UNIVERSIDADE FEDERAL DE SANTA CATARINA

Sistema de gerenciamento de coleta de dados laboratoriais por Arduinos

Caetano Sasia dos Santos

Vinícius Soares Laghi

Florianópolis - SC

2023/1

UNIVERSIDADE FEDERAL DE SANTA CATARINA

DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

CURSO DE SISTEMAS DE INFORMAÇÃO

Sistema de gerenciamento de coleta de dados laboratoriais por Arduinos

Caetano Sasia dos Santos

Vinícius Soares Laghi

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Sistemas de Informação.

Florianópolis - SC

2023/1

Caetano Sasia dos Santos

Vinícius Soares Laghi

Sistema de gerenciamento de coleta de dados laboratoriais por Arduinos

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Sistemas de Informação

Banca Examinadora:

_____

Prof. Alex Sandro Roschildt Pinto

Orientador

_____

Prof. Jose Eduardo de Lucca

Avaliador

_____

Prof. Frank Augusto Siqueira

Avaliador

AGRADECIMENTOS

Agradecemos aos nossos familiares por todo o apoio durante essa jornada em uma nova graduação, ao nosso orientador professor Alex por aceitar nos orientar nesse projeto e aos membros da banca examinadora.

RESUMO

A Internet das Coisas tem sido cada vez mais usada para a automatização de sistemas. O uso de dispositivos Arduino permite a construção de sistemas automatizados de baixo custo que servem como uma substituição barata de sistemas comerciais caros e muitas vezes inacessíveis. Neste projeto foi construído um sistema automatizado de de armazenamento de dados coletados por um Arduino conectado a sensores para ser utilizado em laboratórios de pesquisa, utilizando-se Arduino para ser utilizado em laboratórios com um sistema de autenticação para melhor segurança. O sistema desenvolvido utiliza-se da tecnologia MERN para o desenvolvimento do back end e aplicação Web. O resultado final foi um sistema que permite a inserção de novos sensores no Arduíno sem a necessidade de alteração no código do back end para acomodar o novo sistema de dados enviados, com exibição dos gráficos com os dados em tempo real na aplicação Web além de possibilitar a extração dos dados em um arquivo csv.

Palavras-chave: Arduino, Internet das Coisas, IoT, MERN, sensores

ABSTRACT

The Internet of Things has become popular in its use for building automated systems. The use of Arduino devices allows for automated systems to be built with a low cost, working as a cheap alternative for commercial systems that are often too expensive. In this project we built an automated system that collects data from sensors connected to an Arduino device to be used in research laboratories, with an authentication system for better security. The system uses MERN technology for the development of the back-end and the web application. The final result was a system that allows connecting new sensors to the Arduino without the need to do any change in the back end to accommodate the new data format being sent, with real time charts being displayed in the web application, with the possibility of extracting the available data in a csv file.

Keywords: Arduino, Internet of Things, IoT, MERN, sensors.

LISTA DE FIGURAS

SUMÁRIO

1. Introdução

A Internet das Coisas (IoT) é uma tecnologia emergente, nas quais dispositivos ao nosso redor conseguem se comunicar através do uso da internet (BEHERA, GUPTA, *et al.,* 2015). A popularização dessa tecnologia só foi possível graças aos avanços tecnológicos de áreas como a Informática, a Internet e a Eletrônica (VANELLI, DA SILVA, *et al.,* 2017).

O uso de dispositivos IoT permite a solução de diversos problemas existentes, dentre eles a automatização de sistemas manuais. Devido a isso, essa tecnologia é promissora para a automatização de sistemas em laboratórios de pesquisa, principalmente na área biotecnológica, que utilizam sensores que são fundamentais aos seus processos, que normalmente não estão amplamente disponíveis em laboratórios devido ao seu custo alto de aquisição e manutenção (CARVALHO, 2022).

Carvalho (2022) desenvolveu um sistema IoT automatizado utilizando arduinos conectados a sensores, como uma solução para automatização de coleta de dados utilizados no cultivo de microalgas, apresentando uma economia de 95% em seu custo baseado em um sistema comercial similar. Apesar de funcional e eficiente para aquele processo, o sistema desenvolvido possui limitações quanto a adição ou remoção de sensores, uso paralelo em outros experimentos, falta de segurança entre outros fatores.

Baseado nesse projeto de Carvalho (2022), o objetivo deste trabalho é o desenvolvimento de um novo sistema automatizado de coleta de dados enviados por Arduinos, com uma maior flexibilidade permitindo adição e remoção de sensores sem a necessidade de alterar o banco de dados para suportar mudanças, melhorias em relação a segurança através de um sistema de autenticação e token para envio

dos dados pelo Arduino e que permite que dados de diferentes experimentos sejam coletados em paralelo, permitindo que diferentes experimentos sejam automatizados simultaneamente.

1.1. “INTERNET OF THINGS EM BIOTECNOLOGIA: ARDUINO NO CULTIVO DE MICROALGAS” por Carvalho (2022).

Para realizar o cultivo de microalgas é necessário fazer o acompanhamento das condições em que elas se encontram. Este acompanhamento é importante para obter um melhor desempenho em seu crescimento e multiplicação. Durante o acompanhamento diversos fatores devem ser levados em consideração, como concentração, presença ou ausência de elementos nutricionais e fatores externos. Dentre alguns dos fatores necessários para o crescimento adequado temos o pH, a temperatura, turbidez da água e luminosidade. Normalmente são necessárias medições manuais diárias, porém Carvalho (2022) desenvolveu um sistema com o uso de arduinos com tecnologia IoT para a coleta automatizada a partir de sensores e um protoboard, além de uma placa NodeMCU ESP8266 para acesso à internet.

Além do arduíno, a autora também desenvolveu um sistema Web, o qual consiste em uma API feita em NodeJs para receber os dados e inseri-los em um banco de dados Postgres. Esta mesma API é responsável por disponibilizar os dados para um front-end feito em ReactJs e que utiliza a biblioteca gráfica Echarts (Baidu) para visualização dos dados. Todo esse sistema foi implementado na plataforma de nuvem Heroku e a interface ficou disponível para acesso livre.

As medidas coletadas pelo sistema desenvolvido se mostraram fiéis às mesmas coletadas manualmente, com a vantagem de poder fazer múltiplas coletas

ao longo do dia de acordo com as configurações necessárias no sistema. Outro ponto positivo observado pela autora, foi a economia do custo desse sistema quando comparado ao uso de ferramentas comerciais para medições automatizadas em laboratórios, apresentando uma economia de 95% quando comparado o custo das peças do sistema e o custo de equipamentos comerciais para realizar as mesmas medições automaticamente.

Apesar das diversas vantagens apresentadas pela autora, foram apontadas algumas melhorias a serem feitas ao sistema, como segurança na comunicação do sistema Web e segurança para acesso aos dados do experimento pela interface. Além disso, a estrutura utilizada no banco de dados não permite a adição ou remoção de novos sensores sem a necessidade de alterar o esquema no banco de dados.

2. Objetivos

2.1 Objetivos Gerais

Desenvolver um sistema que permita o armazenamento automático de dados coletados através de sensores conectados a arduinos.

2.2 Objetivos Específicos

- Desenvolver uma interface web que permita a visualização e download dos dados armazenados no banco de dados;
- Desenvolver um sistema back-end que suporte a conexão de novos sensores automaticamente;

- Desenvolver um sistema de autenticação para acesso aos dados dos sensores, que forneça um token único na interface web, necessário para o envio de dados pelos arduinos para o banco de dados;
- Desenvolver um sistema de autenticação de usuário para acesso à interface web

3. Levantamento Bibliográfico

O uso de dispositivos Arduino conectado a sensores para automatização da coleta de dados é amplamente utilizado em diversos sistemas diferentes, como na agricultura (ARDIANSAH *et al.*, 2020; ARVINDAN *et al.,* 2016), monitoramento do meio ambiente (FERDOUSH *et al.*, 2014), ou até mesmo para monitoramento da potabilidade da água em sistemas de filtragem automático (IRAWAN *et al.*, 2021), comprovando a eficiência desses dispositivos para monitoramento remoto de sensores. Entretanto, muitos desses sistemas contam como um servidor local, como por exemplo um Raspberry Pi (FERDOUSH *et al.*, 2014), dentro do laboratório de pesquisa, para armazenamento e envio dos dados para a internet (Figura 1) (GOLINELLI *et al.*, 2018). A fim de simplificar e baratear o custo do sistema, foi optado por utilizar um servidor em nuvem para armazenamento e envio dos dados no sistema desenvolvido.

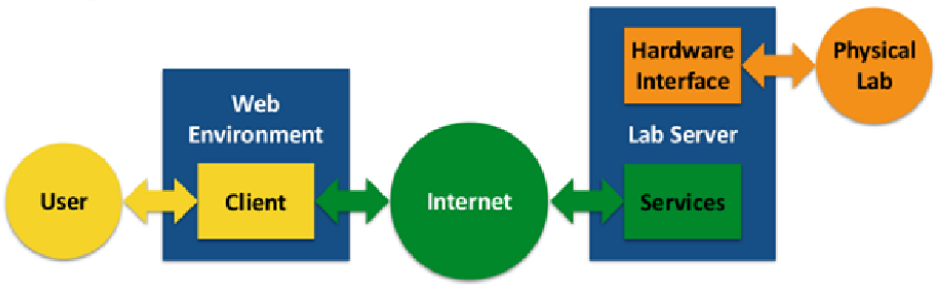

Figura 1 - Arquitetura de laboratórios remotos. Fonte: SALZMANN *et al.* (2015).

Em relação ao desenvolvimento da aplicação Web para o monitoramento remoto dos sensores, o uso da tecnologia MERN (Mongo, Express, React, NodeJS) tem sido amplamente utilizado para desenvolvimento de dashboards para monitoramento em tempo real de dados (LIU, 2021; KHUE *et al*., 2017). Esta tecnologia permite a criação de aplicações que realizam atualizações assíncronas e em tempo real, além de serem altamente escaláveis (KHUE *et al*., 2017), o que a torna ideal para o desenvolvimento do sistema proposto.

4. Fundamentos Tecnológicos

4.1. Internet das Coisas

Dispositivos com tecnologia IoT possuem um potencial em processar e arquivar um grande volume de dados, o que gera um problema em relação ao armazenamento dos mesmos (VANELLI, DA SILVA, *et al.,* 2017). Uma das categorias de dispositivos que apresentam esse potencial são os dispositivos de sensores, que se beneficiam fortemente da tecnologia IoT (BEHERA, GUPTA, *et al.,* 2015). A fim de sanar esse problema no armazenamento dos dados produzidos, é possível utilizar a tecnologia da computação em nuvem (VANELLI, DA SILVA, *et al.,* 2017).

Um dos dispositivos mais utilizados para desenvolvimento de sistemas IoT é o Arduino, que é um microcontrolador barato de código aberto desenvolvido em 2005 (LOUIS, 2016).

4.1.1 Arduino

Arduino é uma plataforma eletrônica de código aberta, onde uma placa eletrônica é capaz de ler ou receber dados de entrada, através de acessórios conectados como sensores ou botões e os e torná-lo em um sinal de saída, podendo ser uma postagem online, ativação de um motor, entre outros (ARDUINO.CC, 2023).

O Arduino Uno é o modelo utilizado como referência para os modelos mais modernos, pois foi o primeiro a ser lançado com suporte USB além de ser lançado juntamente com a versão 1.0 do software de desenvolvimento do Arduino (Arduino IDE). Esse modelo possui 14 pinos de entrada e saída, 6 entradas analógicas, 1 *header* ICSP, 1 ressonador cerâmico de 16 MHz, 1 conector de alimentação, 1 conector USB, 1 botão de reset e um microcontrolador ATmega328 (Figura 2) (LOUIS, 2016; ARDUINO.CC, 2023)

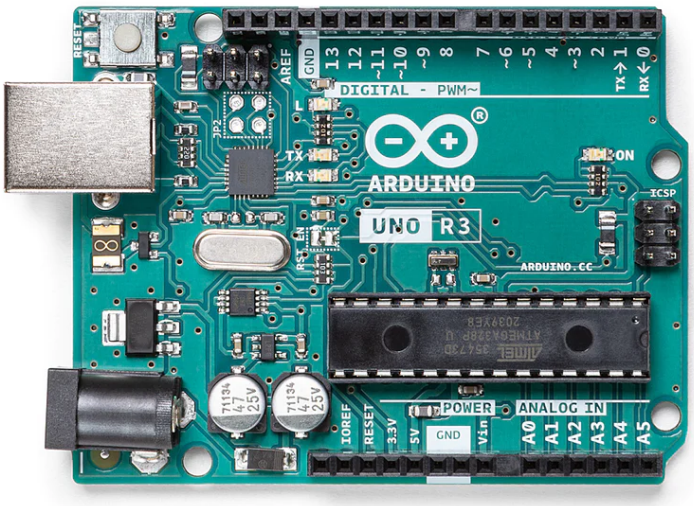

Figura 2 - Arduino Uno. (Fonte: Arduino.cc)

O software de desenvolvimento (Arduino IDE) é publicado como uma ferramenta de código aberto baseado em C, que permite a instalação de bibliotecas em C++ (ARDUINO.CC, 2023). O código é escrito usando uma forma simplificada da linguagem C++, e possui duas principais partes em sua estrutura: o void setup() e o void loop(). O void setup() é a primeira parte do código a ser lida ao inicializar o Arduino, essa parte do código será lida apenas uma vez e serve para inicializar todos os pinos que serão utilizados no programa como entrada ou saída. Posteriormente inicia-se o void loop(), que corresponde à parte do código que irá rodar continuamente e utiliza os pinos configurados durante o void setup() (LOUIS, 2016).

No sistema desenvolvido, o Arduino é responsável pela leitura e coleta de dados contínua através de sensores conectados ao mesmo, enviando-os para o back-end a cada leitura realizada através de um *post* que possui token de validação único previamente gerado no sistema em seu *header*. O uso do modelo Uno se dá ao fato de ser o modelo base utilizado para os modelos mais modernos, sendo de simples uso e ao mesmo tempo atendendo todas as necessidades do sistema.

4.1.2 Sensores utilizados

Para os testes com o sistema desenvolvido foram utilizados um sensor de luminosidade fotoresistor LDR e um o sensor de temperatura DS18B20 à prova d'água de 2 metros.

4.2. Tecnologia MERN

O uso de tecnologia MERN (MongoDB, Express, React, Nodejs) para desenvolvimento de aplicativos WEB tem se tornado cada vez mais popular, pois o

uso dessa tecnologia permite criar uma aplicação de alta performance, que realiza atualizações em tempo real de forma assíncrona e facilmente escalável (KHUE, *et al.,* 2017). O uso do React para o desenvolvimento do front-end gera uma maior eficiência no projeto devido ao uso de um DOM virtual por parte do React, o que minimiza a operação do DOM (LIU, 2021). Além disso, a separação entre front-end e back-end (Figura 3) permite uma comunicação entre os dois através de chamadas AJAX, onde o back-end envia os dados em formato json, reduzindo a demanda em cima do back-end no sistema (LIU, 2021).

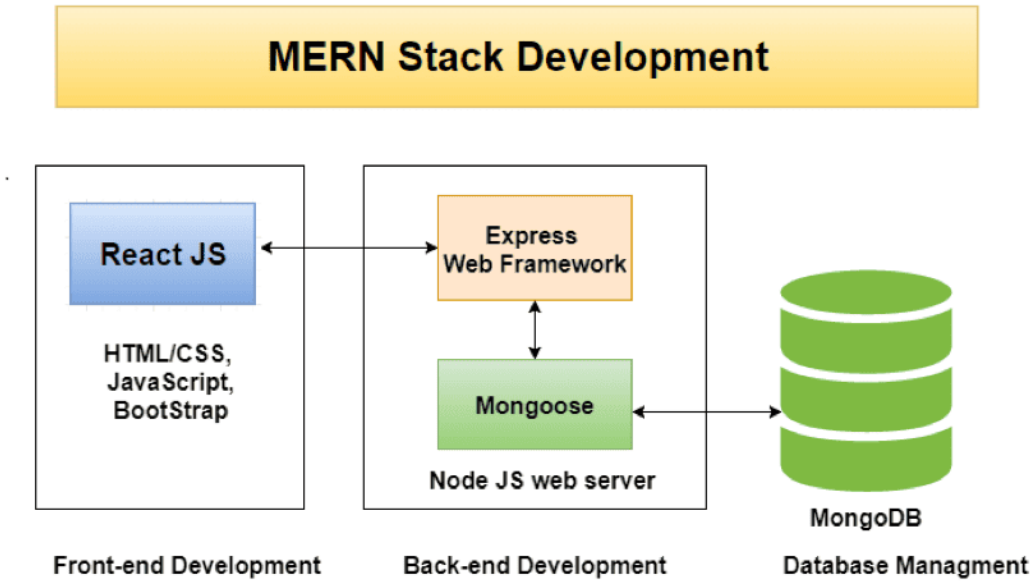

Figura 3 - Diagrama exemplificando tecnologia MERN. Fonte: https://www.bocasay.com/how-does-the-mern-stack-work/

4.3 Back-end

Souto (2023) define back-end como “aquilo que tem por trás de uma aplicação”, trabalhando como uma ponte entre os dados do banco de dados e as aplicações ou navegadores, aplicando regras de negócios, validações e outras regras pertinentes ao sistema.

4.3.1 NodeJS

O NodeJS é um ambiente de execução single thread para javascript, construído a partir da engine V8 do Google (Krol et. al., 2015). O NodeJS funciona de forma assíncrona além de ser orientado a eventos (Krol et. al., 2015), o que permite o processamento de várias solicitações em paralelo.

O uso de NodeJS permite que a aplicação inteira, tanto o front-end quanto o back-end seja desenvolvida inteiramente usando a mesma linguagem de programação (javascript), o que foi um dos pontos que foi levado em consideração para a escolha do mesmo para o desenvolvimento do back-end. Krol et. al. (2015) também classifica NodeJS como ideal para aplicações com requisições em tempo real, que requerem poucos ciclos de CPU para o processamento das requisições, e que também suporta grandes quantidades de conexões em paralelo, tornando-o ideal para esse projeto.

4.3.2 MongoDB

O MongoDB é um banco de dados não relacional de código aberto, que utiliza JSON para o armazenamento e retirada de dados, o que o torna ideal para o

uso com Javascript pois dessa forma é possível utilizar o mesmo formato de dados do cliente ao servidor e eventualmente ao banco de dados (Krol et. al., 2015).

Outro ponto importante do MongoDB, é o que por ser um banco de dados não relacional, os dados são armazenados sem um esquema, ou seja, a quantidade de colunas, tamanho e tipo de dado de cada campo não precisa ser definido de antemão, e novos campos com definições diferentes podem ser adicionados ao longo do projeto (Krol et. al., 2015).

A facilidade do uso com o Javascript e a natureza não relacional do MongoDB foram dois critérios importantes para a escolha desse banco de dados para o sistema, pois dessa forma novos sensores podem ser adicionados a qualquer momento no projeto sem necessidade de alterar a estrutura do banco de dados para acomodar os mesmos.

4.3.3 Express

O site Expressjs.com (2023) define Express como um “framework de aplicação web para NodeJS que fornece um conjunto de recursos robustos para aplicações web e mobile”. O Express é um framework que essencialmente recebe solicitações HTTP e retorna solicitações HTTP, o que o torna flexível. Além disso, por padrão o Express fornece inicialmente o mínimo possível de framework e funcionalidades para sua aplicação, e novas funcionalidades podem ser adicionadas de acordo com a necessidade e complexidade do projeto (BROWN, 2020).

O uso do Express no sistema auxiliou a execução e controle das funções do back-end, de acordo com as requisições realizadas pela aplicação web.

4.3.4 PostgreSQL

PostgreSQL é um sistema de gerenciamento de banco de dados que permite o uso de diferentes linguagens de programação para o desenvolvimento de funções e procedimentos como C, SQL, Python, Javascript, entre outros (OBE & HSU, 2018). Uma outra característica forte do PostgreSQL é o fato de além de possuir um grande suporte para diferentes tipos de dados existentes como números inteiros, texto, Booleans, ele também permite a criação e definição de novos tipos de dados para atender as necessidades do sistema (OBE & HSU, 2018).

O PostgreSQL foi utilizado para a criação da tabela de usuários no sistema desenvolvido, utilizando o email cadastrado como chave primária da tabela, complementando o MongoDB no banco de dados do projeto.

4.3.5 Mongoose

Mongoose é uma ferramenta para modelagem de objetos para MongoDB e NodeJS, que permite a definição do modelo de dados utilizado no projeto dentro do próprio código, em um único lugar, através da criação de um schema em formato JSON (HOLMES, 2013).

4.3.6 Knex.js

O Knex.js é um construtor de queries SQL para PostgreSQL, CockroachDB, MSSQL, MySQL, MariaDB, SQLite3, Better-SQLite3, Oracle, e Amazon Redshift (KNEXJS.ORG, 2023). A função do Knex.js no projeto foi gerenciar o PostgreSQL, criar as *migrations* para as tabelas de usuário e experimentos e tokens de verificação de email para ser consumido.

4.3.7 Autenticação e segurança

A geração de tokens para confirmação do e-mail utilizado no cadastro, validação do login e para o header dos dados enviados pelo Arduino foi feita utilizando o jsonwebtoken. As senhas são criptografadas para serem salvas no banco de dados utilizando a biblioteca node.bcrypt.js.

4.4 Front-end

Souto (2023) define front-end como a parte visual que conseguimos interagir de um sistema. No caso do sistema desenvolvido, o front-end se resume à aplicação web visualizada pelo navegador, com os dados coletados pelo arduíno e os gráficos desenvolvidos com o mesmo.

4.4.1 ReactJS

O ReactJS é uma biblioteca JavaScript mantida pelo Facebook, usada para desenvolver interfaces de usuário interativas. Ele permite criar componentes independentes e reutilizáveis, facilitando a manutenção do código. Com o uso do Virtual DOM, o ReactJS atualiza apenas as partes necessárias da interface, melhorando o desempenho. Sua abordagem declarativa simplifica o desenvolvimento, descrevendo como a interface deve ser exibida com base no estado da aplicação (REACT.DEV, 2023).

4.4.2 Apache eCharts

O Apache Echarts é uma biblioteca JavaScript de visualização de dados de código aberto. Com o Echarts, é possível criar gráficos interativos e personalizados para exibir dados de maneira clara. Ele é flexível, compatível com diversas

plataformas e capaz de lidar com grandes volumes de dados. O Echarts é amplamente utilizado por desenvolvedores que desejam apresentar informações de forma visualmente atraente e compreensível. (ECHARTS.APACHE.ORG, 2023)

5. Desenvolvimento do sistema

O sistema desenvolvido consiste de uma interface web (front-end), acessível pelo usuário através de uma interface de login, conectada a um back-end que controla o fluxo de dados armazenado no banco de dados, e que além de enviar dados para o front-end também recebe dados oriundo de arduinos conectados à internet. O sistema web foi desenvolvido através do uso da tecnologia MERN, além de um banco de dados relacional, sendo ele o PostgreSQL.

A aplicação web foi hospedada utilizando o Heroku, disponível no site https://tcc-vini-cae.herokuapp.com/, e o banco de dados utilizado foi o AWS Dynamo, com isso fechando a arquitetura geral do sistema (Figura 4).

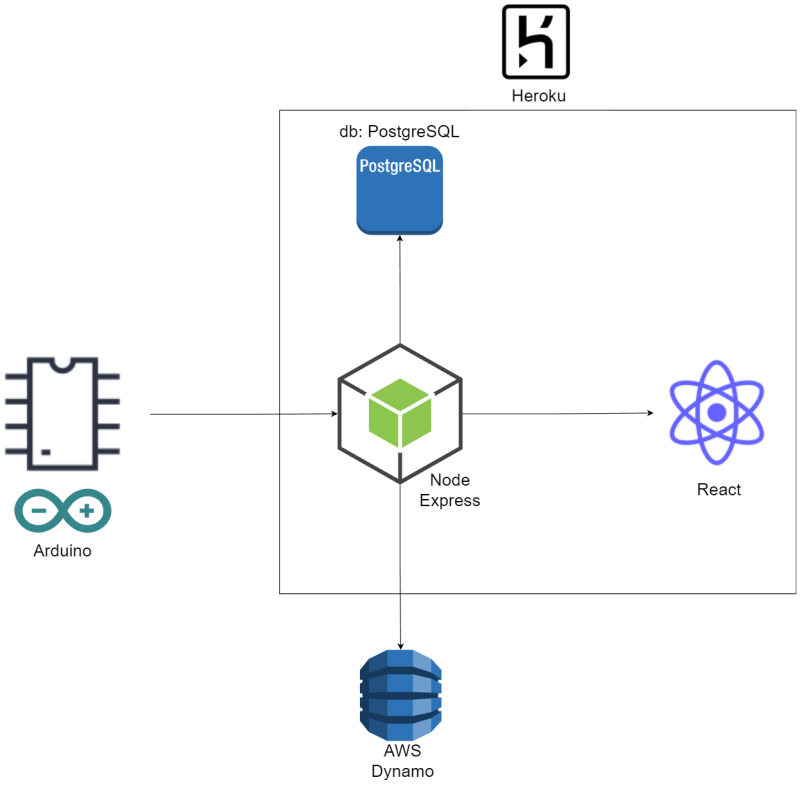

Figura 4 - Arquitetura da aplicação desenvolvida para o projeto, mostrando a integração de componentes. O ícone do Arduino representa o dispositivo físico utilizado. O ícone do Node.js representa o back-end da aplicação, enquanto o ícone do DynamoDB representa o banco de dados NoSQL. O ícone do PostgreSQL representa o banco de dados relacional. O ícone do React representa a interface de usuário. O quadrado indica que o React, Node.js e PostgreSQL estão hospedados no Heroku.

5.1 Hospedagem do sistema

Parte da aplicação foi hospedada na plataforma de nuvem Heroku. Essa plataforma fornece um ambiente de hospedagem simplificado e eficiente para implantar tanto o back-end quanto o front-end da aplicação. Ao optar por hospedar a aplicação no Heroku, conseguimos aproveitar os benefícios de uma implantação simplificada e escalabilidade oferecidos pela plataforma. Além disso, o Heroku

também disponibiliza o PostgreSQL como um serviço de banco de dados, tornando a integração entre a aplicação e o banco de dados mais fácil e eficiente. Em resumo, a escolha de hospedar o back-end em Node.js com Express, o front-end em React e utilizar o PostgreSQL fornecido pelo Heroku proporcionou uma configuração integrada e eficiente para a aplicação. Essa abordagem facilitou a implantação, escalabilidade e gerenciamento dos componentes da aplicação na plataforma de nuvem Heroku.

Foi utilizado o serviço MongoDB Cloud para armazenar e gerenciar os dados dos experimentos de forma dinâmica por meio de um banco de dados NoSQL. Além disso, foi integrado esse banco de dados com o DynamoDB, um serviço da AWS (Amazon Web Services). O MongoDB Cloud é uma plataforma de banco de dados NoSQL baseada em documento, conhecida por sua flexibilidade e escalabilidade. Optamos por utilizar essa solução devido à sua capacidade de lidar com dados não estruturados ou semiestruturados, oferecendo um modelo de dados mais flexível para a aplicação.

Por sua vez, o DynamoDB é um serviço de banco de dados NoSQL totalmente gerenciado fornecido pela AWS. Ele é conhecido por sua escalabilidade automática, alta disponibilidade e desempenho rápido. Integramos o MongoDB Cloud com o DynamoDB para aproveitar as vantagens de ambos os serviços e obter uma solução de armazenamento de dados robusta e eficiente. Essa abordagem permitiu que utilizássemos o MongoDB Cloud como uma camada de acesso aos dados, enquanto o DynamoDB fornecia o armazenamento e a escalabilidade necessários para lidar com grandes volumes de informações. Ao integrar o MongoDB Cloud com o DynamoDB, conseguimos aproveitar as vantagens de cada serviço. O MongoDB Cloud ofereceu uma interface amigável e recursos avançados

para manipular os dados, enquanto o DynamoDB garantiu a escalabilidade horizontal e o desempenho necessário para atender às demandas da nossa aplicação. Em resumo, a combinação do MongoDB Cloud e do DynamoDB nos permitiu utilizar um banco de dados NoSQL flexível e escalável em conjunto com um serviço altamente disponível e de alto desempenho da AWS. Essa integração proporcionou uma solução robusta e eficiente para o armazenamento e gerenciamento de dados no contexto deste projeto.

5.2 Cadastro e criação de novos experimentos

A aplicação proposta possui uma interface de login que oferece ao usuário três opções: acesso à tela inicial, criação de uma nova conta ou recuperação de senha (Figura 5). Ao selecionar a opção de criação de conta, uma tela é exibida contendo um formulário no qual o usuário deve preencher seu nome, email e senha desejada. No caso de seleção da opção de recuperação de senha, a tela exibida apresentará um campo para inserção do email. Se o email estiver registrado no banco de dados, um email contendo um link (token composto por uma rota do back-end e um token a ser consumido) será enviado ao usuário, permitindo que ele realize a troca de senha.

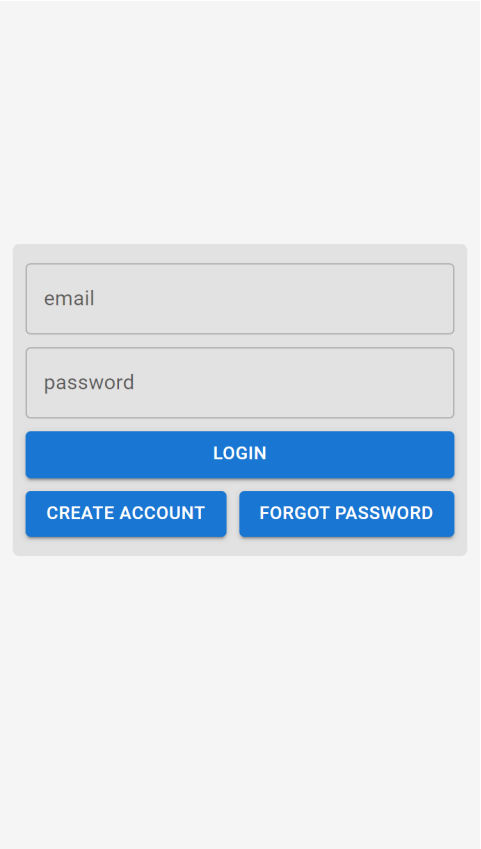

Figura 5 - Tela inicial de login da aplicação. Fonte: Criação própria

Após efetuar o login, o usuário será apresentado a uma interface contendo seu nome, um botão de logout e um botão adicional com um sinal de adição ("+"), destinado ao registro de novos experimentos. Ao clicar no botão de cadastro de experimento, o usuário será direcionado a um formulário onde deverá preencher o nome e uma descrição do experimento. Ao concluir o cadastro, o experimento será exibido na interface inicial, logo abaixo do botão de cadastro de experimentos (Figura 6). Além disso, o usuário receberá um e-mail contendo um token, o qual será utilizado pelo dispositivo Arduino para o envio dos dados correspondentes ao experimento. Ainda nesta interface, o usuário terá a possibilidade de (i) deletar o experimento (ii) realizar o download dos dados do experimento em um arquivo csv

(iii) reenviar para o seu e-mail o token que deve ser utilizado com o arduino (iv) botão para ver os detalhes do experimento.

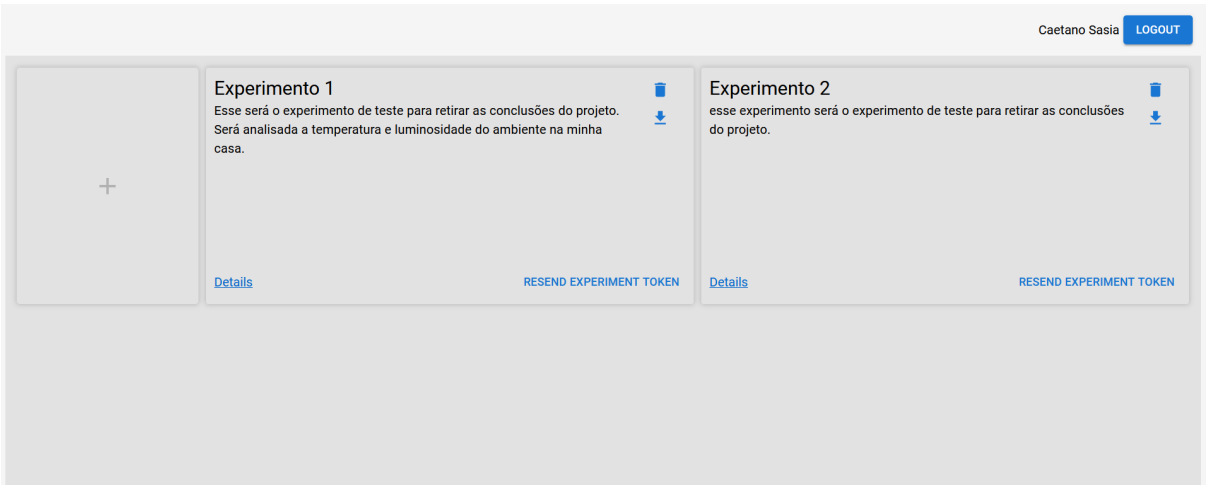

Figura 6 - Tela de gerenciamento de experimentos. Fonte: Criação própria

Ao clicar para ver os detalhes do experimento, o usuário será direcionado para uma nova tela. Caso o experimento ainda não possua dados será mostrada uma mensagem avisando que o experimento ainda não possui dados. Se o experimento já tiver dados, será mostrado um gráfico de linha para cada sensor que enviou dados naquele experimento (Figura 7).

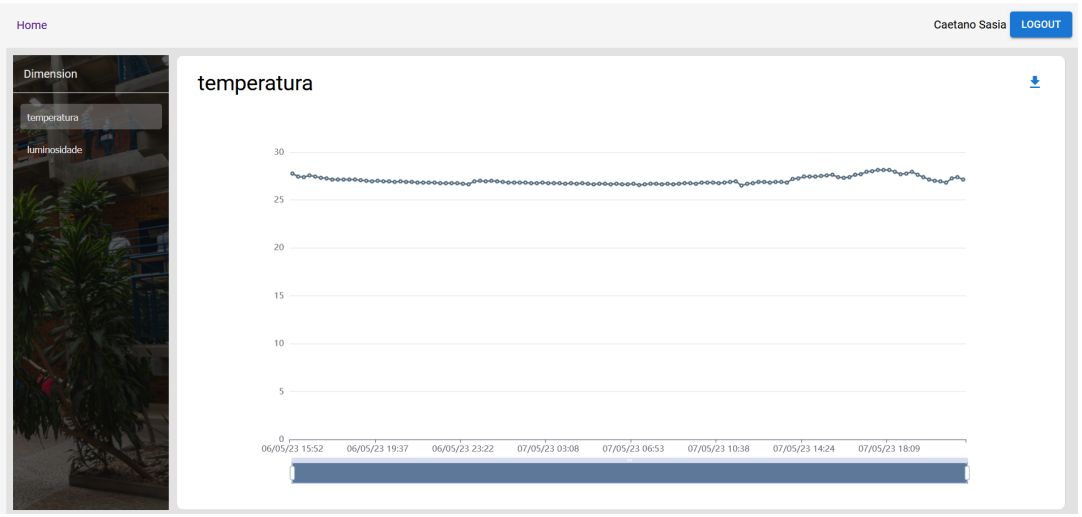

Figura 7 - Gráfico gerado com dados coletados do Arduino com intervalos de 15 minutos. Fonte: Criação própria

5.3 Coleta de dados

Após ter o token da aplicação (enviado por e-mail), esse token deve ser incorporado no Header do json que o arduino enviará, junto com os outros dados, por método HTTPS POST na rota: “/insert-data”. Ao receber o método post o back-end chama o controlador dos dados e verifica o token recebido na requisição. Caso o token seja válido, será extraída do mesmo a informação do id do experimento e e-mail do usuário. Se o experimento não for válido, apenas retorna um erro. Validando o id do experimento e e-mail do usuário, realizamos a extração das keys do objeto com os dados para serem inseridos como uma nova linha no banco não relacional.

O schema criado para receber os dados do Arduino é composto de um objeto chamado ‘data’, onde os dados dos sensores chegam permitindo o dinamismo do sistema, seguido do email cadastrado (utilizado pelo PostgreSQL como chave primária da tabela de usuários) e do ID do experimento (utilizado pelo MongoDB como chave primária da tabela de experimentos), e por fim um objeto chamado updatedAt com a hora em que foi recebido os dados (Figura 8). Os dados dos sensores no objeto ‘data’ devem seguir o formato chave-valor, onde a chave é o nome do sensor e o valor é a medição realizada no momento.

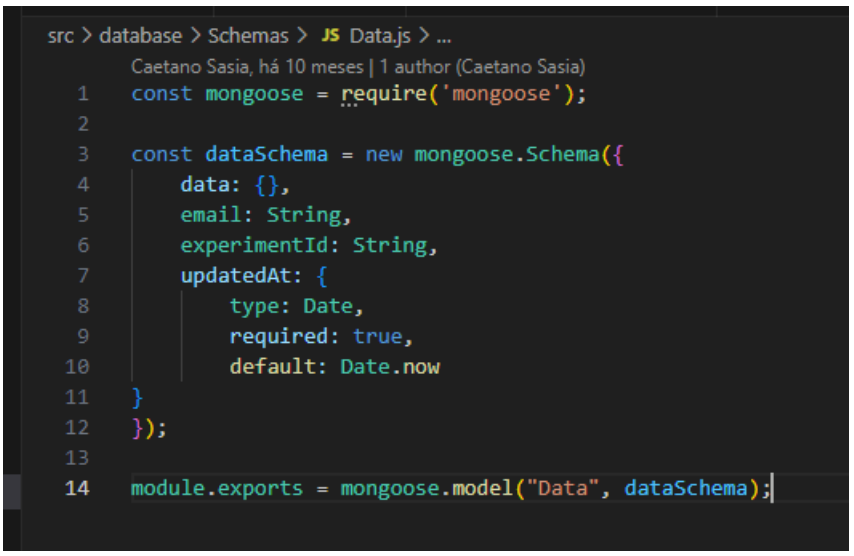

Figura 8 - Schema utilizado pelo Mongoose para recebimento dos dados.

Fonte: Criação própria

5.4 Gráficos e exportação de dados

Na interface de início podemos ver a lista de todos os experimentos que o usuário possui. Em cada experimento podemos observar um botão de exportar, o qual exporta todos os dados deste experimento. Como nesta tela ainda não temos os dados de dentro do experimento, apenas seu título e descrição, foi necessária uma rota para o botão de exportar. Este botão obtém os dados para exportar realizando um GET na rota “/experiment-to-export/:experimentId”, enviando o id do experimento como parâmetro. Após receber os dados para exportar, a tela executa uma função para gerar um arquivo CSV com os dados de todos os experimentos (Figura 9).

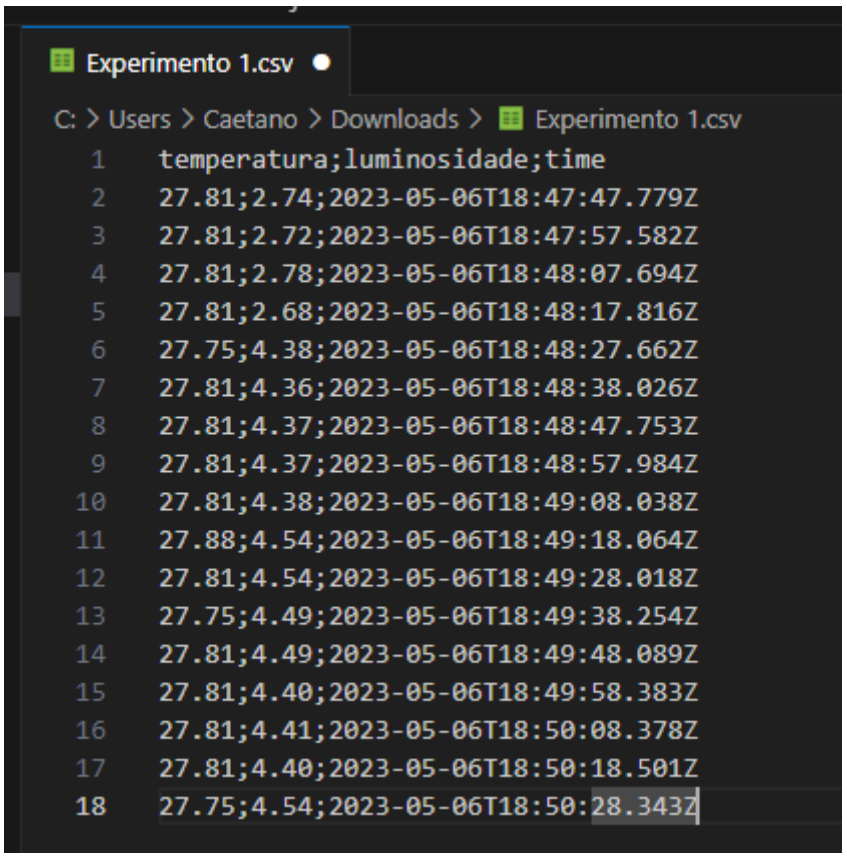

Figura 9 - Arquivo CSV gerado com os dados de todos os sensores do experimento. Fonte: Criação própria

Ao acessar os detalhes de um experimento, para renderizar os gráficos na tela, o front-end necessita realizar um GET na seguinte rota: “/get-data/:experimentId” enviando o id do experimento como parâmetro e recebendo todos os dados do experimento de uma vez. Ao ver o gráfico, também podemos ver um botão para exportar os dados, que irá gerar um CSV somente com os dados que foram utilizados para gerar o gráfico, ou seja, de apenas um sensor (Figura 10).

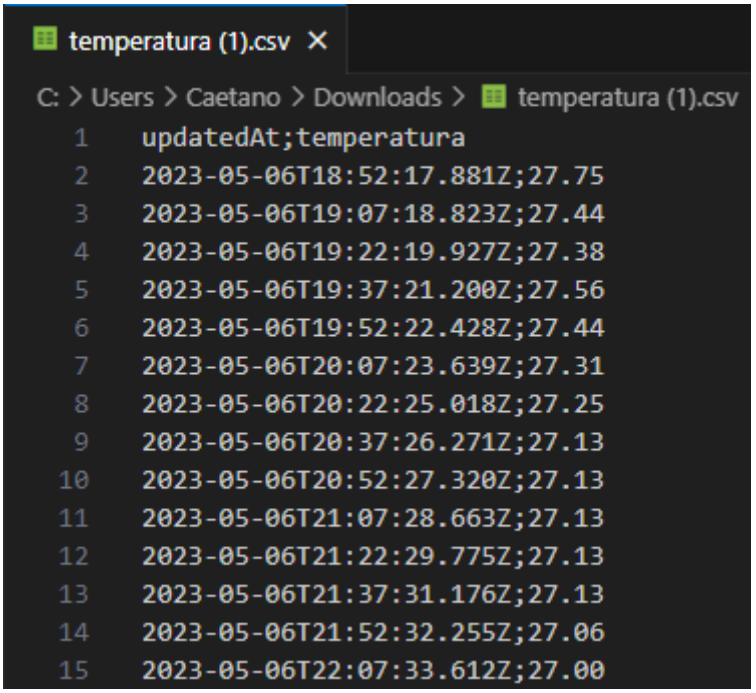

Figura 10 - Arquivo CSV gerado com os dados de apenas um dos sensores.

Fonte: Criação própria

5.5 Autenticação e segurança

Em comparação ao projeto de CARVALHO (2022), o primeiro ponto de segurança é que este projeto possui um sistema de autenticação de usuário, fazendo com que os dados dos experimentos fiquem visíveis apenas para os donos de suas contas. Este sistema de autenticação se inicia com o botão “create account” na tela de login, que permite ao usuário a criação de uma conta própria. Na interface para criar conta haverá um formulário que precisa ser preenchido com o nome do usuário, e-mail e senha desejada. Ao submeter o formulário, essas informações serão enviadas ao back-end através de um POST na rota “/create-user”, a qual realizará as seguintes tarefas: (i) criptografar a senha utilizando a biblioteca Bcrypt (ii) verificar se o e-mail não existe na tabela users (iii) inserir o usuário na tabela (iv)

enviar um e-mail com um link que consome um token para verificar se o usuário realmente é o dono deste e-mail (Figura 11).

Figura 11 - E-mail de verificação enviado para criação de conta. Fonte: Criação própria

A tela de login possui um formulário que deve ser preenchido com e-mail e senha de um usuário já existente, que quando submetido realizará um POST na rota “/login”. Este POST irá desencadear as seguintes tarefas: (i) busca do e-mail na tabela usuários, se existir continua (ii) comparação da senha fornecida pelo usuário com a senha do banco utilizando a função *compare* da biblioteca Bcrypt, e, caso não ocorra erro, (iii) geração de um token com validade de 1 dia que é enviado como resposta do login (Figura 12).

```
async login(request, response){
    try {
        const { email, password } = request.body;
        const result = await connection('users').select('*').where({ email });
        if(result.length === 0) return response.status(403).send({ msg: 'Falha na autenticação' });
        bcrypt.compare(password, result[0].password, (err, res) => {
            if(err) return response.status(401).send({ msg: 'Falha na autenticação' });
            if(res) {
                const token = jwt.sign({ email, name: result[0].name }, process.env.JWT_KEY, { expiresIn: '1d' });
                return response.status(200).send({ token, user: {
                    "email": result[0].email,
                    "verified": result[0].verified,
                    "name": result[0].name
                } });
            }
            return response.status(401).send({ msg: 'Falha na autenticação' });
        })

    } catch(err) {
        return response.status(500).send({ msg: err });
    }
},
```

Figura 12 - Função de autenticação realizada pelo back-end. Fonte: Criação própria

Quando o usuário já realizou login, as rotas do back-end utilizadas pelo front-end passam por um middleware (Figura 13) para realizar a verificação do token que passa a ser necessário em todas as requisições. Caso o token esteja faltando ou não seja válido, o front-end receberá o erro 403 (forbidden).

```
JS Logged.js ×
src > Middleware > JS Logged.js > <unknown> > exports
      Caetano Sasia, há 10 meses | 1 author (Caetano Sasia)
1     const jwt = require('jsonwebtoken');
2
3     module.exports = (req, res, next) => {
4         try {
5             const token = req.headers.authorization.split(' ')[1];
6             const decode = jwt.verify(token, process.env.JWT_KEY);
7             req.usuario = decode;
8             next();
9         } catch(err) {
10            return res.status(403).send({ mensagem: 'Falha na autenticação' });
11        }
12    }       Caetano Sasia, há 12 meses • initial commit
```

Figura 13 - Middleware responsável pela verificação do token gerado no login. Fonte: Criação própria

Como segundo ponto de segurança em comparação ao projeto de CARVALHO (2022), após o usuário realizar o login, o mesmo possui a possibilidade de criar um experimento, através do preenchimento de um formulário com o título e a descrição do experimento. O front-end irá realizar um post na rota “/experiments” e o back-end irá realizar as seguintes tarefas: (i) criar um id e token para o experimento, onde o token irá possuir o id do experimento e e-mail do usuário (ii) enviar um e-mail para o usuário com o token que deverá ser utilizado ao enviar os dados dos sensores (iii) inserção do experimento na tabela experiments. Então além de ter a possibilidade de ter diversos experimentos com quantidade variáveis de sensores, também existe a validação de um token para a inserção de dados em um experimento, fazendo com que a rota de inserção de dados seja mais segura.

5.6. Responsividade

O desenvolvimento do front-end foi realizado levando em consideração a responsividade da página web, tornando possível acessar o sistema facilmente em dispositivos móveis.

Na página de gerenciamento de experimentos, os experimentos disponíveis são exibidos em uma única coluna embaixo do botão de adicionar novos experimentos, sendo possível visualizar todos os experimentos disponíveis utilizando apenas a rolagem do navegador (Figura 14).

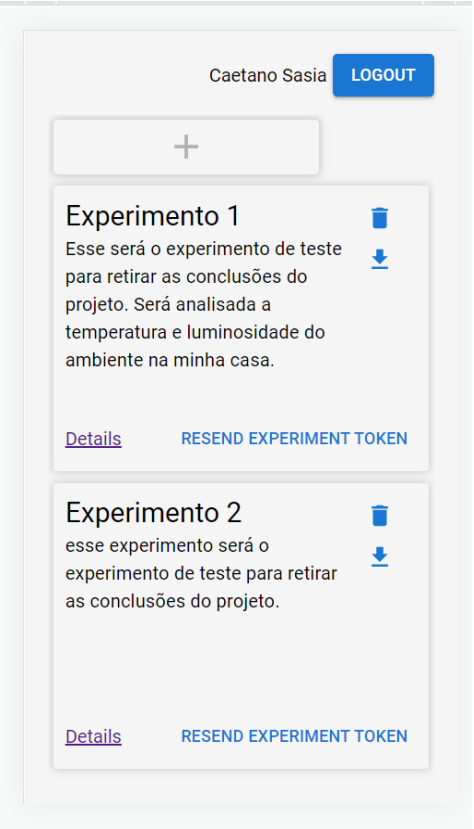

Figura 14 - Responsividade da tela de gerenciamento de experimentos. Fonte: Criação própria

Quando acessado em dispositivos móveis, a página de visualização dos gráficos também é exibida em uma única coluna, com as opções para alternar entre os diferentes sensores do experimento no topo, e a visualização dos gráficos abaixo. Além disso, a biblioteca eCharts também possui responsividade, sendo possível utilizar a função de ampliar os gráficos para exibir um intervalo maior ou menor de tempo no gráfico, sendo possível “navegar” pelo gráfico facilitando a visualização em dispositivos com telas menores (Figura 15).

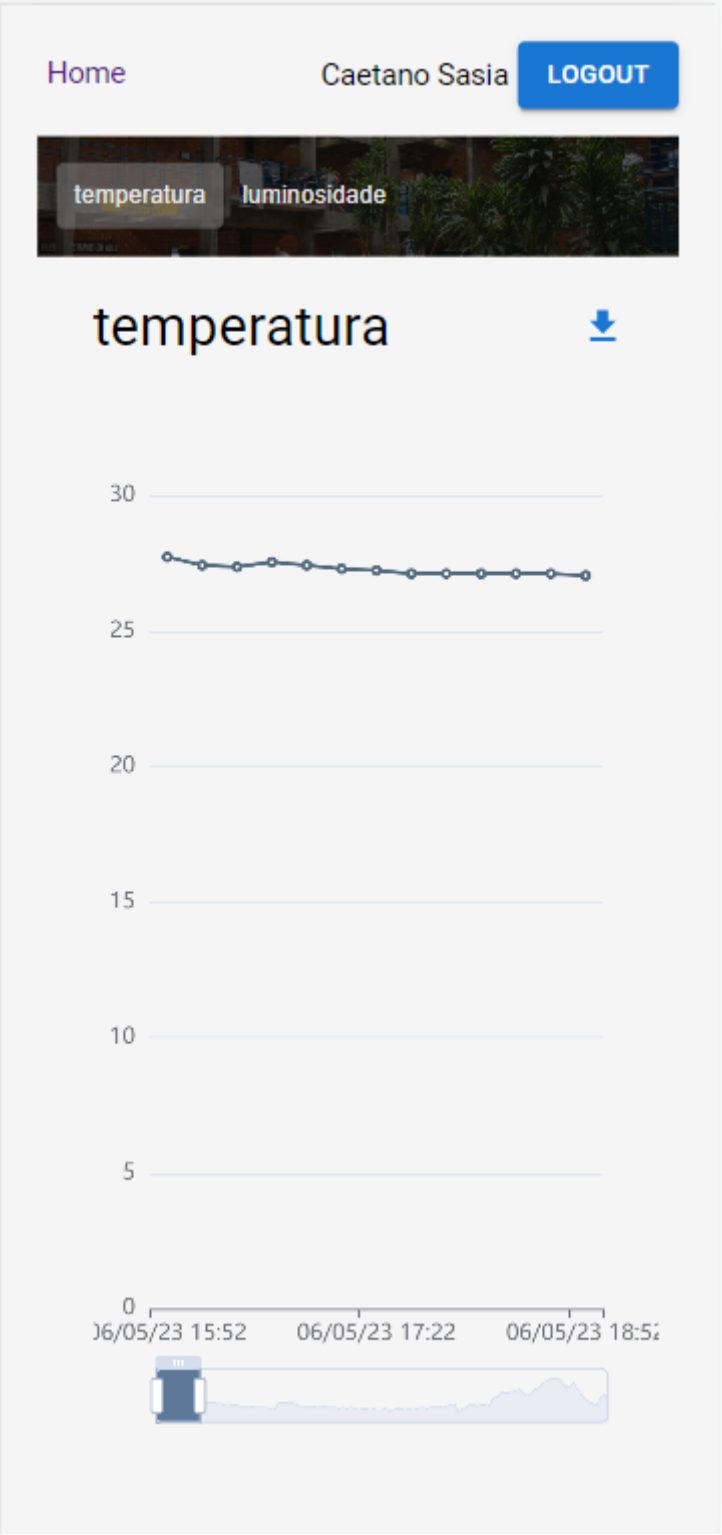

Figura 15 - Responsividade da tela de exibição de gráficos. Note a barra de ampliação abaixo do gráfico, selecionada de forma para exibir poucos dados no intervalo de tempo entre 06/05/23 15:52 e 06/05/23 18:52. Fonte: Criação própria

5.7. Testes realizados

Foram realizados testes com um Arduino que fez a coleta de dados de sensores de temperatura e luminosidade. No Experimento II, realizado entre 06/05/2023 às 18:52:17 e 08/05/2023 às 00:24:23, foram obtidas um total de 119 leituras. Essas leituras foram recebidas pelo back-end e gravadas com sucesso. Com base nesses dados, foram gerados 2 gráficos que podem ser visualizados ao acessar os detalhes do experimento na aplicação (figuras 16 e 17).

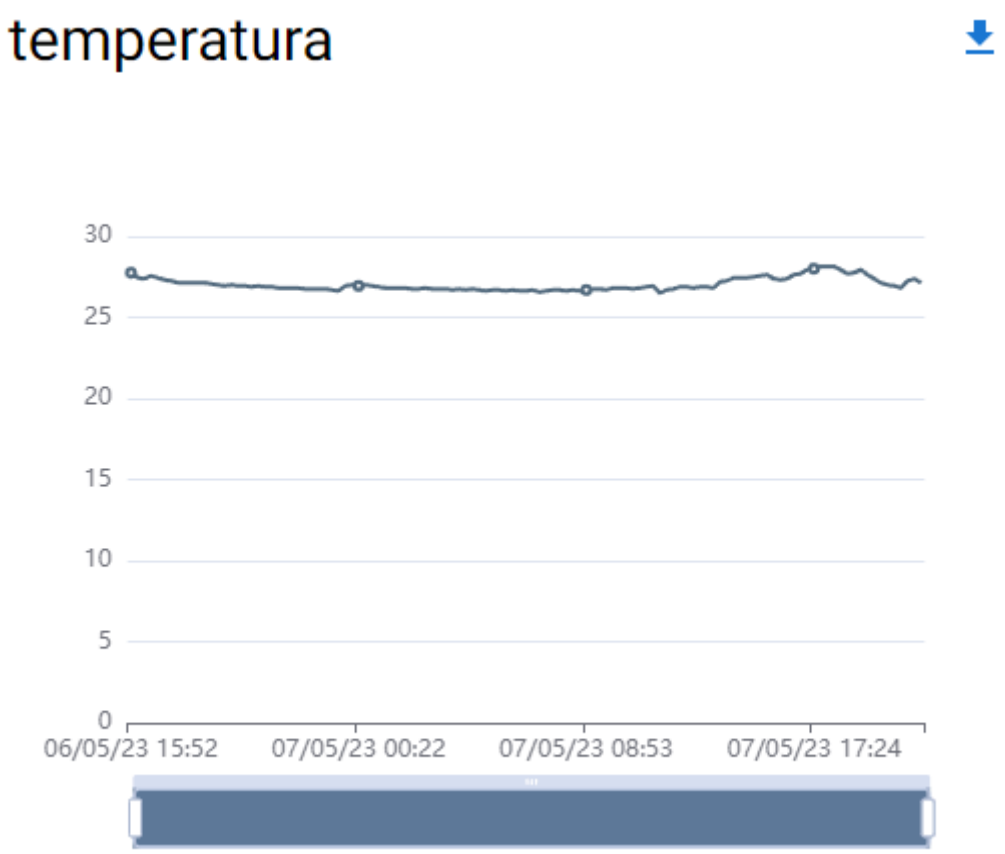

Figura 16 - Gráfico de Temperatura em função do tempo. O eixo x representa o tempo, enquanto o eixo y representa a temperatura em graus Celsius. Fonte: criação própria.

Figura 17 - Gráfico de luminosidade em função do tempo. O eixo x representa o tempo, enquanto o eixo y representa a luminosidade. Fonte: criação própria.

5.8. Dificuldades encontradas

A primeira dificuldade encontrada neste projeto foi a de como armazenar os dados de uma maneira dinâmica, aceitando quantos sensores fossem necessários para cada experimento. No projeto desenvolvido por CARVALHO (2022), os dados eram armazenados em uma tabela Data em um banco de dados relacional de uma maneira não dinâmica, com uma coluna por sensor, porém para tornar possível a adição e remoção de sensores sem precisar alterar o banco de dados, seria necessário buscar outra maneira de implementação. No começo estudamos a hipótese de criar uma coluna chamada Data com o tipo TEXT, pois o mesmo contém

um tamanho máximo aproximadamente em 1GB, sendo assim, para cada linha da tabela poderíamos inserir um objeto com uma grande quantidade de sensores. Mas ao estudar o conceito MERN, percebemos que utilizando um banco de dados não relacional poderíamos realizar o armazenamento dos dados de uma maneira dinâmica e organizada.

A segunda dificuldade consistiu em problemas ao realizar o deploy do front-end, pois o Heroku na versão Heroku-22 passou a considerar o create-react-app padrão como depreciado. O Heroku exibia a sugestão de utilizar outro framework em conjunto com o ReactJs, o que levaria um esforço considerável para realizar a migração. Porém, após bastante dificuldade, conseguimos realizar o deploy da aplicação utilizando o build final.

6. Considerações finais

Equipamentos automatizados para coletas de sensores para uso em laboratórios, apesar de muito úteis não são muito acessíveis devido ao seu preço. Além de ser um desperdício de tempo ter que realizar coletas manuais de medições como temperatura e pH, pode ser uma atividade muito desgastante dependendo da necessidade do experimento, pois muitas vezes é necessário realizar coletas com intervalos de tempo inconvenientes como a cada 6 horas ou a cada 12 horas dependendo do experimento, forçando o pesquisador a voltar constantemente ao laboratório.

O sistema desenvolvido além de ser de baixo custo, pode ser modificado de acordo com a necessidade de cada experimento, podendo ser adicionado quantos sensores forem necessários e de qualquer tipo no sistema, desde que seja compatível com o arduino utilizado. Além disso, as medidas de segurança tomadas

com o uso de tokens para coleta de dados do arduino, sistema de autenticação com verificação de e-mail, dificultam a invasão do sistema por terceiros, garantindo a integridade dos dados coletados pelos pesquisadores.

Por fim, esse sistema pode ser considerado uma evolução do sistema proposto por CARVALHO (2022), devido a sua flexibilidade, segurança e possibilidade de diferentes pesquisadores o utilizarem simultaneamente devido ao sistema de autenticação.

6.1 Trabalhos Futuros

Nesse sistema foram melhorados os aspectos de segurança e flexibilidade para inserção de novos sensores, utilizando criptografia para armazenamento dos dados de autenticação no back-end. Porém, ainda existem outras possibilidades de aprimoramento para esse sistema que poderia ser realizado em trabalhos futuros, tais como:

- Adicionar criptografia no front-end, para envio de dados criptografados ao back-end, aumentando ainda mais a segurança do sistema;
- Adicionar possibilidade de comparar diferentes sensores na aplicação web, através de gráficos de linhas com múltiplas séries (uma série para cada sensor sendo comparado)
- Adicionar possibilidade de alterar o tipo de gráfico sendo utilizado para visualização dos dados.
- Criar página de documentação com instruções para implementação do código no Arduino.

7. Referências

ARDIANSAH, R.; BAFDAL, N.; SURYADI, E.; BONO, A. Greenhouse Monitoring and Automation Using Arduino: a Review on Precision Farming and Internet of Things (IoT), **International Journal on Advanced Science Engineering Information Technology**, vol. 10, no. 2, 2020.

Arduino.cc. Disponível em: <https://www.arduino.cc/>. Acesso em: 8 mai 2023.

ARVINDAN A. N.; KEERTHIKA, D. Experimental investigation of remote control via Android smart phone of arduino-based automated irrigation system using moisture sensor, **3rd International Conference on Electrical Energy Systems (ICEES)**, 2016

BEHERA, K. B.; GUPTA, S.; GAUTAM, A. Big-data empowered cloud centric Internet of Things. **International Conference on Man and Machine Interfacing (MAMI)**, 2015.

BROWN, E. Webdevelopment with Node and Express. **O'REILLY**, segunda edição, 2020.

CARVALHO, J. S. **Internet of Things em Biotecnologia: arduino no cultivo de microalgas.** Trabalho de conclusão de curso - Universidade Federal de Santa Catarina. Florianópolis. 2022.

Echarts.apache.org. Disponível em: <https://echarts.apache.org>. Acesso em 20 mai 2023.

Expressjs.com. Disponível em: <https://expressjs.com/>. Acesso em: 8 mai 2023.

FERDOUSH, S.; LI, X. Wireless Sensor Network System Design using Raspberry Pi and Arduino for Environmental Monitoring Applications, **Procedia Computer Science,** vol 34, 2014.

GOLINELLI, M. H. M.; SILVA, J. B.; YEVSEYAVA, O. Arquitetura de dispositivos inteligentes para laboratórios remotos: uma revisão sistemática da literatura, **RETEC,** Ourinhos, v. 11, n. 2, 2018.

HOLMES, S. Mongoose for Application Development, **Packt Publishing**, 2013

IRAWAN, Y.; Febriani FEBRIANI, A.; WAHYUNI, R.; DEVIS, Y. Water Quality Measurement and Filtering Tools Using Arduino Uno, PH Sensor and TDS Meter Sensor, **Journal of Robotics and Control (JRC)**, Vol 2, issue 5, 2021.

KHUE, T. D. *et al.* Design and Implementation of MEARN Stack-based Real-time Digital Signage System. **Journal of Korea Multimedia Society** Vol. 20, No. 5, pp. 808-826, 2017.

Knexjs.org. Disponível em: <https://knexjs.org/>. Acesso em 8 mai 2023.

KROL, M. S. J.; D'MELLO, B. J. Web Development with MongoDB and NodeJS. **Packt Publishing**. Ed 2, 2015.

LIU, P. C. An Admin Management Dashboard with MERN for SME E-commerce. **Summer 2021 CS 687 Capstone Project Progress Report**, 2021.

LOUIS, L. Working principle of arduino and using it as a tool for study and research. **International Journal of Control, Automation, Communication and Systems (IJCACS)**, Vol.1, No.2, 2016.

OBE, R. O.; HSU, L. S. PostgreSQL: Up and Running. **O'REILLY**, terceira edição, 2018.

React.dev. Disponível em: <https://react.dev/>. Acesso em 20 mai 2023.

SALZMANN, C. et al., The Smart Device Specification for Remote Labs, **International Journal Of Online Engineering (ijoe)**, v. 11, n. 4, 2015.

SOUTO, Mario. O que é front-end e back-end? **Alura Cursos Online**, 17 jan 2023. Disponível em:

<https://www.alura.com.br/artigos/o-que-e-front-end-e-back-end>.

Acesso em: 09 mai. 2023.

VANELI, B. *et al.* Internet of Things Data Storage Infrastructure in the Cloud Using NoSQL Databases. **IEEE Latin America Transactions**, vol. 15, no. 4, pp. 737-743, 2017.

Apêndice A - Artigo do Trabalho

Sistema de gerenciamento de coleta de dados laboratoriais por Arduinos

Caetano Sasia dos Santos[1], Vinícius Soares Laghi[1]

[1] Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC) Caixa Postal 476 – 88040-900 – Florianópolis – SC – Brasil

caetanosasia@gmail.com, vinicius.laghi@gmail.com

Abstract. *The present work aims to introduce an automatic data collection system, built using an Arduino, for laboratory use. This system aims to make the automatization of data collection more accessible, since commercial systems are often too expensive. The system has a web interface with an authentication system, built using the MERN technology, allowing for real-time monitoring of the data collected through charts, in addition to being able to export all data collected in a .csv file.*

Resumo. *O presente trabalho tem como proposta apresentar um sistema automatizado de coleta de dados com Arduino para uso em laboratórios. Tal sistema tem o objetivo de baratear a automatização de coleta de dados visto que sistemas comerciais possuem um custo muito elevado. O sistema conta com uma interface web com um sistema de autenticação, desenvolvida utilizando a tecnologia MERN, sendo possível realizar um acompanhamento em tempo real dos dados coletados através de gráficos, além de ser possível exportar os dados coletados em um arquivo .csv.*

1.Introdução

A Internet das Coisas (IoT) é uma tecnologia emergente, nas quais dispositivos ao nosso redor conseguem se comunicar através do uso da internet [Behera, Gupta, et al., 2015]. A popularização dessa tecnologia só foi possível graças aos avanços tecnológicos de áreas como a Informática, a Internet e a Eletrônica [Vanelli, Da Silva, et al., 2017].

O uso de dispositivos IoT permite a solução de diversos problemas existentes, dentre eles a automatização de sistemas manuais. Devido a isso, essa tecnologia é promissora para a automatização de sistemas em laboratórios de pesquisa, principalmente na área biotecnológica, que utilizam sensores que são fundamentais aos seus processos, que normalmente não estão amplamente disponíveis em laboratórios devido ao seu custo alto de aquisição e manutenção [Carvalho, 2022].

Carvalho (2022) desenvolveu um sistema IoT automatizado utilizando arduinos conectados a sensores, como uma solução para automatização de coleta de dados utilizados no cultivo de microalgas, apresentando uma economia de 95% em seu custo baseado em um sistema comercial similar. Apesar de funcional e eficiente para aquele processo, o sistema desenvolvido possui limitações quanto a adição ou remoção de sensores, uso paralelo em outros experimentos, falta de segurança entre outros fatores.

Baseado nesse projeto de Carvalho (2022), o objetivo deste trabalho é o desenvolvimento de um novo sistema automatizado de coleta de dados enviados por Arduinos, com uma maior flexibilidade permitindo adição e remoção de sensores sem a necessidade de alterar o banco de dados para suportar mudanças, melhorias em relação a segurança através de um sistema de autenticação e token para envio dos dados pelo Arduino e que permite que dados de diferentes experimentos sejam coletados em paralelo, permitindo que diferentes experimentos sejam automatizados simultaneamente.

2.Objetivos

2.1. Objetivos Gerais

Desenvolver um sistema que permita o armazenamento automático de dados coletados através de sensores conectados a arduinos.

2.2. Objetivos Específicos

Como objetivos específicos foram definidos: (i) desenvolver uma interface web que permita a visualização e download dos dados armazenados no banco de dados; (ii) desenvolver um sistema back-end que suporte a conexão de novos sensores automaticamente; (iii) desenvolver um sistema de autenticação para acesso aos dados dos sensores, que forneça um token único na interface web, necessário para o envio de dados pelos arduinos para o banco de dados; e (iv) desenvolver um sistema de autenticação de usuário para acesso à interface web;

3. Tecnologias Utilizadas

Para a coleta dos dados foi utilizado um Arduino modelo UNO. No sistema desenvolvido, o Arduino é responsável pela leitura e coleta de dados contínua através de sensores conectados ao mesmo, enviando-os para o back-end a cada leitura realizada através de um post que possui token de validação único previamente gerado no sistema em seu header. O uso do modelo Uno se dá ao fato de ser o modelo base utilizado para os modelos mais modernos, sendo de simples uso e ao mesmo tempo atendendo todas as necessidades do sistema. Para os testes com o sistema desenvolvido foram utilizados um sensor de luminosidade fotorresistor LDR e um o sensor de temperatura DS18B20 à prova d'água de 2 metros.

O desenvolvimento da aplicação web foi feita através da tecnologia MERN (Mongo, Express, React, Node), resultando em uma aplicação de alta performance que realiza atualizações em tempo real de forma assíncrona e facilmente escalável [Khue, et al., 2017]. Além disso, para o back-end também foi utilizado PostgreSQL para a criação da tabela de usuários no sistema desenvolvido, utilizando Knex.js para gerenciamento do mesmo, e para criar as migrations para as tabelas de usuário e experimentos e tokens de verificação de email para ser consumido. A modelagem dos objetos para MongoDB e NodeJS foi feita através do Mongoose. A parte de segurança foi utilizado jsonwebtoken para geração de tokens a serem consumidos e node.bcrypt.js para a criptografia das senhas a serem salvas. Por fim, foi utilizado o Apache Echarts para a construção dos gráficos a serem exibidos no front-end.

4. Utilizando o Sistema

Ao acessar o sistema, o usuário se depara com uma tela de login, onde é possível realizar o cadastro de um novo usuário, sendo necessário utilizar um email válido para criação de uma nova conta. Após feito o login, é possível cadastrar um novo experimento,

fornecendo-o um nome e uma breve descrição. Tendo um experimento cadastrado, o usuário receberá um Token em seu email, que deve ser utilizado pelo Arduino para o envio dos dados coletados para rota: "/insert-data" através de um HTTPS POST, utilizando o token no header para a validação dos dados enviados.

Quando o usuário possui experimentos cadastrados com dados coletados, é possível acessar os mesmos e verificar os gráficos gerados utilizando os dados coletados pelos diferentes sensores, sendo possível visualizar individualmente cada sensor (figura 1), assim como exportar os dados dos mesmos em um arquivo .csv.

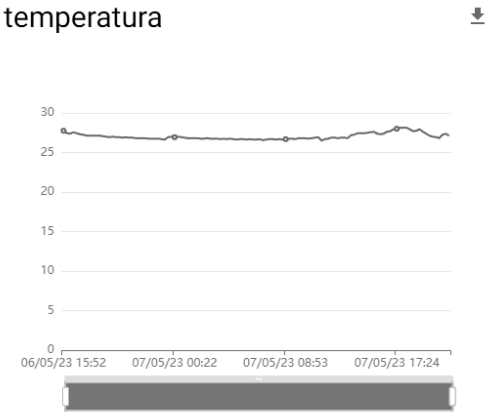

Figura 1. Gráfico de temperatura gerado pelo sistema nos testes realizados

Por fim, toda a aplicação web foi desenvolvida tendo como prioridade a responsividade do sistema, podendo ser acessada e facilmente navegada através de dispositivos móveis (figura 2), permitindo o monitoramento dos sensores a partir de qualquer dispositivo.

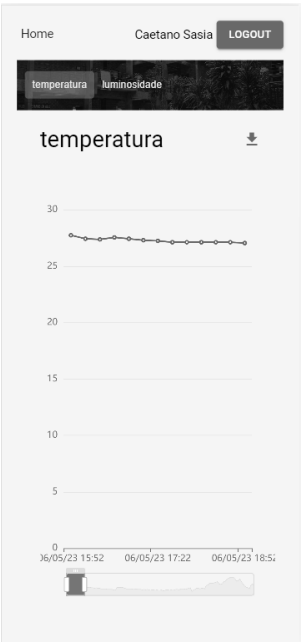

Figura 2. Tela de visualização de um experimento em um dispositivo móvel.

5. Considerações Finais

Equipamentos automatizados para coletas de sensores para uso em laboratórios, apesar de muito úteis não são muito acessíveis devido ao seu preço. O sistema desenvolvido além de ser de baixo custo, pode ser modificado de acordo com a necessidade de cada experimento, podendo ser adicionado quantos sensores forem necessários e de qualquer tipo no sistema, desde que seja compatível com o arduino utilizado. Além disso, as medidas de segurança tomadas com o uso de tokens para coleta de dados do arduino, sistema de autenticação com verificação de e-mail, dificultam a invasão do sistema por terceiros, garantindo a integridade dos dados coletados pelos pesquisadores.

Por fim, esse sistema pode ser considerado uma evolução do sistema proposto por Carvalho (2022), devido a sua flexibilidade, segurança e possibilidade de diferentes pesquisadores o utilizarem simultaneamente devido ao sistema de autenticação.

Como sugestões para melhorias futuras pode ser citado: (i) adicionar criptografia no front-end, para envio de dados criptografados ao back-end, aumentando ainda mais a segurança do sistema; (ii) adicionar possibilidade de comparar diferentes sensores na aplicação web, através de gráficos de linhas com múltiplas séries (uma série para cada sensor sendo comparado); (iii) adicionar possibilidade de alterar o tipo de gráfico sendo utilizado para visualização dos dados; e (iv) criar página de documentação com instruções para implementação do código no Arduino.

6. Referências

Behera, K. B.; Gupta, S.; Gautam A. (2015) A. Big-data empowered cloud centric Internet of Things. International Conference on Man and Machine Interfacing (MAMI).

Carvalho, J. S. (2022) Internet of Things em Biotecnologia: arduino no cultivo de microalgas. Trabalho de conclusão de curso - Universidade Federal de Santa Catarina. Florianópolis..

Khue, T. D. et al. (2017) Design and Implementation of MEARN Stack-based Real-time Digital Signage System. Journal of Korea Multimedia Society Vol. 20, No. 5, pp. 808-826.

VANELI, B. et al. (2017) Internet of Things Data Storage Infrastructure in the Cloud Using NoSQL Databases. IEEE Latin America Transactions, vol. 15, no. 4, pp. 737-743.

Apêndice B - Código do back-end

src\Controllers\DataController.js

```
const connection = require('../database/connection');
const jwt = require('jsonwebtoken');
const mongoose = require('mongoose');
const Data = require('../database/Schemas/Data');
mongoose.connect("mongodb://localhost:");

module.exports = {
async index(request, response) {
try {
const token = request.headers.authorization.split(' ')[1];
const { email } = jwt.verify(token, process.env.JWT_KEY);
const { experimentId } = request.params;

const data = await Data.find({ email, experimentId });
let cleanData = {};
data.forEach((item) => {
const { updatedAt, id } = item;
if(!item.data) return;
const itemData = Object.values(item.data);
itemData.forEach(subItem => {
const { key, value } = subItem;
if(!cleanData[key]) {
cleanData[key] = {
key,
data: [{
value,
updatedAt,
id
}]
};
```

```
} else {
cleanData[key].data.push({
value,
updatedAt,
id
});
}
})
});
return response.status(200).send(cleanData);
} catch (err) {
return response.status(500).send({ msg: err, func: 'get' });
}
},
async create(request, response){
try {
const { body } = request;
const { token } = request.headers;
const { email, id } = jwt.verify(token, process.env.EXP_SECRET);

const verifyToken = await
connection('experiments').select('*').where({ id });

if(verifyToken.length === 0) return response.status(404).send({ msg:
'Usuário inexistente' });
if(verifyToken[0].token !== token) return response.status(401).send({
msg: 'Token expirado' });

const values = Object.values(body);
const keys = Object.keys(body);
const data = {};
keys.forEach((key, i) => {
data[key] = {
key,
value: values[i],
}
});
const row = await Data.create({ data, email, experimentId: id });
```

```
65  await row.save();
66  return response.status(200).send({ msg: 'Dado inserido no BD com
    sucesso' });
67  } catch (err) {
68  return response.status(500).send({ msg: err, func: 'create' });
69  }
70
71  },
72  async drop(request, response) {
73  try {
74  await Data.remove({});
75  return response.status(200).send({ msg: 'Dados deletados com sucesso'
    });
76  } catch (err) {
77  return response.status(500).send({ msg: err, func: 'delete' });
78  }
79  }
80
81  }
```

src\Controllers\ExperimentController.js

```
1  const connection = require('../database/connection');
2  const crypto = require('crypto');
3  const Data = require('../database/Schemas/Data');
4  const jwt = require('jsonwebtoken');
5  const nodemailer = require('nodemailer');
6  const mongoose = require('mongoose');
7  mongoose.connect("mongodb://localhost:");
8
9  module.exports = {
10 async index(request, response) {
11 try {
12 const sessionToken = request.headers.authorization.split(' ')[1];
13 const { email } = jwt.verify(sessionToken, process.env.JWT_KEY);
```

```
14  const experiments = await
    connection('experiments').select('*').where({ email });
15  return response.status(200).send({ experiments });
16  } catch (err) {
17  return response.status(500).send({ msg: err });
18  }
19  },
20  async create(request, response){
21  try {
22  const { name, description } = request.body;
23
24  if(!name) return response.status(500).send({ msg: 'missing name' });
25  const id = crypto.randomBytes(4).toString('HEX');
26  const token = request.headers.authorization.split(' ')[1];
27  const { email } = jwt.verify(token, process.env.JWT_KEY);
28  const experimentToken = jwt.sign({ email, name, id },
    process.env.EXP_SECRET);
29  console.log('teste')
30  sendEmailWithExperimentToken(email, experimentToken, name);
31  await connection('experiments').insert({
32  id,
33  email,
34  token: experimentToken,
35  name,
36  description
37  })
38  return response.status(200).send({ msg: 'Experimento criado com
    sucesso', createdExperiment: { id } });
39  } catch (err) {
40  return response.status(500).send({ msg: err });
41  }
42  },
43  async resendExperimentToken(request, response){
44  try {
45  console.log('teste');
46  const { experimentId } = request.body;
47  const experiment = await
    connection('experiments').select('*').where({ id: experimentId });
```

```
48 if(experiment.length === 0) return response.status(404).send({ msg:
   'Experimento inexistente' });
49 const experimentToken = experiment[0].token;
50 const name = experiment[0].name;
51 sendEmailWithExperimentToken(experiment[0].email, experimentToken,
   name);
52 return response.status(200).send({ msg: 'Token reenviado com
   sucesso' });
53 } catch(err) {
54 return response.status(500).send({ msg: err });
55 }
56 },
57 async experimentToExport(request, response) {
58 try {
59 const token = request.headers.authorization.split(' ')[1];
60 const { email } = jwt.verify(token, process.env.JWT_KEY);
61 const { experimentId } = request.params;
62 const keys = {};
63 const data = await Data.find({ email, experimentId });
64 let cleanData = [];
65 data.forEach((item) => {
66 if(!item.data) return;
67 const itemData = Object.values(item.data);
68 itemData.forEach(subItem => {
69 const { key } = subItem;
70 if(keys[key] === undefined) {
71 keys[key] = null;
72 }
73 })
74 });
75 data.forEach((item) => {
76 const { updatedAt } = item;
77 if(!item.data) return;
78 const itemData = {};
79 const data = Object.values(item.data);
80 data.forEach(a => {
81 itemData[a.key] = a.value
82 });
```

```
cleanData.push({
...keys,
...itemData,
time: updatedAt
});
});
console.log(cleanData);
return response.status(200).send(cleanData);
} catch (err) {
return response.status(500).send({ msg: err, func: 'get' });
}
},
async deleteExperiment(request, response){
try {
const sessionToken = request.headers.authorization.split(' ')[1];

const { email } = jwt.verify(sessionToken, process.env.JWT_KEY);

const { experimentId } = request.body;
const experiment = await
connection('experiments').select('*').where({ id: experimentId });
if(email != experiment[0].email) {

return response.status(403).send({ msg: 'Usuário inválido' });

}

if(experiment.length === 0) return response.status(404).send({ msg:
'Experimento inexistente' });
await connection('experiments').select('*').where({ id: experimentId
}).delete();
Data.find({ experimentId }).deleteMany().exec();

const name = experiment[0].name;

sendMessage(experiment[0].email, `O experimento "${name}" foi
deletado com sucesso.`, 'Experimento deletado');
return response.status(200).send({ msg: 'Experimento deletado com
sucesso' });
} catch(err) {

```

```
111 return response.status(500).send({ msg: err });
112 }
113 },
114 }
115
116
117 function sendMessage(email, message, title) {
118 const transporter = nodemailer.createTransport({
119 host: "smtp-mail.outlook.com",
120 port: 587,
121 secureConnection: false,
122 auth: {
123 user: process.env.EMAIL_TCC,
124 pass: process.env.EMAIL_PASSWORD
125 },
126 tls: {
127 ciphers:'SSLv3'
128 }
129 });
130 const mailOptions = {
131 from: process.env.EMAIL_TCC,
132 to: email,
133 subject: title,
```

```
html: message

};

transporter.sendMail(mailOptions, (err, info) => {

if(err) return console.log(err);

console.log(info);

} );

}



function sendEmailWithExperimentToken(email, token, name) {

const transporter = nodemailer.createTransport({

host: "smtp-mail.outlook.com",

port: 587,

secureConnection: false,

auth: {

user: process.env.EMAIL_TCC,

pass: process.env.EMAIL_PASSWORD

},

tls: {

ciphers:'SSLv3'

}

});

const mailOptions = {

from: process.env.EMAIL_TCC,
```

```
to: email,

subject: `Experimento: "${name}"`,

html: `Oi, o token para o experimento "${name}" é: <br>${token}<br>`

};

transporter.sendMail(mailOptions, (err, info) => {

if(err) return console.log(err);

console.log(info);

} );

}
```

src\Controllers\UserController.js

```
const connection = require('../database/connection');
const bcrypt = require('bcrypt');
const jwt = require('jsonwebtoken');
const nodemailer = require('nodemailer');



module.exports = {
async create(request, response){
const { name, password, email } = request.body;
bcrypt.hash(password, 10, async (errBcrypt, hash) => {
try {
if(errBcrypt) return response.status(500).send(({ error: errBcrypt
}));
const result = await connection('users').select('*').where({ email
});
if(result.length > 0) return response.status(409).send({ msg: 'User
already exists' });
await connection('users').insert({
email,
password: hash,
name,
```

```
verified: false
})
const emailToken = jwt.sign({ email }, process.env.EMAIL_KEY);

await connection('verify_email').insert({
token: emailToken,
send_date: new Date(),
})
sendEmailVerification(email, emailToken);
return response.status(201).send({ msg: 'Usuário criado com
sucesso', createdUser: { email } });
} catch (err) {
return response.status(500).send({ msg: err });
}
});
},
async drop(request, response){
try {
const { email } = request.body;
const result = connection('users').select('*').where({ email });


if(result.length === 0) return response.status(404).send({ msg:
'Usuário inexistente' });

await connection('users').where('email', email).delete();


return response.status(200).send({ msg: 'Usuário deletado com
sucesso', deletedUser: { email } });
} catch (err) {
return response.status(500).send({ msg: err });
}
},
async login(request, response){
try {
const { email, password } = request.body;
const result = await connection('users').select('*').where({ email
});
```

```
51 if(result.length === 0) return response.status(403).send({ msg:
   'Falha na autenticação' });
52 bcrypt.compare(password, result[0].password, (err, res) => {
53 if(err) return response.status(401).send({ msg: 'Falha na
   autenticação' });
54 if(res) {
55 const token = jwt.sign({ email, name: result[0].name },
   process.env.JWT_KEY, { expiresIn: '1d' });
56 return response.status(200).send({ token, user: {
57 "email": result[0].email,
58 "verified": result[0].verified,
59 "name": result[0].name
60 } });
61 }
62 return response.status(401).send({ msg: 'Falha na autenticação' });

63 })
64

65 } catch(err) {
66 return response.status(500).send({ msg: err });
67 }
68 },
69 async session(request, response){
70 try {
71 const { token } = request.body;
72 const decode = jwt.verify(token, process.env.JWT_KEY);
73 const user = await connection('users').select('*').where({ email:
   decode.email });
74 return response.status(200).send({ msg: 'Sessão válida', user: {

75 "email": user[0].email,
76 "verified": user[0].verified,
77 "name": user[0].name
78 }});
79 } catch(err) {
80 return response.status(401).send({ msg: 'Falha na autenticação, faça
   login novamente' });
81 }
82 },
```

```
83  async generateToken(request, response){
84  try {
85  const token = request.headers.authorization.split(' ')[1];
86  const { email, name } = jwt.verify(token, process.env.JWT_KEY);
87  const tokenToReturn = jwt.sign({ email, name },
    process.env.SERVER_SECRET, { expiresIn: '1y' });
88  await connection('users').update({ token: tokenToReturn }).where({
    email });
89  return response.status(200).send({ token: tokenToReturn });
90  } catch(err) {
91  return response.status(500).send({ msg: err });
92  }
93  },
94  async patchPassword(request, response){
95  const { email, password, role } = request.body;
96  if(!password) return response.status(400).send({ msg: 'Senha
    inválida'});
97
98  bcrypt.hash(password, 10, async (errBcrypt, hash) => {
99  try {
10  if(errBcrypt) return response.status(500).send(({ error: errBcrypt
0   }));
10
1
10  const result = await connection('users').select('*').where({ email
2   });
10  if(result.length === 0) return response.status(404).send({ msg:
3   'Usuário não encontrado'})
10
4
10  await connection('users').where({ email }).update({ password: hash,
5   role });
10  return response.status(200).send({ msg: 'Usuário atualizado com
6   sucesso'});
10
7
10  } catch(err) {
8
10  return response.status(500).send({ msg: err });
9
11  }
0
```

```
});

},

async verifyAccount(request, response){

try {

const { token } = request.params;

const { email } = jwt.verify(token, process.env.EMAIL_KEY);

const result = await connection('verify_email').select('*').where({ token });

if(result.length === 0) return response.status(404).send({ msg: 'Token inválido' });

await connection('users').where({ email }).update({ verified: true });

await connection('verify_email').where({ token }).delete();

return response.status(200).send("conta verificada com sucesso");

} catch(err) {

return response.status(500).send({ msg: err });

}

},

async resendEmailVerification(request, response){

try {

const { email } = request.body;

const result = await connection('users').select('*').where({ email });

if(result.length === 0) return response.status(404).send({ msg: 'Usuário não encontrado'});

const emailToken = jwt.sign({ email }, process.env.EMAIL_KEY);

await connection('verify_email').insert({

token: emailToken,
```

```
send_date: new Date(),

})

sendEmailVerification(email, emailToken);

return response.status(200).send({ msg: 'Email reenviado com sucesso' });
} catch (err) {

return response.status(500).send({ msg: err });

}

},

}



function sendEmailVerification(email, token) {

const link = `${process.env.HOST}/verify-account/${token}`

const transporter = nodemailer.createTransport({

host: "smtp-mail.outlook.com",

port: 587,

secureConnection: false,

auth: {

user: process.env.EMAIL_TCC,

pass: process.env.EMAIL_PASSWORD

},

tls: {

ciphers:'SSLv3'

}
```

```
});

const mailOptions = {

from: process.env.EMAIL_TCC,

to: email,

subject: 'Verificação de conta',

html: "Oi,<br> Por favor, clicar no seguinte link para verificar o seu email: <br><a href="+link+">Clique aqui.</a>"

};

console.log(process.env.EMAIL, process.env.EMAIL_PASSWORD)

transporter.sendMail(mailOptions, (err, info) => {

if(err) return console.log(err);

console.log(info);

} );

}
```

src\database\migrations\20220717143907_users.js

```
exports.up = function(knex) {
return knex.schema.createTable('users', function (table)
{
table.string('email').primary();
table.string('password').notNullable;
table.boolean('verified').notNullable;
table.string('name').notNullable;
})
};

exports.down = function(knex) {
return knex.schema.dropTAble('users');
};
```

src\database\migrations\20220807195912_verify_email.js

```
exports.up = function(knex) {
return knex.schema.createTable('verify_email', function (table)
{
table.string('token').primary();
table.string('send_date').notNullable;
table.string('expires_in').notNullable;
})
};

exports.down = function(knex) {
return knex.schema.dropTAble('verify_email');
};
```

src\database\migrations\20220821183158_experiments.js

```
exports.up = function(knex) {
return knex.schema.createTable('experiments', function (table)
{
table.string('id').primary();
table.string('email').notNullable;
table.string('token');
table.string('name').notNullable;
table.string('description').notNullable;
})
};

exports.down = function(knex) {
return knex.schema.dropTAble('experiments');
};
```

src\database\Schemas\Data.js

```
const mongoose = require('mongoose');

const dataSchema = new mongoose.Schema({
data: {},
```

```
email: String,
experimentId: String,
updatedAt: {
type: Date,
required: true,
default: Date.now
}
});

module.exports = mongoose.model("Data",
dataSchema);
```

src\database\connection.js

```
const knex = require('knex');
const configuration = require('../../knexfile');



const connection = knex(configuration[process.env.DB_ENV] ||
configuration.development);
connection.migrate.latest([configuration])




module.exports = connection;



```

src\Middleware\Logged.js

```
const jwt = require('jsonwebtoken');

module.exports = (req, res, next) => {
try {
const token = req.headers.authorization.split(' ')[1];
const decode = jwt.verify(token, process.env.JWT_KEY);
req.usuario = decode;
```

```
next();
} catch(err) {
return res.status(403).send({ mensagem: 'Falha na autenticação' });
}
}
```

src\index.js

```
const express = require('express');
const routes = require('./routes');
const cors = require('cors');

const app = express();

app.use(express.json());

app.use((req, res, next) => {
      //allowing localhost:3000 conection
res.header("Access-Control-Allow-Origin", "*");
      //methods
res.header("Access-Control-Allow-Methods", 'GET,PATCH,POST,DELETE');
res.header("Access-Control-Allow-Headers", ["Content-Type",
"Authorization"]);
app.use(cors());
next();
});

app.use(routes);

// porta preenchida automaticamente pelo heroku em prod
const port = process.env.PORT || 3333;
app.listen(port, () => console.log("Listening on port", port));


```

src\routes.js

```
const express = require('express');

const DataController = require('./Controllers/DataController');
const UserController = require('./Controllers/UserController');
const ExperimentController =
require('./Controllers/ExperimentController');

const routes = express.Router();

const logged = require('./Middleware/Logged');



//DataController
routes.post('/insert-data', DataController.create);
routes.get('/get-data/:experimentId',logged , DataController.index);
routes.delete('/delete-data', DataController.drop);

//UserController
routes.post('/create-user', UserController.create);
routes.delete('/delete-user', UserController.drop);
routes.post('/login', UserController.login);
routes.post('/session', UserController.session);
routes.post('/patch-password', UserController.patchPassword);
routes.get('/verify-account/:token', UserController.verifyAccount);
routes.post('/resend-email-verification', logged,
UserController.resendEmailVerification);

//ExperimentController
routes.get('/experiments',logged, ExperimentController.index);
routes.post('/experiments',logged, ExperimentController.create);
routes.delete('/experiments',logged,
ExperimentController.deleteExperiment);
routes.post('/resend-experiment-token', logged,
ExperimentController.resendExperimentToken);
routes.get('/experiment-to-export/:experimentId', logged,
ExperimentController.experimentToExport);

module.exports = routes;
```

knexfile.js

```
// Update with your config settings.

module.exports = {
development: {
client: 'sqlite3',
connection: {
filename: './src/database/db.sqlite'
},
migrations: {
directory: './src/database/migrations'
},
useNullAsDefault: true,
},

staging: {
client: 'pg',
connection: {
connectionString: process.env.DATABASE_URL,
ssl: { rejectUnauthorized: false }
},
migrations: {
directory: __dirname + '/database/migrations',
}
},

production: {
client: 'pg',
connection: {
connectionString: process.env.DATABASE_URL,
ssl: { rejectUnauthorized: false }
},
migrations: {
directory: './src/database/migrations',
}
}
```

```
};

```

package.json

```
{
"name": "backend",
"version": "1.0.0",
"description": "",
"main": "index.js",
"scripts": {
"start": "node src/index.js",
"dev": "nodemon -r dotenv/config src/index.js "
},
"keywords": [],
"author": "",
"license": "ISC",
"dependencies": {
"bcrypt": "^5.0.1",
"cors": "^2.8.5",
"decimal.js": "^10.3.1",
"dotenv": "^16.0.1",
"express": "^4.17.1",
"express-session": "^1.17.1",
"jsonwebtoken": "^8.5.1",
"knex": "^0.95.4",
"mongoose": "^6.10.5",
"nodemailer": "^6.7.7",
"pg": "^8.6.0",
"socket.io": "^4.4.1",
"sqlite3": "^5.0.2"
},
"devDependencies": {
"nodemon": "^2.0.7"
}
}
```

Apêndice C - Código do front-end

src\api\api.js

```
const apiPath = process.env.REACT_APP_API;

const requestOptions = (method, bodyContent, token) => {
let options = {};
switch (method) {
case 'POST': {
let postHeaders = {
'Content-Type': 'application/json',
};
if (token) {
postHeaders = { ...postHeaders, Authorization: `Bearer ${token}` };

}

options = {
method,
headers: postHeaders,
body: JSON.stringify(bodyContent),
}; }
break;
case 'GET': {
let getHeaders = {
'Content-Type': 'application/json',
};
if (token) {
getHeaders = { ...getHeaders, Authorization: `Bearer ${token}` };
}
options = {
method,
headers: getHeaders,
}; }
break;
```

```
case 'DELETE': {
let delHeaders = {
'Content-Type': 'application/json',
};
if (token) {
delHeaders = { ...delHeaders, Authorization: `Bearer ${token}` };
}
options = {
method,
body: JSON.stringify(bodyContent),
headers: delHeaders,
}; }
break;
case 'PATCH': {
let patchHeaders = {
'Content-Type': 'application/json',
};
if (token) {
patchHeaders = { ...patchHeaders, Authorization: `Bearer ${token}`
};
}
options = {
method,
body: JSON.stringify(bodyContent),
headers: patchHeaders,
}; }
break;
default: {
let defaultHeaders = {
'Content-Type': 'application/json',
};
if (token) {
defaultHeaders = { ...defaultHeaders, Authorization: `Bearer
${token}` };
}

options = {
```

```
method,
headers: defaultHeaders,
body: JSON.stringify(bodyContent),
}; }
break;
}

return options;
};

const api = {
getHomeData: () => new Promise((resolve) => {
const options = requestOptions('GET', null,
sessionStorage.getItem('session-token'));
fetch(`${apiPath || 'http://localhost:3333'}/experiments`, options)
.then(async (response) => {
const isJson =
response.headers.get('content-type')?.includes('application/json');
const data = isJson && await response.json();
// check for error response
if (!response.ok) {
// get error message from body or default to response status
resolve(response);
}
resolve({ response, data: data.data || data });
})
.catch((error) => {
// don't return anything => execution goes the normal way
console.error('There was an error!', error);
});
}),
createExperiment: (name, description) => new Promise((resolve) => {
const options = requestOptions('POST', { name, description },
sessionStorage.getItem('session-token'));
fetch(`${apiPath || 'http://localhost:3333'}/experiments`, options)
.then(async (response) => {
const isJson =
response.headers.get('content-type')?.includes('application/json');
```

```
const data = isJson && await response.json();

// check for error response

if (!response.ok) {

// get error message from body or default to response status

resolve(response);

}

resolve({ response, data: data.data || data });

})

.catch((error) => {

// don't return anything => execution goes the normal way

console.error('There was an error!', error);

});

}),

deleteExperiment: (experimentId) => new Promise((resolve) => {

const options = requestOptions('DELETE', { experimentId },
sessionStorage.getItem('session-token'));

fetch(`${apiPath || 'http://localhost:3333'}/experiments`, options)

.then(async (response) => {

resolve(response);

})

.catch((error) => {

// don't return anything => execution goes the normal way

console.error('There was an error!', error);

});
```

```
  }),

  resendVerificationEmail: (email) => new Promise((resolve) => {

  const options = requestOptions('POST', { email },
  sessionStorage.getItem('session-token'));

  fetch(`${apiPath ||
  'http://localhost:3333'}/resend-email-verification`, options)

  .then(async (response) => {

  resolve(response);

  })

  .catch((error) => {

  // don't return anything => execution goes the normal way

  console.error('There was an error!', error);

  });

  }),

  resendExperimentToken: (experimentId) => new Promise((resolve) => {

  const options = requestOptions('POST', { experimentId },
  sessionStorage.getItem('session-token'));

  fetch(`${apiPath ||
  'http://localhost:3333'}/resend-experiment-token`, options)

  .then(async (response) => {

  resolve(response);

  })

  .catch((error) => {

  // don't return anything => execution goes the normal way

  console.error('There was an error!', error);

  });

  }),
```

```
createAccount: ({ email, name, password }) => new Promise((resolve) => {
const options = requestOptions('POST', { name, email, password });
fetch(`${apiPath || 'http://localhost:3333'}/create-user`, options)
.then(async (response) => {
resolve(response);
})
.catch((error) => {
// don't return anything => execution goes the normal way
console.error('There was an error!', error);
});
}),
changePassword: ({ password, token }) => new Promise((resolve) => {
console.log('token', token);
console.log('password', password);
const options = requestOptions('POST', { token, password });
fetch(`${apiPath || 'http://localhost:3333'}/change-password`, options)
.then(async (response) => {
resolve(response);
})
.catch((error) => {
// don't return anything => execution goes the normal way
console.error('There was an error!', error);
});
```

```
}),

recoverPassword: ({ email }) => new Promise((resolve) => {

const options = requestOptions('POST', { email });

fetch(`${apiPath || 'http://localhost:3333'}/recover-password`,
options)

.then(async (response) => {

resolve(response);

})

.catch((error) => {

// don't return anything => execution goes the normal way

console.error('There was an error!', error);

});

}),

getExperimentById: (id) => new Promise((resolve) => {

const options = requestOptions('GET', null,
sessionStorage.getItem('session-token'));

fetch(`${apiPath || 'http://localhost:3333'}/get-data/${id}`,
options)

.then(async (response) => {

const isJson =
response.headers.get('content-type')?.includes('application/json');

const data = isJson && await response.json();

// check for error response

if (!response.ok) {

// get error message from body or default to response status

resolve(response);

}
```

```
resolve({ response, data: data.data || data });

})

.catch((error) => {

// don't return anything => execution goes the normal way

console.error('There was an error!', error);

});

}),

getDataToExport: (id) => new Promise((resolve) => {

const options = requestOptions('GET', null,
sessionStorage.getItem('session-token'));

fetch(`${apiPath ||
'http://localhost:3333'}/experiment-to-export/${id}`, options)

.then(async (response) => {

const isJson =
response.headers.get('content-type')?.includes('application/json');

const data = isJson && await response.json();

// check for error response

if (!response.ok) {

// get error message from body or default to response status

resolve(response);

}

resolve({ response, data: data.data || data });

})

.catch((error) => {

// don't return anything => execution goes the normal way

console.error('There was an error!', error);
```

```
});

}),

login: (username, password) => new Promise((resolve) => {

console.log(username, password);

const options = requestOptions('POST', { email: username, password });

fetch(`${apiPath || 'http://localhost:3333'}/login`, options)

.then(async (response) => {

const isJson = response.headers.get('content-type')?.includes('application/json');

const data = isJson && await response.json();

// check for error response

if (!response.ok) {

resolve(response);

}

sessionStorage.setItem('session-token', data.token);

resolve({ ok: response.ok, data });

})

.catch((error) => {

// don't return anything => execution goes the normal way

console.error('There was an error!', error);

});

}),

verifySession: () => new Promise((resolve) => {

const options = requestOptions('POST', { token: sessionStorage.getItem('session-token') });
```

```
fetch(`${apiPath || 'http://localhost:3333'}/session`, options)
.then(async (response) => {
const isJson = response.headers.get('content-type')?.includes('application/json');
const data = isJson && await response.json();
// check for error response
if (!response.ok) {
// get error message from body or default to response status
const error = { error: (data && data.msg) || response.statusText || response.status };
resolve(error);
}
resolve({ ok: response.ok, data });
})
.catch((error) => {
// don't return anything => execution goes the normal way
console.error('There was an error!', error);
});
}),
};


export { api, apiPath, requestOptions };
```

src\pages\ChangePassword.jsx

```
import React, { useState, useEffect } from 'react';
import { useParams, useHistory } from 'react-router-dom';
import {
Button, Alert, TextField,
} from '@mui/material';
import { api } from '../api/api';

function ChangePassword() {
const [msg, setMsg] = useState(null);
const [successMsg, setSuccessMsg] = useState(null);
const history = useHistory();
const [password, setPassword] = useState(null);
const { token } = useParams();
const [verifyPassword, setVerifyPassword] = useState(null);
useEffect(() => {
if (password !== verifyPassword && verifyPassword !== null) {
setMsg('Passwords do not match');
} else {
setMsg(false);
}
}, [verifyPassword, password]);

function handleSetPassword(e) {
setPassword(e.target.value);
}
function handleSetVerifyPassword(e) {
setVerifyPassword(e.target.value);
}
function handleSubmit() {
api.changePassword({ password, token }).then((response) => {
console.log(response);
if (response.status === 403) {
setMsg('Invalid token');
}
if (response.ok) {
setSuccessMsg('Password changed');
```

```
}
});
}
function handleGoBack() {
history.push('/');
}
return (
<div style={{ display: 'flex', flexDirection: 'column' }}>
<TextField
style={{ margin: '5px 0' }}
onChange={handleSetPassword}
id="password"
type="password"
label="New password"
variant="outlined"
/>
<TextField
style={{ margin: '5px 0' }}
onChange={handleSetVerifyPassword}
id="verifyPassword"
type="password"
label="Repeat password"
variant="outlined"
/>
<Button
style={{ margin: '5px 0' }}
onClick={handleSubmit}
variant="contained"
>
Change
</Button>
<Button
style={{ margin: '5px 0' }}
onClick={handleGoBack}
variant="contained"
>
Back
```

```
</Button>
{msg && (
<Alert
style={{
position: 'fixed', bottom: '30px', right: '30px', maxWidth:
'calc(100% - 90px)',
}}
onClose={() => setMsg(null)}
severity="error"
>
{msg}
</Alert>
)}
{successMsg && (
<Alert
style={{
position: 'fixed', bottom: '80px', right: '30px', maxWidth:
'calc(100% - 90px)',
}}
onClose={() => setSuccessMsg(null)}
severity="success"
>
{successMsg}
</Alert>
)}
</div>
);
}


export default ChangePassword;
```

src\pages\CreateAccount.jsx

```
import React, { useState, useEffect } from 'react';
import {
```

```
3  Button, Alert, TextField,
4  } from '@mui/material';
5  import { api } from '../api/api';
6
7  function CreateAccount() {
8  const [email, setEmail] = useState(null);
9  const [msg, setMsg] = useState(null);
10 const [successMsg, setSuccessMsg] = useState(null);
11 const [password, setPassword] = useState(null);
12 const [verifyPassword, setVerifyPassword] = useState(null);
13 const [name, setName] = useState(null);
14 // eslint-disable-next-line max-len
15 const emailRegex =
   /[a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:\.[a-z0-9!#$%&'*+/=?^_`{|}~-]+)*@(?:
   [a-z0-9](?:[a-z0-9-]*[a-z0-9])?\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])?/g
   i;
16
17 useEffect(() => {
18 if (password !== verifyPassword && verifyPassword !== null) {
19 setMsg('Passwords do not match');
20 } else {
21 setMsg(false);
22 }
23 }, [verifyPassword, password]);
24
25 function handleSetName(e) {
26 setName(e.target.value);
27 }
28 function handleSetEmail(e) {
29 setEmail(e.target.value);
30 }
31 function handleSetPassword(e) {
32 setPassword(e.target.value);
33 }
34 function handleSetVerifyPassword(e) {
35 setVerifyPassword(e.target.value);
36 }
37
```

```
function handleSubmit() {
if (!emailRegex.test(email)) {
setMsg('Email is not valid');
return;
}
if (!name) {
setMsg('Missing name');
return;
}
api.createAccount({ email, password, name }).then((response) => {
console.log(response);
if (response.status === 409) {
setMsg('Email already in use');
}
if (response.ok) {
setSuccessMsg('Account created');
}
});
}
function handleGoBack() {
window.history.back();
}
return (
<div style={{ display: 'flex', flexDirection: 'column' }}>
<TextField
style={{ margin: '5px 0' }}
onChange={handleSetName}
id="name"
label="Name"
variant="outlined"
/>
<TextField
style={{ margin: '5px 0' }}
onChange={handleSetEmail}
id="email"
label="Email"
variant="outlined"
```

```
/>
<TextField
style={{ margin: '5px 0' }}
onChange={handleSetPassword}
id="password"
type="password"
label="Password"
variant="outlined"
/>
<TextField
style={{ margin: '5px 0' }}
onChange={handleSetVerifyPassword}
id="verifyPassword"
type="password"
label="Repeat password"
variant="outlined"
/>
<Button
style={{ margin: '5px 0' }}
onClick={handleSubmit}
variant="contained"
>
Create
</Button>
<Button
style={{ margin: '5px 0' }}

onClick={handleGoBack}

variant="contained"


>

Back

</Button>

{msg && (

```

```
<Alert

style={{

position: 'fixed', bottom: '30px', right: '30px', maxWidth:
'calc(100% - 90px)',

}}

onClose={() => setMsg(null)}

severity="error"

>

{msg}

</Alert>

)}

{successMsg && (

<Alert

style={{

position: 'fixed', bottom: '80px', right: '30px', maxWidth:
'calc(100% - 90px)',

}}

onClose={() => setSuccessMsg(null)}

severity="success"

>

{successMsg}

</Alert>

)}

</div>

);
```

```
}


export default CreateAccount;
```

src\pages\ExperimentDetail.jsx

```jsx
import React, { useEffect, useState } from 'react';
import { useParams, Link } from 'react-router-dom';
import {
Alert,
} from '@mui/material';
import { api } from '../api/api';
import styles from './styles.module.css';
import ChartCard from '../components/ChartCard/ChartCard';
import SideBar from '../components/sidebar/SideBar';
import MobileDimensionsBar from
'../components/sidebar/MobileDimensionsBar';

function ExperimentDetail() {
const [isMobile] = useState(window.innerWidth < 768);
const [chartData, setChartData] = useState(null);
const [selectedDimension, setSelectedDimension] = useState(null);
const [msg, setMsg] = useState(null);
const { id } = useParams();
useEffect(() => {
if (id) {
api.getExperimentById(id).then(({ data }) => {
if (Object.values(data).length !== 0) {
setSelectedDimension(Object.values(data)[0]?.key);
setChartData(data);
} else {
setMsg('Não existem dados para esse experimento');
}
});
```

```
28  }
29  }, [id]);
30  return (
31  <div className={isMobile ? styles.page_mobile : styles.page}>
32  <div style={{ position: 'absolute', left: '25px', top: isMobile ?
    '28px' : '25px' }}>
33  <Link style={{ textDecoration: 'none' }} to="/">Home</Link>
34  </div>
35  {msg && (
36  <Alert
37  style={{
38  position: 'fixed', bottom: '30px', right: '30px', maxWidth:
    'calc(100% - 90px)',
39  }}
40  onClose={() => setMsg(null)}
41  severity="error"
42  >
43  {msg}
44  </Alert>
45  )}
46  {chartData && !isMobile && (
47  <SideBar
48  data={chartData}
49  selectedDimension={selectedDimension}
50  setSelectedDimension={setSelectedDimension}
51  />
52  )}
53  {chartData && isMobile && (
54  <MobileDimensionsBar
55  data={chartData}
56  selectedDimension={selectedDimension}
57  setSelectedDimension={setSelectedDimension}
58  />
59  )}
60  {chartData && <ChartCard data={chartData}
    selectedDimension={selectedDimension} />}
61  </div>
62  );
```

```
}

export default ExperimentDetail;

```

src\pages\ForgotPassword.jsx

```
import React, { useState } from 'react';
import {
Button, Alert, TextField,
} from '@mui/material';
import { api } from '../api/api';

function ForgotPassword() {
const [email, setEmail] = useState(null);
const [msg, setMsg] = useState(null);
const [successMsg, setSuccessMsg] = useState(null);
// eslint-disable-next-line max-len
const emailRegex = /[a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:\.[a-z0-9!#$%&'*+/=?^_`{|}~-]+)*@(?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])?/gi;

function handleSetEmail(e) {
setEmail(e.target.value);
}

function handleSubmit() {
if (!emailRegex.test(email)) {
setMsg('Email is not valid');
return;
}
api.recoverPassword({ email }).then((response) => {
console.log(response);
if (response.ok) {
setSuccessMsg('If this email exists, you will receive an email with instructions to recover your password');
} else {
setMsg('Something went wrong');
```

```
}
});
}
function handleGoBack() {
window.history.back();
}
return (
<div style={{ display: 'flex', flexDirection: 'column' }}>
<TextField
style={{ margin: '5px 0' }}
onChange={handleSetEmail}
id="email"
label="Email"
variant="outlined"
/>

<Button
style={{ margin: '5px 0' }}
onClick={handleSubmit}
variant="contained"
>
Recover
</Button>
<Button
style={{ margin: '5px 0' }}
onClick={handleGoBack}
variant="contained"
>
Back
</Button>
{msg && (
<Alert
style={{
position: 'fixed', bottom: '30px', right: '30px', maxWidth:
'calc(100% - 90px)',
}}
onClose={() => setMsg(null)}
```

```
severity="error"
>
{msg}
</Alert>
)}
{successMsg && (
<Alert
style={{
position: 'fixed', bottom: '80px', right: '30px', maxWidth:
'calc(100% - 90px)',
}}
onClose={() => setSuccessMsg(null)}
severity="success"
>
{successMsg}
</Alert>
)}
</div>
);
}

export default ForgotPassword;

```

src\pages\Home.jsx

```
import React, { useEffect, useState } from 'react';
import {
Typography, Button, Alert, Modal, Box, TextField,
} from '@mui/material';
import { Add } from '@mui/icons-material';
import HomeCard from '../components/HomeCard/HomeCard';
import styles from './styles.module.css';
import { api } from '../api/api';

const style = {
position: 'absolute',
```

```
12  display: 'flex',
13  flexDirection: 'column',
14  top: '50%',
15  left: '50%',
16  transform: 'translate(-50%, -50%)',
17  width: window.innerWidth < 768 ? 250 : 400,
18  bgcolor: 'background.paper',
19  border: '2px solid #000',
20  boxShadow: 24,
21  p: 4,
22  };
23
24  function Home({ logged }) {
25  const [isMobile] = useState(window.innerWidth < 768);
26  const [modalDeleteExperiment, setModalDeleteExperiment] = useState({
    open: false });
27  const [experiments, setExperiments] = useState([]);
28  const [successMsg, setSuccessMsg] = useState(null);
29  const [msg, setMsg] = useState(null);
30  const [modal, setModal] = useState(false);
31  const [experimentName, setExperimentName] = useState('');
32  const [experimentDescription, setExperimentDescription] =
    useState('');
33  function getData() {
34  api.getHomeData().then(({ response, data }) => {
35  if (response.ok) {
36  setExperiments(data.experiments);
37  }
38  });
39  }
40  useEffect(() => {
41  if (logged) {
42  getData();
43  }
44  }, [logged]);
45  function handleCreateExperiment() {
46  if (!logged.verified) {
```

```
47 setMsg('Please, verify your email before creating an experiment');
48 return;
49 }
50 setModal(true);
51 }
52
53 function handleClose() {
54 setModal(false);
55 }
56 function handleSetExperimentName(e) {
57 setExperimentName(e.target.value);
58 }
59 function handleExperimentDescription(e) {
60 setExperimentDescription(e.target.value);
61 }
62 function handleSubmit() {
63 api.createExperiment(experimentName, experimentDescription).then(({
   response }) => {
64 if (response.ok) {
65 setSuccessMsg('Experiment created');
66 getData();
67 handleClose();
68 }
69 });
70 }
71 function handleResendExperimentToken(experimentId) {
72 api.resendExperimentToken(experimentId).then((response) => {
73 if (response.ok) {
74 setSuccessMsg('Token sent');
75 } else {
76 setMsg('Error sending token');
77 }
78 });
79 }
80 function handleDeleteExperiment() {
81 api.deleteExperiment(modalDeleteExperiment.id).then((response) => {
82 if (response.ok) {
```

```
setSuccessMsg('Experiment deleted');
getData();
setModalDeleteExperiment({ open: false });
}
});
}
return (
<div className={isMobile ? styles.page_mobile : styles.page}>
<div style={{ margin: '5px', width: '230px', maxHeight: '300px' }}>
<Button onClick={handleCreateExperiment}
className={styles.create_experiment}>
<Add sx={{ color: '#b1b1b1' }} fontSize="large" />
</Button>
</div>
{experiments.length > 0 && experiments.map((experiment) => (
<HomeCard
key={experiment.id}
experiment={experiment}
setModalDeleteExperiment={setModalDeleteExperiment}
handleResendExperimentToken={handleResendExperimentToken}
/>
))}
{msg && (
<Alert
style={{
position: 'fixed', bottom: '30px', right: '30px', maxWidth:
'calc(100% - 90px)',
}}
onClose={() => setMsg(null)}
severity="error"
```

```
>

{msg}

</Alert>

)}

{successMsg && (

<Alert

style={{

position: 'fixed', bottom: '30px', right: '30px', maxWidth:
'calc(100% - 90px)',

}}

onClose={() => setSuccessMsg(null)}

severity="success"

>

{successMsg}

</Alert>

)}

<Modal

open={modal}

onClose={handleClose}

aria-labelledby="modal-modal-title"

aria-describedby="modal-modal-description"

>

<Box sx={style}>

<TextField
```

```
style={{ margin: '5px 0' }}
onChange={handleSetExperimentName}
id="experiment-name"
label="Experiment name"
variant="outlined"
/>
<TextField
style={{ margin: '5px 0' }}
onChange={handleExperimentDescription}
id="description"
label="Description"
variant="outlined"
/>
<Button
style={{ margin: '5px 0' }}
onClick={handleSubmit}
variant="contained"
>
Create
</Button>
</Box>
</Modal>
<Modal
```

```
open={modalDeleteExperiment.open}
onClose={handleClose}
aria-labelledby="modal-modal-title"
aria-describedby="modal-modal-description"
>
<Box sx={style}>
<Typography
variant="h5"
>
{`Are you sure you want to delete this experiment
(${modalDeleteExperiment.name})?`}
</Typography>
<Button
style={{ margin: '5px 0' }}
onClick={handleDeleteExperiment}
variant="contained"
>
Yes
</Button>
<Button
style={{ margin: '5px 0' }}
onClick={() => setModalDeleteExperiment(false)}
variant="contained"
>
```

```
No
</Button>
</Box>
</Modal>
</div>
);
}


export default Home;


```

src\pages\Layout.jsx

```jsx
import React, { useEffect, useState } from 'react';
import { Button, Alert } from '@mui/material';
import {
Route, BrowserRouter as Router, Switch, Redirect,
} from 'react-router-dom';
import styles from './styles.module.css';
import ExperimentDetail from './ExperimentDetail';
import Home from './Home';
import CreateAccount from './CreateAccount';
import ForgotPassword from './ForgotPassword';
import LoginScreen from './Login';
import SideBar from '../components/sidebar/SideBar';
import { api } from '../api/api';
import ChangePassword from './ChangePassword';

function Layout() {
const [data] = useState(null);
const [successMsg, setSuccessMsg] = useState(null);
```

```
const [redirectState, setRedirectState] = useState(false);
const [logged, setLogged] = useState(null);
const [selectedDimension, setSelectedDimension] = useState(null);

useEffect(() => {
const token = sessionStorage.getItem('session-token');
if (token && !logged) {
api.verifySession().then((response) => {
if (response.ok) {
setRedirectState(false);
setLogged(response.data?.user);
} else {
setRedirectState(true);
}
});
} else {
setRedirectState(true);
}
}, []);

useEffect(() => {
if (!data) return;
const arr = Object.values(data.data);
setSelectedDimension(arr[0].key);
}, [data]);

useEffect(() => {
console.log(logged);
}, [logged]);

function handleLogout() {
sessionStorage.removeItem('session-token');
setLogged(null);
setRedirectState(true);
}

```

```
function handleResendVerificationEmail() {
api.resendVerificationEmail(logged.email).then((response) => {
console.log(response);
if (response.ok) {
setSuccessMsg('Email enviado com sucesso');
}
});
}
const redirect = redirectState ? <Redirect path="/" to="/login" /> :
'';
return (
<div className={styles.app_wrapper}>
{data && selectedDimension && logged
&& <SideBar selectedDimension={selectedDimension}
setSelectedDimension={setSelectedDimension} data={data.data} />}
<div className={styles.page_content}>
{logged && (
<div className={styles.header_main}>
{!logged.verified && <Button
onClick={handleResendVerificationEmail}>Resend verification
email</Button>}
<span style={{ marginRight: '5px' }}>{logged.name}</span>
<Button onClick={handleLogout} variant="contained">Logout</Button>

</div>
)}
<Router>
<Switch>
<Route path="/create-account">
<CreateAccount />
</Route>
<Route path="/forgot-password">
<ForgotPassword />
</Route>
<Route path="/change-password/:token">
<ChangePassword />
</Route>
<Route path="/login">
<LoginScreen
```

```
setRedirectState={setRedirectState}
logged={logged}
setLogged={setLogged}
/>
</Route>
{redirect}
<Route path="/experiment/:id">
<ExperimentDetail logged={logged} handleLogout={handleLogout} />
</Route>
<Route path="/">
<Home logged={logged} handleLogout={handleLogout} />
</Route>
</Switch>
</Router>
</div>
{successMsg && (
<Alert
style={{
position: 'fixed', bottom: '30px', right: '30px', maxWidth:
'calc(100% - 90px)',
}}
onClose={() => setSuccessMsg(null)}
severity="success"
>
{successMsg}
</Alert>
)}
```

```
</div>

);

}


export default Layout;


```

src\pages\Login.jsx

```
import React, { useState, useEffect } from 'react';
import { Button, TextField, Alert } from '@mui/material';
import {
useHistory,
} from 'react-router-dom';
import styles from './styles.module.css';
import { api } from '../api/api';

function Login({
setLogged, logged, setRedirectState,
}) {
const history = useHistory();
const [username, setUsername] = useState('');
const [password, setPassword] = useState('');
const [msg, setMsg] = useState(false);

const handleUser = (event) => {
setUsername(event.target.value);
};
const handlePassword = (event) => {
setPassword(event.target.value);
};

const handleSubmit = () => {
```

```
api.login(username, password).then((response) => {
console.log(response);
if (response.status === 401) {
setMsg('User or password incorrect');
}
if (response.ok) {
setLogged(response.data?.user);
}
});
};

function handleCreateAccount() {
history.push('/create-account');
}

function handleForgotPassword() {
history.push('/forgot-password');
}

useEffect(() => {
if (logged) {
history.push('/');
setRedirectState(false);
}
}, [logged]);

function submitWithEnter(e) {
if (msg) {
setMsg(false);
}
if (e.key === 'Enter') {
handleSubmit();
}
}

return (
```

```
<div className={window.innerWidth < 820 ? styles.page_mobile_login : styles.page_login}>
<TextField
style={{ margin: '5px 0' }}
onKeyDown={submitWithEnter}
onChange={handleUser}
id="user"
label="email"
variant="outlined"
/>
<TextField
style={{ margin: '5px 0' }}
onKeyDown={submitWithEnter}
onChange={handlePassword}
type="password"
id="password"
label="password"
variant="outlined"
/>
<Button
style={{ margin: '5px 0' }}
onClick={handleSubmit}
variant="contained"
>
Login

</Button>
<div>
<Button
style={{ margin: '5px 5px 5px 0' }}
onClick={handleCreateAccount}
variant="contained"
>
Create Account
</Button>
<Button
style={{ margin: '5px 0 5px 5px' }}
```

```
onClick={handleForgotPassword}
variant="contained"
>
Forgot Password
</Button>
</div>


{msg && (
<Alert
style={{
position: 'fixed', bottom: '30px', right: '30px', maxWidth:
'calc(100% - 90px)',
}}
onClose={() => setMsg(null)}
severity="error"
>
{msg}
</Alert>
)}
</div>
);
}


export default Login;
```

src\pages\styles.module.css

```css
.app_wrapper {
margin: 0;
box-sizing: border-box;
height: 100%;
display: flex;
font-family: "Roboto";
box-sizing: border-box;
}
.header_main {
display: flex;
padding: 15px 0;
font-family: "Roboto";
align-items: center;
width: calc(100% - 20px);
display: flex;
justify-content: flex-end;
}

.experiment_box{
min-width: 230px;
padding: 10px;
margin: 5px;
min-height: 120px;
max-height: 300px;
flex: 1;
border-radius: 3px;
display: flex;
flex-direction: column;
box-sizing: border-box;
box-shadow: rgb(0 0 0 / 18%) 0px 0px 7px 1px;
justify-content: space-between;
}
.create_experiment {
width: 100%;
height: 100%;
```

```
box-shadow: rgb(0 0 0 / 18%) 0px 0px 7px 1px;
}

.page_content {
display: flex;
box-sizing: border-box;
flex: 1;
flex-direction: column;
padding: 0 10px 10px 10px;
align-items: center;
justify-content: center;
background-color: #f5f5f5;
}

.nav_item {
color: black!important;
font-weight: 500;
text-transform: uppercase!important;
font-size: 20px;
text-decoration: none!important;
margin: 0 10px!important;
}

.nav_main {
display: flex;
align-items: center;
}

.title_nav {
display: flex;
align-items: center;
justify-content: space-between;
flex: 1;
}

.dual_card_holder {
```

```
display: flex;
flex: 1;
}

.dual_card_holder_mobile {
display: flex;
flex-direction: column;
}

.main_content {
flex: 1;
display: flex;
}

.all_content {
height: 100%;
display: flex;
flex-direction: column;
}

.page, .page_mobile {
padding: 10px;
max-height: calc(100% - 87px);
flex: 1;
flex-wrap: wrap;
background-color: #e3e2e2;
display: flex;
width: calc(100% - 20px);
font-family: "Roboto";

}



.page_mobile{

padding: 0px 0 10px 0;
```

```
display: flex;

background-color: transparent;

flex-direction: column;

}



.page_mobile > .card_main {

margin: 0;

max-width: 100%;

border: 0;

}



.page_login, .page_mobile_login {

padding: 10px;

background-color: #e3e2e2;

display: flex;

flex-direction: column;

font-family: "Roboto";

margin-bottom: 30px;

border-radius: 5px;

}



.page_mobile{

display: flex;
```

```
overflow: auto;

flex-wrap: nowrap;

flex-direction: column;

}



.cheap_scientist, .cheap_scientist_mobile {

font-family: "Roboto";

font-size: 20px;

margin: 0;

font-weight: bold;

}



.cheap_scientist_mobile{

font-size: 30px;

}



@font-face {

font-family: "Dancing Script";

src:
url("../assets/Fonts/DancingScript-VariableFont_wght.ttf"),

}



.export_button_icon {

font-size: 30px;
```

```
}


.export_button_container {
align-items: center;
display: flex;
justify-content: flex-end;
font-size: 26px;
}


.export_button_container>button {
padding: 0;
color: black
}


.contatos,
.contatos_mobile {
text-align: center;
flex: 1;
display: flex;
justify-content: center;
align-items: center;


}
```

```
174

175  .contatos_mobile {
176  flex-direction: column;
177  }
178

179  .contato_foto, .contato_foto_mobile{
180

181  height: 400px;
182  border-radius: 30px;
183  }
184

185  .contato_foto_mobile{
186

187  height: 300px;
188  }
189

190  .contato_texto, .contato_texto_mobile {
191  font-size: 25px;
192  font-family: "Roboto";
193        display: inline-block;
194  margin-left: 60px;
195  }
196
```

```
.contato_texto_mobile{
margin-left: 0;
}

.meu_nome {
font-family: "Dancing Script";
font-size: 70px;
}

.meu_nome:first-child {
      font-weight: bold;
}

.telefone:before{
content: "📞 ";
}

.modal {
position: fixed;
z-index: 8;
width: 360px;
top: calc(50% - 100px);
right: calc(50% - 180px);
```

```
220 background-color: #526a7e;
221 border-radius: 15px;
222 padding: 20px;
223 display: flex;
224 align-items: center;
225 flex-direction: column;
226 font-family: "Roboto";
227
228 }
229
230 .confirm_button {
231 color: red
232 }
233
234 .modal_buttons {
235 width: 300px;
236 display: flex;
237 justify-content: space-around;
238 }
239
240 .modal_phrase {
241 margin-bottom: 30px;
242 }
```

```
243

.to_close_modal {
244
z-index: 6;
245
position: fixed;
246
top: 0;
247
bottom: 0;
248
right: 0;
249
left: 0;
250
background-color: transparent;
251
border: none;
252
}
253

254

.email:before{
255
content: "📩 ";
256
}
257
```

src\App.jsx

```
1  import React from 'react';
2  import Layout from
   './pages/Layout';
3
4  function App() {
5  return (
6  <Layout />
7  );
8  }
9
```

```
export default App;

```

src\index.jsx

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';
import reportWebVitals from './reportWebVitals';

ReactDOM.render(
<React.StrictMode>
<App />
</React.StrictMode>,
document.getElementById('root'),
);

// If you want to start measuring performance in your app, pass a
function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more:
https://bit.ly/CRA-vitals
reportWebVitals();

```

src\reportWebVitals.js

```
const reportWebVitals = (onPerfEntry) => {
if (onPerfEntry && onPerfEntry instanceof Function)
{
import('web-vitals').then(({
getCLS, getFID, getFCP, getLCP, getTTFB,
}) => {
getCLS(onPerfEntry);
getFID(onPerfEntry);
getFCP(onPerfEntry);
getLCP(onPerfEntry);
getTTFB(onPerfEntry);
});
```

```
}
};

export default reportWebVitals;

```

src\setupTests.js

```
// jest-dom adds custom jest matchers for asserting on DOM nodes.
// allows you to do things like:
// expect(element).toHaveTextContent(/react/i)
// learn more: https://github.com/testing-library/jest-dom
import '@testing-library/jest-dom';

```

src\Utils.jsx

```
const getDate = (timestamp) => {
var date = new Date(timestamp * 1000);
const time = date.toLocaleDateString("pt-BR");
return time;
}

const dateToTimeStamp = (date) => {
const dateWithoutTime = date.setHours(0, 0, 0, 0);
const timestamp = Math.floor(dateWithoutTime / 1000);
return timestamp;
}

const Utils = {
getDate: getDate,
dateToTimeStamp: dateToTimeStamp
};

export default Utils;
```

package.json

```
{
"name": "frontend",
"version": "0.1.0",
"private": true,
"engines": {
"node": "17.4.0"
},
"dependencies": {
"@date-io/date-fns": "^1.3.13",
"@emotion/react": "^11.7.1",
"@emotion/styled": "^11.6.0",
"@material-ui/core": "^4.11.4",
"@material-ui/data-grid": "^4.0.0-alpha.37",
"@material-ui/icons": "^4.11.2",
"@material-ui/lab": "^4.0.0-alpha.58",
"@material-ui/pickers": "^3.3.10",
"@mui/icons-material": "^5.11.16",
"@mui/material": "^5.4.0",
"@mui/styles": "^5.0.1",
"@mui/x-data-grid": "^5.0.0-beta.2",
"@testing-library/jest-dom": "^5.11.4",
"@testing-library/react": "^11.1.0",
"@testing-library/user-event": "^12.1.10",
"date-fns": "^2.28.0",
"echarts-for-react": "^3.0.2",
"express": "^4.18.2",
"react": "^17.0.2",
"react-dom": "^17.0.2",
"react-number-format": "^4.5.5",
"react-router-dom": "^5.2.0",
"react-scripts": "^5.0.1",
"web-vitals": "^1.0.1"
},
"scripts": {
"start": "react-scripts start",
"build": "react-scripts --openssl-legacy-provider build",
```

```
"test": "react-scripts test",
"eject": "react-scripts eject"
},
"eslintConfig": {
"extends": [
"react-app",
"react-app/jest"
]
},
"browserslist": {
"production": [
">0.2%",
"not dead",
"not op_mini all"
],
"development": [
"last 1 chrome version",
"last 1 firefox version",
"last 1 safari version"
]
},
"devDependencies": {
"eslint": "^8.19.0",
"eslint-config-airbnb": "^19.0.4",
"eslint-plugin-import": "^2.26.0",
"eslint-plugin-jsx-a11y": "^6.6.0",
"eslint-plugin-react": "^7.30.1",
"eslint-plugin-react-hooks": "^4.6.0"
}
}

```

Procfile

```
web: node scripts/heroku-start.js
```

Apêndice D - Código do Arduino

```
// Definição das bibliotecas utilizadas e portas para temperatura
#include <OneWire.h>
#include <DallasTemperature.h>
#define ONE_WIRE_BUS 8
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);
DeviceAddress sensorTemperatura;

#include <Wire.h>
#include <SoftwareSerial.h>
#include <ArduinoHttpClient.h>
#include <ArduinoJson.h>

SoftwareSerial s(5, 6);

// ------ CONEXÃO COM WIFI ------

const char* ssid = "SEU_SSID";
const char* password = "SUA_SENHA";
const char* host = "tcc-cae-vini-backend.herokuapp.com";
const int port = 80;

WiFiClient client;
HttpClient httpClient = HttpClient(client, host, port);

// ------ SENSOR DE LUZ ------

#define analogLDR A0
int sensorLDR = 0.0;
float voltagemLDR;

// ------ GERAL ------

void setup() {
  Serial.begin(9600);

  // ------ SENSOR DE LUZ ------
  pinMode(analogLDR, INPUT);

  // ------ SENSOR DE TEMPERATURA ------
  sensors.begin();
  sensors.getAddress(sensorTemperatura, 0);

  // ------ CONEXÃO COM WIFI ------
```

```
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
  Serial.println("Connected to WiFi");
}

void loop() {
  // ------ SENSOR DE LUZ ------
  sensorLDR = analogRead(analogLDR);
  voltagemLDR = sensorLDR * (5.0/1023.0);

  // ------ SENSOR DE TEMPERATURA ------
  sensors.requestTemperatures();
  float temperatura = sensors.getTempC(sensorTemperatura);

  String log_final = "t=" + String(temperatura) + "l=" + String(voltagemLDR) + "@";
  Serial.print(log_final);

  if (WiFi.status() == WL_CONNECTED) {
    String dataToPost = "{\"temperatura\":" + String(temperatura) + ",\"luminosidade\":" +
String(voltagemLDR) + "}";
    httpClient.setDefaultHeaders("token: SEU_TOKEN");
    httpClient.post("/insert-data", "application/json", dataToPost);
    delay(1000);
  }

  delay(900000);
}
```