****

**FACULDADE DE TECNOLOGIA DE TAUBATÉ**

**AISLAN LIMA PEREIRA MARTINS**

**GABRIEL BATISTA ROCHA**

##### **PROTÓTIPO DE DISPOSITIVO MICROCONTROLADOR PARA SISTEMA DE NOTIFICAÇÃO DE ESTADO DE CLAMP**

**TAUBATÉ**

**2022**

****

**FACULDADE DE TECNOLOGIA DE TAUBATÉ**

**AISLAN LIMA PEREIRA MARTINS**

**GABRIEL BATISTA ROCHA**

##### **PROTÓTIPO DE DISPOSITIVO MICROCONTROLADOR PARA SISTEMA DE NOTIFICAÇÃO DE ESTADO DE CLAMP**

Trabalho de Graduação apresentado à Coordenação do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas do Centro Estadual de Educação Tecnológica Paula Souza para a obtenção do diploma de Tecnólogo em Análise e Desenvolvimento de Sistemas.

**Orientador: Prof.** **Dra. Divani Barbosa Gavinier**

**Coorientador: Me. Michel Robert Veiga**

**TAUBATÉ**

**2022**

**AISLAN LIMA PEREIRA MARTINS**

**GABRIEL BATISTA ROCHA**

##### **PROTÓTIPO DE DISPOSITIVO MICROCONTROLADOR PARA SISTEMA DE NOTIFICAÇÃO DE ESTADO DE CLAMP**

Trabalho de Graduação apresentado a Faculdade de Tecnologia de Taubaté, como parte das exigências para a obtenção do diploma de Tecnólogo em Análise e Desenvolvimento de Sistemas.

**Orientador: Prof. Dra. Divani Barbosa Gavinier**

**Coorientador: Me. Michel Robert Veiga**

Taubaté, 12 de dezembro de 2022.

**BANCA EXAMINADORA**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Prof. Dra. Divani Barbosa Gavinier

Fatec Taubaté

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Prof. Me. Michel Robert Veiga

Fatec Taubaté

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Prof. Me. Luiz Eduardo Souza Evangelista

Fatec Taubaté

**RESUMO**

Com o avanço da tecnologia, surge um novo campo denominado “Health Tech”, cujo objetivo é realizar a junção do meio da saúde com os meios de tecnologias, gerando assim novos procedimentos, produtos e serviços. Existe uma gama de produtos que unem as duas áreas, desde aparelhos para aferir pressão até inteligências artificiais que colaboram junto ao médico em diagnósticos. Em meio uma área que abrange diversos projetos, com grandes aportes, ainda há necessidade no que diz respeito às tecnologias que facilitem o dia a dia nos hospitais. Tecnologias essas que sejam viáveis e de fácil implementação para pronto funcionamento.

Realiza-se então, o desenvolvimento de uma aplicação web integrada a um microcontrolador que tem como objetivo enviar requisições para a aplicação web informando desta forma, o estado atual (aberto ou fechado) de dispositivos hospitalares como drenos e sondas. Evitando desta forma, uma obstrução não intencional destes dispositivos.

**Palavras-Chave**: Saúde. Tecnologia. Drenos. Sondas. NodeMCU.

**ABSTRACT**

With the advancement of technology, a new field called “Health Tech” emerges, whose objective is to stimulate the health environment with the means of technology, thus generating new procedures, products and services. There is a range of products that unite the two areas, from devices to measure pressure to artificial intelligence that collaborate with the doctor in diagnoses. In the midst of an area that encompasses several projects, with large contributions, there is still a need for technologies that facilitate daily life in hospitals. Technologies that are viable and easy to implement for ready operation.

Then, the development of a web application integrated to a microcontroller is carried out, which aims to send requests to the web application, thus informing the current status (open or closed) of hospital devices such as drains and probes. Thus avoiding an unintentional sequence of these devices.

**Key words**: Health. Technology. Drains. Probes. NodeMCU.

**LISTA DE ILUSTRAÇÕES**

[Figura 1 - Dispositivo de drenagem pleural com dispositivo clamp 14](#_Toc444609424)

[Figura 2 - Dispositivo NodeMCU utilizando botão para mudança de estado 18](#_Toc444609425)

[Figura 3 - Ilustração de um clamp oclusor utilizado em drenos e sondas 18](#_Toc444609426)

[Figura 4 -  Tela de Login 22](#_Toc444609427)

[Figura 5 - Tela painel administrativo 22](#_Toc444609428)

[Figura 6 – Tela cadastro novo usuário 23](#_Toc444609428)

[Figura 7 – Tela cadastro novo leito 23](#_Toc444609428)

[Figura 8 – Tela cadastro novo dispositivo clamp 23](#_Toc444609428)

[Figura 9 - Trecho do código responsável por atualizar o estado do dispositivo clamp...... 24](#_Toc444609429)

[Figura 10 - Trecho do código do painel administrativo responsável por listar os leitos e seus dispositivos atrelados 24](#_Toc444609430)

[Figura 11 -  Status dispositivo clamp “fechado” (o status na request é "1" e o sinal luminoso do dispositivo fica apagado) 25](#_Toc444609430)

[Figura 12 -  Função GET na plataforma de teste POSTMAN com a requisição de “dispositivo fechado”. 25](#_Toc444609430)

[Figura 13 - Status dispositivo clamp “aberto” (o status na request é "0" e o sinal luminoso do dispositivo fica aceso) 25](#_Toc444609430)

[Figura 14 -  Função GET na plataforma de teste POSTMAN com a requisição após mudar o estado do dispositivo para “aberto” 26](#_Toc444609430)

[Figura 15 -  Servidor que simula botão de “abrir/fechar” do dispositivo. 26](#_Toc444609430)

**LISTA DE TABELAS**

[Tabela 1 - Requisitos funcionais 16](#_Toc444609424)

[Tabela 2 - Requisitos não-funcionais 16](#_Toc444609425)

**LISTA DE ABREVIATURAS E SIGLAS**

|  |  |
| --- | --- |
| **API** | *Application Programming Interface* |
| **CSS** | *Cascading Style Sheets* |
| **HTML** | *HyperText Markup Language* |
| **HTTP** | *Hypertext Transfer Protocol* |
| **JSON** | *JavaScript Object Notation* |

SUMÁRIO

[1 INTRODUÇÃO 09](#_Toc444609424)

[2 PROBLEMA 11](#_Toc444609425)

[3 OBJETIVOS 12](#_Toc444609431)

[3.1 Objetivo geral 12](#_Toc444609428)

[3.2 Objetivo específico 12](#_Toc444609428)

[4 JUSTIFICATIVA 13](#_Toc444609428)

[5 REVISÃO DA LITERATURA 14](#_Toc444609428)

[5.1 A influência do uso do “clamp” ou braçadeira no acúmulo de coágulos em drenos pleurais tubulares. 14](#_Toc444609428)

[5.2 Saúde e tecnologia 14](#_Toc444609428)

[6 MATERIAIS E MÉTODO 16](#_Toc444609428)

[6.1 Requisitos Funcionais 16](#_Toc444609428)

[6.2 Requisitos não-funcionais 16](#_Toc444609428)

[6.3 Desenvolvimento da API 17](#_Toc444609428)

[6.4 Tecnologias utilizadas 18](#_Toc444609428)

[7 RESULTADOS 21](#_Toc444609428)

8 [RESULTADOS OBTIDOS 22](#_Toc444609432)

[9 CONCLUSÕES 29](#_Toc444609433)

[REFERÊNCIAS 30](#_Toc444609433)

[APÊNDICE A – CÓDIGO *BACK-END* DA APLICAÇÃO 32](#_Toc444609434)

[APÊNDICE B – CÓDIGO *FRONT-END* DA APLICAÇÃO 41](#_Toc444609433)

[APÊNDICE C – *LINK* DA DEMONSTRAÇÃO DO SISTEMA NA PLATAFORMA DO YOUTUBE 51](#_Toc444609433)

[APÊNDICE D – *LINK* DA DEMONSTRAÇÃO DO MICROCONTROLADOR NA PLATAFORMA DO YOUTUBE 52](#_Toc444609433)

APÊNDICE E – *LINK* DA ACESSO AO REPOSITÓRIO DO PROJETO NO GITHUB, CONTENDO TODOS OS DOCUMENTOS, CÓDIGOS E APRESENTAÇÃO DO PROJETO............................................................................................................53

1. INTRODUÇÃO

Devido ao constante avanço das tecnologias nos hospitais e no que diz respeito à saúde, cada vez mais dispositivos microcontroladores que possibilitem um dia a dia mais seguro neste ambiente são cogitados como uma solução para o futuro.

Hoje, pacientes podem morrer por falta de alertas que um sistema integrado pode resolver. Um grande problema encontrado no meio hospitalar, mais especificamente nas unidades de terapia intensiva - UTI está com o fato de dispositivos clamps serem fechados em dispositivos de drenagem para no momento de sua manutenção ou atendimento dos serviços de saúde e serem esquecidos de tal forma, impossibilitando a volta da sucção e uma efetiva drenagem, o que pode causar desconfortos e danos ao paciente internado e até mesmo levá-lo a óbito. Surge então, a seguinte pergunta: como desenvolver um sistema utilizando um microcontrolador que identifica e notifica se o dispositivo de clamp está aberto ou fechado e há quanto tempo ele está em seu estado atual, gerando uma base de dados e relatórios com seus históricos.

Por meio de tecnologias modernas para desenvolvimento de sistemas *web*, banco de dados e serviços de hospedagem, que visam performance e segurança das informações, o projeto tem como objetivo desenvolver um sistema integrado que capture os estados em tempo real de clamps de drenos de pacientes em um hospital e armazene-os, visando análises e controle dos dispositivos em utilização.

O que se deseja alcançar com o desenvolvimento do trabalho proposto, implica no desenvolvimento de uma aplicação computacional que faça interação com um dispositivo microcontrolador acoplado a um dispositivo de clampeamento.

A criação de um *software* integrado que una uma aplicação *web* e um sistema de microcontrolador capaz de obter o estado do clamp e enviar essa informação, abre portas para uma maior eficiência nos hospitais, controle sobre todos os dispositivos em utilização, podendo identificar seu *id* e seu respectivo local para caso necessário, haja uma intervenção manual.

Para realizar a leitura do estado do clamp, será utilizado um microcontrolador NODEMCU ESP8266, que recebe tensões de energia quando o estado da ferramenta muda (aberto/fechado), este controlador é responsável por enviar requisições indicando seus respectivos estados e *ids*. Para o armazenamento será utilizado MongoDB, um software de banco de dados orientado a documentos livre, de código aberto e multiplataforma. A linguagem que fará a interação entre banco e sistema será o Node.js, um *framework* de JavaScript.

O projeto visa então criar um sistema para monitoramento dos estados de clamps de pacientes em um hospital. Utilizando do microcontrolador para envio de informações para o sistema *web*, onde o *back-end* será feito com Node.js e o *front-end* com React.js. Desta forma, acreditamos que podemos chegar em excelentes resultados com um protótipo funcional.

# 2 PROBLEMA

O projeto de uma aplicação web em conjunto a um microcontrolador capaz de enviar requisições para o sistema com o objetivo de gerar informações em uma interface para o usuário.

O projeto foi determinado após presenciado uma situação-problema de atendimento a drenos e sondas de pacientes por profissionais da saúde em um hospital. Afinal, como um sistema de notificação e status de clampeamento pode evitar manter um sistema de aspiração contínua obstruído por um longo tempo?

# 3 OBJETIVOS

Desenvolvimento de um sistema *Web* para monitorar estados (aberto ou fechado) de dispositivos de clampeamento de drenos e sondas de pacientes em um hospital, através de um sistema *Web* desenvolvido especificamente para receber requisições e atualizar os estados do dispositivo em uma interface para o usuário, utilizando o microcontrolador NodeMCU para o envio dos dados para o sistema *web.*

**3.1 OBJETIVO GERAL**

O projeto tem como um objetivo geral o desenvolvimento de uma aplicação *web* para monitorar estados de dispositivos de clampeamento de sondas e drenos em um hospital utilizando o recurso de um microcontrolador, afins desta forma evitar eventuais danos a pacientes internados.

**3.2 OBJETIVOS ESPECÍFICOS**

De acordo com SOUSA (2018) o levantamento de requisitos é o início de qualquer projeto de *software* independente da metodologia utilizada, pois é a partir dos resultados dessa etapa que será possível definir como as próximas etapas de desenvolvimento serão executadas.

Requisitos de *Software* são as ações que devem ser executadas, possuindo características e condições próprias (CASTRO, 2017).

Para CUNHA (2020) “Os requisitos funcionais são todos os problemas e necessidade que devem ser atendidos e resolvidos pelo *software* por meio de funções e serviços. Os requisitos não funcionais são todos aqueles relacionados a forma como o software tornará realidade o que está sendo planejado.”

O projeto visa o desenvolvimento de um sistema *Web* integrado a um microcontrolador para enviar requisições de estado de clamps em drenos e sondas. Este projeto além de ser utilizado como trabalho de graduação na Faculdade de Tecnologia de Taubaté – FATEC, os alunos pretendem utilizá-lo posteriormente como protótipo de um produto para a área da saúde. Tendo em vista os evidentes possíveis danos causados por um sistema de aspiração contínua não intencionalmente fechado por um longo período.

# 4 JUSTIFICATIVA

   O desenvolvimento de um protótipo de dispositivo microcontrolador de estado de clampeamento de drenos e sondas é motivado pelo amplo desenvolvimento dos meios de tecnologia e saúde, podendo desta forma gerar um possível produto que poderia auxiliar no dia a dia de profissionais da saúde. Desta forma, temos como principal justificativa o humilde e nobre desejo de contribuir com o meio de desenvolvimento tecnológico hospitalar e diminuir desta maneira possíveis danos causados pelo atendimento de saúde e sistemas de sondagens e drenagens contínuas em pacientes internados em um hospital. Um dos alunos deste projeto além de estudante na Faculdade de Tecnologia de Taubaté – FATEC, é também técnico de enfermagem em uma Unidade de Terapia Intensiva - UTI.

         Além dos pontos citados acima, é importante salientar que o elemento principal que motivou o tema deste trabalho foram os cursos ofertados pela Faculdade de Tecnologia de Taubaté, onde foi possível conciliar orientações e dicas de professores do curso de Análise e Desenvolvimento de Sistemas no que diz respeito ao desenvolvimento da aplicação *Web* e do professor de Eletrônica Automotiva Michel Veiga no que diz respeito a toda a parte eletrônica e de microcontrolador.

        Também busca-se otimizar a aplicação *Web* e o protótipo para que possa ser utilizado em futuros projetos que tenham interesse em integrar estas tecnologias.

# 5 revisão da literatura

Segundo Lima AG, Rocha ERF, Seabra JCT, Mussi RK, Santos JGd e Toro IFC (2008) o uso indiscriminado da braçadeira talvez seja o mais sério destes erros de manejo, pois a oclusão do dreno pleural através da braçadeira é um procedimento proscrito e que deve ser desencorajado por todos médicos, para toda equipe. No entanto, por vício de uso ou por falta de informação, muitos serviços continuam a usar rotineiramente este dispositivo, para mobilização e transporte, deambulação ou saída do leito, a despeito das orientações médicas e educação continuada.

**5.1 A INFLUÊNCIA DO USO DO “CLAMP” OU BRAÇADEIRA NO ACÚMULO DE COÁGULOS EM DRENOS PLEURAIS TUBULARES**

O uso da braçadeira pode ter consequências agudas catastróficas, como pneumotórax hipertensivo, parada cardíaca e hemotórax retido. No entanto, além dessas complicações, muito pouco se sabe acerca da influência da interrupção intermitente dos drenos tubulares pleurais sobre o funcionamento em médio prazo.

Figura 1. Dispositivo de drenagem pleural com dispositivo clamp.



Fonte: Sistema para drenagem torácica mediastinal Lac-Medical – 2000 Ml – Com dreno Nr 32. Bisturi Material Hospitalar. Rio de Janeiro, abr. de 2020. Disponível em: <https://www.bisturi.com.br/sistema-para-drenagem-toracica-mediastinal-n34-lac-medical/p>. Acesso em: 01 Dez. de 2022.

**5.2 SAÚDE E TECNOLOGIA**

Barra (2009) introduz que avanços tecnológicos na área da saúde trouxeram diversos benefícios, como os aparelhos modernos e sofisticados, que ajudam com grande velocidade no tratamento de doenças. Uma tecnologia criada pelo homem, a serviço do homem, que pode se reverter em melhores condições de saúde para o paciente.

Peixoto (1994) diz que as instituições de saúde, bem como os profissionais que atuam nesse setor, não ficaram alheias a esse processo. Assim como nas organizações produtivas, onde a utilização da tecnologia permite a manutenção de uma posição de igualdade ou superioridade em relação a concorrentes, as organizações de prestações de serviços têm o mesmo comportamento, visto que, em sociedades capitalistas, os serviços passam, também, a ser considerados bens vendáveis, reproduzindo, assim, a lógica das leis de mercado competitivo.

# 6 MATERIAIS E MÉTODO

Após pesquisas relacionadas ao âmbito de tecnologia hospitalar, onde o trabalho tem seu foco, as tecnologias relacionadas ao uso de microcontroladores e seus potenciais funcionalidades, as tecnologias para desenvolvimento de uma aplicação *web*, foram escolhidas diversas ferramentas para o desenvolvimento.

**6.1 REQUISITOS FUNCIONAIS**

Sommerville (2011) define os Requisitos funcionais como declarações de serviços que o sistema deve fornecer, de como o sistema deve reagir a entradas específicas e de como o sistema deve se comportar em determinadas situações.

Tabela 1: Requisitos funcionais

|  |  |  |
| --- | --- | --- |
| **CÓD.** | **Descrição** | **Prioridade** |
| 1 | Comunicar-se com o sistema *web* através da rede *Wifi* utilizando o microcontrolador NodeMCU; | Alta |
| 2 | Emitir notificação para o sistema *web* notificando desta forma o profissional da saúde; | Alta |
| 3 | Permitir a configuração de novos dispositivos de clamps no sistema *Web*; | Alta |
| 4 | Permitir a exclusão de dispositivos inutilizados; | Alta |
| 5 | Permitir o cadastro de novos usuários profissionais da saúde; | Alta |
| 6 | Comunicar-se com o sistema web através da rede *Wifi* utilizando o microcontrolador NodeMCU; | Alta |
| 7 | Armazenar dados de status anteriores do dispositivo. | Média |

**6.2 REQUISITOS NÃO FUNCIONAIS**

Sommerville (2011) Requisitos não funcionais. São restrições aos serviços ou funções oferecidas pelo sistema. Incluem restrições de timing, restrições no processo de desenvolvimento e restrições impostas pelas normas.

Tabela 2: Requisitos não funcionais

|  |  |  |
| --- | --- | --- |
| **Cód.** | **Descrição** | **Prioridade** |
| 1 | Acesso a um computador próximo ao setor; | Alta |
| 2 | Acesso a rede *Wifi;* | Alta |
| 3 | Sistema Operacional Windows, Linux, Mac; | Alta |
| 4 | Acesso ao *Google Chrome, Mozilla Firefox, Microsoft Edge,* *Internet Explorer*; | Alta |
| 5 | Dispositivo NodeMCU acoplado ao dispositivo de clampeamento. | Alta |

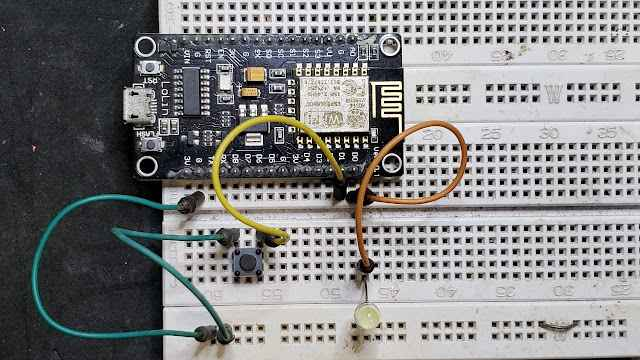
**6.3 DESENVOLVIMENTO DA API**

Os primeiros aspectos a serem considerados no desenvolvimento dos dispositivos de notificação de estados de dispositivos de clampeamento é o sistema que será utilizado para receber as requisições e o microcontrolador a ser utilizado. Como dispositivo será utilizado o microcontrolador NodeMCU ESP8266 devido seu leque de possibilidades, funções e valor acessível.

Programar para NodeMCU possibilita a utilização de bibliotecas de Arduino, o que facilita em pesquisas e até mesmo no desenvolvimento. Há também as próprias bibliotecas, com seus conjuntos de comandos, funções e bibliotecas para lidar com *WiFI* e os protocolos de rede.

Para o acionamento do dispositivo, será implantado um botão que nas mudanças de estado do dispositivo de clampeamento, será enviado uma requisição para o sistema *Web*, gerando assim, uma informação de estado atual. Conforme ilustra a figura 2.

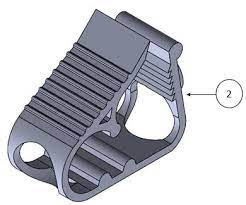
Figura 2. Dispositivo NodeMCU utilizando botão para mudança de estado.

****

Fonte: (MARQUES J. Como utilizar o Botão Flash do NodeMCU ESP8266, 2019)

Outro aspecto importante de se avaliar nas etapas iniciais são os componentes que serão utilizados, além do dispositivo microcontrolador, deverá ser considerado a utilização de um dispositivo de clampeamento utilizado em drenos e sondas para a criação de um potencial protótipo usual conforme ilustra a figura 3:

Figura 3. Ilustração de um clamp oclusor utilizado em drenos e sondas.



Fonte: (kit de barreira para tubo endotraqueal, 2020)

Para a comunicação do dispositivo com a API será desenvolvido uma aplicação *web* cuja criação do *back-end* que receberá as requisições do dispositivo, será utilizado a linguagem de programação JavaScript.

**6.4 TECNOLOGIAS UTILIZADAS**

Flanagan (1996) define o JavaScript como uma linguagem moderna, presente em todos os navegadores, seja em *consoles* de jogos, celulares ou computadores. Como todo navegador tem um interpretador de JavaScript, a linguagem tornou-se a mais onipresente da história.

Flanagan (1996) compreende que o JavaScript pertence à tríade de tecnologias que um desenvolvedor web deve conhecer, sendo elas: HTML, CSS e JavaScript.

JavaScript é uma linguagem desenvolvida para rodar no lado do cliente, isto é, a interpretação e o funcionamento da linguagem dependem de funcionalidades hospedadas no navegador do usuário. Isso é possível porque existe um interpretador JavaScript hospedado no navegador.

Para o desenvolvimento da parte visual das telas da aplicação, onde será possível identificar os dispositivos e seus estados atuais, será utilizado a ferramenta de desenvolvimento React.js.

Fedosejev (2015) introduz que hoje a maneira e o caminho para o desenvolvimento *web* é diferente. O React.js surge como uma solução para o desenvolvimento front-end, principalmente com relação às animações, muito conhecidas pela biblioteca de JavaScript, JQuery. Logo o React.js surge como uma opção para o desenvolvimento de interfaces *web* de maneira rápida, modular e escalável.

Combinando React.js com a arquitetura Flux, temos o cenário ideal não apenas para desenvolvedores experientes, mas também para todos que querem iniciar na carreira de desenvolvimento *web*.

Como ambiente de desenvolvimento utilizando a linguagem de programação JavaScript, a melhor opção neste projeto foi utilizar o NodeJS.

Segundo Ribeiro (2016) vivemos em uma era onde os usuários acessam informações de diversas plataformas, sejam elas computadores, celulares, *tablets* etc. Para atender as necessidades de troca de informações, devemos desenvolver uma *web service*, conhecidos como API 's *(Application Program Interface*).

"API 's são sistemas *back-end* que têm o objetivo de trabalhar apenas com dados, de forma centralizada, permitindo que sejam desenvolvidos, separadamente, aplicações clientes que possuem interfaces para o usuário final”.

Para armazenamento dos dados e informações geradas pelo sistema, foi utilizado o banco de dados MongoDB.

Segundo Kyle (2011), o MongoDB é um banco de dados orientado a documentos, diferente dos bancos de dados tradicionais que seguem o modelo relacional. Este banco de dados tem como característica ser código-fonte aberto licenciado pela *GNU AGPL* (*Affero General Public License*) versão 3.0, possuir alta performance, não possuir esquemas, ser escrito em C++, multiplataforma e ser formado por um conjunto de aplicativos JSON. Diversas linguagens e plataformas já possuem drivers para o MongoDB, entre elas destacam-se: C, C#, C++, Haskell, Java, JavaScript, Perl, PHP, Python, Ruby e Scala. Além disso, o MongoDB possui binários para diversas plataformas como Windows, Mac OS X, Linux e Solaris.

A fim de realizar testes da API durante seu desenvolvimento, a melhor opção escolhida pelos desenvolvedores foi a utilização da ferramenta Postman.

O Postman é um API *Client* que facilita aos desenvolvedores criar, compartilhar, testar e documentar APIs. Isso é feito, permitindo aos usuários criar e salvar soluções HTTP e HTTPs simples e complexas, bem como ler suas respostas. Para a implantação da API foi utilizado a plataforma HEROKU. O heroku é uma plataforma com diversas funcionalidades como: criação de *pipelines*, hospedagem de banco de dados, *deploy* de aplicações. Neste projeto foi usado para o *deploy do back-end.*

# 7. RESULTADOS

Espera-se que após a conclusão do projeto, seja possível alcançar um resultado que possibilite transmitir o conhecimento para as futuras gerações de graduandos ou a quem possua interesse neste âmbito de desenvolvimento, afins de que futuramente seja implantado de forma funcional em sua área de destino.

No apêndice C deste projeto, situado na página 51 encontra-se o *link* do vídeo na plataforma do youtube onde é possível visualizar o resultado obtido através do funcionamento do sistema *web* e suas interfaces.

Situado na página 52 deste projeto, o apêndice D conta com o *link* do vídeo na plataforma do youtube possibilitando desta forma a visualização do funcionamento e teste de conexão do dispositivo NodeMcu ESP8266 com a plataforma *web* desenvolvida.

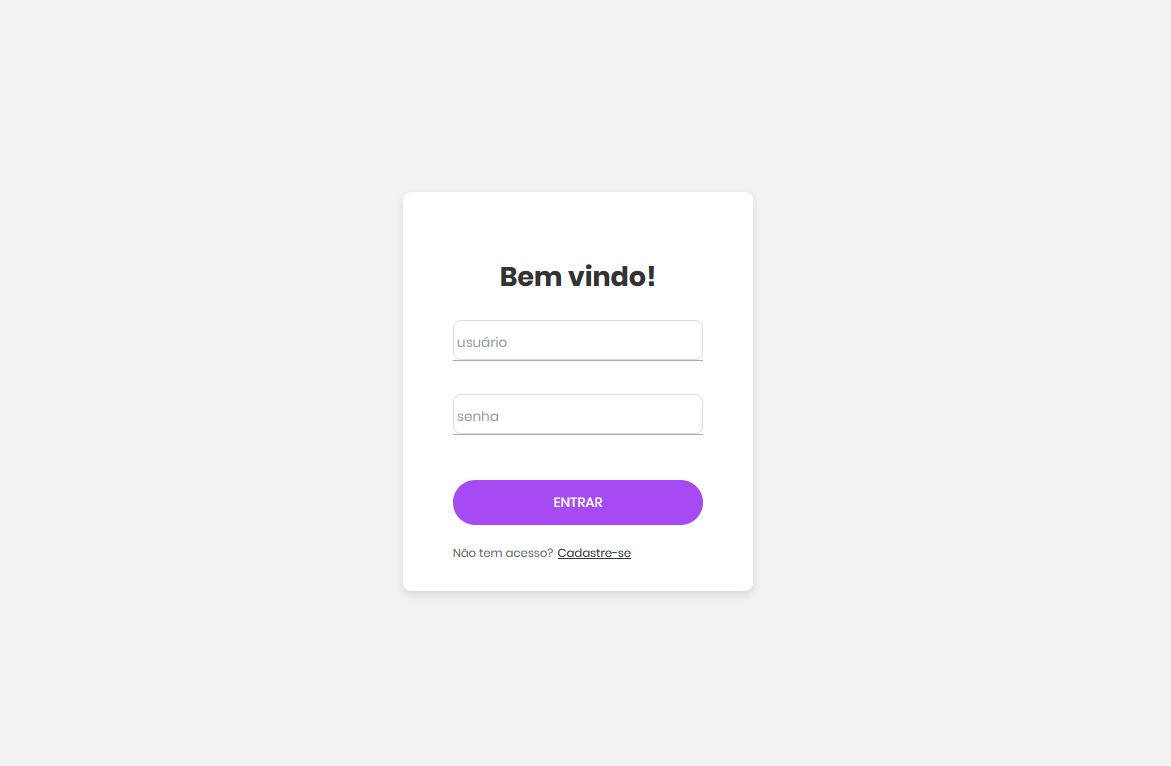
De início, será utilizado como trabalho de graduação na Faculdade de Tecnologia de Taubaté. Posteriormente, será apresentado para profissionais diretamente ligados aos assuntos abordados neste trabalho.

# 8. RESULTADOS OBTIDOS

Nesta seção serão abordadas cada uma das telas da aplicação *web* e suas respectivas funcionalidades.

A figura 4 ilustra a tela de *login*, onde o usuário já cadastrado irá realizar seu acesso e após logado, o sistema irá situá-lo para o painel administrativo da aplicação, conforme ilustra a imagem 5.

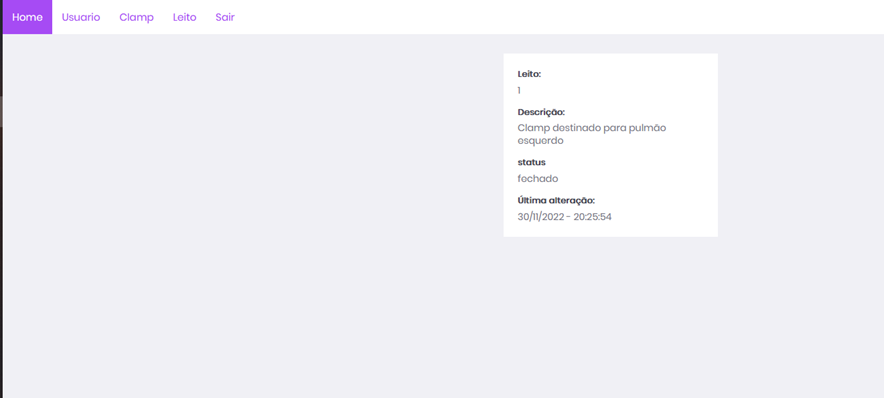
Figura 4. Tela de *Login*

****

Fonte: Desenvolvimento próprio (2022).

A figura 5 ilustra o painel administrativo da aplicação, onde o usuário após logado é possível já visualizar os estados atuais dos dispositivos atrelados ao sistema, suas descrições contendo data e horário da última alteração.

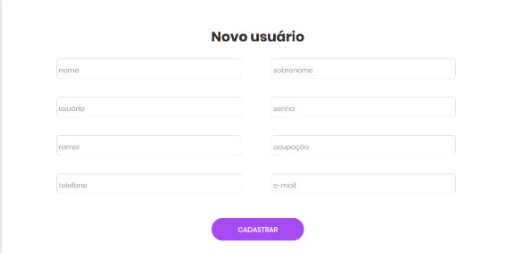
Figura 5. Tela painel administrativo



Fonte: Desenvolvimento próprio (2022).

A figura 6 demonstra a tela de cadastro de novo usuário, onde o usuário com sua respectiva permissão possui a possibilidade de adicionar um novo usuário para o acesso.

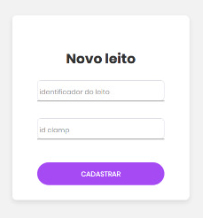
Figura 6. Tela cadastro novo usuário



Fonte: Desenvolvimento próprio (2022).

A figura 7 demonstra a tela de cadastro de novo leito, onde o usuário através do *ID* do dispositivo pode atrelar um dispositivo clamp a um respectivo leito.

Figura 7. Tela cadastro novo leito



Fonte: Desenvolvimento próprio (2022).

A figura 8 demonstra a tela de cadastro de novo dispositivo clamp, onde através das informações de *Mac Address* e de sua respectiva descrição é possível cadastrar um novo dispositivo à plataforma *web*.

Figura 8. Tela cadastro novo dispositivo clamp



Fonte: Desenvolvimento próprio (2022).

A figura 9 ilustra um dos principais trechos de código da aplicação, que é o responsável por atualizar o estado do dispositivo clamp. No apêndice A deste trabalho, situado na página 32, é possível visualizar a codificação completa da aplicação.

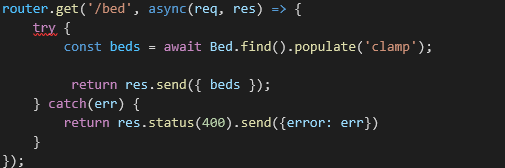
Figura 9. Trecho do código responsável por atualizar o estado do dispositivo clamp.

.

Fonte: Desenvolvimento próprio (2022).

A figura 10 ilustra o trecho de código responsável por listar os leitos e os dispositivos atrelados a ele, contendo sua descrição, estados atuais e última alteração.

Figura 10. Trecho do código do painel administrativo responsável por listar os leitos e seus dispositivos atrelados.



Fonte: Desenvolvimento próprio (2022).

A figura 11 ilustra o funcionamento do dispositivo microcontrolador onde demonstra o dispositivo com seu respectivo sinal luminoso apagado, onde identifica através do servidor simulador o estado de dispositivo fechado.

Figura 11. Status dispositivo clamp “fechado” (o *status* na *request* é "1" e o sinal luminoso do dispositivo fica apagado).



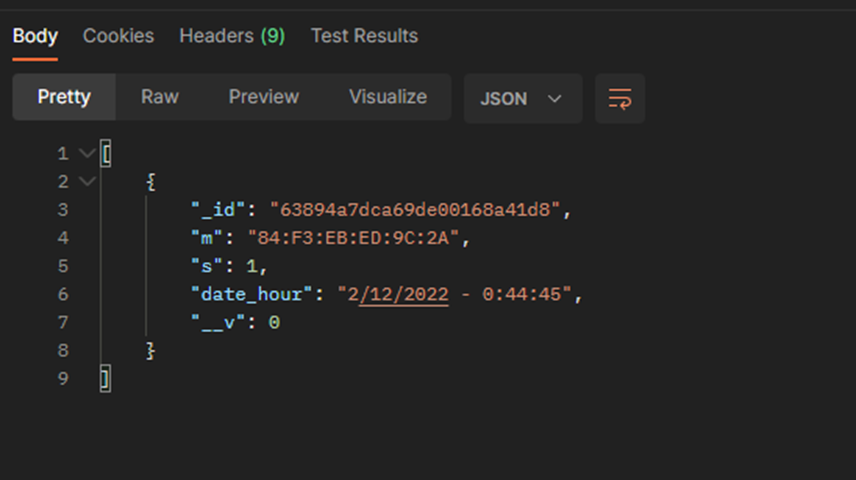
Fonte: Desenvolvimento próprio (2022).

A figura 12 demonstra o teste utilizando a função *GET* na plataforma Postman, onde é enviado uma requisição de “dispositivo fechado” para o sistema.

Figura 12. Função *GET* na plataforma de teste Postman com a requisição de “dispositivo fechado”.

“m” = *mac address* do dispositivo, tem como função ser um identificador único.

“s” = status aberto ou fechado (0 ou 1).



Fonte: Desenvolvimento próprio (2022).

A figura 13 demonstra o estado de dispositivo “aberto”, onde neste caso, o sinal luminoso do dispositivo fica aceso indicando para o profissional que o dispositivo se encontra “aberto”.

Figura 13. Status dispositivo clamp “aberto” (o *status* na *request* é "0" e o sinal luminoso do dispositivo fica aceso).



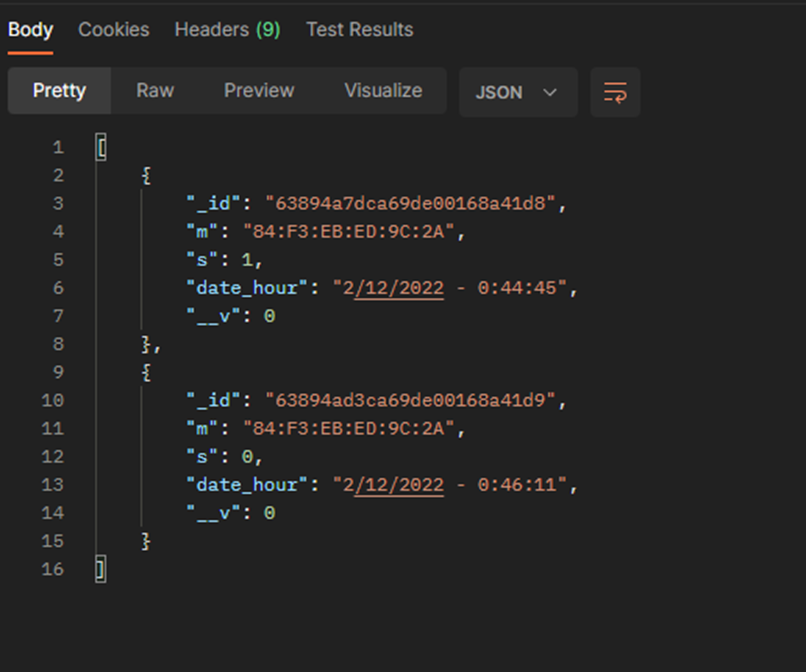
Fonte: Desenvolvimento próprio (2022).

A figura 14 demonstra o teste utilizando a função *GET* na plataforma Postman, onde é enviado uma requisição de “dispositivo aberto” para o sistema.

Figura 14. Função *GET* na plataforma de teste POSTMAN com a requisição após mudar o estado do dispositivo para “aberto”.

“m” = *mac address* do dispositivo, tem como função ser um identificador único.

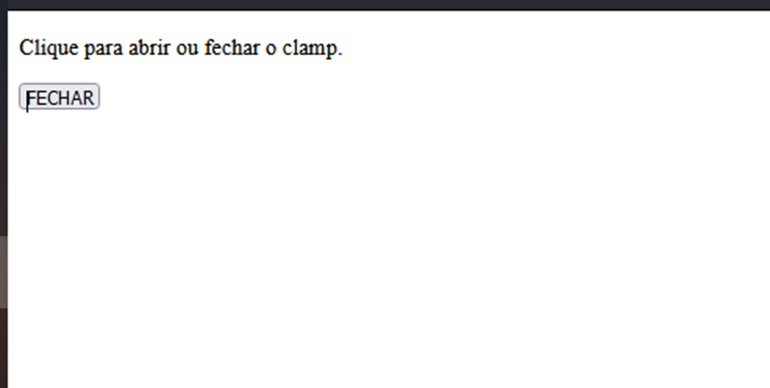
“s” = *status* aberto ou fechado (0 ou 1).



Fonte: Desenvolvimento próprio (2022).

A figura 15 demonstra o servidor que foi utilizado para simular um botão de “abrir e fechar” do dispositivo.

Figura 15. Servidor que simula botão de “abrir/fechar” dispositivo.

Fonte: Desenvolvimento próprio (2022).

# 9 ConclusÔES

Com o avanço dos meios da tecnologia e os meios da saúde, estão sendo gerados diversos produtos e serviços. Muitos hospitais já estão se adequando ao uso de novas tecnologias que auxiliem na diminuição de falhas e danos causados. Neste mesmo propósito foi proposto pelos alunos a criação de um sistema *Web* integrado a um microcontrolador. O sistema utilizará sensores em abraçadeiras mais conhecidas no âmbito hospitalar como “clamp”, que são utilizados em drenos e sondas. Como principal função, o microcontrolador irá enviar para o sistema *web* requisições informando os estados dos dispositivos, sendo dois: “aberto ou fechado”.

Durante o desenvolvimento deste trabalho, foram encontradas dificuldades que foi necessário desta forma, realizar algumas alterações e mudanças de trajeto a ser percorrido. Foi realizado alteração no sistema de armazenamento dos dados e informações, de um banco de dados relacional para banco de dados orientado a documentos livre, de código aberto e multiplataforma para que melhorasse o desenvolvimento dos alunos e para que fosse mais bem adequado ao sistema. Foi decidido pelos alunos neste momento não realizar a modelagem própria da abraçadeira e a acoplagem do dispositivo microcontrolador devido ao grau de dificuldade e a ausência do recurso neste momento. Por isso, foi utilizado um servidor simulando o acionamento do dispositivo mostrando seu funcionamento.

Foram alcançados satisfatoriamente os objetivos primórdios propostos pelos alunos no desenvolvimento deste trabalho. Pretendemos dar continuidade e compartilhar em diante com futuros graduandos o desenvolvimento deste produto, tendo em vista a seriedade no desenvolvimento para a área da saúde e a precisão que ele deve ter, para que um dia este, além de um simples protótipo, venha auxiliar a reduzir danos e até mesmo salvar vidas de pessoas que necessitem da utilização destes dispositivos durante uma internação hospitalar.

###### REFERÊNCIAS

BARRA, Daniela. Evolução histórica e impacto da tecnologia na área da saúde e da enfermagem. Revista Eletrônica de Enfermagem, Goiânia, v. 08, p. 422-430, 2006.

KYLE Banker, MongoDB in Action. Manning, 2011. Disponível em: <<https://www.devmedia.com.br/introducao-ao-mongodb/30792/>.> Acesso em: 02 dez. 2020.

CASTRO, Eduardo. O que são requisitos? E requisitos de software? Disponível em: http://rederequisitos.com.br/o-que-sao-requisitos-e-requisitos-de-software/. Acesso em 01/12/2022.

CUNHA, Fernando. Requisitos funcionais e não funcionais: o que são?Disponível em: https://mestresdaweb.com.br/fabrica-de-software/requisitos-funcionais-e-nao-funcionais-o-que-sao/. Acesso em 01/12/2022.

FLANAGAN, David. JavaScript: O Guia Definitivo. 6° edição. São Paulo: Bookman, 2013.

FEDOSEJEV, Artemij. React.js Essentials. 1° edição. Birmingham: Packt, 2015.

PEREIRA, Caio. Construindo APIs REST com Node.js. 1° edição. São Paulo, 2016.

FRIZZARIN, Fernando. NodeMCU: 15 passos para se tornar um mestre em IoT. 1° edição. [S.I], 2019.

LIMA, Alexandre, ROCHA Elen, SEABRA José, MUSSI Ricardo, SANTOS José, TORO Ivan (2008). A influência do uso do “clamp” ou braçadeira no acúmulo de coágulos em drenos pleurais tubulares. Disponível em: <https://www.scielo.br/j/rcbc/a/CZZd8p9VMqWFd8H5hrgkp7f/?lang=pt>. <Acesso em: 02 de dez. de 2020>.

Lima AG, Toro IFC, Tincani AJ, Barreto G. A drenagem pleural pré-hospitalar: apresentação de mecanismo de válvula unidireccional. Rev. Col. Bras Cir. 2006; 33(2):101-6

PEIXOTO, Marisa. O uso da tecnologia no processo diagnóstico-terapêutico: ótica do enfermeiro e do usuário. Revista USP, São Paulo, v. 28, p. 257-69, dez. 1994.

SOMMERVILLE, Ian. Engenharia de Software / Ian Sommerville; tradução Ivan Bosnic e Kalinka G. de O. Gonçalves; revisão técnica Kechi Hirama. — 9. ed. — São Paulo: Pearson Prentice Hall, 2011. Título original: Software engineering. ISBN 978 - 85 -7936 - 108 -1.

SOUSA, Julio. Levantamento de Requisitos – O ponto de partida do projeto de software. Disponível em: https://blog.cedrotech.com/levantamento-de-requisitos-o-ponto-de-partida-do-projeto-de-software/. Acesso em 01/12/2022.

###### APÊNDICE A – CÓDIGOS *BACK-END* DA APLICAÇÃO

*Server.js* - Trecho de código responsável por configurar o projeto:

         const clamps = await Clamp.find();

         return res.send({ clamps });

    } catch(err) {

     return res.status(400).send({ error: err});

    }

 });

router.post('/clamp', async(req, res) => {

    try {

        const m = await Clamp.findOne({m: req.body.m});

const express = require('express');

const server = express();

const cors = require('cors');

const body\_parser = require("body-parser");

server.use(cors());

server.use(body\_parser.json());

server.use(body\_parser.urlencoded({extended: false}));

require('./controllers/clampController')(server);

require('./controllers/bedController')(server);

require('./controllers/userController')(server);

require('./controllers/adminController')(server);

server.listen(process.env.PORT || 4000);

module.exports = server;

Fonte: Desenvolvimento próprio

*Procfile -* Trecho de código responsável por configurar comandos para rodar no Heroku:

const mongoose = require('../db/database');

const UsuarioSchema = new mongoose.Schema({

    name: {

        type: String,

        require: true,

    },

    lastName: {

        type: String,

        require: true,

    },

    user: {

        type: String,

        require: true,

    },

    password: {

        type: String,

        require: true,

    },

    branch: {

        type: String,

        require: true,

    },

    occupation: {

        type: String,

        require: true,

    },

    phone: {

        type: String,

        require: true,

    },

    email: {

        type: String,

        require: true,

    },

    createdAt: {

        type: Date,

        default: Date.now

    },

});

var usuario = mongoose.model('usuario', UsuarioSchema);

module.exports = usuario;

Fonte: Desenvolvimento próprio.

*UserController* - Trecho de código responsável pelas lógicas e *CRUD* de usuário:

const express = require('express');

const User = require('../model/user');

const router = express.Router();

router.get('/user', async(req, res) => {

   try {

        const users = await User.find();

        return res.send({ users });

   } catch(err) {

    return res.status(400).send({ error: 'Erro ao listar usuários'});

   }

});

router.get('/user/:userId', async(req, res) => {

    try {

        const user = await User.findById(req.params.userId);

        return res.send({ user });

    } catch(err) {

        return res.status(400).send({ error: 'Erro ao recuperar usuário'});

    }

});

router.post('/user', async(req, res) => {

    try {

        const userName = await User.findOne({user: req.body.user});

        const email = await User.findOne({email: req.body.email});

        if (userName || email) {

            return res.status(400).send("usuário já cadastrado");

        } else {

            const user = await User.create(req.body);

            return res.send(user);

        }

    } catch (err) {

        return res.status(400).send({ error: 'Erro ao criar usuário'});

    }

});

router.put('/user/:userId', async(req, res) => {

    try {

        const { name, lastName, user, password, branch, occupation, phone, email} = req.body

        const updatedUser = await User.findByIdAndUpdate(req.params.userId, {

            name,

            lastName,

            user,

            password,

            branch,

            occupation,

            phone,

            email

        }, {new: true});

        return res.send(updatedUser);

    } catch (err) {

        return res.status(400).send({ error: 'Erro ao autalizar usuário'});

    }

});

router.delete('/user/:userId', async(req, res) => {

    try {

        await User.findByIdAndRemove(req.params.userId);

        return res.send();

    } catch(err) {

        return res.status(400).send({ error: 'Erro ao apagar usuário'});

    }

});

router.post('/login', async(req, res) => {

    try {

        const user = req.body.user;

        const password = req.body.password;

        const userLogged = await User.findOne({user: user, password: password});

        if (!userLogged) {

            return res.status(401).send("usuário ou senha incorreto");

        } else {

            return res.send(userLogged);

        }

    } catch(err) {

        return res.status(400).send({ error: err});

    }

});

module.exports = server => server.use('/api', router);

Fonte: Desenvolvimento próprio.

Clamp - Trecho de código referente ao *model* de um clamp:

const mongoose = require('../db/database');

const ClampSchema = new mongoose.Schema({

    m: {

        type: String,

        required: true,

    },

    s: {

        type: Number,

        required: false,

        default: 0

    },

    statusDescription: {

        type: String,

        required: false,

    },

    createdAt: {

        type: Date,

        default: Date.now

    },

    updateAt: {

        type: String,

        required: false

    },

});

var clamp = mongoose.model('clamp', ClampSchema);

module.exports = clamp;

Fonte: Desenvolvimento próprio.

*ClampController* - Trecho de código responsável pelas lógicas e *CRUD* de clamps:

const express = require('express');

const Clamp = require('../model/clamp');

const router = express.Router();

 router.get('/clamp', async(req, res) => {

    try {

        if (m) {

            return res.status(400).send("clamp já cadastrado");

        } else {

            const status = req.body.s;

            var statusDescription;

            if (status === 0) {

                statusDescription = "fechado";

            } else {

                statusDescription = "aberto";

            }

            var clamp = {

                m: req.body.m,

                s: req.body.s,

                statusDescription: statusDescription

            }

            const save = await Clamp.create(clamp);

            return res.send(save);

        }

    } catch (err) {

        return res.status(400).send({ error: err});

    }

});

router.post('/clamp/update', async(req, res) => {

    try {

        var currentdate = new Date();

        var updateAt = currentdate.getDate() + "/"

            + (currentdate.getMonth()+1)  + "/"

            + currentdate.getFullYear() + " - "

            + currentdate.getHours() + ":"

            + currentdate.getMinutes() + ":"

            + currentdate.getSeconds();

            const status = req.body.s;

            var statusDescription;

            if (status === 0) {

                statusDescription = "fechado";

            } else {

                statusDescription = "aberto";

            }

        const updatedClamp = await Clamp.findOneAndUpdate(

            {m: req.body.m},

            {

                s: req.body.s,

                statusDescription: statusDescription,

                updateAt: updateAt

            },

            {new: true}

        );

        return res.send(updatedClamp);

    } catch (err) {

        return res.status(400).send({ error: err});

    }

});

module.exports = server => server.use('/api', router);

Fonte: Desenvolvimento próprio.

*Bed* - Trecho de código responsável pelo *model* do leito:

const mongoose = require('../db/database');

const BedSchema = new mongoose.Schema({

    bedIdentifier: {

        type: Number,

        required: true,

    },

    createdAt: {

        type: Date,

        default: Date.now

    },

    clamp: {

        type: mongoose.ObjectId,

        ref: 'clamp'

    },

});

var leito = mongoose.model('bed', BedSchema);

module.exports = leito;

Fonte: Desenvolvimento próprio.

*BedController* - Trecho de código responsável pelas lógicas e *CRUD* de leitos:

const express = require('express');

const Bed = require('../model/bed');

const router = express.Router();

router.get('/bed', async(req, res) => {

    try {

        const beds = await Bed.find().populate('clamp');

         return res.send({ beds });

    } catch(err) {

        return res.status(400).send({error: err})

    }

});

router.get('/bed/:bedId', async(req, res) => {

    try {

        const bed = await Bed.findById(req.params.bedId);

        return res.send({ bed });

    } catch(err) {

        return res.status(400).send({ error: 'Erro ao recuperar usuário'});

    }

});

router.post('/bed', async(req, res) => {

    try {

        const bedId = await Bed.findOne({bedIdentifier: req.body.bedIdentifier});

        if (bedId) {

            return res.status(400).send("leito já cadastrado");

        } else {

            const bed = await Bed.create(req.body);

            return res.send(bed);

        }

    } catch (err) {

        return res.status(400).send({ error: err});

    }

});

router.put('/bed/:bedIdentifier', async(req, res) => {

    try {

        const {clamp} = req.body

        const updateBed = await Bed.findOneAndUpdate(

            {bedIdentifier: req.params.bedIdentifier},

            {clamp: clamp},

            {new: true}

        );

        return res.send(updateBed);

    } catch (err) {

        return res.status(400).send({ error: err});

    }

});

router.delete('/bed/:bedId', async(req, res) => {

    try {

        await Bed.findByIdAndRemove(req.params.bedId);

        return res.send();

    } catch(err) {

        return res.status(400).send({ error: 'Erro ao apagar leito'});

    }

});

module.exports = server => server.use('/api', router);

Fonte: Desenvolvimento próprio.

*AdminController* - Trecho de código responsável por retornar os leitos e clamps relacionados para o *admin*:

clamp');

const router = express.Router();

router.get('/admin', async(req, res) => {

    try {

        const bed = await Bed.find().populate('clamp');

        return res.json(bed);

    } catch(err) {

     return res.status(400).send({ error: err});

    }

 });

 module.exports = server => server.use('/api', router);

Fonte: Desenvolvimento próprio.

###### APÊNDICE B – CÓDIGOs *front-end* da aplicação

*Routes* - Trecho de código responsável por configurar as rotas da aplicação:

import React from 'react';

import {BrowserRouter, Route, Switch} from 'react-router-dom';

import Login from './pages/login';

import Usuario from './pages/usuario';

import Admin from './pages/admin';

import Clamp from './pages/clamp';

import Leito from './pages/leito'

export default function Routes(){

    return(

        <BrowserRouter>

            <Switch>

                <Route path="/" exact component={Login}/>

                <Route path="/usuario" exact component={Usuario}/>

                <Route path="/admin" exact component={Admin}/>

                <Route path="/clamp" exact component={Clamp}/>

                <Route path="/leito" exact component={Leito}/>

            </Switch>

        </BrowserRouter>

    );

}

Fonte: Desenvolvimento próprio.

*Index.js* - Trecho de código responsável por configurar o *index* inicial:

import React from 'react';

import ReactDOM from 'react-dom';

import App from './App';

ReactDOM.render(

  <React.StrictMode>

    <App />

  </React.StrictMode>,

  document.getElementById('root')

);

Fonte: Desenvolvimento próprio.

Trecho de código responsável por configurar a utilização das rotas:

import React from 'react';

import './global.css';

import Routes from './routes';

function App() {

  return (

    <div>

      <Routes/>

    </div>

  );

}

export default App;

Fonte: Desenvolvimento próprio.

Trecho de código responsável pelo cadastro de usuários:

import React  from 'react';

import {useHistory} from 'react-router-dom';

import './styles.css';

import { useState } from 'react';

import api from '../../services/api';

export default function CadastroUsuarios(){

    const history = useHistory();

    const[name, setName] = useState('');

    const[lastName, setLastName] = useState('');

    const[user, setUser] = useState('');

    const[password, setPassword] = useState('');

    const[branch, setBranch] = useState('');

    const[occupation, setOccupation] = useState('');

    const[phone, setPhone] = useState('');

    const[email, setEmail] = useState('');

    async function handleCadUsuario(e){

        e.preventDefault();

        try{

            await api.post('api/user', {name, lastName, user, password, branch, occupation, phone, email});

            history.push('/admin');

        } catch(err){

            alert(err);

        }

    return(

        <div class="limiter-usuario">

        <div class="container-usuario100">

            <div class="wrap-usuario100">

                <form class="usuario100-form validate-form" onSubmit={handleCadUsuario}>

                    <span class="usuario100-form-title p-b-26">

                        Novo usuário

                    </span>

                    <div class="input-group-usuario">

                        <div class="wrap-input-usuario100 validate-input" data-validate = "user name">

                            <input class="input100" type="text" name="name"

                            value={name}

                            onChange={e => setName(e.target.value)}

                            />

                            <span class="focus-input100" data-placeholder="nome"></span>

                        </div>

                        <div class="wrap-input-usuario100 validate-input" data-validate="User last name">

                            <span class="btn-show-pass">

                                <i class="zmdi zmdi-eye"></i>

                            </span>

                            <input class="input100" type="text" name="last\_name"

                            value={lastName}

                            onChange={e => setLastName(e.target.value)}

                            />

                            <span class="focus-input100" data-placeholder="sobrenome"></span>

                        </div>

                    </div>

                    <div class="input-group-usuario">

                        <div class="wrap-input-usuario100 validate-input" data-validate="User nickname">

                            <span class="btn-show-pass">

                                    <i class="zmdi zmdi-eye"></i>

                            </span>

                            <input class="input100" type="text" name="user\_nickname"

                            value={user}

                            onChange={e => setUser(e.target.value)}

                                />

                            <span class="focus-input100" data-placeholder="usuário"></span>

                        </div>

                        <div class="wrap-input-usuario100 validate-input" data-validate="password">

                            <span class="btn-show-pass">

                                <i class="zmdi zmdi-eye"></i>

                            </span>

                            <input class="input100" type="password" name="password"

                            value={password}

                            onChange={e => setPassword(e.target.value)}

                            />

                            <span class="focus-input100" data-placeholder="senha"></span>

                        </div>

                    </div>

                    <div class="input-group-usuario">

                        <div class="wrap-input-usuario100 validate-input" data-validate="branch">

                            <span class="btn-show-pass">

                                <i class="zmdi zmdi-eye"></i>

                            </span>

                            <input class="input100" type="text" name="branch"

                            value={branch}

                            onChange={e => setBranch(e.target.value)}

                            />

                            <span class="focus-input100" data-placeholder="ramal"></span>

                        </div>

                        <div class="wrap-input-usuario100 validate-input" data-validate="occupation">

                            <span class="btn-show-pass">

                                <i class="zmdi zmdi-eye"></i>

                            </span>

                            <input class="input100" type="text" name="occupation"

                            value={occupation}

                            onChange={e => setOccupation(e.target.value)}

                            />

                            <span class="focus-input100" data-placeholder="ocupação"></span>

                        </div>

                    </div>

                    <div class="input-group-usuario">

                        <div class="wrap-input-usuario100 validate-input" data-validate="phone">

                            <span class="btn-show-pass">

                                <i class="zmdi zmdi-eye"></i>

                            </span>

                            <input class="input100" type="text" name="phone"

                            value={phone}

                            onChange={e => setPhone(e.target.value)}

                            />

                            <span class="focus-input100" data-placeholder="telefone"></span>

                        </div>

                        <div class="wrap-input-usuario100 validate-input" data-validate="email">

                            <span class="btn-show-pass">

                                <i class="zmdi zmdi-eye"></i>

                            </span>

                            <input class="input100" type="email" name="email"

                            value={email}

                            onChange={e => setEmail(e.target.value)}

                            />

                            <span class="focus-input100" data-placeholder="e-mail"></span>

                        </div>

                    </div>

                    <div class="container-usuario100-form-btn">

                        <div class="wrap-usuario100-form-btn">

                            <div class="usuario100-form-bgbtn"></div>

                            <button class="usuario100-form-btn" type="submit">

                                cadastrar

                            </button>

                        </div>

                    </div>

                </form>

            </div>

        </div>

    </div>

    );

}

Fonte: Desenvolvimento próprio.

Trecho de código responsável pelo *login:*

import React, { useState }  from 'react';

import {Link, useHistory} from 'react-router-dom';

import api from '../../services/api';

import './styles.css';

export default function Login(){

    const [user, setUser] = useState('');

    const [password, setPassword] = useState('');

    const history = useHistory();

    async function handleLogin(e){

        e.preventDefault();

        try{

            const response = await api.post('/api/login', {user, password});

            localStorage.setItem('usuarioId', response.data.userId);

            localStorage.setItem('nome', response.data.name);

            history.push('/admin')

        } catch(err){

            alert(err);

        }

    }

    return(

        <div class="limiter-login">

        <div class="container-login100">

            <div class="wrap-login100">

                <form class="login100-form validate-form" onSubmit ={handleLogin}>

                    <span class="login100-form-title p-b-26">

                        Bem vindo!

                    </span>

                    <div class="wrap-input100 validate-input" data-validate = "user">

                        <input class="input100" type="text" name="user"

                        value={user}

                        onChange={e=> setUser(e.target.value)}

                        />

                        <span class="focus-input100" data-placeholder="usuário"></span>

                    </div>

                    <div class="wrap-input100 validate-input" data-validate="password">

                        <input class="input100" type="password" name="password"

                        value={password}

                        onChange={e=> setPassword(e.target.value)}

                        />

                        <span class="focus-input100" data-placeholder="senha"></span>

                    </div>

                    <div class="container-login100-form-btn">

                        <div class="wrap-login100-form-btn">

                            <div class="login100-form-bgbtn"></div>

                            <button class="login100-form-btn" type="submit"> entrar </button>

                        </div>

                    </div>

                    <div class="container-singup">

                        <span class="txt1">

                            não tem acesso?

                        </span>

                        <Link class="txt2" to = "/usuario">

                            cadastre-se

                        </Link>

                    </div>

                </form>

            </div>

        </div>

    </div>

    );

}

Fonte: Desenvolvimento próprio.

Trecho de código responsável pelo cadastro de leitos:

import React from 'react';

import {useHistory} from 'react-router-dom';

import './styles.css';

import { useState } from 'react';

import api from '../../services/api';

export default function CadastroLeito(){

    const history = useHistory();

    const [bedIdentifier, setBed] = useState('');

    const [clamp, setClamp] = useState('');

    async function handleCadLeito(e){

        e.preventDefault();

        try{

            if (clamp === "") {

                await api.post('api/bed', {bedIdentifier});

            } else {

                await api.post('api/bed', {bedIdentifier, clamp});

            }

            history.push('/admin');

        } catch(err){

            alert(err);

        }

    }

    return(

        <div class="limiter-login">

        <div class="container-login100">

            <div class="wrap-login100">

                <form class="login100-form validate-form" onSubmit={handleCadLeito}>

                    <span class="login100-form-title p-b-26">

                        Novo leito

                    </span>

                    <div class="wrap-input100 validate-input" data-validate = "bed id">

                        <input class="input100" type="text" name="bed id"

                        value={bedIdentifier}

                        onChange={e=> setBed(parseInt(e.target.value))}

                        />

                        <span class="focus-input100" data-placeholder="identificador do"></span>

                    </div>

                    <div class="wrap-input100 validate-input" data-validate = "clamp id">

                        <input class="input100" type="text" name="clamp id"

                        value={clamp}

                        onChange={e=> setClamp(e.target.value)}

                        />

                        <span class="focus-input100" data-placeholder="leito"></span>

                    </div>

                    <div class="container-login100-form-btn">

                        <div class="wrap-login100-form-btn">

                            <div class="login100-form-bgbtn"></div>

                            <button class="login100-form-btn" type='submit'>

                                cadastrar

                            </button>

                        </div>

                    </div>

                </form>

            </div>

        </div>

    </div>

    );

}

Fonte: Desenvolvimento próprio.

Trecho de código responsável pelo cadastro de clamp:

import React from 'react';

import {useHistory} from 'react-router-dom';

import './styles.css';

import { useState } from 'react';

import api from '../../services/api';

export default function CadastroClamp(){

    const history = useHistory();

    const [macAddress, setMacAddress] = useState('');

    const [description, setDescription] = useState('');

    async function handleCadClamp(e){

        e.preventDefault();

        try{

            await api.post('api/clamp', {macAddress, description});

            history.push('/admin');

            alert(macAddress);

        } catch(err){

            alert(err);

        }

    }

    return(

        <div class="limiter-login">

        <div class="container-login100">

            <div class="wrap-login100">

                <form class="login100-form validate-form" onSubmit={handleCadClamp}>

                    <span class="login100-form-title p-b-26">

                        Novo clamp

                    </span>

                    <div class="wrap-input100 validate-input" data-validate = "mac address">

                        <input class="input100" type="text" name="mac address"

                        value={macAddress}

                        onChange={e => setMacAddress(e.target.value)}/>

                        <span class="focus-input100" data-placeholder="mac address"></span>

                    </div>

                    <div class="wrap-input100 validate-input" data-validate="description">

                        <span class="btn-show-pass">

                            <i class="zmdi zmdi-eye"></i>

                        </span>

                        <input class="input100" type="text" name="description"

                        value={description}

                        onChange={e => setDescription(e.target.value)}

                        />

                        <span class="focus-input100" data-placeholder="descrição"></span>

                    </div>

                    <div class="container-login100-form-btn">

                        <div class="wrap-login100-form-btn">

                            <div class="login100-form-bgbtn"></div>

                            <button class="login100-form-btn" type='submit'>

                                cadastrar

                            </button>

                        </div>

                    </div>

                </form>

            </div>

        </div>

    </div>

    );

}

Fonte: Desenvolvimento próprio.

Trecho de código responsável pelo painel *Admin:*

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

import {useHistory} from 'react-router-dom';

import './styles.css';

import api from '../../services/api';

    export default function Admin(){

        const history = useHistory();

        const [beds, setBeds] = useState([]);

        useEffect(() => {

            api.get('api/admin').then(response =>{

                setBeds(response.data);

            })

        });

        function handleLogout(){

            localStorage.clear();

            history.push('/');

        }

        return(

            <div className="container">

                <div class="topnav">

                    <a class="active" href="#home">Home</a>

                    <a href="../usuario">Usuario</a>

                    <a href="../clamp">Clamp</a>

                    <a href="../leito">Leito</a>

                    <a onClick={handleLogout}>Sair</a>

                </div>

                <div className="profile-container">

                    <ul>

                        {beds.map(bed => (

                            <li key={bed.bedId}>

                                <strong>Leito:</strong>

                                <p>{bed.bedIdentifier}</p>

                                <strong>status:</strong>

                                <p>{bed.clamp.statusDescription}</p>

                                <strong>Última alteração:</strong>

                                <p>{bed.clamp.updateAt}</p>

                            </li>

                        ))}

                    </ul>

                </div>

            </div>

        );

    }

Fonte: Desenvolvimento próprio.

###### APÊNDICE C – *link* da demonstração do sistema na plataforma do youtube:

<https://www.youtube.com/watch?app=desktop&v=GzgXFCPAMUo&feature=youtu.be>

###### APÊNDICE D – *LINK* DA DEMONSTRAÇÃO DO FUNCIONAMENTO DO MICROCONTROLADOR NA PLATAFORMA DO YOUTUBE:

<https://www.youtube.com/watch?v=uO-IqqFy1Cg>

###### APÊNDICE E – *Link* de acesso ao repositório do projeto no github, contendo todos os documentos, códigos e apresentação do projeto: <https://github.com/aislanlima13/TG>