Desenvolvimento da Aplicação Flask

Lucas Puhl Gasperin Renan Pamplona Renan Czervinski Tiago Follador

Abstrato:

Neste documento, apresentamos uma descrição detalhada do desenvolvimento do Gasperin Web Services, uma aplicação inovadora que utiliza o framework Flask para proporcionar aos postos de saúde públicos um meio eficiente de gerenciar o fluxo de pessoas dentro de suas instalações, ao mesmo tempo que monitora as condições climáticas internas.

O desenvolvimento de um projeto direcionado aos postos de saúde representa um desafio significativo, dada a importância crítica desses serviços para milhões de brasileiros. Qualquer melhoria na infraestrutura pode ter um impacto direto na qualidade do atendimento e na eficiência operacional. No entanto, a infraestrutura atual frequentemente se mostra obsoleta e incapaz de acompanhar as demandas crescentes. A falta de investimento adequado por parte do governo também limita severamente as oportunidades de inovação neste setor vital.

O objetivo principal deste documento é explorar como enfrentamos um dos desafios mais ocorrentes enfrentados pelos postos de saúde através da aplicação de recursos da ciência da computação. Além de discutir a implementação prática do "Gasperin Web Services", também abordamos as potenciais melhorias e expansões que podem ser incorporadas no futuro. Isso inclui não apenas aprimorar a capacidade de gerenciamento de fluxo de pacientes, mas também explorar novas funcionalidades que poderiam melhorar significativamente a experiência tanto para pacientes quanto para profissionais de saúde.

1 Introdução

Nos bastidores da saúde pública, os postos de saúde desempenham um papel fundamental na oferta de cuidados acessíveis e eficazes à população brasileira. No entanto, a gestão eficiente desses estabelecimentos enfrenta desafios constantes, desde o controle preciso do fluxo de pacientes até o gerenciamento das condições ambientais internas. Em resposta a essas necessidades críticas, surge a $Gasperin\ Web\ Services$, uma inovadora aplicação desenvolvida sobre o framework Flask, projetada para revolucionar como os postos de saúde monitoram e otimizam suas operações diárias.

Nós criamos uma aplicação que não apenas monitora em tempo real a ocupação dos espaços físicos, ajustando automaticamente a climatização conforme a demanda, mas também visa resolver um dos desafios mais persistentes enfrentados pelos postos de saúde públicos brasileiros. A combinação de tecnologia avançada com princípios de gestão eficiente permite não apenas melhorar o conforto dos pacientes e funcionários, mas também contribuir significativamente para a redução do consumo energético e a sustentabilidade ambiental.

Desenvolver um projeto voltado para um setor tão crucial como a saúde pública não é tarefa simples. Cada decisão tomada durante o desenvolvimento do "Gasperin Web Services" visa não apenas otimizar a operação diária dos postos de saúde, mas também garantir que cada ajuste tecnológico seja um passo em direção a um atendimento mais eficiente e equitativo para todos os usuários do sistema de saúde pública.

Durante seu desenvolvimento, buscamos criar uma aplicação que não apenas oferecesse uma solução ao problema, mas também a possibilidade de ampliar ainda mais seus recursos, a fim de, possivelmente, fornecer soluções a problemas ainda maiores. Isso inclui discutir como o "Gasperin Web Services" pode

ser adaptado para enfrentar desafios futuros e oportunidades de expansão, promovendo uma melhoria contínua na entrega de serviços de saúde pública no Brasil.

Em resumo, a $Gasperin\ Web\ Services$ não é apenas uma solução tecnológica; é um compromisso com a inovação e a qualidade no setor de saúde pública. Ao integrar avanços da ciência da computação com as necessidades urgentes do atendimento médico, esperamos não apenas melhorar as operações dos postos de saúde, mas também estabelecer um padrão de excelência que inspire futuras iniciativas de saúde digital em todo o país.

2 Objetivos

2.1 Objetivo Geral

O objetivo é criar uma aplicação que combine recursos de software e hardware visando inovar os postos de saúde públicos a fim de um controle de fluxo de pessoas e um monitoramento de temperatura ambiente optimizados. Para tal, utilizaremos o framework Flask em conjunto com um banco de dados relacionalMySQL, bem como um dispositivo IOT via protocolo MQTT. Ademais, combinaremos tecnologias de front e back-end para criar uma aplicação interativa, dinâmica e de simples uso.

2.2 Objetivos Específicos

- Construir protótipos em EPP com sensores e atuadores.
- Codificar sistemas web utilizando conceitos de arquitetura de software, por meio de requisições e respostas aos serviços síncronos e assíncronos.
- Construir bancos de dados com os requisitos de dados da aplicação e realizar a integração com o sistema web utilizando framework de Mapeamento Objeto Relacional (ORM – Object Relational Mapper).
- Realizar a integração de sistemas web, projetos de banco de dados e microcontroladores utilizando interfaces de comunicação para solucionar demandas da IoT.

3 Metodologia

Este projeto foi desenvolvido utilizando o $framework\ Flask$, e seguindo a arquitetura $Model-View-Controller$ (MVC), em conjunto com diversas tecnologias de $front$ e $back-end$. Além do fato de que foi requisitado, esta escolha providencia uma aplicação web robusta, escalável e de fácil manutenção.

3.1 Camadas da Aplicação

- A camada $Model$ é responsável pelo gerenciamento dos dados, utilizando o banco de dados $MySQL$.
- A camada $View$ se refere aos $templates\ html$ da aplicação.
- A camada $Controllers$ gerencia as rotas da aplicação.

3.2 Hardware

Utilizamos uma ESP32 para gerenciar os equipamentos e enviar dados via Wi-Fi. Os componentes principais incluem um sensor de temperatura (DHT22) e um leitor de tags para detectar a passagem de pessoas. Dois servomotores controlam a abertura e fechamento de portas e janelas, agilizando o serviço e prevenindo a propagação de doenças. A comunicação entre o hardware e o software é realizada via protocolo MQTT, garantindo a transmissão eficiente de dados.

3.3 Banco de Dados

Para o armazenamento dos dados, optamos pelo MySQL, um sistema de gerenciamento de banco de dados relacional (RDBMS) conhecido por sua eficiência e confiabilidade. O banco de dados foi projetado para suportar todas as operações necessárias da aplicação. A conexão com o banco de dados é estabelecida através da biblioteca PyMySQL e as interações são gerenciadas pela biblioteca SQLAlchemy.

Ao iniciar a aplicação, o banco de dados é automaticamente reiniciado, mantendo apenas os valores definidos na própria aplicação. Esses valores iniciais são especificados no arquivo *initial_insert.py*. Não é necessário configurar a conexão ou criar um usuário específico para o MySQL manualmente; basta rodar o MySQL na máquina local e a aplicação se encarregará do restante.

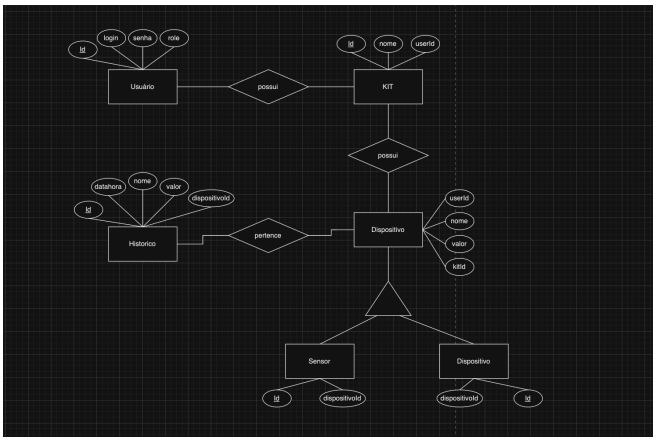

Fig. 1. Modelo Conceitual

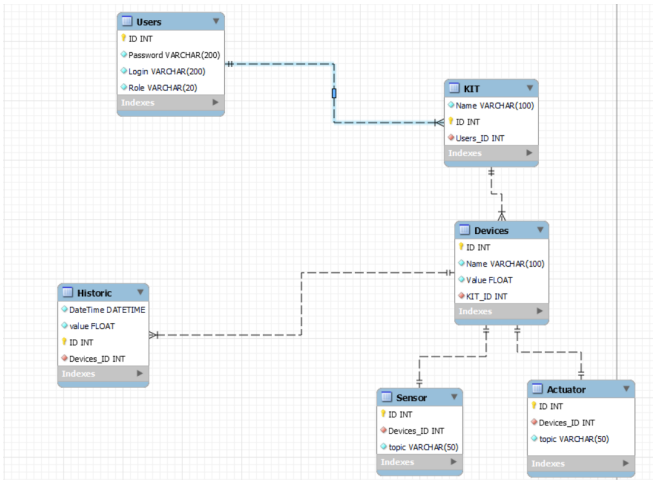

Fig. 2. Modelo Logico

3.4 Front-End

O $front-end$ da aplicação foi desenvolvido utilizando $HTML$ e $Jinja2$ para os $templates$, permitindo a renderização dinâmica das páginas web. Além disso, utilizamos CSS para estilização e $JavaScript$ para adicionar interatividade e efeitos visuais. Essa combinação de tecnologias $front-end$ garantiu uma interface de usuário responsiva e intuitiva.

3.5 Implementação

A implementação seguiu uma abordagem iterativa e incremental, com revisões e melhorias contínuas ao longo do processo de desenvolvimento. O uso do Flask, juntamente com o MySQL, CSS, JavaScript, HTML e Jinja2, proporcionou uma base sólida para a criação de uma aplicação eficiente e funcional, atendendo aos objetivos propostos de controle de fluxo e conforto nos postos de saúde públicos.

3.6 Configuração do Projeto

A aplicação utiliza diversas tecnologias, e para suas instalações localmente, é recomendado acessar o Repositório Oficial do projeto e seguir os passos abaixo.

```

# Clone o repositorio
git clone https://github.com/Lucas-LPG/gws

# Instale os requisitos
pip install -r requirements.txt
```

Listing 1. Project Setup

3.7 Estrutura da Aplicação

O projeto possui 6 diretórios relevantes: $ESP32$, $controllers$, db, $models$, $static$ e $views$. A estrutura de diretórios da aplicação como um todo é a seguinte:

```

.
    /ESP32
    /__pycache
    /controllers
        /__pycache__
    /db
        /__pycache__
    /models
        /__pycache__
    /static
        /css
        /img
        /js
    /views
        /actuators
        /devices
        /historic
        /kits
        /layouts
        /sensors
        /users
```

Listing 2. Estrutura dos Diretórios

Além daqueles já listados acima - pertencentes a arquitetura MVC, o diretório $ESP32$ contém os arquivos necessários para lidar com a conexão entre a aplicação e o dispositivo IOT, via protocolo $MQTT$. O diretório db lida com todas as operações necessárias para iniciar o banco de dados da aplicação, o que significa estabelecer a conexão, criar o usuário e as tablas, gerenciar os $triggers$ e executar a inserção inicial. Por fim, o diretório $static$ lida com os elementos estáticos da aplicação, portanto arquivos css, $JavaScript$ e imagens.

3.7.1 Models

O diretório models contém todas as classes que representam os objetos a serem persistidos no banco de dados, juntamente com

seus respectivos métodos. As classes incluídas são: Actuators, Devices, Historic, Kits, Sensors e Users. Cada uma dessas classes está relacionada a uma tabela específica do banco de dados e define os atributos e métodos necessários para manipulação dos dados.

3.7.1a Users

A classe Users é responsável por armazenar e gerir as informações sobre os usuários. A classe armazena dados como nome, senha e o cargo que o usuário exerce.

Os atributos da classe são:

- id: Utilizado para identificar cada usuário de maneira rápida e eficiente, sendo um número único.
- $name$: Nome do usuário em questão, deve ser um nome único e não pode ser nulo.
- $password$: Senha do usuário para realizar o login de maneira segura, não pode ser nula.
- $role$: O cargo que o usuário está exercendo no gerenciamento dos dados, sendo esses cargos limitados a admin, estatístico e operador.

Os métodos da classe são, em geral, para realizar busca, validação, inserção, atualização e remoção dos usuários.

- $validate\_user$: Valida o usuário com base no nome e senha passados como parâmetros. Realiza uma busca e verifica se esses parâmetros estão contidos no banco de dados.
- $insert\_into\_users$: Realiza a inserção de um novo usuário no banco de dados, passando seu nome, senha e cargo como parâmetros.
- $select\_all\_information\_from\_users$: Busca e mostra todas as informações sobre todos os usuários presentes no banco de dados. As informações de retorno são nome de usuário, cargo, ID de usuário e nome do kit.
- $select\_all\_from\_users$: Retorna todas as informações dos objetos presentes na classe Users, como seus IDs, nomes, senhas e cargos.
- $select\_from\_users$: Realiza uma busca com base na condição passada como parâmetro (por exemplo, um ID de usuário específico) e retorna o usuário, se existir.
- $select\_user\_by\_id$: Retorna o usuário com ID igual ao passado como parâmetro na função, caso exista.
- $select\_user\_by\_name$(name): Retorna o usuário com nome igual ao passado como parâmetro na função, caso exista.
- $update\_given\_user$: Realiza a atualização dos dados de um usuário específico com os novos dados passados como parâmetros (ID de usuário, nome de usuário, senha do usuário e cargo).
- $delete\_user\_by\_id$: Realiza a remoção de um usuário com base no ID dele, passado como parâmetro.

3.7.1b Kits

A classe Kit armazena os dados dos kits, que sao os nomes dos kits e o id do usuario que tem acesso a ele.

Os atributos da classe são:

- id: Utilizado para identificar cada kit de maneira rápida e eficiente, sendo um número único e autoincrementado.
- $name$: Nome do kit em questão, deve ser um nome único e não pode ser nulo.
- $user\_id$: Utiliza o ID do usuário para identificar qual usuário tem acesso a esse kit.

Os métodos da classe são, em geral, para realizar busca, inserção, atualização e remoção dos kit.

- $select\_all\_from\_kits$: Seleciona todas as informações sobre os kits, incluindo o ID do kit, seu nome, o nome do usuário que possui o kit, o total de sensores e o total de atuadores.
- $select\_kit\_by\_id$: Realiza uma busca e seleciona o kit que tenha o ID igual ao passado como parâmetro.
- $select\_kit\_by\_name$: Realiza uma busca e seleciona o kit que tenha o nome igual ao passado como parâmetro.
- $update\_given\_kit$: Atualiza o nome e o ID do usuário de um kit específico, identificado pelo seu ID. Todas as informações necessárias para a atualização, como o novo nome do kit e o novo ID do usuário, são passadas como parâmetros.
- $delete\_kit\_by\_id$: Remove o kit que possui o ID igual ao passado como parâmetro.

3.7.1c Devices

A classe Device armazena os dados dos dispositivos utilizados pelos kits, como seu ID, nome, valor e o ID do kit ao qual pertence.

Os atributos da classe são:

- id: Utilizado para identificar cada dispositivo de maneira rápida e eficiente, sendo um número único e autoincrementado.
- $name$: Nome do dispositivo em questão, não pode ser nulo.
- $value$: Armazena o valor do dispositivo em questao.
- kit_id: Armazena o ID do kit ao qual o dispositivo pertence.

Os métodos da classe são para realizar buscas especificas, com base no id e no nome.

- $select\_device\_by\_name$: Realiza uma busca e seleciona o dispositivo com base no nome, que é passado como parâmetro.
- $select\_device\_by\_id$: Realiza uma busca e seleciona o dispositivo com o ID igual ao passado como parâmetro.

3.7.1d Actuators

A classe Actuator armazena os dados dos atuadores, incluindo o seu ID, tópico para receber informações e o ID do dispositivo ao qual está associado.

Os atributos da classe são:

- id: Utilizado para identificar cada atuador de maneira rápida e eficiente, sendo um número único e autoincrementado.
- $topic$: Armazena o tópico através do qual o atuador se comunica com o servidor via MQTT. Não pode ser nulo.
- $device\_id$: Armazena o ID do dispositivo ao qual o atuador pertence.

Os métodos da classe são, em geral, para realizar busca, inserção, atualização e remoção dos atuadores.

- $insert\_actuator$: Realiza a inserção de um novo atuador, utilizando como parâmetros o nome do kit, o ID do kit, o nome do dispositivo, o valor e o tópico. Se o dispositivo já existir, associa o atuador a ele; caso contrário, cria um novo dispositivo e associa o atuador a ele.
- $select\_all\_from\_actuators$: Realiza uma busca e retorna todas as informações sobre os atuadores, incluindo o tópico, o ID dos atuadores, o ID do dispositivo, o nome do dispositivo, o valor do dispositivo e o nome do kit.
- $update\_given\_actuator$: Realiza a atualização do atuador com base no ID do dispositivo e do atuador, que são passados como parâmetros. Ele modifica o nome do dispositivo, valor do dispositivo, ID do kit, tópico e ID do dispositivo, todos passados como parâmetros.
- $update\_actuator\_by\_id$: Realiza a atualização do atuador com base no ID do atuador, passado como parâmetro. Modifica o nome do dispositivo, valor do dispositivo e tópico do atuador, todos passados como parâmetros.
- $select\_actuators\_by\_id$: Realiza uma busca e seleciona os atuadores com base no ID do dispositivo, que é passado como parâmetro, retornando informações como o tópico do atuador, o ID do atuador, o ID do dispositivo, o nome do dispositivo, o valor do dispositivo e o nome do kit.
- $select\_single\_actuator\_by\_id$: Realiza uma busca e seleciona o atuador com o ID igual ao passado como parâmetro.
- $select\_device\_by\_actuator\_id$: Realiza uma busca e seleciona o dispositivo associado ao atuador com base no ID do atuador, passado como parâmetro.
- $update\_actuator\_button\_value$: Atualiza o valor de um atuador específico, identificado pelo ID do dispositivo, incrementando o valor atual pelo novo valor passado como parâmetro.
- $delete\_actuator\_by\_id$: Remove o atuador que possui o ID igual ao passado como parâmetro.

3.7.1e Sensors

A classe Sensor armazena os dados dos sensores, incluindo o seu ID, tópico para receber informações e o ID do dispositivo ao qual está associado.

Os atributos da classe são:

- id: Utilizado para identificar cada sensor de maneira rápida e eficiente, sendo um número único e autoincrementado.
- $topic$: Armazena o tópico através do qual o sensor se comunica com o servidor via MQTT. Não pode ser nulo.

- $device\_id$: Armazena o ID do dispositivo ao qual o sensor pertence.

Os métodos da classe são, em geral, para realizar busca, inserção, atualização e remoção dos sensores.

- $insert\_sensor$: Realiza a inserção de um novo sensor, utilizando como parâmetros o nome do kit, o ID do kit, o nome do dispositivo, o valor e o tópico. Se o dispositivo já existir, associa o sensor a ele; caso contrário, cria um novo dispositivo e associa o sensor a ele.
- $select\_all\_from\_sensors$: Realiza uma busca e retorna todas as informações sobre os sensores, incluindo o tópico, o ID dos sensores, o ID do dispositivo, o nome do dispositivo, o valor do dispositivo e o nome do kit.
- $update\_given\_sensor$: Realiza a atualização do sensor com base no ID do dispositivo e do sensor, que são passados como parâmetros. Modifica o nome do dispositivo, valor do dispositivo, ID do kit, tópico e ID do dispositivo, todos passados como parâmetros.
- $update\_sensor\_by\_id$: Realiza a atualização do sensor com base no ID do sensor, passado como parâmetro. Modifica o nome do dispositivo, valor do dispositivo e tópico do sensor, todos passados como parâmetros.
- $select\_sensors\_by\_id$: Realiza uma busca e seleciona os sensores com base no ID do dispositivo, que é passado como parâmetro, retornando informações como o tópico do sensor, o ID do sensor, o ID do dispositivo, o nome do dispositivo, o valor do dispositivo e o nome do kit.
- $select\_single\_sensor\_by\_id$: Realiza uma busca e seleciona o sensor com o ID igual ao passado como parâmetro.
- $select\_device\_by\_sensor\_id$: Este método realiza uma busca e seleciona o dispositivo associado ao sensor com base no ID do sensor passado como parâmetro.
- $select\_from\_sensors$: Este método realiza uma busca nos sensores com base na condição passada como parâmetro e retorna os sensores que correspondem a essa condição.
- $update\_sensor\_value$: Este método atualiza o valor de um sensor específico, identificado pelo ID do dispositivo passado como parâmetro, para o novo valor também passado como parâmetro.
- $delete\_sensor\_by\_id$: Remove o sensor que possui o ID igual ao passado como parâmetro.

3.7.1f Historic

A classe Historic armazena os históricos dos atuadores e dos sensores sempre que um objeto da classe Device sofrer alguma alteração no valor. Isso inclui o valor anterior, a data em que ocorreu a alteração e o ID do dispositivo.

Os atributos da classe são:

- id: Utilizado para identificar cada histórico de maneira rápida e eficiente, sendo um número único e autoincrementado.
- $value$: Armazena o valor anterior do dispositivo antes de sofrer alguma alteração no atributo value(valor) da classe Device.
- $datetime$: Armazena a data e horário do último valor antes da alteração.
- $device\_id$: Armazena o ID do dispositivo ao qual o sensor ou atuador pertence.

Os métodos da classe são, no geral, para realizar buscas nos históricos dos dispositivos.

- $select\_all\_from\_historic$: Realiza uma busca e retorna todos os registros de históricos, incluindo informações como nome do kit, nome do dispositivo, valor do dispositivo e data e horário da alteração.
- $select\_all\_from\_sensor\_historic$: Realiza uma busca e retorna todos os registros de históricos dos sensores, incluindo informações como nome do kit, nome do dispositivo, valor do dispositivo e data e horário da alteração, ordenados por data de forma decrescente.
- $select\_all\_from\_sensor\_historic$: Realiza uma busca e retorna todos os registros de históricos dos sensores, incluindo informações como nome do kit, nome do dispositivo, valor do dispositivo e data e horário da alteração, ordenados por data de forma decrescente.
- $select\_by\_datetime\_from\_Sensor\_historic$: Realiza uma busca e retorna os registros de históricos dos sensores que ocorreram dentro do intervalo de tempo especificado, utilizando uma data de início e uma data

de fim passadas como parâmetros. As informações retornadas incluem o nome do kit, o nome do dispositivo, o valor do dispositivo e a data e horário da alteração, ordenados por data de forma decrescente.
- *select_by_datetime_from_Actuator_historic*: Realiza uma busca e retorna os registros de históricos dos atuadores que ocorreram dentro do intervalo de tempo especificado, utilizando uma data de início e uma data de fim passadas como parâmetros. As informações retornadas incluem o nome do kit, o nome do dispositivo, o valor do dispositivo e a data e horário da alteração, ordenados por data de forma decrescente.
- *select_by_datetime_from_historic*: Realiza uma busca e retorna os registros de históricos dos dispositivos que ocorreram dentro do intervalo de tempo especificado, utilizando uma data de início e uma data de fim passadas como parâmetros. As informações retornadas incluem o nome do kit, o nome do dispositivo, o valor do dispositivo e a data e horário da alteração, ordenados por data de forma decrescente.
- *select_datetime_by_device_id*: Retorna a data e horário da última alteração registrada para o dispositivo identificado pelo ID do dispositivo passado como parâmetro.
- *select_historic_by_device_id*: Realiza uma busca e retorna o registro de histórico mais recente para o dispositivo identificado pelo ID do dispositivo passado como parâmetro.

3.7.2 Views

O diretório *views* é bastante simples. Contendo todos os templates *HTML* da aplicação, ele está organizado de forma que grupos de templates que fazem referência a um tipo especial de página, como aquelas relacionadas a sensores e atuadores, por exemplo, se encontram dentro de subdiretórios, enquanto templates individuais se encontram na *root* do diretório.

3.7.2a Estrutura do Diretório

A estrutura do diretório *views*, com seus arquivos e subdiretórios é da seguinte forma:

```
.
  /actuators
      /actuators.html
      /register_actuator.html
      /update_actuator.html
  /devices
      /devices.html
      /edit_device.html
      /register_device.html
  /historic
      /data-historic.html
      /data_history.html
  /kits
      /edit_kits.html
      /kits.html
      /register_kit.html
  /layouts
      /layout.html
  /sensors
      /register_sensor.html
      /sensors.html
  /users
      /edit_user.html
      /register_user.html
      /users.html
  about.html
  home.html
  landing.html
  login.html
  real_time.html
```

Listing 3. Estrutura do Diretório

3.7.2b Métodos e Técnologias

Com a exclusão do template *login.html*, todos os arquivos extendem o layout *layout.html*. Este aceita como argumentos dois blocos: *title* e *content*. O bloco *title* se refere aos elementos únicos do elemento *HTML head* da página que extende o layout, incluindo título, imports, metadados, etc. Já o bloco *content* contém o conteúdo principal da página.

Por exemplo, para criar um novo template que utiliza o layout padrão, o arquivo pode ser estruturado da seguinte maneira:

```

  {% block title %}
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=
      device-width, initial-scale=1.0" />
  <link rel="stylesheet" href="../static/
      css/home.css" />
  <script src="../static/js/home.js" defer
      ></script>
  <title>Gasperin Web Services</title>
  {% endblock %}
```

Listing 4. Snippet de home.html

Enquanto o segundo se refere ao conteúdo da página, que estará localizado abaixo do

header incluso em *layout.html*. Ademais, diversos layouts fazem o uso da linguagem *Jinja2* para mostrar informações de forma dinâmica ao usuário. Por exemplo, o código a seguir cria um elemento *user–card* para cada user passado ao template.

```
1
2   {% for user in users %}
3   <div class="user--card">
4     <div class="user-name">
5       <span>{{ user.name }}</span>
6     </div>
7     <div class="user-role">
8       <span>{{user.role}} </span>
9     </div>
10    <div class="user-actions">
11      <a class="user-edit" href="/
            edit_user?user_id={{ user.id }}"
            >Editar</a>
12      <a class="user-delete" href="/
            delete_user?user_id={{ user.id
            }}"
13        >Deletar</a
14      >
15    </div>
16  </div>
17  {% endfor %}
```

Listing 5. Snippet de users/users.html

3.7.3 Controllers

O diretório controllers é usado para armazenar as rotas e organizar o código, visando melhor legibilidade e manutenção. Ao separar funcionalidades específicas em arquivos dedicados, o código se torna mais modular e fácil de gerenciar.

3.7.3a $\_\_init\_\_$

O arquivo $\_\_init\_\_.py$ é equivalente a $app\_controller.py$ configura o aplicativo Flask, integrando o MQTT para comunicação em tempo real, gerenciamento de login e rotas diversas para manipulação de kits, dispositivos e histórico de dados.

- $create\_app$: Este método inicializa a aplicação Flask com configurações específicas, como o gerenciamento de sessões e a integração com o MQTT para comunicação em tempo real. Ele também define o gerenciador de login e registra várias rotas essenciais para a manipulação de kits, dispositivos, histórico de dados e gerenciamento de usuários. Essa configuração centralizada garante que todas as funcionalidades do aplicativo estejam prontas para uso, facilitando a manutenção e a extensão do código.
- $index$: A rota $index()$ renderiza a página principal do aplicativo. Essa função serve como ponto de entrada para os usuários, onde eles podem acessar outras funcionalidades do sistema. É uma rota simples que redireciona os usuários para a interface principal da aplicação.
- $handle\_mqtt\_message$:O método $handle\_mqtt\_message()$ lida com mensagens recebidas via MQTT, um protocolo de comunicação leve. Quando uma mensagem MQTT é recebida, este método processa a mensagem, atualizando os valores dos sensores e atuadores conforme necessário. Isso permite que o sistema reaja em tempo real às mudanças e eventos detectados pelos dispositivos conectados.
- $kits$:A rota $kits()$ é protegida e renderiza uma página que exibe a lista de kits associados ao usuário logado. Esta função facilita a visualização e o gerenciamento dos kits, fornecendo uma interface onde os usuários podem acessar rapidamente informações detalhadas sobre cada kit.
- $edit\_kit$: O método $edit\_kit()$ permite a edição dos detalhes de um kit específico. Ao acessar esta rota, o usuário pode modificar as informações associadas ao kit selecionado. Isso é essencial para manter os dados dos kits atualizados e corretos.
- $data\_history$: A $rotadata\_history()$ é protegida e permite que os usuários visualizem o histórico de dados dos sensores e atuadores. Esta função é crucial para monitorar o desempenho ao longo do tempo e identificar padrões ou anomalias nos dados coletados.
- $edit\_given\_kit$: O método $edit\_given\_kit()$ valida a existência do kit e do usuário antes de permitir a edição dos detalhes do kit específico. Isso assegura que apenas kits válidos e pertencentes ao usuário logado possam ser modificados, garantindo a integridade dos dados.
- $remove\_kit$: A rota $remove\_kit()$ é protegida e permite que o usuário remova um kit especificado. Ao acessar esta função, o usuário pode deletar permanentemente um kit e todas as suas informações associadas do sistema.

- add_kit: A rota $add\_kit()$ permite a adição de um novo kit ao sistema, validando a existência do kit e do usuário. Isso assegura que apenas kits válidos sejam registrados, mantendo a consistência dos dados.
- $devices$: A função $devices()$ é protegida e renderiza uma página com a lista de sensores e atuadores associados ao usuário logado. Esta interface permite que os usuários visualizem rapidamente todos os dispositivos conectados e seus estados.
- $edit\_device$: O método $edit\_device()$ facilita a edição dos detalhes de sensores e atuadores. Ao acessar esta rota, o usuário pode modificar as informações específicas de cada dispositivo, garantindo que os dados estejam sempre atualizados.
- $edit\_given\_device$: A função $edit\_given\_device()$ valida a existência do dispositivo e do kit antes de permitir a edição. Isso garante que apenas dispositivos válidos, pertencentes a kits existentes, possam ser modificados.
- $delete\_device$: A rota $delete\_device()$ é protegida e permite que o usuário delete um dispositivo especificado. Esta função remove permanentemente o dispositivo do sistema, incluindo todas as suas informações associadas.
- $register\_device$: A função $register\_device()$ renderiza um formulário protegido para registrar novos dispositivos. Isso facilita a adição de novos sensores e atuadores ao sistema, garantindo que todas as informações necessárias sejam fornecidas pelo usuário.
- add_device: A rota $add\_device()$ permite a adição de um novo dispositivo ao sistema, validando a existência do kit e do dispositivo. Isso assegura que apenas dispositivos válidos sejam registrados, mantendo a integridade dos dados.
- $publish\_message$: A função $publish\_message()$ permite a publicação de uma mensagem via MQTT. Isso facilita a comunicação em tempo real entre o sistema e os dispositivos conectados, permitindo o envio de comandos e atualizações instantâneas.
- $real\_time$: A rota $real\_time()$ renderiza a página de monitoramento em tempo real, exibindo informações como a temperatura e a contagem de pessoas. Esta interface permite que os usuários monitorem os dados em tempo real, facilitando a tomada de decisões imediatas.
- $handle\_connect$: O método $handle\_connect()$ lida com a conexão bem-sucedida ao broker MQTT, subscrevendo aos tópicos relevantes. Isso assegura que o sistema esteja pronto para receber e processar mensagens MQTT assim que a conexão for estabelecida.
- $handle\_disconnect$: A função $handle\_disconnect()$ lida com a desconexão do broker MQTT. Isso permite que o sistema reaja apropriadamente a uma perda de conexão, tentando reconectar ou notificando os usuários sobre o problema.
- $load\_user\_from\_request$: O método $load\_user\_from\_request()$ carrega o usuário a partir de uma requisição, tentando autenticar via chave API ou autenticação básica. Isso facilita a identificação e autenticação de usuários em diferentes contextos de requisição.
- $load\_user$: A função $load\_user()$ carrega o usuário a partir do ID do usuário. Isso permite que o sistema obtenha informações detalhadas sobre o usuário, facilitando a personalização e gerenciamento de sessões.

3.7.3b $actuators$

O arquivo $actuators.py$ gerencia todas as operações relacionadas aos atuadores no sistema. Ele permite o registro, listagem, atualização e remoção de atuadores, fornecendo formulários protegidos para que os usuários possam adicionar novos atuadores e visualizar ou modificar os existentes. Este módulo assegura que todas as informações dos atuadores estejam atualizadas e corretamente associadas aos usuários.

- $register\_actuators$: A função $register\_actuators()$ renderiza um formulário protegido para registrar novos atuadores. Isso permite que os usuários adicionem atuadores ao sistema de maneira estruturada e validada.

- $list\_actuators$:A rota $list\_actuators$ é protegida e lista todos os atuadores associados ao usuário logado. Esta função facilita a visualização e gerenciamento de todos os atuadores disponíveis.
- $add\_actuators$: A função $add\_actuators()$ permite a adição de um novo atuador ao sistema, processando os dados do formulário submetido pelo usuário. Isso garante que os novos atuadores sejam corretamente registrados e associados ao usuário.
- $update\_actuator$: A rota $update\_actuator()$ permite a atualização das informações de um atuador existente. Esta função facilita a manutenção e atualização dos dados dos atuadores, garantindo que estejam sempre corretos e atualizados.
- $update\_actuator$: A rota $update\_actuator()$ permite a atualização das informações de um atuador existente. Esta função facilita a manutenção e atualização dos dados dos atuadores, garantindo que estejam sempre corretos e atualizados.
- $del\_actuator$: A função $del\_actuator()$ permite que o usuário delete um atuador especificado. Isso remove permanentemente o atuador do sistema, incluindo todas as suas informações associadas.

3.7.3c $sensors$

O arquivo $sensors.py$ lida com as operações relacionadas aos sensores, incluindo o registro, listagem, atualização e remoção de sensores. Ele oferece formulários protegidos para adicionar novos sensores e interfaces para visualizar e gerenciar os sensores já registrados. Esse módulo garante a consistência e integridade dos dados dos sensores no sistema.

- $register\_sensors$: A função $register\_sensors()$ renderiza um formulário protegido para registrar novos sensores. Isso facilita a adição de novos sensores ao sistema, garantindo que todas as informações necessárias sejam fornecidas pelo usuário.
- $list\_sensors$: A rota $list\_sensors()$ é protegida e lista todos os sensores associados ao usuário logado. Esta função facilita a visualização e gerenciamento de todos os sensores disponíveis.
- $add\_sensors$: A função $add\_sensors()$ permite a adição de um novo sensor ao sistema, processando os dados do formulário submetido pelo usuário. Isso garante que os novos sensores sejam corretamente registrados e associados ao usuário.
- $update\_sensor$: A rota $update\_sensor()$ permite a atualização das informações de um sensor existente. Esta função facilita a manutenção e atualização dos dados dos sensores, garantindo que estejam sempre corretos e atualizados.
- $remove\_sensor$: A função $remove\_sensor()$ renderiza a página para remover sensores. Isso facilita a exclusão de sensores específicos do sistema, fornecendo uma interface para seleção e confirmação da remoção.
- del_sensor:A função $del\_sensor()$ permite que o usuário delete um sensor especificado. Isso remove permanentemente o sensor do sistema, incluindo todas as suas informações associadas.

3.7.3d $login$

O arquivo $login.py$ é responsável pelo gerenciamento de usuários, incluindo autenticação de login, registro de novos usuários e listagem dos usuários existentes. Ele processa as credenciais submetidas para autenticação, garante acesso seguro às áreas protegidas do sistema e permite que administradores adicionem ou removam usuários conforme necessário.

- $login\_func$: A função $login\_func()$ processa as credenciais submetidas pelo usuário e autentica o login. Isso garante que apenas usuários válidos possam acessar o sistema, protegendo as informações e funcionalidades.
- $home$: A rota $home()$ é protegida e renderiza a página inicial após o login, exibindo dados relevantes como kits e usuários associados. Esta função fornece uma visão geral rápida e acessível das informações principais do sistema.
- $register\_user$: A função $register\_user()$ renderiza um formulário protegido para registrar novos usuários. Isso facilita a adição de novos usuários ao sistema,

garantindo que todas as informações necessárias sejam fornecidas.
- add_users: A função $add\_users()$ permite a adição de um novo usuário ao sistema, processando os dados do formulário submetido. Isso assegura que novos usuários sejam corretamente registrados e autenticados.
- $list\_users$: A rota $list\_users()$ é protegida e lista todos os usuários registrados no sistema. Esta função facilita a visualização e gerenciamento de todos os usuários disponíveis.
- $remove\_user$: A função $remove\_user()$ permite que o administrador delete um usuário especificado. Isso remove permanentemente

3.7.4 db

O diretório db é fundamental para a configuração e gerenciamento do banco de dados. Utilizando arquivos como $\_\_init\_\_.py$, $clean\_db.py$, $events.py$ e $initial\_insert.py$, o sistema realiza desde a inicialização e limpeza do banco de dados até a criação de estruturas, como tabelas e gatilhos, para capturar eventos importantes, como inserções e exclusões. Além disso, são realizadas inserções iniciais para preencher o banco de dados com dados básicos. Essas operações garantem a integridade e funcionalidade do sistema de banco de dados ao longo do tempo.

3.7.4a __init__

Este arquivo desempenha um papel crucial na inicialização do aplicativo Flask e na configuração do contexto do aplicativo. Aqui, são importados modelos de dados (models) e módulos que lidam com operações de banco de dados, como limpeza, criação de gatilhos e inserção inicial de dados.

3.7.4b clean_db

O arquivo $clean\_db.py$ é responsável por limpar o banco de dados antes de sua inicialização. Ele utiliza SQL raw para executar operações como:

- $drop\_database\_stmt$: Remove o banco de dados existente, se houver.
- $create\_database\_stmt$: Cria um novo banco de dados com o nome especificado, no caso $puhl\_gasperin\_health$.
- $use\_database\_stmt$: Manda utilizar o banco de dados com o nome especificado
- $drop\_user\_stmt$: Remove um usuário de banco de dados, se já existir.
- $create\_user\_stmt$: Cria um novo usuário com permissões específicas para interagir com o banco de dados.
- $grant\_stmt$: Concede todas as permissões para o usuário criado, garantindo acesso completo ao banco de dados.

3.7.4c connection

No arquivo $connection.py$, a conexão com o banco de dados é configurada usando SQLAlchemy, uma biblioteca de mapeamento objeto-relacional (ORM) para Python. As etapas incluem:

- Configuração dos parâmetros de conexão como usuário, senha, servidor, porta e nome do banco de dados $puhl\_gasperin\_health$.
- Verificação da existência do banco de dados. Se não existir, ele é criado usando $create\_database$ da SQLAlchemy Utils.
- Essa configuração garante que o ambiente de banco de dados esteja pronto para ser utilizado pelo aplicativo.

3.7.4d events

O arquivo $events.py$ gerencia a criação de gatilhos no banco de dados. Gatilhos são procedimentos automáticos que são acionados por eventos específicos no banco de dados, como inserções, atualizações e exclusões. Este arquivo contém:

- $create\_historic\_trigger$: Este método cria um gatilho que é acionado após a inserção de um novo registro na tabela devices. Quando um dispositivo é adicionado, o gatilho insere automaticamente um registro correspondente na tabela historic, incluindo o valor do dispositivo, a data e hora da inserção ($NOW()$) e o ID do dispositivo. Isso permite rastrear eventos históricos de novos dispositivos registrados no sistema.
- $update\_historic\_trigger$: Este método define um gatilho que é acionado após a atualização de um registro na tabela devices. Sempre que um dispositivo é atualizado, o gatilho insere um novo registro

na tabela historic, contendo o novo valor do dispositivo, a data e hora da atualização ($NOW()$) e o ID do dispositivo. Isso ajuda a manter um histórico detalhado das alterações feitas nos dispositivos.
- $handle\_device\_deletion$: Este método cria um gatilho que é executado antes da exclusão de um registro na tabela devices. O gatilho garante que todas as referências ao dispositivo excluído sejam removidas das tabelas sensors, actuators e historic. Isso impede inconsistências no banco de dados, garantindo que não restem dados órfãos relacionados ao dispositivo excluído.
- $handle\_user\_deletion$: Este método define um gatilho que é acionado antes da exclusão de um registro na tabela users. O gatilho remove todos os kits associados ao usuário que está sendo excluído, garantindo que não fiquem kits sem referência a um usuário válido no sistema.
- $handle\_kit\_deletion$: Este método cria um gatilho que é executado antes da exclusão de um registro na tabela kits. O gatilho remove todos os dispositivos associados ao kit que está sendo excluído, assegurando que os dispositivos não permaneçam no banco de dados sem estar associados a um kit válido.

3.7.4e initial_insert

O arquivo $initial\_insert.py$ é responsável pela inserção inicial de dados no banco de dados. Isso inclui:

- $\_populate\_users$: Insere registros de usuários (User) no banco de dados. Cada usuário tem um nome, senha e papel no sistema.
- $\_populate\_kits$: Insere registros de kits (Kit) no banco de dados. Cada kit está associado a um usuário específico.
- $\_populate\_devices$: Insere registros de dispositivos (Device) no banco de dados. Cada dispositivo tem um nome, estado inicial e é associado a um kit.
- $\_populate\_actuators$ e $\_populate\_sensors$: Insere registros de atuadores e sensores no banco de dados, respectivamente. Cada um está associado a um dispositivo específico.

3.7.5 ESP32

O ESP32, integrado neste projeto, desempenha um papel crucial na coleta de dados ambientais e no controle de dispositivos periféricos. Este microcontrolador Wi-Fi e Bluetooth é central para a funcionalidade do sistema, permitindo a comunicação com a Internet e interação com sensores e atuadores.

3.7.5a Conexão Wi-Fi e MQTT

O ESP32 se conecta à rede Wi-Fi Wokwi-GUEST usando a interface network.WLAN do MicroPython. Após a conexão bem-sucedida, ele se conecta ao servidor MQTT hospedado em mqtt-dashboard.com. Isso facilita a troca de dados entre o dispositivo e outros serviços na nuvem.

3.7.5b Sensores e Atuadores

- $SensorDHT22$: Monitora temperatura e umidade ambiente. O sensor é configurado usando a biblioteca dht e fornece leituras precisas através do método sensor.measure()
- $BotesdeEntradaeSada$: Gerenciam o controle de fluxo de pessoas com os botões conectados aos pinos 25 e 26. A cada pressionamento, o sistema atualiza o número de pessoas e aciona o servo motor.
- $ServoMotores$: Controlados pelos pinos 22 (porta) e 27 (janela), ajustam a abertura da porta e da janela com base na quantidade de pessoas e nas condições climáticas.

3.7.5c Comunicação MQTT

Utiliza o cliente MQTT para enviar e receber mensagens do tópico cz/enviar e cz/degar. O método client.publish() envia dados de temperatura, número de pessoas e estado dos botões para o servidor MQTT, enquanto client.subscribe() recebe comandos para controlar o ar-condicionado e a janela.

3.7.5d Monitoramento de Conexão

A função check_connection() verifica periodicamente a conexão com o servidor MQTT,

reconectando-se automaticamente em caso de interrupções.

3.7.6 static

O diretório static é crucial para o funcionamento do website, pois nele estão armazenados todos os elementos estáticos essenciais, como imagens, arquivos CSS e JavaScript. Esses recursos desempenham um papel fundamental na estética e na usabilidade da aplicação.

As imagens contidas no diretório static incluem elementos visuais como logotipos, ícones e banners, que são fundamentais para a identidade visual do site e para melhorar a experiência do usuário ao tornar a navegação mais atraente e informativa.

Os arquivos CSS são responsáveis por estilizar as páginas web, definindo aspectos como cores, fontes e layout. Isso não apenas proporciona uma interface visualmente agradável, mas também garante consistência no design do site, independentemente do dispositivo usado pelo usuário.

Os scripts JavaScript presentes no diretório static adicionam interatividade às páginas, permitindo funcionalidades dinâmicas como menus deslizantes, validação de formulários e animações. Esses elementos não só melhoram a usabilidade, tornando o site mais intuitivo e responsivo, mas também proporcionam uma experiência de usuário mais envolvente.

Ao concentrar esses recursos em um único diretório, o static facilita o desenvolvimento e a manutenção do website, garantindo que atualizações e modificações nos elementos visuais e funcionais sejam gerenciadas de maneira eficiente.

4 Resultados

4.1 Evolução da Aplicação

As etapas de evolução pelas quais o projeto passou até chegar ao seu estado atual podem ser divididas em 4 principais:

- Inicialização dos recursos de hardware
- Inicialização da comunicação entre hardware e software
- Aplicação do $back-end$ da aplicação utilizando um banco de dados $MySQL$
- Optimização da aplicação

4.2 Inicialização dos recursos de hardware

Inciando o projeto não utilizamos nenhum recurso de software. No lugar, começamos apenas implementando o hardware descrito préviamente. Utilizando tanto peças de hardware físicas quanto a ferramenta Wokwi.

4.3 Web

Ao longo do desenvolvimento do projeto, realizamos várias melhorias para aprimorar a interação do usuário e tornar a aplicação mais dinâmica e intuitiva.

4.3.1 Tela de Início

A tela de início original apresentava uma interface simples, oferecendo poucas opções de navegação e informações básicas. Com o objetivo de proporcionar uma experiência mais envolvente e agradável para o usuário, decidimos implementar uma reformulação completa do design, adotando uma abordagem moderna e minimalista.

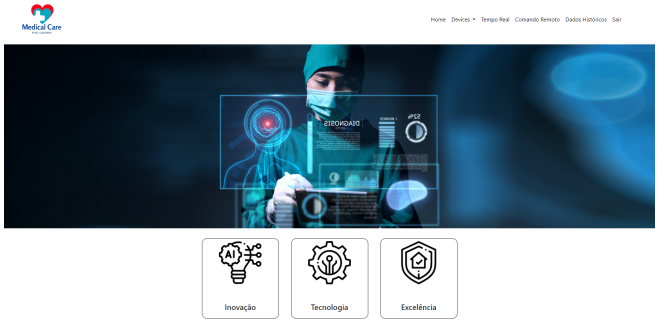

Fig. 3. Tela de início antiga

Fig. 4. Tela de início nova

4.3.2 Página Administrativa

A página reservada aos administradores da empresa era limitada e pouco intuitiva, com menus básicos pouco responsíveis que não ofereciam muitas funcionalidades avançadas. A falta de recursos interativos e a interface simplificada dificultavam a gestão eficiente do sistema. Isso impactava diretamente na capacidade de análise detalhada dos dados e na tomada de decisões informadas. Além disso, a ausência de gráficos detalhados e relatórios precisos limitava a compreensão completa do desempenho do sistema. Com a reformulação, procuramos criar um ambiente administrativo mais intuitivo e eficaz. Introduzimos um painel de controle completo com gráficos interativos que proporcionam uma visualização clara e em tempo real dos dados, facilitando a análise e o monitoramento contínuo. Essas mudanças visam não apenas melhorar a usabilidade, mas também aumentar a eficiência operacional e a capacidade de resposta às demandas dos usuários.

Fig. 5. Tela Console Antiga

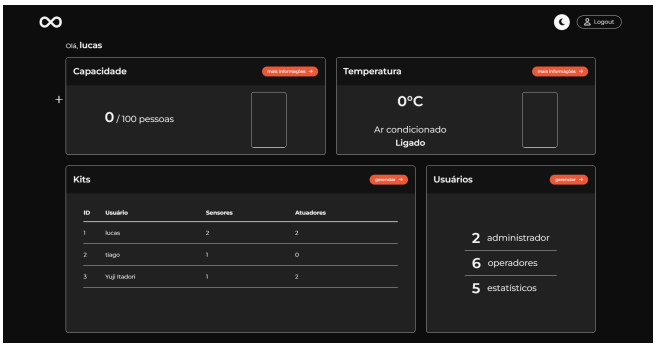

Fig. 6. Tela Console Nova

4.3.3 Nome do projeto

A empresa era inicialmente chamado de Medical Care Puhl Gasperin, um nome que poderia sugerir uma associação direta com serviços médicos e hospitalares. Todavia, apos considerarmos, alteramos o nome para Gasperin Web Services para refletir mais claramente nosso objetivo de fornecer uma solução tecnológica para hospitais e outros tipos de negócios, sem implicar envolvimento direto com a prática médica. Esse novo nome comunica melhor nosso foco em serviços tecnológicos e gerenciamento de ambientes.

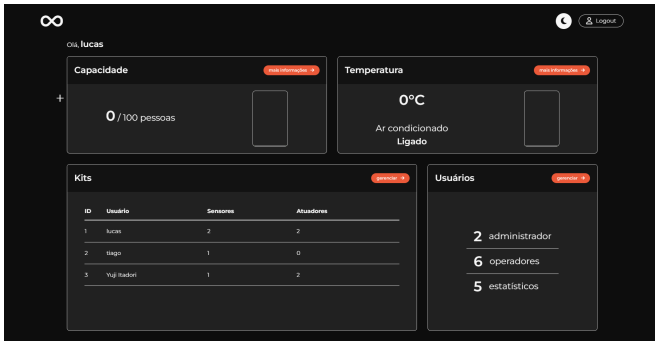

Fig. 7. Tela Console Dark

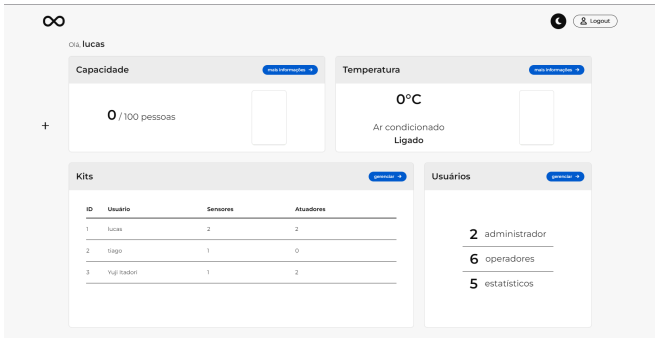

Fig. 8. Tela Console Light

4.3.4 Modos de Exibição

Decidimos criar algo simples que deixasse a interação com o website mais divertida e intuitiva. Para tal, implementamos a opção de modos de exibição dark e light, permitindo que o usuário escolha a interface que melhor se adapta às suas preferências. Essa funcionalidade pode ser facilmente ajustada a qualquer momento, proporcionando maior flexibilidade e conforto visual ao usuário.

4.4 Resultados Finais

Evidencia-se, portanto, que o website como um todo passou por uma transformação significativa e aprimorada. As mudanças não foram

feitas apenas para atender aos critérios estabelecidos, mas também com o objetivo de criar uma aplicação mais estética, minimalista e funcional, proporcionando uma experiência de uso superior.

4.5 Banco de dados

Durante o desenvolvimento do projeto, a implementação do sistema de login passou por uma significativa evolução. Inicialmente, a verificação de usuários era realizada sem o uso de banco de dados, utilizando apenas dicionários em Python para armazenar informações básicas de login.

No início, o sistema de login utilizava dicionários em Python para verificar as credenciais dos usuários. Este método era simples e rápido de implementar, mas apresentava limitações em termos de escalabilidade e segurança. As informações dos usuários, como nomes de usuário e senhas, eram armazenadas diretamente no código, o que não é uma prática recomendada para sistemas em produção.

Com a evolução do projeto, migramos o sistema de login para utilizar um banco de dados MySQL, acompanhada da biblioteca flask_login. Esta mudança permitiu um armazenamento mais seguro e escalável das informações dos usuários. Criamos tabelas específicas para armazenar os dados dos usuários e suas respectivas roles. Os usuários são agora categorizados em diferentes roles, como admin, user e operario. Isso permite um controle de acesso mais refinado e seguro, onde cada role tem permissões e funcionalidades específicas dentro da aplicação. A integração com o banco de dados também nos possibilitou implementar práticas de segurança mais robustas, como a utilização de tokens para sessões de login. A evolução do Flask Login reflete nosso compromisso em melhorar a segurança, escalabilidade e usabilidade da aplicação. Ao migrar de um sistema baseado em dicionários para um banco de dados estruturado, conseguimos oferecer um sistema de autenticação mais robusto e confiável para nossos usuários. Ademais, alguns templates presentes no diretório *Views* utilizam do Flask Login para verificar o que o usuário atual pode ou não acessar dentro da página em específico. Por exemplo, o *snippet* abaixo altera o link de referência de acordo com o role do usuário atual.

```
1
2   <a
3   class="card--link"
4   onmouseover="activateCursor()"
5   onmouseleave="deactivateCursor()"
6   href="{% if current_user.role != '
        operador' %} /real_time {% else %} /
        data_history {% endif %}"
7   >mais informacoes <i class="fa-solid fa-
        arrow-right"></i
8 ></a>
```

Listing 6. Snippet de home.html

5 Discussão

O desenvolvimento da aplicação web para o monitoramento do fluxo de pessoas em postos de saúde representa um avanço significativo na gestão e eficiência desses ambientes. A possibilidade de controlar a abertura e fechamento de portas conforme a demanda de pacientes permite uma organização mais dinâmica e adaptável às necessidades reais, melhorando a experiência dos usuários e otimizando os recursos humanos e materiais disponíveis. Além disso, a integração de sistemas para abertura e fechamento de janelas e controle do ar condicionado conforme a temperatura ambiente não só proporciona um ambiente mais confortável para pacientes e profissionais de saúde, mas também promove a eficiência energética, reduzindo os custos operacionais dos estabelecimentos.

Os resultados preliminares indicam uma melhoria na fluidez do atendimento e uma redução no tempo de espera dos pacientes. A capacidade de monitoramento em tempo real permite uma resposta rápida a situações de alta demanda, evitando aglomerações e garantindo um atendimento mais organizado e seguro, especialmente em tempos de pandemia. No entanto, algumas falhas foram observadas durante o processo de implementação. Problemas de conectividade foram identificados como pontos críticos que necessitam de soluções para garantir a confiabilidade do sistema. Como solução, deve-se investir em equipamentos duradouros e de qualidade, a fim de providenciar um funcionamento constante e duradouro.

As implicações futuras dessa tecnologia são promissoras. A possibilidade de criar um aplicativo móvel que não apenas mostre os dados de ocupação em tempo real, mas também direcione os pacientes para o posto de

saúde mais adequado conforme a demanda atual, pode revolucionar o acesso aos serviços de saúde. Essa funcionalidade reduziria ainda mais os tempos de espera e distribuiria melhor os pacientes entre as diversas unidades, evitando sobrecargas em determinados locais e subutilização em outros. Ademais, a análise dos dados coletados pode fornecer $insights$ valiosos sobre padrões de utilização dos serviços de saúde, permitindo uma melhor gestão e planejamento.

Entretanto, a implementação desse sistema requer a consideração de diversos fatores. Visto que existem diversos sistemas já em produção dentro dos postos de saúde públicos, é fundamental assegurar a compatibilidade do novo sistema com estes já existentes, promovendo uma integração eficiente e sem falhas. Investimentos em infraestrutura tecnológica e treinamento de pessoal são igualmente essenciais para o sucesso do projeto a longo prazo.

Em resumo, a aplicação web desenvolvida apresenta um potencial significativo para melhorar a gestão dos postos de saúde, proporcionando um atendimento mais eficiente e confortável para os pacientes. As falhas identificadas são superáveis com investimentos adequados em tecnologia e infraestrutura. As futuras expansões, incluindo o desenvolvimento de um aplicativo móvel e a análise de dados para aprimoramento contínuo dos serviços, delineiam um caminho promissor para a modernização e otimização do sistema de saúde pública.

Ademais, a proposta da empresa $Gasperin\ Web\ Services$ abre portas para atender diversos outros clientes posteriormente, não apenas gerenciando o controle automatizado do fluxo de pessoas e clima ambiente, mas também outros meios de inovação via serviços web. Através desse projeto, a GWS demonstra sua competência em áreas-chave que são altamente relevantes para uma variedade de outros setores e mercados globais.

6 Conclusão

Em resumo, a aplicação web desenvolvida apresenta um potencial significativo para melhorar a gestão dos postos de saúde, proporcionando um atendimento mais eficiente e confortável para os pacientes. As falhas identificadas são superáveis com investimentos adequados em tecnologia e infraestrutura. As futuras expansões, incluindo o desenvolvimento de um aplicativo móvel e a análise de dados para aprimoramento contínuo dos serviços, delineiam um caminho promissor para a modernização e otimização do sistema de saúde pública.

O projeto aqui descrito demonstra a viabilidade e a utilidade de soluções tecnológicas avançadas na gestão de serviços de saúde. Com a capacidade de monitorar e ajustar o fluxo de pacientes em tempo real, abre-se uma nova era de eficiência operacional e qualidade no atendimento. A implementação de sistemas automatizados para controle de portas, janelas e climatização não só melhora o conforto dos usuários como também contribui para a sustentabilidade dos recursos energéticos. O próximo passo envolve a ampliação das funcionalidades da aplicação, incluindo a criação de um aplicativo que possa informar os usuários sobre a lotação dos postos de saúde e direcioná-los de forma inteligente para as unidades mais adequadas.

A incorporação dessas tecnologias emergentes no dia a dia dos serviços de saúde pode levar a uma transformação significativa, onde a gestão de fluxo e a alocação de recursos se tornem mais eficazes e responsivas. Para alcançar esse futuro, é essencial um compromisso contínuo com a inovação, o investimento em tecnologia e a formação de profissionais capacitados para lidar com esses novos sistemas. Dessa forma, poderemos garantir que os avanços tecnológicos se traduzam em benefícios reais e duradouros para a sociedade.