre elas, a **`<WiFi.h>`**, biblioteca usada para a conexão Wifi do ESP. No diretório foi criado uma pasta de componente local para o Wi-fi, contendo os arquivos `wifi.h` e `wifi.cpp`, que contêm a função de inicialização ou conexão:

```
void initWiFi() {
  WiFi.begin("SSID", "Senha");
  int timeout_counter = 0;
  Serial.print("Connecting to WiFi ..");
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print('.');
```

```
7       delay(1000);
8       timeout_counter++;
9       if(timeout_counter >= 10){
10      ESP.restart();
11      }
12    }
13    Serial.println(WiFi.localIP());
14    delay(1000);
15  }
```

Aqui quando a função é chamada, as credencias de Wi-fi, SSID e senha, são passadas para a função `Wifi.begin()`, a qual irá autentificar a conexão, um contador é inicializado e um laço aguarda o status de conexão, caso o laço tente 10 vezes e não receba a confirmação de conexão um softwere reset é chamado (linha 10) e o ESP é reiniciado. Essa função irá ser utilizada na `main.c` para a conexão wi-fi.

Após a conexão Wi-fi, é necessário conectar o ESP ao banco de dados do Firebase, o Realtime database, para a conexão, leitura, escrita e manipulação dentro do banco de dados foi utilizado a biblioteca **`<Firebase_ESP_Client.h>`**. Essa biblioteca permite o acesso ao Firebase Realtime Database e ao Firebase Authentication diretamente do ESP32, possibilitando o envio e recebimento de dados em tempo real e a autenticação de usuários. O repositório dessa biblioteca está em <https://github.com/mobizt/Firebase-ESP-Client>.

Dentro do projeto foi criado um componente *FirabaseComponente*, com os arquivos `"FirebaseComponent.h"` e `"FirebaseComponent.cpp"`, que definem a função de inicialização da conexão com o Firebase, e uma função para ler um valor Booleano dentro do banco de dados `estadoUser`, que indica se há um usuário ativo na aplicação ou não, essa informação é importante para definir em que momento o ESP poderá entrar em sono profundo (*deep sleep*).

Função de inicialização:

```
String initFirebase() {
  config.api_key = "chave api do Firebase";
  auth.user.email = "email do usuario";
  auth.user.password = "senha do usuario";
  config.database_url = "Url do banco de dados";

  Firebase.reconnectWiFi(true);

  config.token_status_callback = tokenStatusCallback;
  config.max_token_generation_retry = 5;

  Firebase.begin(&config, &auth);

  while ((auth.token.uid) == "") {
    delay(1000);
  }
```

```
17
18   uid = auth.token.uid.c_str();
19   return uid;
20 }
```

Esse trecho de código tem a função de inicializar a conexão do ESP32 com o banco de dados do Firebase. Ele começa, linha 2 à 5, configurando algumas informações importantes como a chave de API, o URL do banco de dados e o endereço de email e senha do usuário para fazer a autenticação. Em seguida, ele chama a função `Firebase.reconnectWiFi(true)`, linha 7, que faz com que o ESP32 reconecte ao WiFi caso a conexão seja perdida. Depois disso, são configuradas algumas outras opções importantes, linha 9 e 10, como a função de retorno de chamada de status do token e o número máximo de tentativas de geração de token. Por fim, ele chama a função Firebase.begin() para iniciar a conexão com o Firebase. O loop while aguarda a geração do token de autenticação e o armazena na variável uid, que é retornada pela função para que possa ser usada posteriormente em outras funções que precisam acessar o banco de dados. A partir dessa variável uid, o usuário é identificado unicamente dentro do banco de dados, o que garante que ele tera acesso somente aos seus dados e poderá modifica-los.

A função `initFirebase()` é chamada na `main.cpp` para estabelecer a conexão com o banco de dados, outras funções do código irão depender do valor da uid (*user id*) para funcionar corretamente.

Já a função de leitura do estado do usuário:

```
1    bool estadoUser(){
2        estadoPath = "/UsersData/" + String(uid) + "/Estado/ativo";
3        Firebase.RTDB.getBool(&fbdo, estadoPath);
4        return fbdo.boolData();
5    }
```

Primeiramente a função mapeia o caminho para o nó dentro do banco de dados onde está a variável boolena que indica a atividade do usuário da aplicação, depois com o método `getBool()` é realizada a leitura no banco e guarda o respectivo valor dentro do objeto fdbo. Por fim esse valor é retornado. No `main`, essa função é usada para verificar se há um usuário ativo utilizando a aplicação na Web, caso negativo o ESP pode entrar no modo sono profundo.

Após, esses dois componentes, foi desenvolvido um componente para o registro do epcoch timer, que é o número de segundos que passaram desde 1º de janeiro de 1970 (meia-noite UTC/GMT), esse componente tem a função de registar a data exata em que os dados foram lidos, essa característica permite que posteriormente sejam desenvolvidas análises temporais sobre os dados coletados, como registros históricos e projeções.

O componente epoch possui dois arquivos, `epoch.h` e `epoch.cpp`, e a biblioteca utilizada na sua construção foi a **`<time.h`**, que fornece funções para manipular datas e horas em C. O componente tem duas funções, uma de inicialização e uma de aquisição do timer.

```
void epoch_init() {
    const char* ntpServer = "pool.ntp.org";
    configTime(0, 0, ntpServer);
}
```

A função acima é a inicialização, a função `configTime` configura o timer a ser utilizado, e recebe como argumento uma string que especifica o servidor NTP (Network Time Protocol) a ser utilizado para sincronizar o relógio do sistema. Nesse caso, foi utilizado o servidor "pool.ntp.org", que é um servidor público de NTP que fornece a hora precisa a partir de um conjunto de servidores distribuídos geograficamente, os outros parâmetros definem que o fuso horário será obtido automaticamente a partir da conexão com a rede.

```
unsigned long getTime() {
  time_t now;
  struct tm timeinfo;
  if (!getLocalTime(&timeinfo)) {
    return(0);
  }
  time(&now);
  return now;
}
```

Já a função `getTime()` retorna o Epoch do tempo atual, para obter informações de data e hora locais do dispositivo ela armazena essas informações na estrutura `timeinfo`. Se `getLocalTime()` retornar falso, indicando que não foi possível obter informações de data e hora, a função retorna 0. Caso contrário, a função chama time() para obter o tempo atual em segundos.

Essa função será chama sempre que uma nova leitura de dados for feita pelos sensores, dessa forma os dados enviados para o banco de dados terão um atributo **timestamp**, que indica em que momento aquele dado foi adquirido, para posterior aplicação de ciência de dados esse atributo é fundamental, como não mais de um dado pode ser solicitado por segundo nessa aplicação, esse atributo exerce a função de chave primária dentro de um banco NoSQL, pois identifica unicamente aquele documento.

Nesse projeto, para cada sensor usado foi desenvolvido um componente com arquivos .h e .cpp. Para o SHT75 foi utilizada a biblioteca **`<SHT1x-ESP.h>`** que permite a leitura dos dados de sensores de temperatura e umidade SHT1x, e fornece funções para inicializar o sensor, ler os valores de temperatura e umidade e calcular os valores corretos com base nas constantes de calibração do sensor. Também inclui funções para configurar o pino de dados e o pino de clock usados para se comunicar com o sensor. Seu diretório está em <https://github.com/beegee-tokyo/SHT1x-ESP>.

O arquivo sht75.cpp é composto primeiramente por:

```
String SHT_Path;
String STH_tempcPath = "/TemperaturaC";
```

```
    String STH_tempfPath = "/TemperaturaF";
    String STH_umidPath = "/umidade";
    String STH_timePath = "/timestamp";
    String Request_SHT_Path;
    String parentePathSHT;

    FirebaseJson jsonSHT;

    int timestampSHT;

    #define dataPin  21
    #define clockPin 22
    SHT1x sht1x(dataPin, clockPin, SHT1x::Voltage::DC_3_3v);
```

Onde são definidos as variáveis globais desse componente, os caminhos para os nós atributos do banco de dados, para o mapeamento dos dados referentes a temperatura, umidade e tempo epoch, ainda um objeto json é definido, esse objeto serve para estruturar o dado que será enviado, uma variável inteira é definida para guarda o tempo epoch, e ainda os pinos de dados e clock, e sua inicialização através da estrutura sht1x();

A função de inicialização é definida como:

```
void initSHT75(String uid){
  SHT_Path = "/UsersData/" + String(uid) + "/SHT";
  Request_SHT_Path = "/UsersData/" + String(uid) + "/Request/SHT_request";
}
```

A função recebe um parâmetro de string chamado uid. Ela é responsável por inicializar as variáveis `SHT_Path` e `Request_SHT_Path` usadas para armazenar e acessar os dados coletados pelo sensor SHT75. A primeira é usada para armazenar o caminho do nó Firebase Realtime Database onde os dados do sensor serão salvos, criando um caminho único para incluir os dados de sensores SHT. Já a segunda, é usada para verificar se há uma solicitação de dados do sensor SHT, o valor booleano armazenado nesse nó é usado para determinar se uma solicitação foi feita para enviar os dados do sensor SHT para o Firebase Realtime Database.

A função responsável por verificar as requisições de novas leituras pelo banco de dados é definida como:

```
    bool Request_SHT75(){
      Firebase.RTDB.getBool(&fbdo, Request_SHT_Path);
      bool aux = fbdo.boolData();
      Firebase.RTDB.setBool(&fbdo, Request_SHT_Path, false);
      return aux;
    }
```

Essa função obtém a leitura da variável boolena no nó Request SHT dentro do banco de dados, se esse valor for positivo significa que uma nova leitura foi solicitada por outra

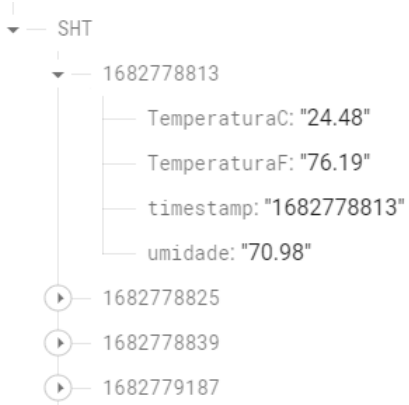
aplicação, função salva esse valor em uma variável auxiliar, seta seu valor para False, pois já leu essa requisição e retorna o valor do nó. No `main.cpp` ela será útil para a interação entre a aplicação WEB e o microcontrolador, pois realiza a leitura de uma informação proveniente da do web app e usa-a no ESP.

Já a função de aquisição de dados e envio é definida conforme:

```
1   void Resposta_SHT75(){
2     timestampSHT = getTime();
3     parentePathSHT= SHT_Path + "/" + String(timestampSHT);
4
5     jsonSHT.set(STH_tempcPath.c_str(), String(sht1x.readTemperatureC()));
6     jsonSHT.set(STH_tempfPath.c_str(), String(sht1x.readTemperatureF()));
7     jsonSHT.set(STH_umidPath.c_str(), String(sht1x.readHumidity()));
8     jsonSHT.set(STH_timePath.c_str(), String(timestampSHT));
9     Serial.printf("Envia json... %s\n", Firebase.RTDB.setJSON(&fbdo,
    parentePathSHT.c_str(), &jsonSHT) ? "ok" : fbdo.errorReason().c_str());
10  }
```

Primeiramente é obtido o tempo atual através da função getTime(), como visto anteriormente no componente Epoch. O caminho é concatenado com o valor do timestamp para identificação única dentro do banco. Depois o documento Json é estruturado com as leituras do sensores e por fim, enviado para o banco de dados através do método `setJson()`.

Figura 14 – Leitura SHT no banco de dados

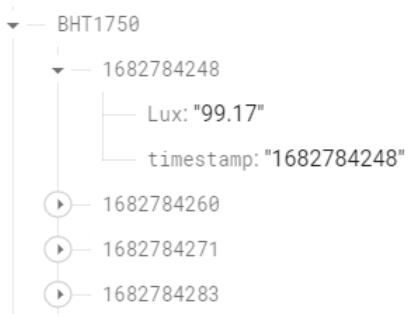


Fonte: Própio autor

Para o sensor BH1750 e para o módulo GPS foram desenvolvidos componentes com funções similares as do SHT75, de inicialização, leitura do requisito no banco de dados, aquisição e envio de dados. Para o BH1750 foram utilizadas as bibliotecas `<Wire.h>` para a conexão I2C, e `<BH1750.h>` para a leitura do sensor de luminosidade, seu diretório está em <https://github.com/claws/BH1750>. Já o módulo GPS utilizou a biblioteca `"SoftwareSerial.h"` para criar uma porta serial de software, e a `<TinyGPSPlus.h>` para

processar dados de GPS definida em <https://github.com/mikalhart/TinyGPSPlus>. A seguir como os dados dos respectivos sensores ficaram estruturados no banco de dados.

Figura 15 – Leitura do BH1750 no banco de dados



Fonte: Própio autor

Assim, todos os componentes foram estruturados para o projeto. Entretanto um desafio apresentou-se na concepção da `main.c`. Para aplicações IOT é importante ter a eficiência e economia de energia como diretrizes do projeto, nesse sentido é muito comum o uso da técnica sono profundo (*deep sleep*), um recurso que permite que o dispositivo entre em um estado de baixo consumo de energia, consumindo uma quantidade mínima de corrente elétrica, e a maioria de suas funções é desativada, incluindo a CPU, os periféricos, o relógio e a memória. O microcontrolador só é ativado novamente quando uma interrupção externa, como um sinal de entrada ou uma contagem de tempo programada, é detectada.

Entretanto nesse projeto, um maior nível de interatividade foi desenvolvido, de modo que o usuário pode solicitar dados em tempo real a partir da aplicação WEB, essa característica contrasta com a diretriz de economia de energia, assim manter a interatividade em tempo real e utilizar o recurso de baixo consumo de energia não podem ser simultaneamente. Para lidar com isso, considerou-se que a maior parte do tempo a aplicação irá coletar dados e enviar para o banco de dados sem nenhuma interação do usuário, e que a interatividade com o usuário não é o operacional da aplicação. Então, o ESP está normalmente em modo sono profundo, acordando periodicamente e enviando os dados para o banco. Entretanto quando um usuário logar na aplicação Web uma variável booleana é enviada para o banco, e através da função `estadoUser` o algoritmo identifica esse evento, e impede que o ESP entre em sono profundo, mantendo a funcionalidade de resposta em tempo real, e volta para o sono profundo somente quando o usuário sair da aplicação Web. Portanto os dois requisitos de projeto foram satisfeitos, o modo de economia de energia quando nenhum usuário está utilizando o app, e modo em tempo real quando há.

Por fim, a main ficou:

```
1    #include <Arduino.h>
2 #include "WiFi/WiFi.h"
3 #include "FirebaseComponent/FirebaseComponent.h"
```

```
#include "Epoch/Epoch.h"
#include "SHT75/sht75.h"
#include "BH1750/bh1750.h"
#include "GPS/gps.h"

FirebaseData fbdo;

void setup(){
  Serial.begin(115200);
  initWiFi();
  String user = initFirebase();
  initSHT75(user);
  initBH1750(user);
  initGPS(user);
  epoch_init();

  if(!estadoUser()){
    Resposta_SHT75();
    Resposta_BH1750();
    Resposta_GPS();
    esp_sleep_enable_timer_wakeup( 5 * 60 * 1000000);
    esp_deep_sleep_start();
  }
}
void loop(){
  if(Request_SHT75()){
    Resposta_SHT75();
  }

  if(Request_BH1750()){
    Resposta_BH1750();
  }

  if(Request_GPS()){
    Resposta_GPS();
  }

  if(!estadoUser()){
    esp_sleep_enable_timer_wakeup(5 * 60 * 1000000);
    esp_deep_sleep_start();
  }
}
```

Inicialmente foi feito a inclusão de todos os componentes desenvolvidos anteriormente. Dentro da função `setup()`, foi realizado sequencialmente, a inicialização da comunicação serial, a conexão Wi-fi, a inicialização da instância do Firebase e obtenção do ID de usuáriuo, a inicialização do sensores SHT17, BH1750 e GPS e do módulo Epoch. Após

as inicializações, ocorreu a verificação do estado do usuário. Se o usuário estiver ativo, o ESP permanecerá no loop principal do código para continuar enviando dados para o Firebase, e entrará no `void loop()`, onde verifica continuamente se há uma solicitação de leitura de sensores do Firebase. Se o usuário estiver inativo, o dispositivo realizará uma leitura de todos os sensores, enviará os dados para o Firebase e entrará no modo de sono profundo por 5 minutos, usando a função `esp_deep_sleep_start()`. Ainda no loop há a verificação se o usuário continua ativo, caso negativo o ESP entra em sono profundo e sai do loop.

A lógica desenvolvida está conforme os objetivos do projeto, um sistema IOT para o monitoramento de grandezas agrometeorológicas, entre as principais características destaca-se a marcação temporal na aquisição de dados, a qual permite analises temporais dos dados e um monitoramento mais elaborado, com dados históricos para a tomada de decisão. Ainda o monitoramento imediato ou em tempo real, possibilitado pela aplicação WEB e incorporado pelo algoritmo no ESP.

3.5 Desenvolvimento da API em Python

Outro processo de desenvolvimento nesse trabalho, foi uma API em Python para análises históricas dos dados coletados. Os dados são lidos pelos sensores e armazenados no banco de dados NoSQL, a API tem como objetivo então conectar-se ao banco de dados, transformar os dados não estruturados em estruturados, de maneira tabular através de um DataFrame, a partir desse dado tabular plotar gráficos que representem os comportamentos dos dados.

3.5.1 Pré-processamento dos dados

Antes de desenvolver a API, foi necessário avaliar a realidade dos dados coletados, que não eram suficientes para uma análise histórica mais robusta, muito pelo período de coleta, o sistema deveria coletar dados de maneira ininterrupta por meses para estabelecer dados suficientes. Assim foi necessário popular o banco com dados históricos anteriores a realização do projeto, para permitir uma análise mais precisa. E o mesmo algoritmo utilizado para os dados históricos é utilizado para os novos dados coletados, o que garante a uniformidade na análise. Portanto justifica-se a utilização de dados externos para a conclusão desse trabalho, pois é uma maneira de garantir a precisão e confiabilidade das análises realizadas.

A aquisição desses dados foi feita a partir do *History Bulk* encontrado em <https://openweathermap.org/history-bulk>, que é um produto que permite extrair dados históricos de clima por hora por mais de 40 anos para qualquer local ou coordenada escolhida, escolhe-se então o período dos dados, as coordenadas de localização, e o formato de re-

torno dos dados. Nesse projeto foi escolhido dados do dia 01/01/2023 à 30/04/2023, de temperatura e umidade, no formato CSV e JSON. O arquivo gerado tem em torno de 7Gb de memória, com 2880 instâncias de dados, a seguir um exemplo de uma instância:


```
{"dt":1672531200,"dt_iso":"2023-01-01 00:00:00 +0000 UTC",
"timezone":-10800,"main":{"temp":24.65,"humidity":84},
"lat":-23.31973,"city_name":"Londrina","lon":-51.166201}
```


Primeiramente, o algoritmo conectou-se ao Realtime Firebase, e em seguida abriu o arquivo contendo os dados históricos Json utilizando a função open da biblioteca padrão do Python para abrir o arquivo em modo leitura, e em seguida carregou-se os dados em um objeto em Python, que pode ser facilmente manipulado pelo código:

```
with open('dado_hist.json') as f:
    json_data = f.read()
data_hist = json.loads(json_data)
```

Entretanto as instâncias de dados contém atributos que não são necessários para o banco de dados, então criou-se um novo objeto `data` que irá conter somente os dados de interesse: a temperatura, a umidade e o timestamp, através de um laço foi estruturado o novo objeto, filtrando somente esses três atributos simplificando o processamento, processo importante quando se trabalha com um grande volume de dados.

```
data = []

for data_point in data_hist:
    temperatura = data_point['main']['temp']
    umidade = data_point['main']['umidade']
    timestamp = data_point['dt']

    data.append({"timestamp": timestamp, "temperatura": temperatura, "
    umidade": umidade})
```

Agora o objeto contém somente os atributos de interesse para o banco, entretanto ao analisar os dados, percebe-se que seu período é a cada uma hora um novo dado, o que contrasta com o período de aquisição de dados feito pelos sensores, a cada 5 minutos. Trabalhar com períodos de dados discrepantes torna o algoritmo de análise de dados ineficiente e pode ser uma fonte de problemas. Pensando-se nisso foi desenvolvido uma interpolação de dados, uma técnica útil para preencher lacunas nos dados, ajudando a manter o período de aquisição de dados consistente e a melhorar a qualidade da análise de dados.

A técnica de interpolação desenvolvida foi a interpolação linear, utilizando a média móvel e a distribuição normal. A interpolação linear é uma técnica que estima os valores intermediários entre dois pontos conhecidos, utilizando uma fórmula matemática que de-

fine a relação entre a posição dos pontos e seus valores, a função `np.interp` da biblioteca `numpy` que utiliza seguinte formula:

$$Valor\_intermed = Valor\_inicial + (\frac{\Delta_t}{\Delta_{total}}) * (Valor\_final - Valor\_inicial)$$

Onde Δ_t é a diferença entre o timestamp final e o timestamp do novo ponto, e Δ_{total} é a diferença entre o timestamp final e o timestamp do inicial. E como argumento a função recebe o valor no qual se deseja interpolar, e dois valores de extremidade conhecidos, nesse projeto, como os valores extremos têm entre si uma distancia de 1 hora e deseja-se obter valores a cada 5 minutos, é necessário interpolar 12 novos dados entre as extremidades. A implementação desse algoritmo segue:

```
dado_interp = []

for i in range(len(data)-1):
    timestamp_inicial = data[i]['timestamp']
    timestamp_final = data[i+1]['timestamp']
    temperatura_inicial = data[i]['temperatura']
    temperatura_final = data[i+1]['temperatura']
    umidade_inicial = data[i]['umidade']
    umidade_final = data[i+1]['umidade']

    for j in range(1, 12):
        timestamp = timestamp_inicial + j*300
        temperatura = np.interp(timestamp, [timestamp_inicial,
    timestamp_final], [temperatura_inicial, temperatura_final])
        umidade = np.interp(timestamp, [timestamp_inicial, timestamp_final
    ], [umidade_inicial, umidade_final])
        dado_interp.append({"timestamp": timestamp, "temperatura":
    temperatura, "umidade": umidade})
```

Assim para cada ponto novo adiciona-se 300 ao timestamp, representando os 5 minutos, e um novo dado interpolado para temperatura e umidade é calculado e salvo. Uma vez obtido os dados interpolados é necessário aplicar o processo de suavização, porque a interpolação ajuda a preencher as lacunas, mas pode criar dados artificiais que não refletem a realidade. A suavização ajuda a reduzir o impacto desses valores artificiais, tornando os dados preenchidos mais realistas e confiáveis.

A técnica de suavização aplicada nesse projeto foi a média móvel, na qual um conjunto de pontos é suavizado por meio do cálculo da média aritmética de um subconjunto de pontos, reduzindo suas flutuações de curto prazo, através da substituição de cada valor no sinal pela média dos valores vizinhos, segue o código desse processo:

```
temperatura_suav = [dado_interp[i+j]['temperatura'] for j in range(1, 11)]
umidade_suav = [dado_interp[i+j]['umidade'] for j in range(1, 11)]
temperatura_media = np.mean(temperatura_suav)
```

```
umidade_media = np.mean(umidade_suav)
```

São selecionados 10 pontos de temperatura e umidade interpolados e consecutivos para que seja feita a suavização. Em seguida, é feito o cálculo da média desses valores que serão utilizados para substituir o valor da temperatura e umidade original no ponto central do intervalo selecionado. Esse processo é repetido até que todos os pontos estejam suavizados.

Após a aplicação da média móvel, é realizada uma distribuição normal nos dados suavizados. A distribuição normal é usada para simular variações aleatórias, que podem ocorrer devido a diversas razões, como erros de medição ou flutuações naturais nos dados, assim ao aplica-la,o conjunto de dados torna-se mais realista e próximo do que poderia ser observado na prática e por consequência mais qualificados para algoritmos de análises e previsões.

```
temperatura_std = np.std(temperatura_suav)
umidade_std = np.std(umidade_suav)
for j in range(11):
    temperatura = np.random.normal(temperatura_media, temperatura_std)
    umidade = np.random.normal(umidade_media, umidade_std)
    dado_interp[i+j]['temperatura'] = round(temperatura, 2)
    dado_interp[i+j]['umidade'] = round(umidade, 2)
```

Primeiramente, são calculados os desvios padrões dos dados de temperatura e umidade, em seguida para cada subconjunto de dados interpolados, são gerados valores aleatórios com destruição gaussiana de média e e desvio padrão iguais a média dos subconjuntos suavizados. Depois que esses valores aleatórios são gerados, eles são arredondados para duas casas decimais e armazenados no objeto `dado_interp`.

Após esses processos, os dados históricos estão preparados e com qualidade suficiente para popular o banco de dados, para essa transmissão utiliza-se a biblioteca `firebase_admin` que possuí funcionalidades de transmissão de dados, como segue abaixo:

```
    for d in data:
    ref.child(str(d['timestamp'])).set({
        "TemperaturaC": str(d['temperatura']),
        "timestamp": str(d['timestamp']),
        "umidade": str(d['umidade'])
    })
```

O laço percorre a lista data que contém os dados interpolados, suavizados e normalizados gerados anteriormente pelo algoritmo. Para cada elemento d nessa lista, a função `ref.child` é chamada com o argumento `str(d['timestamp'])`. A função set é chamada na referência retornada e recebe um objeto que contém os valores a serem salvos no banco de dados. Esses valores são uma string que representa a temperatura ("TemperaturaC"), uma string que representa o timestamp ("timestamp") e uma string que representa a umidade ("umidade").

Por conta da interpolação, de inicialmente 2880 instância de dados, adicionando-se 12 pontos entre cada 2 pontos desse conjunto, resultou em cerca de 34.550 instâncias de dados, o que levou cerca de 116 minutos para a transmissão para o banco de dados. Como esse algoritmo é compilado somente uma única vez, esse longo tempo de execução não é um problema, dado o grande volume de dados transmitidos para o banco.

3.5.2 Plotagem dos gráficos

Após a conclusão do processo de preenchimento do banco de dados, foi desenvolvida uma API de ciência de dados, com o objetivo de facilitar a manipulação, agrupamento e visualização de gráficos baseados nos dados armazenados. Através desta API, é possível acessar informações atualizadas, o que torna o processo de análise e tomada de decisões mais eficiente e eficaz. A manipulação dos dados se dá de forma flexível, permitindo que o usuário execute consultas personalizadas e obtenha resultados relevantes para sua necessidade. Os gráficos gerados pela API oferecem uma visualização clara e objetiva dos dados.

Para o desenvolvimento dessa API, foi baseado no framework Flask, documentado em <https://flask.palletsprojects.com/en/2.3.x/>, esse framework permite definir rotas para manipular solicitações HTTP recebidas, bem como definir lógica para retornar respostas. Isso o torna uma ótima opção para criar APIs RESTful. Com ele define-se as rotas de recursos disponíveis e suas ações (GET, POST, PUT e DELETE). Nesse trabalho são utilizadas ações de GET, ou seja a API recebe requisições da aplicação WEB através desse método, e retorna o dado de interesse. Ainda, esse processo foi dividido em 2 arquivos um chamdo `api.py` para lidar com as requisições e respostas via HTTP utilizando o Flask, e o outro `plot.py` para a construção dos gráficos a serem transmitidos utilizando as bibliotecas de ciência de dados.

Segue o `api.py`:

```
app = Flask(__name__)
@app.route('/plots/temp', methods=['GET'])
def graphs_temp():
    args = request.args
    datemin = args.get('datemin')
    datemax = args.get('datemax')
    medida = args.get('medida')
    resolucao = args.get('resolucao')
    bytes_obj = do_plot_temp(datemin, datemax, medida, resolucao)
    return send_file(bytes_obj,
                     download_name='plot.png',
                     mimetype='image/png')

@app.route('/plots/umid', methods=['GET'])
def graphs_umid():
```

```
16      args = request.args
17      datemin = args.get('datemin')
18      datemax = args.get('datemax')
19      bytes_obj = do_plot_umid(datemin, datemax)
20      return send_file(bytes_obj,
21                       download_name='plot.png',
22                       mimetype='image/png')
23
24  if __name__ == '__main__':
25      app.run(debug=True)
```

Esse código possui duas rotas: `/plots/temp` que recebe como parâmetros datemin, datemax, medida e resolucao via query string. Esses parâmetros são usados pela função `do_plot_temp( )` para gerar um gráfico de temperatura e retorna o gráfico em formato de imagem PNG. E a rota `/plots/umid` que recebe como parâmetros datemin e datemax via query string. Esses parâmetros são usados pela função `do_plot_umid( )` para gerar um gráfico da umidade e retorna o gráfico em formato de imagem PNG. Assim, cada rota utiliza uma função específica para gerar o gráfico desejado. Para a chamada da API, basta que a aplicação especifique a rota correta com os respectivos parâmetros de chamada, e recebera como retorno a imagem PNG do gráfico, na interface do usuário esse parâmetros são definidos.

Primeiramente, tem-se a seleção da medida de interesse da temperatura, tendo como opções a temperatura média, a mínima, a máxima, ou todas as medidas em conjunto. Já a resolução define de que modo os dados serão agrupados, e suas opções são 1 hora, 1 dia, 1 semana e 1 mês. Já a data de início e data de fim definem o período em que serão analisados os dados. Ao clicar no botão *Submit*, a requisição RESTful é enviada com os respectivos parâmetros.

As funções geradoras dos gráficos estão definidas no arquivo `plot.py`, o qual é responsável pela leitura dos dados diretamente no banco de dados, e sua manipulação e geração dos gráficos. Para essa tarefa foram utilizadas as bibliotecas de manipulação de dados em Python, `numpy.py` que fornece suporte para computação científica com arrays multidimensionais e matrizes, a `pandas.py` para trabalhar com estruturas relacionais de dados, e `matplotlib.py` que é uma ferramenta de visualização de dados e geração de gráficos.

Primeiramente sobre a função `do_plot_temp`, que plota os dados de temperatura, com os parâmetros `datemin` e `datemax`, é realizado uma consulta (*query*) no banco de dados dentro dessa faixa. Anteriormente todos os dados estavam sendo carregados do banco e posteriormente filtrados com base no período definido, porém notou-se um ganho substancial de processamento ao especificar o período já no processo da consulta, que segue:

```
1 ref = db.reference('/UsersData/'+ str(user.uid)+'/SHT/')
2 query = ref.order_by_child('timestamp').start_at(datemin_timestamp).end_at(
    datemax_timestamp)
```

```
3 data = query.get()
```

Com os dados carregados, se dá o primeiro processo de manipulação, como os dados são de origem de um banco de dados não relacional (*NoSQL*), seu formato é JSON, é importante passa-los para um formato tabular, para a continuidade das análises. Então foram usadas as funções da biblioteca `pandas` para transforma-los em DataFrame. Algumas outras transformações foram feitas, a criação de uma coluna 'data' que transforma o tempo epoch, recebido no formato data, e a coluna 'TemperaturaC' transformada de string para float para facilitar as operações matemáticas além de ter seus valores anomalias (`outliers`) removidos:

```
df = pd.DataFrame.from_dict(data, orient='index')
df["data"] = pd.to_datetime(df["timestamp"], unit='s')
df["TemperaturaC"] = df["TemperaturaC"].astype(float)
df = df.drop(df[df['TemperaturaC'] < 0].index)
```

O próximo processo, é agregação dos dados, que é usada para resumir grandes quantidades de dados de maneira mais gerenciável, por meio uma função que realiza um cálculo estatístico em um conjunto de dados, produzindo um único resultado que representa uma visão geral do conjunto. Aqui os parâmetros de resolução e medida serão utilizados, eles definem a frequência com que os dados serão agrupados e a função estatística aplicada. Após a aplicação dessa função o gráfico é gerado, pois já está conformado no período escolhido, e agrupado conforme os parametros:

```
df = df.groupby(pd.Grouper(key='data', freq=resolucao)).agg({"TemperaturaC"
    : medida}).reset_index()
ax.plot(df["data"], df["TemperaturaC"])
```

Após a geração do gráfico são feitas alguns ajustes da plotagem, como a definição dos limites do eixo x, a definição dos rótulos e das legendas dos eixos. Por fim a imagem do gráfico gerada precisa ser salvar em um buffer python que será retornado da função.

```
buf = io.BytesIO()
fig.savefig(buf, format='png')
buf.seek(0)
return buf
```

Já para o desenvolvimento da função `do_plot_umid`, que plota os dados referentes a umidade, os processos de manipulação de dados são similares aos da função de temperatura, leitura dos dados e transformação tabular. A diferença está na natureza do gráfico, enquanto o da temperatura é uma plotagem linear da variação da temperatura ao longo do tempo, o da umidade é um histograma que agrupa as frequências relativas de acordo com as faixas de umidades definidas. Assim o processo de agregação não é utilizado aqui, pois os dados serão agrupados já na construção do gráfico, segue a definição das faixas de umidade e a geração do gráfico:

```
bins_umid = list(range(0, 110, 10))
```

```
ax.hist(df["umidade"], bins=bins_umid, rwidth=0.9, weights=np.zeros_like(df
    ["umidade"]) + 1. / df["umidade"].size)
ax.set_xlabel("Umidade")
ax.set_ylabel("Frequencia")
```

Assim, as faixas (*bins*) são definidos entre 0 e 100, divida em faixas de 10. A função `ax.hist`, recebe a coluna de dados, as faixas, o tamanho das barras, e os pesos que calcula as frequências relativas de cada faixa. A seguir um exemplo do gráfico de umidade gerado:

3.5.3 Função para baixar os dados

Aproveitou-se a infraestrutura criada para a plotagem dos gráficos para incluir mais uma ferramenta no sistema, o acesso aos dados salvos no banco de dados para download nos formatos csv e json. Então definiu-se a função `download()`, que recebe o tipo de arquivo e o tempo em que os dados serão coletados.

Para o tempo, as opções são de 1 dia, 1 semana, 1 mês. Com o referencial final sendo a data atual, esse parâmetro define a partir de quando os dados serão baixado, esse mecanismo é feito utilizando a variável *qnt* que será subtraída da data hoje, e assim obtém-se o período de aquisição dos dados:

```
    hoje = dt.now()
    hojeTimestamp = str(hoje.timestamp())
    qnt = 1 if tempo == 'd' else 7 if tempo == 'w' else 30
    comeco = hoje - timedelta(days=qnt)
    comecoTimestamp = str(comeco.timestamp())
    ref = db.reference('/UsersData/'+ str(user.uid)+'/SHT/')
    query = ref.order_by_child('timestamp').start_at(comecoTimestamp).
   end_at(hojeTimestamp)
    data = query.get()
```

Após isso é definido o Quadro de dados e formatado as colunas de interesse. Por fim utiliza-se o parâmetro **tipo**, para definir o tipo de arquivo como segue, e em flask uma:

```
    if tipo == 'csv':
        arquivo = df.to_csv(index=False)
        mimetype = 'text/csv'
    elif tipo == 'json':
        arquivo = df.to_json(orient='records')
        mimetype = 'application/json'
    else:
        raise ValueError('Tipo de arquivo invalido')
    return arquivo, mimetype
```

Como mostrado acima, a função retorna o arquivo e o seu tipo, em flask esses dados são utilizados para estruturar o mecanismo de download. Assim sempre que essa função for chamada, ela irá gerar um arquivo e fará o seu download na ponto inicial da solicitação HTTP. A seguir é criado um objeto do tipo Response, que representa a resposta HTTP

que será enviada de volta ao cliente. Nesse objeto, é passado o arquivo de dados e o seu tipo MIME (*Multipart Internet Mail Extension*), que indica o tipo de arquivo que está sendo enviado. Por fim, é adicionado um cabeçalho na resposta para definir o nome do arquivo que será baixado pelo usuário:

```
1     arquivo, formato = plot.download(tipo, tempo)
2     return Response(arquivo,
3                     mimetype=formato,
4                     headers={
5                     'Content-Disposition': f'attachment; filename="dados.{
    tipo}"'
6                     })
```

Após a finalização dessa função, a API foi hospedada em um serviço de nuvem. O critério para a escolha do servidor foi a facilidade do uso, de maneira a manter simples os mecanismos da API. Então. o servidor foi o Google cloud, e a API ficou hospedada no servidor mais próximo, no caso foi o servidor do Rio de Janeiro. A url da API é <https://agriot-9b824.rj.r.appspot.com>, e as requisições são feitas seguindo a arquitetura REST.

3.6 Desenvolvimento do aplicativo web

A aplicação web foi primeiramente hospedada pelo Firebase, o projeto foi vinculado ao Firebase Console e em seguida configurado o domínio do seu site (<https://agriot-9b824.firebaseapp.com/>). Assim por meio da linha de comando dentro do Visual Studio é feito o upload dos arquivos do projeto para o host. Dessa forma é possível fazer alterações no projeto local e realizar o upload para o host somente quando essas alterações foram validadas. Uma vez que o seu site está hospedado no Firebase Hosting, ele é distribuído em vários servidores em todo o mundo para garantir que ele seja entregue rapidamente aos usuários, independentemente de onde eles estejam localizados.

Para a construção da aplicação foram utilizados ferramentas de front-end, especificamente arquivos JavaScript, CSS e HTML. Primeiro o arquivo HTML, index.html é definido na raiz do projeto, esse arquivo define a estrutura e o conteúdo de uma página da web e é responsável por renderizar a página que é exibida no navegador. O arquivo também inclui a configuração do Firebase, que é usado para autenticação de usuários e armazenamento de dados em tempo real.

Uma vez definido o conteúdo da pagina no arquivo HTML, o arquivo CSS, style.css, define o estilo, design e layout do projeto. Aqui, foram definidos as disposições dos elementos na telas, as cores de fundo, fontes, imagens de fundo, aparência dos botões, e a estilização dos formulários de dados de entrada.

Já em JavaScript foram construídos dois scripts para o projeto, um para a autentificação de usuário e outro para a aplicação em si. O de autentificação, auth.js, utiliza o

Firebase Authentication para gerenciar o login e logout de usuários em um website. Já o index.js, implementa a lógica da aplicação, utilizando orientação a objeto, cada objeto é a representação de um sensor, e seus métodos são funções responsáveis por requisitar dados, e mostrar esses dados na interface do usuário.

A interface foi dividida em 6 componentes, onde cada componente possui uma lógica interna de comunicação e leitura. O desafio da construção da interface está no desenvolvimento individual de cada componente e a integração de todos em uma única página. Os componentes tem funções específicas dentro da pagina. Um é para apresentar dados meteorológicos baseados nas coordenadas de GPS. Dois são para mostras os dados das ultimas leituras dos sensores SHT75 e BH1750 respectivamente. Um componente é um dashboard com gráficos referente aos dados. Há também um componente que plota um mapa da localização do sistema. E por fim, um componente que permite baixar os dados armazenado.

A construção de cada componente depende da integração de diversas tecnologias diferentes, como API's de código aberto, API's privadas, API's criadas para esse projeto e fundamentalmente os dados coletados pelos sensores. Para esse projeto foi utilizado Javascript puro, ou seja, sem nenhum fremework. Existem freameworks como React e o Angular que lidam com o desenvolvimento de componentes mais facilmente do que o javascript puro, entretanto o tempo de aprendizagem dessas plataformas é grande, o que impossibilitou seu uso nesse trabalho.

A aplicação consiste 2 estados, um estado para quando um usuário é autentificado e outro para quando não, o estado não autentificado é composto por um formulário de login com a seguinte lógica:

```
1   document.addEventListener("DOMContentLoaded", function(){
2   var UserState;
3   auth.onAuthStateChanged(user => {
4       if (user) {
5         setupUI(user);
6         var uid = user.uid;
7         UserState = db.ref('UsersData/' + uid.toString()+ '/Estado');
8         UserState.set({ativo : true});
9       } else {
10        setupUI();
11      }
12  });
```

Aqui um evento de escuta é gerado, ou seja espera pela mudança do estado de autentificação, e quando essa mudança ocorre ele verifica se o usuário foi gerado, caso sim, a página de conteúdo é carrega, caso contrário a aplicação permanece na página de login. A variável de estado do banco de dados também é modificada aqui quando o usuário entra, e ela é usada para impedir que o ESP entre em sono profundo enquanto o usuário estiver logado na aplicação.

O script auth.js vai escutar as mudanças no status de autenticação de usuários do Firebase. Um usuário é autenticado, quando realiza o login, e esse login de usuário e esse usuário está registrado no Firebase Console do projeto, de forma que para cada usuário um numero de identificação é gerado (*User Id - UID*) e a partir dele a página principal é carrega. Essa característica permite construir diversas aplicações IOT, utilizando essa mesma infraestrutura, alterando somente o conteúdo das páginas principais conforme a necessidade do usuário.

Uma vez finalizado o processo de autentificação, o algoritmo index.js irá remover o formulário de login e carregar os conteúdos da página, bloqueando todos os elementos de estilo do formulário de login:

```
1    if (user) {
2      loginElement.style.display = 'none';
3      contentElement.style.display = 'block';
4      authBarElement.style.display ='block';
5      userDetailsElement.style.display ='block';
6      userDetailsElement.innerHTML = user.email;
```

E por fim há o botão de sair, utilizado para o logout do usuário, assim a aplicação deve retornar para a página inicial do formulário de login, esse evento é acionado pelo click do mouse, e a variável de estado do banco de dados volta para o valor falso, permitindo que o ESP volte ao sono profundo, pois já não há mais nenhum usuário logado na aplicação:

```
1     const logout = document.querySelector('#logout-link');
2    logout.addEventListener('click', (e) => {
3        e.preventDefault();
4        UserState.set({ativo : false});
5        auth.signOut();
6    });
```

3.6.1 API de dados meteorológicos

O componente de dados meteorológicos utiliza dados provenientes do módulo GPS através do ESP e realiza uma chamada para a API, obtendo os dados de satélites.

Foi utilizado a API OpenWeather, uma API que através da longitude e latitude retorna dados meteorológicos de satélite para serem plotados na aplicação. Para tal é necessária uma chave obtida no site oficial da API: <https://openweathermap.org/api>. A lógica desse componente consistem em o módulo GPS através do ESP envia os dados referentes a latitude e longitude para o banco de dados, e o algoritmo em javascript lê esses dados dentro do banco e envia uma requisição para a API passando-os como parâmetros, a API retorna um objeto JSON com os dados meteorológicos, que serão transformados e plotados na interface do usuário. Para a implementação foi definido um objeto *meteorologia* responsável pela lógica do componente, os seus respectivos métodos:

- **readGPS**: método que lê a localização GPS mais recente do usuário no banco de dados.

- **fetchWeather**: método que faz uma solicitação HTTP para o servidor da OpenWeatherMap para buscar informações climáticas com base nas coordenadas fornecidas.

- **displayWeather**: método que recebe as informações climáticas da API da OpenWeatherMap e as exibe em uma página HTML usando o DOM. Ele mostra informações como a cidade, descrição do tempo, temperatura, umidade, velocidade do vento, nascer do sol e pôr do sol.

A implementação dos respectivos métodos seguiu as orientações da documentação oficial da API, encontrada em: <https://openweathermap.org/current>.

O objetivo desse componente é mostrar como uma informação obtida através do módulo GPS no ESP-32, é utilizada como parâmetro para obter-se informações relevantes a partir de uma API.

3.6.2 Plotagem dos dados do sensor SHT75

O componente que plota os dados colhidos pelo sensor SHT75 tem sua lógica baseada na requisição e leitura dos dados. Dentro do banco de dados há uma variável booleana chamada de SHT-Request, sempre que seu valor for verdadeiro, o ESP-32 faz a coleta dos dados do SHT e modifica a variável para falso, portanto para solicitar novos dados basta enviar uma valor verdadeiro para a varíavel SHT-Request, que o ESP irá atender a requisição. O algoritmo possui um objeto chamado **SHT** responsável por controlar a requisição de novos dados do sensor, a leitura, e a plotagem dentro do componente, sendo seus métodos:

- **getData**: método que mapeia a variável booleana *SHT-request* dentro do banco de dados e seta o seu valor para True, criando uma nova requisição de dados, portanto toda vez que esse método é evocado uma nova requisição é enviada para o ESP.

- **readData**: método responsável por ler os dados do SHT enviados ao banco de dados e exibi-los nos componentes HTML, os dados plotados são referentes a temperatura, umidade do ar, e ainda recebe um epoch time enviado pelo ESP no momento da leitura do sensor, assim esse epoch time é convertido e plotado mostrando o momento da ultima leitura do sensor.

O método getData é chamado de duas formas, a primeira por meio de um temporizador javascript, `setInterval(função, tempo)`, o qual recebe como argumento a função a ser chamada e de quanto em quanto tempo. Nesse contexto, a função a ser chamada foi o getData e o tempo 500000 milissegundos, assim a cada 5 minutos o programa chama a

função getData que faz a requisição de leitura do SHT, dessa maneira o ESP recebe essa requisição e realiza 2 leituras por minuto do SHT.

A segunda maneira de chamar a função getData é através do botão atualizar, dentro do componente, que recebeu um gerenciado de eventos javascript, **`addEventListener(evento, função)`**, que recebe o evento que vai gerar a chamada da função, nesse caso o evento é o click e a função é o getData, assim toda vez que o usuário clicar no botão, o ESP irá realizar uma nova leitura do SHT.

Já o método readSHT é chamado uma única vez dentro do algoritmo, pois ele adiciona um obsevador (*listener*) ao nó especificado no banco de dados Firebase usando o método `on('child_added', snapshot => {})`. Esse observador fica ativo e aguardando novas adições de registros no nó especificado. Assim toda vez que o ESP envia uma nova leitura do SHT para o banco de dados, o observador detecta esse evento e o método readSHT atualiza a página HTML com a nova leitura, garantido que os dados plotados dentro na página WEB são os mais recentes coletados pelo ESP.

O objetivo desse componente é plotar os dados atualizados colhidos pelo sensor SHT, e demonstrar a lógica de requisição de leitura através de funções nativas do JavaScript, no caso um temporizador que aciona o ESP a um período definido, e um gerenciador de eventos que através de um click do usuário requisita uma nova leitura no ESP.

3.6.3 Plotagem dos dados do sensor de luminosidade

Semelhantemente ao componente do SHT75, sua lógica é baseada na requisição e leitura dos dados, com uma variável booleana chamada de BHT1750-Request dentro do banco de dados controlando a requisição de novas leituras para o sensor, e assim para solicitar novos dados basta setar o BHT1750-Request para True, que o ESP irá atender a requisição. O algoritmo possui um objeto chamado Lightsensor responsável por controlar a requisição de novos dados do sensor, a leitura, e a plotagem dentro do componente, sendo seus métodos idênticos aos do SHT:

- **getData**: método que mapeia a variável booleana BH1750-request dentro do banco de dados e seta o seu valor para True, criando uma nova requisição de dados.

- **readData**: método responsável por ler os dados do BH1750 enviados ao banco de dados e exibi-los nos componentes HTML.

  A chamada do getData é também feita através das funções **`setInterval(função, tempo)`** e **`addEventListener(evento, função)`**, sendo utilizada para requisitar a leitura do sensor BH1750 a cada 5 minutos, ou a um clique do usuário no botão atualizar do componente. Ainda com o valor medido do sensor uma ação é tomada dentro da interface do usuário, como medir a luminosidade está muito relacionado a presença ou não de luz solar, caso a luminosidade seja maior que o valor de 100lux,

a imagem de fundo da página é ensolarada, e caso seja menor a imagem torna-se noturna, simplesmente mostrando a mesma informação, luminosidade, de uma forma mais interativa, essa troca se dá mudando a fonte url da imagem de fundo.

Assim, o componente da luminosidade tem um grau maior de interação, uma vez que uma ação é tomada com base na sua leitura.

3.6.4 Dashboard dos gráficos de temperatura e umidade

Aqui são apresentados dois gráficos de análises dos dados presentes no banco de dados, o primeiro gráfico é referente a temperatura, um gráfico de linhas onde são plotadas medidas como temperatura mínima, média e máxima em um período escolhido. O segundo gráfico é um histograma referente a umidade, nele são mostradas as frequências nas faixas de umidade. A partir desses gráficos é possível analisar o comportamento da temperatura ao longo do tempo, e as faixas de operações de umidade mais comuns naquele período.

A obtenção desses gráficos é feita através da chamada da API construída em python. Para tal é necessário a definição de alguns parâmetros. Para a escolha da medida de temperatura a ser analisada foi feito uma caixa de seleção com as opções de temperatura mínima, média, máxima ou todas, a ultima opção plota em um mesmo gráfico as três medidas. Para a resolução, que é o parâmetro que define como os dados serão agrupados, do gráfico também foi criado uma caixa de seleção com as opções 1 hora; 1 dia; 1 semana; 1 mês. Por fim foi feito a seleção da data de inicio e fim do período que pretende-se obter os gráficos por meio de uma selação de calendário.

Uma vez selecionado todos os parâmetros, ao clicar no botão sumbmit, a requisição é feita, conforme a seguir:

```
1 if(submit)
2     submit.addEventListener('click', function() {
3     const medida = document.getElementById('medida').value;
4     const resolucao = document.getElementById('resolucao').value;
5     temperaturaGraf.setAttribute('src', "https://agriot-9b824.rj.r.appspot.
    com/plots/temp?datemin="+datmin.value+"&datemax="+datemax.value+"&medida
    ="+medida+"&resolucao="+resolucao);
6     umidadeGraf.setAttribute('src', "https://agriot-9b824.rj.r.appspot.com/
    plots/umid?datemin="+datmin.value+"&datemax="+datemax.value)
7     });
```

O URL, onde a solicitação é feita, é o local onde a API foi hospedada.

3.6.5 Mapa da localização

O componente da localização utiliza dados provenientes do módulo GPS através do ESP e realiza uma chamada para a API do Google Maps, e recebe um mapa referente

àquelas coordenadas. Para a utilização dessa API é necessária uma chave privada obtida em: <https://developers.google.com/maps?hl=pt-br>

Através do objeto *localização*, foram implementados os seguintes métodos:

- **readGPS**: método que lê a localização GPS mais recente do usuário no banco de dados.
- **initMap**: inicializa o mapa com a localização lida, recebe a latitude e longitude como parâmetros e cria um mapa na página HTML através da API do maps.

Portanto, esse componente é responsável por mostras a localização atual do sistema IoT. Por se tratar de uma API privada e com um custo de requisições, a chamada só ocorre quando a página é carregada. Para uma nova localização é preciso atualizar a página web.

3.6.6 Download dos dados

Esse componente permite que o usuário tenha acesso aos dados coletados pelos sensor SHT75. Com duas caixas de seleção. Na primeira é escolhido o período de coleta dos dados, no último mês, ultima semana ou ultimo dia. E a segunda caixa é selecionado o formato de arquivo, podendo ser em csv ou json. Ao clicar no botão download a requisição é feita conforme o código abaixo, e retornado o arquivo de download:

```
1 const download = document.querySelector('#download');
2     if(download)
3     download.addEventListener('click', function() {
4     const tipo = document.getElementById('tipo').value;
5     const periodo =document.getElementById('periodo').value;
6     window.open("https://agriot-9b824.rj.r.appspot.com/plots/download?
    formato="+tipo+'&periodo='+periodo);
7     });
```

A URL da requisição é a mesma da requisição dos gráficos, isso pois são funções diferentes contidas na mesma API.

4 Resultados

O sistema IoT apresenta uma interface de usuário para comunicar as leituras dos sensores e a integração desses dados com outras tecnologias. Como discutido essa interface foi dividida em componentes, os quais possuem funções específicas dentro do sistema. Será apresentado primeiramente esses componentes e por fim os dados e análises feitos na construção desse sistema.

O componente a seguir apresenta os dados meteorológicos obtidos a partir da API Openweather. Essa API recebe as coordenadas do GPS e retorna um json com diversos dados, foram selecionados os dados de interesse para serem mostrados nesse componente.

Figura 16 – Display API Openweather

Fonte: Próprio autor

Esses dados estão de acordo com o sistema, que se trata de grandezas agrometeorológicas, pois representam as condições climáticas gerais da região em que se encontra o sistema. Não são dados específicos para o sistema como os coletados pelos sensores, mas são uteis para extrair-se as condições climáticas, e outras informações não coletadas por sensores nesse sistema como horário do nascer e por do sol e a velocidade do vento. E demonstra como é obtida informações de fontes externas ao sistema por meio de um dado intermediário (coordenadas do GPS).

Os dados provenientes do sensor SHT75, temperatura e umidade do ar, foram exibidos no componente (Figura 17). Uma característica importante desse componente é sua reação ao evento no banco de dados, sempre que um novo dado é armazenado no nó SHT no banco, o componente atualiza o conteúdo das leituras.

Figura 17 – Plot dos dados do sensor SHT75

Fonte: Próprio autor

Como demonstrado acima o componente mostra os dados colhidos pelo sensor, além da ultima atualização, que é o timstamp registrado no microcontrolador no momento da leitura dos dados. A atualização desses é feita de maneira periódica, a cada 5 minutos, ou ao clicar do botão, o tempo de resposta depende primordialmente da latência da conexão de internet, mas em geral fica entre 2s a 10s, e é a soma do tempo de leitura dos sensores mais o tempo de comunicação com o banco de dados mais o tempo de comunicação do banco com a aplicação web. Nesse estado, o ESP não está entrando em sono profundo o que permite o funcionamento desse componente.

Já o dado recebido do sensor BH1750, a iluminância medida em Lux, é exibido no componente da figura 18. A iluminância também é uma grandeza agrometeorológica pois a luz é uma fonte de energia fundamental para o processo de germinação e crescimento de plantas. Sensores mais específicos podem ser utilizados a depender da aplicação, como sensores de infravermelho ou de raios UV e UVA, mas a integração do sensor utilizado já representa a coleta dessa informação (luminosidade) para o sistema.

Figura 18 – Plot dos dados do sensor BH1750

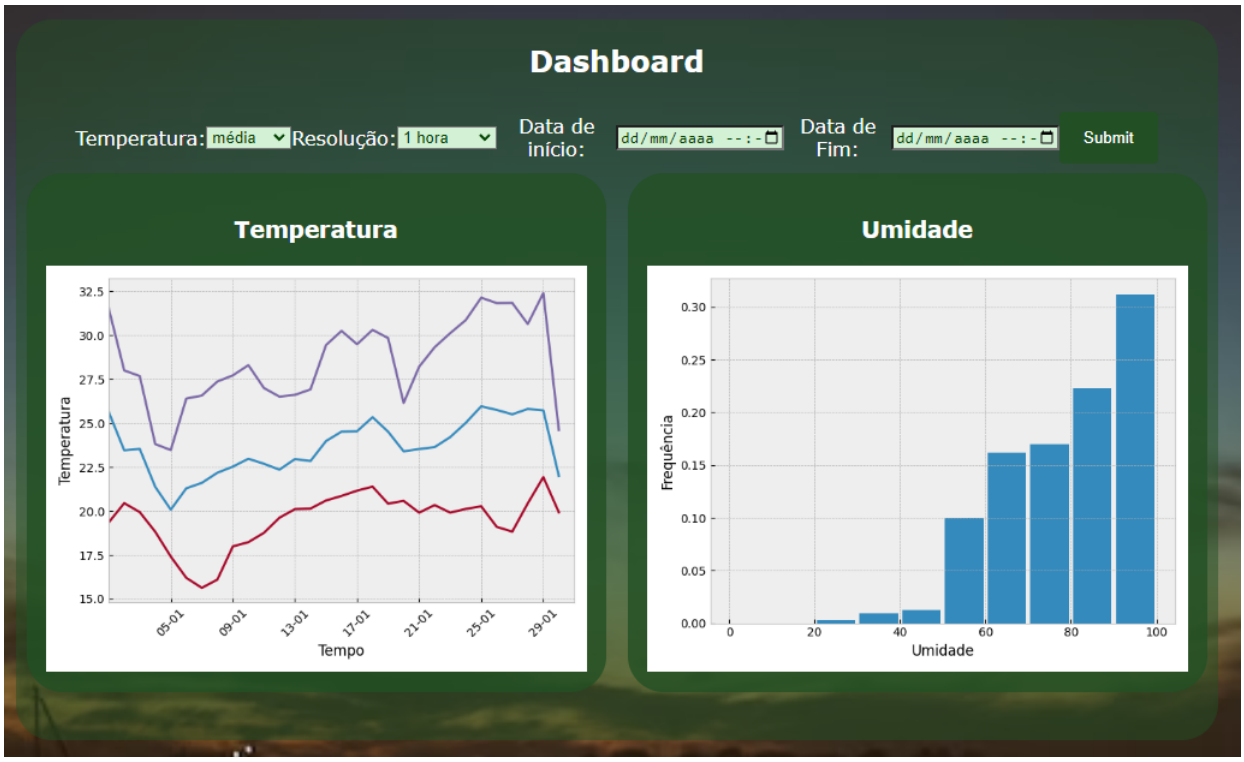
Fonte: Próprio autor

Semelhante ao componente do SHT, novos dados são requisitados a partir do período

definido (5 minutos) ou a partir do clique no botão. Entretanto uma característica foi desenvolvida para esse componente, caso o valor da iluminância seja abaixo de 10 Lux, quantidade de Lux considerada noite, a imagem de fundo da página é trocada por uma noturna. Essa característica comunica visualmente a informação que já anoiteceu. Para sistemas IoT, a capacidade de comunicar informações de maneira indireta é fundamental para garantir a usabilidade do sistema por parte dos usuários.

Até aqui todos os componentes foram de resultados quase imediatos no sistema, entretanto ferramentas de análise histórica dos dados são fundamentais para o sistema IoT. Dado ao fato de que os dados coletados pelos sensores estão sendo armazenados no banco de dados, contruí-se uma API de análise de dados para acessar esses dados no banco e ter como resultados gráficos. Foi feito então um componente de dashboard para a plotagem desses gráficos (Figura 19)

Figura 19 – Dashboard

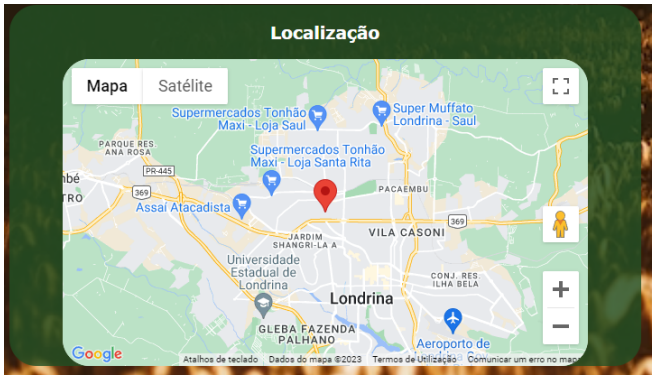


Fonte: Próprio autor

A API é responsiva às configurações de chamada e retornam os gráficos de maneira clara e organizada. Os gráficos são plotados com base nas informações armazenadas, permitindo assim a análise da evolução dos dados ao longo do tempo. Dessa forma, é possível identificar tendências, comportamentos e padrões nos dados coletados, o que pode auxiliar na tomada de decisões e na melhoria do sistema como um todo. A inclusão desse componente no sistema IoT é uma forma de agregar valor e funcionalidades ao sistema, tornando-o mais completo e eficiente para a análise e utilização dos dados coletados.

Como demonstrado a localização proveniente do GPS foi utilizada como um dado intermediário para a aquisição de dados meteorológicos. Mas também foi usada para a exibição de um mapa contendo com a localização do sistema que é uma informação importante para a coleta e análise dos dados agrometeorológicos. A localização pode influenciar diretamente nas condições climáticas e, consequentemente, no comportamento das plantas e culturas. Com a API do Maps, foi construído o componente (Figura 20).

Figura 20 – Mapa

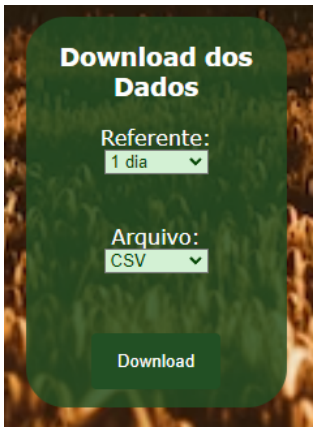


Fonte: Próprio autor

É possível obter informações precisas sobre a localização. Além disso, a visualização do mapa é uma forma clara e intuitiva de representar a localização do dispositivo, facilitando a análise e o monitoramento dos dados coletados.

Apesar do sistema apresentar os dados provenientes dos sensores de diversas maneiras, instantâneas, analíticas e integrado com outras tecnologias. Ainda existem incontáveis possibilidades de analise e integração com outras ferramentas. Não é ambição do sistema comportar todas essas possibilidades de análises, portanto um mecanismo de acesso aos dados foi construído. Através da infraestrutura da API previamente desenvolvida, foi implementado o componente de download dos dados (Figura 21).

Figura 21 – Componente de download dos dados



Fonte: Próprio autor

Através desse componente é possível obter dados referentes até 1 mês anterior, e permite baixar os dados em arquivos do tipo CSV e JSON, pois esses são formatos leves e transitam facilmente pela internet. Tentou-se implementar o download em Excel, entretanto possui uma alta latência no envio desse arquivo, e é facilmente corrompido, como o formato CSV é facilmente convertido para o formato Excel, o esforço torna-se inviável.

Para a analise dos resultados provenientes desse sistema, foi realizado download por meio do componente acima, os dados são referentes a 24 horas de coleta seguidas e interruptas pelo sensor SHT75. O formato que o arquivo foi baixado é o CSV, e posteriormente convertido a uma tabela. Na tabela 1, então, há uma amostra dos 10 primeiros dados dessa tabela:

Tabela 1 – Tabela dos 10 primeiros dados

TemperaturaC	**TemperaturaF**	**timestamp**	**umidade**	**data**
23.14	73.67	1684340563	50.70	17-05 13:22
22.67	72.86	1684340862	52.37	17-05 13:27
23.51	74.36	1684341159	49.29	17-05 13:32
22.85	73.21	1684341457	51.48	17-05 13:37
23.56	74.47	1684341753	49.62	17-05 13:42
23.72	74.74	1684342052	50.27	17-05 13:47
23.63	74.65	1684342351	50.39	17-05 13:52
23.68	74.66	1684342650	50.33	17-05 13:57
22.93	73.31	1684342950	52.50	17-05 14:02
23.30	74.05	1684343249	51.64	17-05 14:07

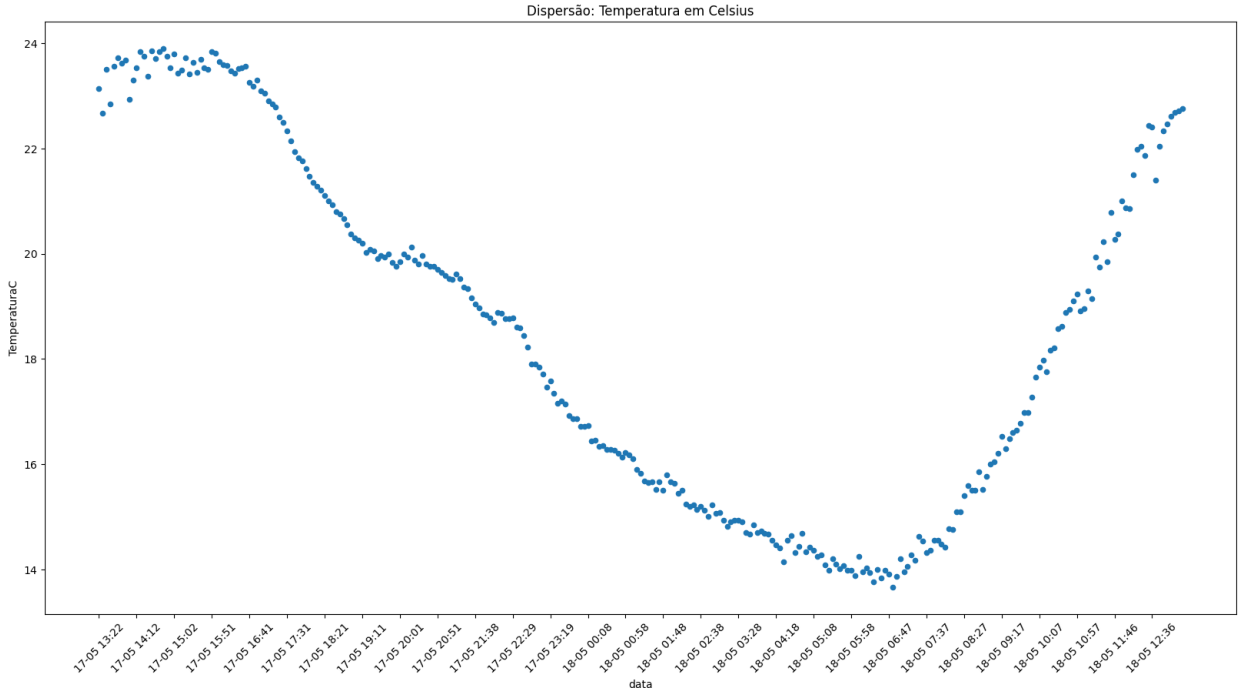
Como esperado a coleta dos dados segue uma periodicidade de 5 minutos, como o tempo de coleta foi 24h a tabela precisa ter no mínimo 288 instâncias. Para extrair medidas resumo desses dados foi aplicado funções em Python, na Tabela 2 há os resultados dessas medidas:

Tabela 2 – Quadro de medidas resumo

	TemperaturaC	**TemperaturaF**	**timestamp**	**umidade**
contagem	289.00000	289.000000	2.890000e+02	289.000000
média	18.46519	65.284221	1.684384e+09	70.102526
desvio padrão	3.31908	5.978780	2.496678e+04	12.770155
min	13.66000	56.660000	1.684341e+09	48.480000
25%	15.23000	59.450000	1.684362e+09	59.300000
50%	18.61000	65.560000	1.684384e+09	69.470000
75%	21.28000	70.340000	1.684405e+09	83.180000
max	23.90000	75.080000	1.684427e+09	87.570000

Assim foram coletados 289 dados, como demonstra o quadro acima, os dados estão coerentes, não houve nenhuma falha de leitura nesse período de leitura, a seguir um gráfico de dispersão desses dados para mostras o comportamento das variáveis coletadas.

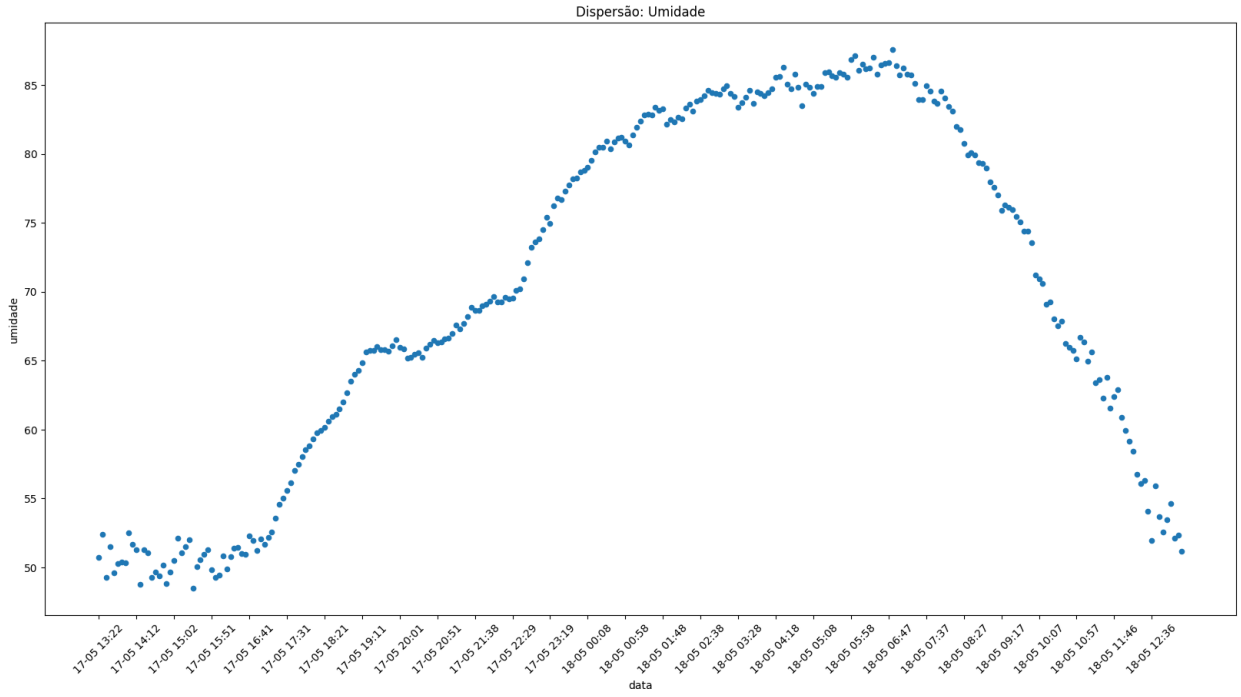
Figura 22 – Dispersão das leituras de temperatura



Fonte: Próprio autor

O Gráfico mostra que o comportamento dos dados de temperatura estão coerentes e não apresentam nenhuma anomalia de leitura, e de processamento, com o sistema entregando dados confiáveis. Já para a umidade:

Figura 23 – Dispersão das leituras de umidade



Fonte: Próprio autor

Novamente, as leituras do sensor de umidade e os processamentos do sistema estão coerentes e não apresentaram nenhuma anomalia nos processos de coleta, armazenamento e processamento dos dados. Mostrando a confiabilidade do sistema, que é a capacidade do sistema em funcionar corretamente e sem falhas ao longo do tempo, mantendo a integridade dos dados.

A segurança do sistema é confiável dada a criptografia e os processos de autentificação em todos os códigos desenvolvidos. O ESP-32 conecta-se somente com um chave de API privada, e com o login de usuário e senha. O desenvolvimento Web que utiliza a hospedagem integrada também precisa da chave API privada e login e senha. Já a API de análise de dados em Python, um certificado criptografado é necessário para estabelecer a conexão com o banco de de dados. E o usuário final somente com autentificação de login e senha, consegue conectar ao sistema.

Para avaliar a usabilidade do sistema, é importante levar em consideração a facilidade de uso, a clareza da interface, a consistência das informações apresentadas, a acessibilidade e a experiência geral do usuário. Para medir de maneira objetiva essas métricas o sistema deveria ter sido implementado e utilizado por diversos usuário de maneira a ter uma amostragem para pesquisas qualitativas sobre esses tópicos. Dado o escopo desse projeto pode-se apenas apontar alguns aspectos da interface de usuário. O componente de login e senha segue o padrão de mercado. Os componentes de plotagem dos gráficos possuem parâmetros para configuração da visualização dos gráficos, essa configuração trás uma maior personalização das análises, entretanto podem perder a intuitividade do usuário. Quanto aos displays dos sensores, todos estão delimitados graficamente e identificadas as origens dos dados. E por fim o mecanismo de baixar os dados permite análises mais específicas.

Já a manutenibilidade possui alguns critérios de avaliação. Primeiramente a modularidade, o sistema ser dividido em módulos distintos que possam ser facilmente substituídos, atualizados ou modificados sem afetar outros componentes do sistema. O trabalho contempla essa característica, pois cada processo de desenvolvimento é independente dos outros, sendo conectados somente através do banco de dados.

Outro critério é a reusabilidade , os componentes do sistema devem ser projetados para serem reutilizáveis em outros projetos e/ou para atender a requisitos futuros. Novamente o critério é atendido, uma vez que todos os componentes do sistema podem ser reutilizáveis em outras aplicações. O banco de dados pode ser integrado a qualquer sistema independente. A aplicação Web é dividida em autentificação e conteúdo o que permite a dado usuário entregar um conteúdo único e personalizado. A API e o ESP exigem adaptação ao contexto inserido mas também são reutilizáveis quanto a conectividade e padronização.

5 Discussões e Conclusões

A partir da fundamentação teórica, foi apontado as principais tecnologias de desenvolvimento IoT. E os processos de construção desse sistema. Desde a camada de percepção, na qual é feita a coleta dos dados através de sensores e um microcontrolador. A camada de rede, onde por meio dos protocolos de comunicação são transmitidos esses dados. A camada de middleware, a qual são feitos os processamentos desses dados e o seu armazenamento. Na camada de aplicação é feita a construção das aplicações Web, mobile ou qualquer outra que comunique as informações do sistema para o usuário. E por fim na camada de negócios, que por meio de análises é gerado um valor para essas informações.

Também são abordadas as tecnologias e ferramentas utilizadas para cada um dos processos. Como os sensores e o microcontrolador, as técnicas de segurança e criptografia, os bancos de dados para esses sistemas e a integração do sistemas com API e técnicas de ciência de dados. O sistema teve como requisitos fundamentais medir grandezas agrometeorológicas, por se tratar de uma aplicação relevante para sistemas IoT, e a partir dos dados coletados com a integração de outras tecnologias extrair informações relevantes e comunica-las de maneira simples e clara.

O sistema foi construído a partir de diversas tecnologias promissoras para sistemas IoT como sensores SHT75 e BH1750, módulo GPS, o microcontrolador ESP-32, banco de dados NoSQL, protocolo de comunicação WebSocket e integração com API's e técnicas de ciência de dados. Essas tecnologias possibilitaram a coleta e o processamento dos dados de forma eficiente, a integração com outras fontes de informação e a extração de conhecimentos relevantes. Assim o sistema desenvolvido atende os critérios estabelecidos nos objetivos desse trabalho, "um sistema IoT para o monitoramento de grandezas agrometeorológicas.

Ainda, o sistema demonstrou sua capacidade de medir e monitorar grandezas agrometeorológicas, como temperatura, umidade do ar, intensidade luminosa e localização. Os dados coletados foram armazenados em tempo real, permitindo um acompanhamento contínuo das condições ambientais. Além disso, a aplicação web desenvolvida proporcionou uma interface intuitiva e acessível. Permitindo a visualização e interação com as informações de forma clara e simples.

A integração do sistema com API's possibilitou a combinação dos dados coletados com informações de outras fontes, como a API do OpenWeatherMap, enriquecendo os dados e fornecendo contexto adicional. Isso permitiu uma análise mais abrangente das condições meteorológicas. Já a API do Maps construiu um mapa da localização atual através das coordenadas enviadas pelo GPS.

Além disso, a aplicação da ciência de dados no sistema IoT permitiu a extração de conhecimentos a partir dos dados coletados. Através da análise temporal da temperatura

e do histograma das faixas de umidade, foi possível identificar padrões sazonais. Essas informações podem auxiliar na tomada de decisões estratégicas. E também foi feito um mecanismo de baixar os dados coletados pelos sensores para permitir análises mais específicas do que as duas apresentadas.

Em suma, o sistema desenvolvido demonstrou a relevância e o potencial das tecnologias utilizadas para sistemas IoT. Através da combinação de sensores, protocolos de comunicação, armazenamento de dados, integração com API's e aplicação de técnicas de ciência de dados, foi possível criar um sistema eficiente, flexível e de fácil acesso para o monitoramento de grandezas agrometeorológicas. Os resultados obtidos destacam a importância dessas tecnologias no contexto do IoT e abrem caminho para futuras aplicações e pesquisas nessa área em constante evolução.

5.1 Trabalhos Futuros

Esse trabalho tem como objetivo a mensuração de grandezas agrometeorológicas gerais. Mas tem grande potencial de ser usado em sistemas mais específicos, como o cultivo de espécies de plantas, ou de controle de condições ambientais em sistemas fechados como greenhouses. Para tais aplicações, é necessário levantar novos requisitos específicos que irão exigir a leitura de outros dados relevantes, para a leitura desses dados novos sensores podem ser integrados no sistema aproveitando a infraestrutura construída. Exemplos de possíveis sensores de interesse a integrar nesse sistema são sensores de umidade do solo, pH do solo ou sensores de luminosidade específicos para determinadas espécies de plantas.

O uso de acionadores também podem ser integrados, como a comunicação no sistema é bidirecional, comandos provenientes dos servidores podem ser enviados ao microcontrolador. Isso abre a possibilidade de implementar controle automático das grandezas ambientais com base nos dados coletados. Por exemplo, é possível acionar sistemas de irrigação automatizados com base na umidade do solo ou ajustar a iluminação com base na intensidade luminosa medida.

Com essas integrações para adaptar o sistema a um cultivo específico ainda é possível o desenvolvimento de modelos preditivos, Utilizando as técnicas de ciência de dados, é possível desenvolver modelos preditivos que relacionem as grandezas agrometeorológicas medidas com o crescimento e desenvolvimento das plantas. Esses modelos poderiam auxiliar na previsão de problemas, como doenças ou deficiências nutricionais, permitindo a tomada de ações preventivas para garantir um cultivo saudável.

Alguns estudos de melhorias nesse sistema podem ser desenvolvidos, como o uso de outros sensores de temperatura e umidade, bem como o uso de outros microcontroladores para avaliar como o sistema se comporta com a diversidade de dispositivos. Também é relevante um estudo sobre o consumo de energia do sistema, para tornar o consumo de energia mais eficiente. Ainda, no desenvolvimento Web, pode-se criar duas aplicações

distintas, uma para o front-end e outra para o back-end, promovendo uma maior segurança no sistema.

Referências

AL-MASRI, E. Investigating messaging protocols for the internet of things (iot). *Engineering and Technology, University of Washington Tacoma*, USA, 2017. 25, 34, 36, 37

ASIMINIDIS, G. K. C. Database systems performance evaluation for iot applications. *International Journal of Database Management Systems (IJDMS)*, Greece, 2018. 26, 42

AZAD, P. The role of structured and unstructured data managing mechanisms in the internet of things. *Springer Nature*, 2019. 42

BAGAA, M. A machine learning security framework for iot systems. *IEEE*, Finland, 2021. 41

CORAK, F. Y. O. B. H. Comparative analysis of iot communication protocols. *Department of Computer Engineering, Gazi University*, Turkey, 2018. 35, 36, 37, 38

DACHYAR, M. Knowledge growth and development: internet of things (iot) research, 2006–2018. *Heliyon*, Indonesia, 2019. 28

FAROOQ, M. W. M. A review on internet of things (iot). *International Journal of Computer Applications*, Pakistan, 2015. 25, 32

FERREIRA, H. G. C. Iot architecture to enable intercommunication through rest api and upnp using ip, zigbee and arduino. *International Workshop on Internet of Things Communications and Technologies*, Brazil, 2013. 26, 42

GARG, H. Securing iot devices and securelyconnecting the dots using rest api and middleware. *IEEE*, India, 2019. 42

GOKHALE, O. B. P. Introduction to iot. *International Advanced Research Journal in Science, Engineering and Technology*, India, 2018. 25, 31, 32

HELAL, S. Integrating the internet of things and data science. *IEEE Internet of Things Magazine*, USA, 2021. 44, 45

KRAIJAK, S. A survey on iot architectures, protocols, applications, security, privacy, real-world implementation and future trends. *King Mongkut's Institute of Technology Ladkrabang*, Thailand, 2013. 27

LUCERO, S. Iot platforms: enabling the internet of things. *IHS TECHNOLOGY*, UK, 2016. 34

MAIER, A. Comparative analysis and practical implementation of the esp32 microcontroller module for the internet of things. *IEEE*, UK, 2017. 48

MORAES, B. N. T. Performance comparison of iot communication protocols. *IEEE International Conference on Systems, Man and Cybernetics (SMC)*, Italy, 2019. 35, 36

MORONEY, L. Book: The definitive guide to firebase. *Apress*, USA, 2017. 43

MORONEY, L. Comparison between mqtt and websocket protocols for iot applications using esp8266. *Apress*, USA, 2017. 44

MURLEY, P. Websocket adoption and the landscape of the real-time web. *IW3C2 (International World Wide Web Conference Committee)*, USA, 2021. 40

NAIK, N. Choice of effective messaging protocols for iot systems: Mqtt, coap, amqp and http. *Defence School of Communications and Information Systems*, UK, 2017. 37, 38

PASHA, S. Thingspeak based sensing and monitoring system for iot with matlab analysis. *International Journal of New Technology and Research (IJNTR)*, India, 2016. 42

PIMENTEL, V. Communicating and displaying real-time data with websocket. *IEEE Computer Society*, Venezuela, 2012. 40

RANJAN, R. Challenges and opportunities fordata science and machine learning in iot systems. *Copublished by the IEEE CS and IEEE ComSoc*, EU, 2018. 44

RAUTMARE, S. Mysql and nosql database comparison for iot application. *IEEE International Conference on Advances in Computer Applications (ICACA)*, India, 2016. 26, 43

ROSELIN, R. Smart agro system using wireless sensor networks. *International Conference on Intelligent Computing and Control Systems ICICCS*, India, 2017. 25, 45

ROY, S. Agro-sense: Precision agriculture using sensor-based) wireless mesh networks. *ITU*, INDIA, 2008. 27, 45

SETHI, S. R. S. P. Internet of things: Architectures, protocols, and applications. *Journal of Electrical and Computer Engineering*, India, 2017. 33

THOMA, M. On iot-services: Survey, classification and enterprise integration. *IEEE International Conference on Green Computing and Communications*, UK, 2012. 27

XU, T. Security of iot systems:design challenges and opportunities. *IEEE*, USA, 2014. 41

YOKOTANI, Y. S. T. Comparison with http and mqtt on required network resources for iot. *International Conference on Control, Electronics, Renewable Energy and Communications (ICCEREC)*, Japan, 2016. 39

Apêndices

APÊNDICE A - Algoritmos ESP32

main.c

```
/*Includes de bibliotecas*/
/*Bibliotecas Arduino.h/ ESP32/ Wifi.h / Firebase/ Sensores*/
#include <Arduino.h>
#include "WiFi/WiFi.h"
#include "FirebaseComponent/FirebaseComponent.h"
#include "Epoch/Epoch.h"
#include "SHT75/sht75.h"
#include "BH1750/bh1750.h"
#include "GPS/gps.h"

/*Objetos Firebase*/
FirebaseData fbdo;
String guarda;

void setup(){
  /*Inicializacao do Serial*/
  Serial.begin(115200);
  /*Inicializacao do WiFi*/
  initWiFi();

  /*Inicializacao do firebase*/
  String user = initFirebase();
  guarda = user;

  /*Inicializacao dos sensores*/
  initSHT75(user);
  initBH1750(user);
  initGPS(user);

  /*Inicializacao do Epoch*/
  epoch_init();

  /*Inicializacao do Deep Sleep caso o a nao esteja ativo na aplicacao Web
    */
  if(!estadoUser()){
    /*Envio dos dados dos sensores*/
    Resposta_SHT75();
    Resposta_BH1750();
    Resposta_GPS();
```

```
39      /*Configuracao do Deep Sleep*/
40      esp_sleep_enable_timer_wakeup( 5 * 58 * 1000000);
41      /*Inicio do Deep Sleep*/
42      esp_deep_sleep_start();
43    }
44  }
45  void loop(){
46    /*Verificacao de requisicao dos sensores*/
47    if(Request_SHT75()){
48      Resposta_SHT75();
49    }
50
51    if(Request_BH1750()){
52      Resposta_BH1750();
53    }
54
55    if(Request_GPS()){
56      initGPS(guarda);
57      Resposta_GPS();
58    }
59
60    /*Verificacao de requisicao do Deep Sleep, caso o usuario tenha saido da
       aplicacao Web*/
61    if(!estadoUser()){
62      esp_sleep_enable_timer_wakeup(5 * 58 * 1000000);
63      esp_deep_sleep_start();
64    }
65  }
```

bh1750.h

```
1  #ifndef BH1750_H
2  #define BH1750_H
3
4  #include "FirebaseComponent/FirebaseComponent.h"
5
6  void initBH1750(String uid);
7  bool Request_BH1750();
8  void Resposta_BH1750();
9
10 extern FirebaseData fbdo;
11
12 #endif
```

bh1750.cpp

```
1  #include "bh1750.h"
2  #include "Epoch/Epoch.h"
3  #include <Wire.h>
```

```
4  #include <BH1750.h>
5
6  /*Variaveis globais BH1750*/
7  String BHT1750_Path;
8  String BH1750_LuxPath = "/Lux";
9  String BH1750_timePath = "/timestamp";
10 String Request_BH1750_Path;
11 String parentPathBH1750;
12
13 /*Objetos BH1750*/
14 BH1750 lightMeter;
15 TwoWire I2CBH1750 = TwoWire(0);
16
17 /*Objetos Firebase*/
18 FirebaseJson jsonBH;
19
20 /*Variaveis auxiliares*/
21 int timestampBH1750;
22
23 /*Funcao de inicializacao do BH1750*/
24 void initBH1750(String uid){
25
26   const int I2C_SDA = 18;
27   const int I2C_SCL = 19;
28
29   I2CBH1750.begin(I2C_SDA, I2C_SCL);
30   lightMeter.begin(BH1750::CONTINUOUS_HIGH_RES_MODE, 0x23, &I2CBH1750);
31
32   BHT1750_Path = "/UsersData/" + uid + "/BHT1750";
33   Request_BH1750_Path = "/UsersData/" + uid + "/Request/BH1750_request/
       BH1750_request";
34
35 }
36
37 /*Funcao de requisicao do BH1750*/
38 bool Request_BH1750(){
39   Firebase .RTDB.getBool(&fbdo, Request_BH1750_Path);
40   bool aux = fbdo.boolData();
41   Firebase.RTDB.setBool(&fbdo, Request_BH1750_Path, false);
42   return aux;
43 }
44
45 /*Funcao de resposta do BH1750*/
46 void Resposta_BH1750(){
47   timestampBH1750 = getTime();
48   parentPathBH1750= BHT1750_Path + "/" + String(timestampBH1750);
49
```

```
  jsonBH.set(BH1750_LuxPath.c_str(), String(lightMeter.readLightLevel()));
  jsonBH.set(BH1750_timePath.c_str(), String(timestampBH1750));
  Serial.printf("envia json... %s\n", Firebase.RTDB.setJSON(&fbdo,
    parentPathBH1750.c_str(), &jsonBH) ? "ok" : fbdo.errorReason().c_str());
}
```

Epoch.h

```
#ifndef EPOCH_H
#define EPOCH_H

#include <time.h>
#include <Arduino.h>

void epoch_init();
unsigned long getTime();

#endif
```

Epoch.cpp

```
#include "epoch.h"

/*Configurando o servidor de timer*/
void epoch_init() {
  const char* ntpServer = "pool.ntp.org";
  configTime(0, 0, ntpServer);
}

/*Funcao de obtencao do tempo*/
unsigned long getTime() {
  time_t now;
  struct tm timeinfo;
  if (!getLocalTime(&timeinfo)) {
    return(0);
  }
  time(&now);
  return now;
}
```

FirebaseComponent.h

```
#ifndef FIREBASE_H
#define FIREBASE_H

#include <Firebase_ESP_Client.h>

String initFirebase();
bool estadoUser();

```

```
9  extern FirebaseData fbdo;
10
11 #endif
```

FirebaseComponent.cpp

```
1  #include "FirebaseComponent.h"
2  #include "addons/TokenHelper.h"
3  #include "addons/RTDBHelper.h"
4  #include "WiFi.h"
5
6  //Objetos Firebase
7  FirebaseAuth auth;
8  FirebaseConfig config;
9  String uid;
10 String estadoPath;
11
12 //Funcao de inicializacao da conexao com o Firebase
13 String initFirebase() {
14   //Inicializacao da conexao com o Firebase
15   config.api_key = "chave da API";
16   auth.user.email = "email do usuario";
17   auth.user.password = "senha do usuario";
18   config.database_url = "url do banco de dados";
19
20   //Configuracao do WiFi
21   Firebase.reconnectWiFi(true);
22
23   //Configuracao do token
24   config.token_status_callback = tokenStatusCallback;
25   config.max_token_generation_retry = 5;
26   Firebase.begin(&config, &auth);
27
28   //Verificacao da conexao com o Firebase
29   while ((auth.token.uid) == "") {
30     delay(1000);
31   }
32
33   //Retorno do UID do usuario
34   uid = auth.token.uid.c_str();
35   return uid;
36 }
37
38 //Funcao de verificacao do estado do usuario
39 bool estadoUser(){
40   estadoPath = "/UsersData/" + String(uid) + "/Estado/ativo";
41   Firebase.RTDB.getBool(&fbdo, estadoPath);
42   return fbdo.boolData();
```

```
}
```

gps.h

```
#ifndef GPS_H
#define GPS_H

#include "FirebaseComponent/FirebaseComponent.h"

void initGPS(String uid);
bool Request_GPS();
void Resposta_GPS();
void Teste();

extern FirebaseData fbdo;

#endif
```

gps.cpp

```
#include "gps.h"
#include <TinyGPSPlus.h>
#include "SoftwareSerial.h"

// Variaveis GPS
String GPS_Path;
String GPS_latPath = "/LAT";
String GPS_lngPath = "/LNG";
String GPS_timePath = "/Date";
String Request_GPS_Path;
TinyGPSPlus gps;
static const int RXPin = 16, TXPin =17;
static const uint32_t GPSBaud = 9600;
SoftwareSerial ss(RXPin, TXPin);

// Variaveis Firebase
FirebaseJson jsonGPS;
String parentPath;

/*Funcao de inicializacao do GPS*/
void initGPS(String uid){
  ss.begin(GPSBaud);
  GPS_Path = "/UsersData/" + uid + "/GPS";
  Request_GPS_Path = "/UsersData/" + uid + "/Request/GPS_request/Requisito"
    ;
}

/*Funcao de requisicao do GPS*/
bool Request_GPS(){
```

```
  Firebase .RTDB.getBool(&fbdo, Request_GPS_Path);
  bool aux = fbdo.boolData();
  Firebase.RTDB.setBool(&fbdo, Request_GPS_Path, false);
  return aux;
}

int controle = 0;

/*Funcao de resposta do GPS*/
void Resposta_GPS() {

  parentPath= GPS_Path + "/GPS_Data";

  Serial.println(ss.read());

   while (ss.available() > 0){
    gps.encode(ss.read());
    if (true){
      Serial.println("Latitude= ");
      Serial.println(gps.location.lat(), 6);
      Serial.println(" Longitude= ");
      Serial.println(gps.location.lng(), 6);

      jsonGPS.set(GPS_timePath.c_str(), String(gps.date.value()) + " " +
   String(gps.time.value())),
      jsonGPS.set(GPS_latPath.c_str(), String(gps.location.lat(), 6));
      jsonGPS.set(GPS_lngPath.c_str(), String(gps.location.lng(), 6));
      Serial.printf("Set json... %s\n", Firebase.RTDB.setJSON(&fbdo,
   parentPath.c_str(), &jsonGPS) ? "ok" : fbdo.errorReason().c_str());

    }
    controle++;
    if(controle > 20){
        break;
    }
  }

  Serial.println("Envia GPS");
}
```

sht75.h

```
#ifndef SHT75_H
#define SHT75_H
#include "FirebaseComponent/FirebaseComponent.h"

void initSHT75(String uid);
bool Request_SHT75();
```

```
7  void Resposta_SHT75();
8
9  extern FirebaseData fbdo;
10
11 #endif
```

sht75.cpp

```
1  #include "SHT75.h"
2  #include "Epoch/Epoch.h"
3  #include <SHT1x-ESP.h>
4
5  /*SHT Variaveis*/
6  String SHT_Path;
7  String SHT_tempcPath = "/TemperaturaC";
8  String SHT_tempfPath = "/TemperaturaF";
9  String SHT_humidityPath = "/umidade";
10 String SHT_timePath = "/timestamp";
11 String Request_SHT_Path;
12 String parentePathSHT;
13
14 FirebaseJson jsonSHT;
15
16 int timestampSHT;
17
18 /*Defenindo pinos do SHT*/
19 #define dataPin  21
20 #define clockPin 22
21 SHT1x sht1x(dataPin, clockPin, SHT1x::Voltage::DC_3_3v);
22
23 /*Funcao de inicializacao do SHT*/
24 void initSHT75(String uid){
25   SHT_Path = "/UsersData/" + String(uid) + "/SHT";
26   Request_SHT_Path = "/UsersData/" + String(uid) + "/Request/SHT_request/
       SHT_request";
27 }
28
29 /*Funcaoo de requisicaoo do SHT*/
30 bool Request_SHT75(){
31   Firebase.RTDB.getBool(&fbdo, Request_SHT_Path);
32   bool aux = fbdo.boolData();
33   Firebase.RTDB.setBool(&fbdo, Request_SHT_Path, false);
34   return aux;
35 }
36
37 /*Funcao de resposta do SHT*/
38 void Resposta_SHT75(){
39   timestampSHT = getTime();
```

```
  parentePathSHT= SHT_Path + "/" + String(timestampSHT);

  jsonSHT.set(SHT_tempcPath.c_str(), String(sht1x.readTemperatureC()));
  jsonSHT.set(SHT_tempfPath.c_str(), String(sht1x.readTemperatureF()));
  jsonSHT.set(SHT_humidityPath.c_str(), String(sht1x.readHumidity()));
  jsonSHT.set(SHT_timePath.c_str(), String(timestampSHT));
  Serial.printf("Envia json... %s\n", Firebase.RTDB.setJSON(&fbdo,
   parentePathSHT.c_str(), &jsonSHT) ? "ok" : fbdo.errorReason().c_str());
}
```

Wifi.h

```
#ifndef WIFI_H
#define WIFI_H

#include <WiFi.h>

void initWiFi();

#endif
```

Wifi.h

```
#include "WiFi.h"

//Funcao de inicializacao do WiFi
void initWiFi() {
  WiFi.begin("SSD", "SENHA");
  int timeout_counter = 0;
  Serial.print("Connecting to WiFi ..");
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print('.');
    delay(1000);
    timeout_counter++;
    if(timeout_counter >= 10){
    ESP.restart();
    }
  }
  Serial.println(WiFi.localIP());
  delay(1000);
}
```

APÊNDICE B - Análise de Dados

API

main.py

```
from firebase_admin import initialize_app, db
from flask import Flask, send_file, request, Response
```

```
3  import plot
4
5  app = Flask(__name__)
6
7  #Funcao de retorno do grafico de temperatura
8  @app.route('/plots/temp', methods=['GET'])
9  def graphs_temp():
10     args = request.args
11     datemin = args.get('datemin')
12     datemax = args.get('datemax')
13     medida = args.get('medida')
14     resolucao = args.get('resolucao')
15     bytes_obj = plot.do_plot_temp(datemin, datemax, medida, resolucao)
16
17     return send_file(bytes_obj,
18                      download_name='plot.png',
19                      mimetype='image/png')
20
21 #Funcao de retorno do grafico de umidade
22 @app.route('/plots/umid', methods=['GET'])
23 def graphs_umid():
24     args = request.args
25     datemin = args.get('datemin')
26     datemax = args.get('datemax')
27     bytes_obj = plot.do_plot_umid(datemin, datemax)
28
29     return send_file(bytes_obj,
30                      download_name='plot.png',
31                      mimetype='image/png')
32
33 #Funcao de retorno do arquivo de download
34 @app.route('/plots/download', methods=['GET'])
35 def download_file():
36     args = request.args
37     tipo = args.get('formato')
38     tempo = args.get('periodo')
39
40     arquivo, formato = plot.download(tipo, tempo)
41
42     return Response(arquivo,
43                      mimetype=formato,
44                      headers={
45                      'Content-Disposition': f'attachment; filename="dados.{
       tipo}"'
46                      })
47
48
```

```
49 if __name__ == '__main__':
50     app.run(debug=True)
```

plot.py

```
1  import firebase_admin
2  from firebase_admin import credentials
3  from firebase_admin import auth
4  from firebase_admin import db
5  import numpy as np
6  import pandas as pd
7  from datetime import datetime as dt
8  from datetime import timedelta
9  import matplotlib.pyplot as plt
10 import matplotlib.dates as mdates
11 import io
12 import PIL.Image as Image
13
14 #Objeto de credenciais do firebase
15 cred_obj = credentials.Certificate("C://arquivo com as credenciais.json")
16 firebase_admin.initialize_app(cred_obj, {
17   'databaseURL': 'URL do banco de dados'
18 })
19
20 #Codigo para pegar o usuario do firebase
21 email = "email de usuario do firebase"
22 user = auth.get_user_by_email(email)
23 plt.style.use("bmh")
24
25 def do_plot_temp(datemin, datemax, medida, resolucao):
26     #Conversao de data para timestamp
27     datemin = dt.strptime(datemin, '%Y-%m-%dT%H:%M')
28     datemax = dt.strptime(datemax, '%Y-%m-%dT%H:%M')
29
30     #Conversao de data para timestamp
31     datemin_timestamp = str(datemin.timestamp())
32     datemax_timestamp = str(datemax.timestamp())
33
34     #consulta dos dados do banco de dados
35     ref = db.reference('/UsersData/'+ str(user.uid)+'/SHT/')
36     query = ref.order_by_child('timestamp').start_at(datemin_timestamp).
       end_at(datemax_timestamp)
37     data = query.get()
38
39     #Criacao do dataframe
40     df = pd.DataFrame.from_dict(data, orient='index')
41     df["data"] = pd.to_datetime(pd.to_numeric(df["timestamp"]), unit='s')
42     df["TemperaturaC"] = df["TemperaturaC"].astype(float)
```

```
43      df = df.drop(df[df['TemperaturaC'] < 0].index)
44
45      #Criacao do grafico
46      fig, ax = plt.subplots()
47      fig.set_tight_layout(True)
48
49      #Funcao de agregacao dos dados de acordo com a frequencia e a medida
        escolhidas
50      if(medida == "all"):
51          df_media = df.groupby(pd.Grouper(key='data', freq=resolucao)).agg({
        "TemperaturaC": "mean"}).reset_index()
52          df_minimo = df.groupby(pd.Grouper(key='data', freq=resolucao)).agg
        ({"TemperaturaC": "min"}).reset_index()
53          df_maximo = df.groupby(pd.Grouper(key='data', freq=resolucao)).agg
        ({"TemperaturaC": "max"}).reset_index()
54
55          ax.plot(df_media["data"], df_media["TemperaturaC"], label="Media")
56          ax.plot(df_minimo["data"], df_minimo["TemperaturaC"], label="Minima
        ")
57          ax.plot(df_maximo["data"], df_maximo["TemperaturaC"], label="Maxima
        ")
58      else:
59          df = df.groupby(pd.Grouper(key='data', freq=resolucao)).agg({"
        TemperaturaC": medida}).reset_index()
60          ax.plot(df["data"], df["TemperaturaC"])
61
62      data_final = dt.fromtimestamp(float(datemax_timestamp))
63
64      #Configuracoes do grafico
65      ax.set_xlim([df['data'].iloc[0], data_final])
66      ax.set_ylabel("Temperatura")
67      plt.xticks(rotation=45)
68      xtick_labels = ax.get_xticklabels()
69      num_labels = len(xtick_labels)
70      max_labels = 10
71      step = max(num_labels // max_labels, 1)
72      new_labels = [xtick_labels[i].get_text() for i in range(0, num_labels,
        step)]
73
74      #Conversao do formato da data de acordo com a resolucao escolhida
75      if len(new_labels[0]) == 8 and new_labels[0][2] == "-" and new_labels
        [0][5] == " ":
76          new_labels = [dt.strptime(date, '%m-%d %H').strftime("%d %H:%M")
        for date in new_labels]
77      elif len(new_labels[0]) == 10 and new_labels[0][4] == "-" and
        new_labels[0][7] == "-":
78          new_labels = [dt.strptime(date, '%Y-%m-%d').strftime("%d-%m") for
```

```
    date in new_labels]
79
80      ax.set_xticklabels(new_labels)
81      ax.set_xlabel("Tempo")
82
83      #Salvando o grafico em um buffer
84      buf = io.BytesIO()
85      fig.savefig(buf, format='png')
86      buf.seek(0)
87      return buf
88
89
90  def do_plot_umid(datemin, datemax):
91      #Conversao de data para timestamp
92      datemin = dt.strptime(datemin, '%Y-%m-%dT%H:%M')
93      datemax = dt.strptime(datemax, '%Y-%m-%dT%H:%M')
94
95      #Conversao de data para timestamp
96      datemin_timestamp = str(datemin.timestamp())
97      datemax_timestamp = str(datemax.timestamp())
98
99      #consulta dos dados do banco de dados
100     ref = db.reference('/UsersData/'+ str(user.uid)+'/SHT/')
101     query = ref.order_by_child('timestamp').start_at(datemin_timestamp).
    end_at(datemax_timestamp)
102     data = query.get()
103
104     #Criacao do dataframe
105     df = pd.DataFrame.from_dict(data, orient='index')
106     df["umidade"] = df["umidade"].astype(float)
107     df = df.drop(df[df['umidade'] < 0].index)
108
109     #Criacao do grafico
110     fig, ax = plt.subplots()
111     fig.set_tight_layout(True)
112
113     #Configuracoes do grafico
114     bins_umid = list(range(0, 110, 10))
115     ax.hist(df["umidade"], bins=bins_umid, rwidth=0.9, weights=np.
    zeros_like(df["umidade"]) + 1. / df["umidade"].size)
116     ax.set_xlabel("Umidade")
117     ax.set_ylabel("Frequencia")
118
119     #Salvando o grAfico em um buffer
120     buf = io.BytesIO()
121     fig.savefig(buf, format='png')
122     buf.seek(0)
```

```
123     return buf
124
125 def download(tipo, tempo):
126     #ConversAo de data para timestamp
127     hoje = dt.now()
128     hojeTimestamp = str(hoje.timestamp())
129
130     #Determinando o perIodo de consulta dos dados
131     qnt = 1 if tempo == 'd' else 7 if tempo == 'w' else 30
132
133     #ConversAo de data para timestamp
134     comeco = hoje - timedelta(days=qnt)
135     comecoTimestamp = str(comeco.timestamp())
136
137     #consulta dos dados do banco de dados
138     ref = db.reference('/UsersData/'+ str(user.uid)+'/SHT/')
139     query = ref.order_by_child('timestamp').start_at(comecoTimestamp).
    end_at(hojeTimestamp)
140     data = query.get()
141
142     #Criacao do dataframe
143     df = pd.DataFrame.from_dict(data, orient='index')
144     df["TemperaturaC"] = df["TemperaturaC"].astype(float)
145     df["data"] = pd.to_datetime(pd.to_numeric(df["timestamp"]), unit='s')
146     df["umidade"] = df["umidade"].astype(float)
147     df = df.drop(df[df['umidade'] < 0].index)
148     df = df.drop(df[df['TemperaturaC'] < 0].index)
149
150     #Criacao do arquivo de acordo com o formato escolhido
151     if tipo == 'csv':
152         arquivo = df.to_csv(index=False)
153         mimetype = 'text/csv'
154     elif tipo == 'json':
155         arquivo = df.to_json(orient='records')
156         mimetype = 'application/json'
157     else:
158         raise ValueError('Tipo de arquivo invalido')
159
160     #Retorno do arquivo e do mimetype
161     return arquivo, mimetype
```

Análise de dispersão

analise.ipynb

```
1 import json
2 import numpy as np
3 import pandas as pd
4 import datetime as dt
```

```
import pytz
import matplotlib.pyplot as plt
timezone = pytz.timezone('America/Sao_Paulo')
with open('dados.json') as f:
    json_data = f.read()

data = json.loads(json_data)
df = pd.DataFrame(data)
df["timestamp"] = df["timestamp"].astype(int)
df["TemperaturaC"] = df["TemperaturaC"].astype(float)
df["umidade"] = df["umidade"].astype(float)
df["TemperaturaF"] = df["TemperaturaF"].astype(float)
df["data"] = pd.to_datetime(df["timestamp"], unit="s").dt.tz_localize(pytz.
    utc).dt.tz_convert(timezone)
df["data"] = df["data"].dt.strftime("%d-%m %H:%M")
print(df.head(10))
   TemperaturaC  TemperaturaF   timestamp  umidade          data
0         23.14         73.67  1684340563    50.70  17-05 13:22
1         22.67         72.86  1684340862    52.37  17-05 13:27
2         23.51         74.36  1684341159    49.29  17-05 13:32
3         22.85         73.21  1684341457    51.48  17-05 13:37
4         23.56         74.47  1684341753    49.62  17-05 13:42
5         23.72         74.74  1684342052    50.27  17-05 13:47
6         23.63         74.65  1684342351    50.39  17-05 13:52
7         23.68         74.66  1684342650    50.33  17-05 13:57
8         22.93         73.31  1684342950    52.50  17-05 14:02
9         23.30         74.05  1684343249    51.64  17-05 14:07
df.describe()
TemperaturaC  TemperaturaF  timestamp umidade
count 289.00000 289.000000  2.890000e+02  289.000000
mean  18.46519  65.284221 1.684384e+09  70.102526
std 3.31908 5.978780  2.496678e+04  12.770155
min 13.66000  56.660000 1.684341e+09  48.480000
25% 15.23000  59.450000 1.684362e+09  59.300000
50% 18.61000  65.560000 1.684384e+09  69.470000
75% 21.28000  70.340000 1.684405e+09  83.180000
max 23.90000  75.080000 1.684427e+09  87.570000
df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 289 entries, 0 to 288
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   TemperaturaC   289 non-null    float64
 1   TemperaturaF   289 non-null    float64
 2   timestamp      289 non-null    int32
 3   umidade        289 non-null    float64
```

```
 4   data           289 non-null    datetime64[ns, America/Sao_Paulo]
dtypes: datetime64[ns, America/Sao_Paulo](1), float64(3), int32(1)
memory usage: 10.3 KB
fig, ax = plt.subplots(figsize=(20, 10))

df.plot.scatter(x="data", y="TemperaturaC", ax=ax, title="Dispersao: Temperatura em Celsius")

# Ajuste dos rotulos do eixo x
xticks = ax.get_xticks()
xticklabels = df["data"].values[::10]  # Seleciona a cada 10 rotulos para evitar sobreposicao
ax.set_xticks(xticks[::10])
ax.set_xticklabels(xticklabels, rotation=45)

plt.show()

fig, ax = plt.subplots(figsize=(20, 10))

df.plot.scatter(x="data", y="umidade", ax=ax, title="Dispersao: Umidade")

# Ajuste dos rotulos do eixo x
xticks = ax.get_xticks()
xticklabels = df["data"].values[::10]  # Seleciona a cada 10 rotulos para evitar sobreposicao
ax.set_xticks(xticks[::10])
ax.set_xticklabels(xticklabels, rotation=45)

plt.show()
```

Pré-processamento

Sensors data.ipynb

```

import requests
import json
import firebase_admin
from firebase_admin import credentials
from firebase_admin import auth
from firebase_admin import db
import numpy as np
import pandas as pd
import datetime as dt
import matplotlib.pyplot as plt
import io
import PIL.Image as Image
#Objeto de credenciais do firebase
cred_obj = credentials.Certificate("C://arquivo com as credenciais.json")
```

```
16 firebase_admin.initialize_app(cred_obj, {
17   'databaseURL': 'URL do banco de dados'
18 })
19
20 #Codigo para pegar o usuario do firebase
21 email = "email de usuario do firebase"
22 user = auth.get_user_by_email(email)
23
24 #Consulta dos dados no banco de dados
25 ref = db.reference('/UsersData/'+ str(user.uid)+'/SHT/')
26 data = ref.order_by_key().get()
27 #Carregando dados historicos do arquivo JSON
28 with open('dado_hist.json') as f:
29     json_data = f.read()
30
31 data_hist = json.loads(json_data)
32 #Conversao dos dados historicos para o formato do banco de dados
33 data = []
34
35 for data_point in data_hist:
36     temperatura = data_point['main']['temp']
37     umidade = data_point['main']['umidade']
38     timestamp = data_point['dt']
39
40     data.append({"timestamp": timestamp, "temperatura": temperatura, "
    umidade": umidade})
41
42 # Loop para interpolar a cada 5 minutos e suavizar com media movel
43 dado_interpolado = []
44
45 for i in range(len(data)-1):
46     timestamp_inicial = data[i]['timestamp']
47     timestamp_final = data[i+1]['timestamp']
48     temperatura_inicial = data[i]['temperatura']
49     temperatura_final = data[i+1]['temperatura']
50     umidade_inicial = data[i]['umidade']
51     umidade_final = data[i+1]['umidade']
52
53     # Interpolacao linear
54     for j in range(1, 12):
55         timestamp = timestamp_inicial + j*300
56         temperatura = np.interp(timestamp, [timestamp_inicial,
    timestamp_final], [temperatura_inicial, temperatura_final])
57         umidade = np.interp(timestamp, [timestamp_inicial, timestamp_final
    ], [umidade_inicial, umidade_final])
58         dado_interpolado.append({"timestamp": timestamp, "temperatura":
    temperatura, "umidade": umidade})
```

```
59
60
61      #Media movel
62      temperatura_suavizada = [dado_interpolado[i+j]['temperatura'] for j in
        range(1, 11)]
63      umidade_suavizada = [dado_interpolado[i+j]['umidade'] for j in range(1,
        11)]
64      temperatura_media = np.mean(temperatura_suavizada)
65      umidade_media = np.mean(umidade_suavizada)
66
67      # Distribuicao normal
68      temperatura_std = np.std(temperatura_suavizada)
69      umidade_std = np.std(umidade_suavizada)
70      for j in range(11):
71          temperatura = np.random.normal(temperatura_media, temperatura_std)
72          umidade = np.random.normal(umidade_media, umidade_std)
73          dado_interpolado[i+j]['temperatura'] = round(temperatura, 2)
74          dado_interpolado[i+j]['umidade'] = round(umidade, 2)
75
76 # Adiciona os dados interpolados e com distribuicao normal ao dicionario '
       data'
77 for item in dado_interpolado:
78      data.append(item)
79
80
81 data = sorted(data, key=lambda k: k['timestamp'])
82 #Envia os dados para o banco de dados
83 for d in data:
84      ref.child(str(d['timestamp'])).set({
85          "TemperaturaC": str(d['temperatura']),
86          "timestamp": str(d['timestamp']),
87          "umidade": str(d['umidade'])
88      })
```

APÊNDICE C - Aplicação Web

index.html

```
1       <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1">
6     <link rel="stylesheet" href="style.css"/>
7     <link rel="preconnect" href="https://fonts.googleapis.com">
8     <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
```

```
9     <link href="https://fonts.googleapis.com/css2?family=Open+Sans&display=
      swap" rel="stylesheet">
10    <title>Iot App</title>
11
12     <!-- conexecao com o firebase -->
13     <script src="https://www.gstatic.com/firebasejs/8.10.0/firebase-app.js
      "></script>
14     <script src="https://www.gstatic.com/firebasejs/8.8.1/firebase-auth.js
      "></script>
15     <script src="https://www.gstatic.com/firebasejs/8.8.1/firebase-
      database.js"></script>
16
17        <script>
18        //Credenciais do banco de dados
19        const firebaseConfig = {
20          apiKey: "Chave da API",
21          authDomain: "URL do dominio",
22          databaseURL: "URL do banco de dados",
23          projectId: "ID do projeto",
24          storageBucket: "Nome do bucket",
25          messagingSenderId: "ID do remetente",
26          appId: "ID do app"
27 };
28        //Inicializacao do app Firebase
29        firebase.initializeApp(firebaseConfig);
30        const auth = firebase.auth()
31        const db = firebase.database();
32    </script>
33
34  </head>
35    <body>
36    <!--Barra-->
37    <div class="topnav">
38      <h1>Sistema IOT</h1>
39    </div>
40
41       <!--Formulario de login-->
42    <form id="login-form" style="display: none;">
43
44      <div class="form-elements-container">
45        <label for="input-email"><b>Email</b></label>
46        <input type="text" placeholder="Enter Email" id="input-email"
      required>
47
48        <label for="input-password"><b>Password</b></label>
49        <input type="password" placeholder="Enter Senha" id="input-password
      " required>
```

```
50
51        <button class="botpad" type="submit" id="login-button">Login</
    button>
52        <p id="error-message" style="color:rgb(255, 50, 50);"></p>
53
54      </div>
55    </form>
56
57    <!--Conteudo-->
58  <div class="content-sign-in" id="content-sign-in" style="display: none;">
59    <div class="conteudo">
60      <div class="box1">
61        <div class="tempconteiner">
62          <div class="card1">
63            <div class="tempo">
64                <h2 class="cidade"></h2>
65                <h1 class="temp"></h1>
66                <div class="flex">
67                    <img src="" alt="" class="icon" >
68                    <div class="descricao"></div>
69                </div>
70                <div class="umidadeSat"></div>
71                <div class="vento"></div>
72                <div class="nascer"></div>
73                <div class="por"></div>
74            </div>
75          </div>
76
77          <div class="sensor">
78            <h2>SHT75</h2>
79            <p>Temperatura: <span id="SHT_tempC"></span></p>
80            <p>Temperatura: <span id="SHT_tempF"></span></p>
81            <p>Umidade: <span id="SHT_umidade"></span></p>
82            <button class="botpad" id="SHT_button">Atualizar</button>
83            <h6>Ultima Atualizacao: <span id="SHT_data"></span></h6>
84          </div>
85
86          <div class="light">
87            <h2>BH1750</h2>
88            <p>Lux: <span id="Lux"></span></p>
89            <button class="botpad" id="BH1750_button">Atualizar</button>
90            <h6>Ultima Atualizacao: <span id="Light_data"></span></h6>
91          </div>
92
93        </div>
94
95    </div>
```

```
96      <div class="box2">
97        <div class="graphconteiner">
98          <h2>Dashboard </h2>
99          <div class="opcoes_graf">
100             <label for="medida">Temperatura:</label>
101             <select id="medida" name="medida">
102                 <option value="mean">media</option>
103                 <option value="min">minima</option>
104                 <option value="max">maxima</option>
105                 <option value="all">todas</option>
106             </select><br><br>
107             <label for="resolucao">Resolucao:</label>
108             <select id="resolucao" name="resolucao">
109                 <option value="1H">1 hora</option>
110                 <option value="1D">1 dia</option>
111                 <option value="1W">1 semana</option>
112                 <option value="1M">1 mes</option>
113             </select><br><br>
114             <label for="datemin">Data de inicio:</label>
115             <input type="datetime-local" id="datemin" name="datemin" min="
    2023-01-01T00:00"><br><br>
116
117             <label for="datemax">Data de Fim:</label>
118             <input type="datetime-local" id="datemax" name="datemax"><br><
    br>
119
120             <button class="botpad" id="submit">Submit</button>
121
122
123         </div>
124
125         <div class="graph">
126           <div class="graf">
127             <h3>Temperatura</h3>
128             <img class="grafico" id="temperaturaGraf" src="https://agriot-9
    b824.rj.r.appspot.com/plots/temp?datemin=2023-01-01T00:00&datemax
    =2023-01-30T23:59&medida=all&resolucao=1D"/>
129           </div>
130           <div class="graf">
131             <h3>Umidade</h3>
132             <img  class="grafico" id="umidadeGraf"src="https://agriot-9b824
    .rj.r.appspot.com/plots/umid?datemin=2023-01-01T00:00&datemax=2023-01-30
    T23:59"/>
133           </div>
134         </div>
135       </div>
136
```

```
137       <div class="map_baixar">
138         <div class="mapconteiner">
139           <h3>Localizacao</h3>
140           <div class="map" id="map"></div>
141         </div>
142         <div class="baixar">
143           <h3>Download dos Dados</h3>
144           <label for="periodo">Referente:</label>
145             <select id="periodo" name="periodo">
146                 <option value="m">1 mes</option>
147                 <option value="w">1 semana</option>
148                 <option value="d">1 dia</option>
149             </select><br><br>
150           <label for="tipo">Arquivo:</label>
151             <select id="tipo" name="tipo">
152                 <option value="csv">CSV</option>
153                 <option value="json">JSON</option>
154             </select><br><br>
155             <button class="botpad" id="download">Download</button>
156         </div>
157       </div>
158     </div>
159     </div>
160   </div>
161
162    <!--Barra de autentificacao-->
163    <div id="authentication-bar" style="display: none;">
164     <p><span id="authentication-status"></span>
165        <span id="user-details">USEREMAIL</span>
166        <button  class="botpad" href="/" id="logout-link">Sair</button>
167     </p>
168   </div>
169
170     <!--inclusao dos arquivos javascript-->
171     <script src="https://maps.googleapis.com/maps/api/js?key=
      AIzaSyCsVMjUUVFcya97oLqI45bEJSPtgA-v8mg"></script>
172     <script src="scripts/auth.js"></script>
173     <script src="scripts/index.js"></script>
174
175   </body>
176 </html>
```

style.css

```
1     html {
2   font-family: Verdana, Geneva, Tahoma, sans-serif;
3   display: inline-block;
4   text-align: center;
```

```
5  }
6
7  /*Formulario de login*/
8  .form-elements-container
9  {
10   background-color: #225024d0;
11   color: #FDFDFD;
12   padding: 2em;
13   border-radius: 20px;
14 }
15
16 #input-email{
17   border-radius: 4px;
18   background-color: rgb(247, 225, 194);
19   color: #225024;
20 }
21
22 #input-password{
23   border-radius: 4px;
24   background-color: rgb(247, 225, 194);
25   color: #225024;
26 }
27
28 #authentication-bar{
29   padding-top: 10px;
30   padding-bottom: 10px;
31   margin-bottom: 5px;
32 }
33
34 .form-elements-container{
35   padding: 16px;
36   width: 250px;
37   margin: 0 auto;
38 }
39
40 input[type=text], input[type=password] {
41   width: 100%;
42   padding: 12px 20px;
43   margin: 8px 0;
44   display: inline-block;
45   border: 1px solid #ccc;
46   box-sizing: border-box;
47 }
48
49 /*Conteudo da pagina*/
50 body {
51   margin: 0;
```

```
52    width: 100%;
53    display: flex;
54    flex-direction: column;
55    background-image: url(https://images.unsplash.com/photo-1563514227147-6
        d2ff665a6a0?ixlib=rb-1.2.1&ixid=
        MnwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8&auto=format&fit=crop&w
        =1471&q=80);
56    background-size: cover;
57  }
58
59  .topnav {
60    height: 50px;
61    overflow: hidden;
62
63    color: white;
64    font-size: 0.7rem;
65    padding: 5px;
66  }
67
68  .conteudo {
69    width: 1350px;
70    height: 100%;
71    display: flex;
72  }
73
74  .box1 {
75    width: max-content;
76    height: max-content;
77    display: flex;
78    flex-direction: column;
79    align-items: center;
80  }
81
82  .box2 {
83    width: max-content;
84    height: 1100px;
85    display: flex;
86    flex-direction: column;
87    justify-content: space-between;
88    align-items: center;
89  }
90
91  .tempconteiner {
92    height: 990px;
93    width: 100%;
94    display: flex;
95    flex-direction: column;
```

```
96    justify-content: space-around;
97    align-items: center;
98  }
99
100 .card1, .sensor, .light{
101   font-size:medium;
102   color: #FDFDFD;
103   background-color: #22502480;
104   padding: 1em;
105   border-radius: 30px;
106   width: 75%;
107   margin: 1em;
108 }
109
110 .graphconteiner {
111   background-color: #22502480;
112   border-radius: 30px;
113   color: #FDFDFD;
114   width: 100%;
115   height: 600px;
116   margin-top: 0px;
117   display: flex;
118   flex-direction: column;
119   align-items: center;
120   justify-content: space-around;
121 }
122
123 .opcoes_graf {
124   display: flex;
125   flex-direction: row;
126   justify-content: space-between;
127   align-items: center;
128   width: 90%;
129 }
130
131 .graph {
132   width: 1000px;
133   height: 900px;
134   display: flex;
135   justify-content: space-around;
136   flex-wrap: wrap;
137 }
138
139 .graf {
140   background-color: #225024d0;
141   padding: 1em;
142   border-radius: 30px;
```

```
143   width: 450px;
144   height: 400px;
145   display: flex;
146   flex-direction: column;
147   justify-content:flex-start;
148 }
149
150 .grafico {
151   width: 450px ;
152   height:400px;
153   object-fit: contain;
154 }
155
156 .map_baixar{
157   width: 100%;
158   height: 450px;
159   margin-top: 0px;
160   display: flex;
161   flex-direction: row;
162   justify-content: space-around;
163 }
164 .mapconteiner{
165   background-color: #225024d0;
166   border-radius: 30px;
167   color: #FDFDFD;
168   width: 70%;
169   height: 400px;
170   margin-top: 0px;
171   display: flex;
172   flex-direction: column;
173   align-items: center;
174   justify-content:flex-start;
175 }
176 .map{
177   background-color: #225024d0;
178   padding: 1em;
179   border-radius: 30px;
180   width: 550px;
181   height: 400px;
182 }
183
184 #map {
185   height: 100%;
186 }
187 .baixar{
188   background-color: #225024d0;
189   border-radius: 30px;
```

```
190   color: #FDFDFD;
191   width: 20%;
192   height: 300px;
193   display: flex;
194   flex-direction: column;
195   align-items: center;
196 }
197
198 .botpad {
199   background-color: #225024;;
200   color: white;
201   padding: 14px 20px;
202   margin: 8px 0;
203   border: none;
204   cursor: pointer;
205   border-radius: 4px;
206 }
207 .botpad:hover {
208  opacity: 0.5;
209 }
210 .deletebtn{
211   background-color: #c52c2c;
212 }
213
214 @media screen and (max-width: 300px) {
215   .cancelbtn, .deletebtn {
216       width: 100%;
217   }
218 }
219
220 #tipo, #periodo{
221   background-color: #d3f1d4;
222   color: rgb(0, 66, 0);
223   width: 40%;
224 }
225
226 #resolucao, #medida{
227   background-color: #d3f1d4;
228   color: rgb(0, 66, 0);
229 }
230
231 #datemin, #datemax{
232   background-color: #d3f1d4;
233   color: rgb(0, 66, 0);;
234 }
235
236 #user-details
```

```
{
  color: #FDFDFD;
}
```

auth.js

```
document.addEventListener("DOMContentLoaded", function(){
    var UserState;
    // Espera por mudancas no estado de autenticacao
    auth.onAuthStateChanged(user => {
        //carrega nova pagina e muda o estado do usuario
        if (user) {
          setupUI(user);
          var uid = user.uid;
          UserState = db.ref('UsersData/' + uid.toString()+ '/Estado');
          UserState.set({ativo : true});
        } else {
          setupUI();
        }
    });

    // formulario de login e autentificacao do usuario
    const loginForm = document.querySelector('#login-form');
    loginForm.addEventListener('submit', (e) => {
        e.preventDefault();
        const email = loginForm['input-email'].value;
        const password = loginForm['input-password'].value;
        auth.signInWithEmailAndPassword(email, password).then((cred) => {
            loginForm.reset();
            console.log(email);
        })
        .catch((error) =>{
            const errorCode = error.code;
            const errorMessage = error.message;
            document.getElementById("error-message").innerHTML =
    errorMessage;
            console.log(errorMessage);
        });
    });

    // logout
    const logout = document.querySelector('#logout-link');
    logout.addEventListener('click', (e) => {
        e.preventDefault();
        UserState.set({ativo : false});
        auth.signOut();
    });
});
```

index.js

```
function epochToJsDate(epochTime){
    return new Date(epochTime*1000);
  }

  // converte epochtime para data e hora
  function epochToDateTime(epochTime){
    var epochDate = new Date(epochToJsDate(epochTime));
    var dateTime = ("00" + epochDate.getDate()).slice(-2) + "/" +
      ("00" + (epochDate.getMonth() + 1)).slice(-2) + "/" +
      epochDate.getFullYear() + " " +
      ("00" + epochDate.getHours()).slice(-2) + ":" +
      ("00" + epochDate.getMinutes()).slice(-2) + ":" +
      ("00" + epochDate.getSeconds()).slice(-2);

    return dateTime;
  }

  const loginElement = document.querySelector('#login-form');
  const contentElement = document.querySelector("#content-sign-in");
  const userDetailsElement = document.querySelector('#user-details');
  const authBarElement = document.querySelector('#authentication-bar');

  // Escutando para mudancas de estado de autenticacao
  const setupUI = (user) => {
    //Estado em que o usuario esta logado
    if (user) {
      loginElement.style.display = 'none';
      contentElement.style.display = 'block';
      authBarElement.style.display ='block';
      userDetailsElement.style.display ='block';
      userDetailsElement.innerHTML = user.email;

      var uid = user.uid;
      var dbPath = 'UsersData/' + uid.toString() + '/SHT';
      var dbRef = firebase.database().ref(dbPath);

      //Objeto referente ao sensor de temperatura e umidade
      let SHT = {
        'databaseSHT': db.ref('UsersData/' + uid.toString() + '/SHT/'),
        'databaseSHT_request': db.ref('UsersData/' + uid.toString() + '/
    Request/SHT_request'),
        'request' : true,
        read_data : function(){
          this.databaseSHT.orderByKey().limitToLast(1).on('child_added',
    snapshot =>{
            let jsonData = snapshot.toJSON(); // example: {temperature:
```

```
    25.02, humidity: 50.20, pressure: 1008.48, timestamp:1641317355}
45            let tempC = jsonData.TemperaturaC;
46            document.getElementById("SHT_tempC").innerHTML = tempC + "C";
47            let tempF = jsonData.TemperaturaF;
48            document.getElementById("SHT_tempF").innerHTML = tempF + "F";
49            let hum = jsonData.umidade;
50            document.getElementById("SHT_umidade").innerHTML = hum + "%";
51            let tempo = jsonData.timestamp;
52            document.getElementById("SHT_data").innerHTML = epochToDateTime
    (tempo);
53          });
54      },
55      getData: function() {
56        this.databaseSHT_request.set({SHT_request : this.request});
57      }
58    }
59
60    //Botao para solicitar dados do sensor de temperatura e umidade
61    const SHT_button = document.getElementById('SHT_button');
62    if(SHT_button)
63    SHT_button.addEventListener('click', function() {
64      SHT.getData();
65    });
66    setInterval(function () {SHT.getData()}, 500000);
67
68    //Objeto referente ao sensor de luminosidade
69    let Lightsensor = {
70      'databaseLight': db.ref('UsersData/' + uid.toString() + '/BHT1750/'
    ),
71      'databaseLight_request': db.ref('UsersData/' + uid.toString() + '/
    Request/BH1750_request'),
72      'request' : true,
73      read_data : function(){
74        this.databaseLight.orderByKey().limitToLast(1).on('child_added',
    snapshot => {
75          let jsonData = snapshot.toJSON();
76          let lum = jsonData.Lux;
77          document.getElementById("Lux").innerHTML = lum + "Lux";
78          let tempo = jsonData.timestamp;
79          document.getElementById("Light_data").innerHTML =
    epochToDateTime(tempo);
80
81          if (parseInt(lum) < 100) {
82            document.body.style.backgroundImage = "url('https://images.
    unsplash.com/photo-1503428009254-8d4f29217f23?ixlib=rb-4.0.3&ixid=
    MnwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8&auto=format&fit=crop&w=870&
    q=80')";
```

```
83            } else {
84              document.body.style.backgroundImage = "url('https://images.
    unsplash.com/photo-1563514227147-6d2ff665a6a0?ixlib=rb-1.2.1&ixid=
    MnwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8&auto=format&fit=crop&w
    =1471&q=80')";
85            }
86          })
87        },
88        getData: function() {
89          this.databaseLight_request.set({BH1750_request : this.request});
90        }
91      }
92
93      //Botao para solicitar dados do sensor de luminosidade
94      const BH1750_button = document.getElementById('BH1750_button');
95      if(BH1750_button)
96      BH1750_button.addEventListener('click', function() {
97        Lightsensor.getData();
98      });
99      setInterval(function () {Lightsensor.getData()}, 500000);
100
101     //Objeto referente a api do openweather
102     let meteorologia = {
103       "apiKey": "apikey",
104       "databaseGPS": db.ref('UsersData/' + uid.toString() + '/GPS/'),
105       read_GPS: function() {
106         this.databaseGPS.orderByKey().limitToLast(1).on('child_added',
    snapshot =>{
107           let jsonData = snapshot.toJSON();
108           console.log(jsonData);
109           let lat = jsonData.LAT;
110           let lng = jsonData.LNG;
111           this.fetchWeather(lat, lng);
112         });
113       },
114       fetchWeather: function(latitude, longitude) {
115           fetch("https://api.openweathermap.org/data/2.5/weather?lat=" +
    latitude+"&lon="+longitude+"&units=metric&lang=pt&appid=" + this.apiKey
116           ).then((response) => response.json())
117           .then((data) => this.displayWeather(data));
118       },
119       displayWeather: function(data) {
120           const {name} = data;
121           const {icon, description} = data.weather[0];
122           const {temp, humidity} = data.main;
123           const {speed} = data.wind
124
```

```
125             const {sunrise} = data.sys;
126             let nascente = new Date(0);
127             nascente.setUTCSeconds(sunrise);
128             let nascer = nascente.toLocaleTimeString([], {hour: '2-digit',
    minute:'2-digit'});
129 
130             const {sunset} = data.sys;
131             let poente = new Date(0);
132             poente.setUTCSeconds(sunset);
133             let por = poente.toLocaleTimeString([], {hour: '2-digit',
    minute:'2-digit'});
134 
135            // console.log(name, icon, description, temp, humidity, speed);
136             document.querySelector('.cidade').innerText = "Clima em " +
    name;
137             document.querySelector('.icon').src = "http://openweathermap.
    org/img/wn/"+icon+".png";
138             document.querySelector('.descricao').innerText = description.
    charAt(0).toUpperCase() + description.slice(1).toLowerCase();;
139             document.querySelector('.temp').innerText = temp + 'C';
140             document.querySelector('.umidadeSat').innerText = "Umidade: " +
     humidity + "%";
141             document.querySelector('.vento').innerText = "Vento: " + speed
    + " km/h";
142             document.querySelector('.nascer').innerText = "Nascer do sol: "
     + nascer;
143             document.querySelector('.por').innerText = "Por do sol: " + por
    ;
144         },
145       }
146 
147       //Objeto referente a api do google maps
148       let localizacao = {
149         "databaseGPS": db.ref('UsersData/' + uid.toString() + '/GPS/'),
150         read_GPS: function() {
151           this.databaseGPS.orderByKey().limitToLast(1).on('child_added',
    snapshot =>{
152             let jsonData = snapshot.toJSON();
153             console.log(jsonData);
154             let lat = parseFloat(jsonData.LAT);
155             let lng = parseFloat(jsonData.LNG);
156             this.initMap(lat, lng);
157 
158           })
159         },
160         initMap: function(lat, lng){
161         var myLatLng = {lat: lat, lng: lng};
```

```
162         var map = new google.maps.Map(document.querySelector('.map'), {
163           zoom: 12,
164           center: myLatLng
165         });
166 
167         var marker = new google.maps.Marker({
168           position: myLatLng,
169           map: map,
170           title: 'Meu local'
171         });
172         }
173       }
174 
175       //Data min e data max do dashboard
176       const datmin = document.querySelector('#datemin');
177       const datemax = document.querySelector('#datemax');
178       const submit = document.querySelector('#submit');
179 
180       const temperaturaGraf = document.querySelector('#temperaturaGraf');
181       const umidadeGraf = document.querySelector('#umidadeGraf');
182 
183       //Atualizacao dos graficos
184       if(submit)
185       submit.addEventListener('click', function() {
186         const medida = document.getElementById('medida').value;
187         const resolucao = document.getElementById('resolucao').value;
188         temperaturaGraf.setAttribute('src', "https://agriot-9b824.rj.r.
    appspot.com/plots/temp?datemin="+datmin.value+"&datemax="+datemax.value+
    "&medida="+medida+"&resolucao="+resolucao);
189         umidadeGraf.setAttribute('src', "https://agriot-9b824.rj.r.appspot.
    com/plots/umid?datemin="+datmin.value+"&datemax="+datemax.value)
190       });
191 
192       //Download dos dados
193       const download = document.querySelector('#download');
194       if(download)
195       download.addEventListener('click', function() {
196         const tipo = document.getElementById('tipo').value;
197       const periodo = document.getElementById('periodo').value;
198         window.open("https://agriot-9b824.rj.r.appspot.com/plots/download?
    formato="+tipo+'&periodo='+periodo);
199       });
200 
201       //Requisicao das leituras
202       SHT.read_data();
203       Lightsensor.read_data();
204       meteorologia.read_GPS();
```

```
205       localizacao.read_GPS();
206
207     // Estado em que o usuario fez log out
208     } else{
209       loginElement.style.display = 'block';
210       authBarElement.style.display ='none';
211       userDetailsElement.style.display ='none';
212       contentElement.style.display = 'none';
213     }
214   }
```