

RELATÓRIO FINAL DA DISCIPLINA DE EXPERIÊNCIA CRIATIVA

BCC 3B Manhã

Alunos: Larissa Raimee, Lucas Galves, Isabela Navarro

- Introdução - Apresentação do Projeto

Lumen Pass é uma empresa comprometida em garantir a segurança dos cidadãos por meio de soluções inovadoras e simples de aplicar em cidades. Acreditamos que pequenas mudanças podem ter um grande impacto, e é por isso que o nosso principal projeto é a luz Lumen Pass - uma luz que acende quando um pedestre atravessa uma faixa de pedestres durante a noite.

Com a Lumen Pass, estamos ajudando a tornar as ruas mais seguras para pedestres e motoristas. Nós entendemos que a segurança é uma prioridade em áreas urbanas, e a nossa solução é uma forma eficaz de melhorar a visibilidade e reduzir o risco de acidentes de trânsito.

Além disso, a Lumen Pass também trabalha em colaboração com governos locais para implementar projetos de iluminação pública em áreas com pouca visibilidade, aumentando a segurança dos cidadãos e melhorando a qualidade de vida em geral.

Na Lumen Pass, estamos comprometidos em tornar as cidades mais seguras para todos os cidadãos, e nossa equipe está sempre empenhada em desenvolver soluções inovadoras para ajudar a tornar isso possível

- Aplicação Flask

O desenvolvimento da aplicação utiliza o framework Flask, que permite a criação de controladores utilizando Blueprints. Os controladores são responsáveis por gerenciar as rotas e as requisições HTTP, enquanto as visualizações (páginas HTML) são renderizadas para exibir as informações ao usuário. A estrutura do projeto é organizada em módulos para facilitar a manutenção e o desenvolvimento.

- Páginas de Cadastro e Listagem

O sistema conta com páginas de cadastro e páginas de listagem referentes à Usuários, Dispositivos e Sensores. Essas páginas são desenvolvidas utilizando HTML, CSS e Jinja2 (template engine do Flask). Cada página de cadastro possui um formulário onde o usuário pode inserir os dados necessários, enquanto as páginas de listagem apresentam uma tabela com os dados previamente cadastrados, permitindo a visualização e busca dessas informações.

- Página Inicial do Projeto

A página inicial do projeto é responsável por fornecer as informações básicas sobre o sistema. Essa página é desenvolvida utilizando HTML e CSS, e contém detalhes sobre o objetivo do sistema, suas funcionalidades principais e informações relevantes para o usuário.

- Controle de Acesso com o Flask-Login

Para garantir a segurança do sistema, foi utilizado o Flask-Login, uma extensão do Flask que gerencia o controle de acesso dos usuários. O Flask-Login permite o gerenciamento de sessões, autenticação de usuários e proteção da rota de visualização de usuários cadastrados, restringindo o acesso apenas a usuários autenticados.

- Banco de Dados e Modelos SQLAlchemy

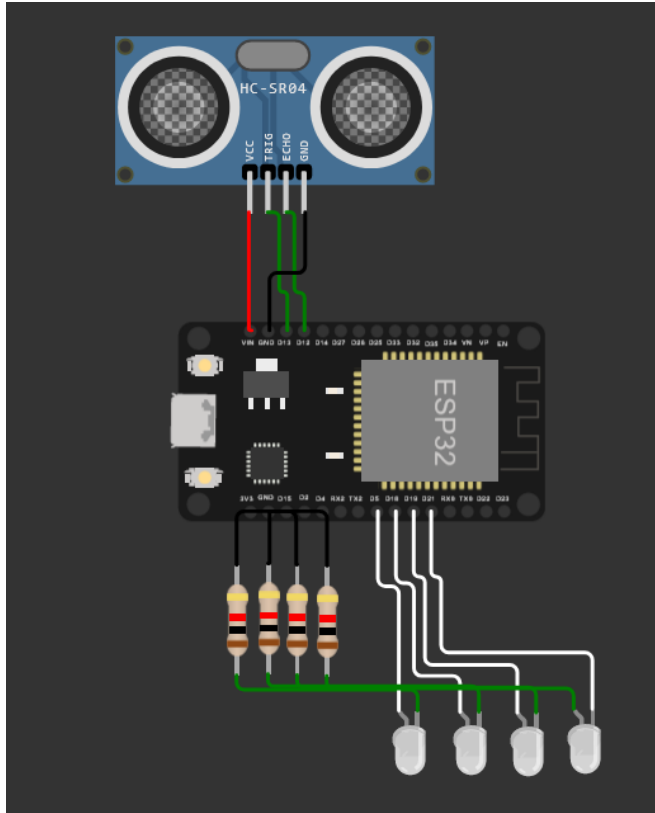
O sistema utiliza um banco de dados para armazenar e gerenciar as informações. Para a integração com o Flask, é utilizado o SQLAlchemy, um mapeador objeto-relacional (ORM) que facilita a manipulação dos dados. São implementados, no mínimo, 6 modelos SQLAlchemy, cada um representando uma entidade no sistema. Esses modelos definem as tabelas do banco de dados e estabelecem os relacionamentos entre elas.

| Tabela | Ação | Registos | Tipo | Agrupamento (Collation) | Tamanho | Suspensão |
|---|-------------|-----------|---------------|---------------------------|-----------------|----------------|
| <input type="checkbox"/> acao | ★ | 0 | InnoDB | utf8mb4_general_ci | 32.0 KB | - |
| <input type="checkbox"/> dispositivo | ★ | 4 | InnoDB | utf8mb4_general_ci | 16.0 KB | - |
| <input type="checkbox"/> read | ★ | 0 | InnoDB | utf8mb4_general_ci | 32.0 KB | - |
| <input type="checkbox"/> roles | ★ | 6 | InnoDB | utf8mb4_general_ci | 16.0 KB | - |
| <input type="checkbox"/> sensor | ★ | 0 | InnoDB | utf8mb4_general_ci | 32.0 KB | - |
| <input type="checkbox"/> users | ★ | 1 | InnoDB | utf8mb4_general_ci | 64.0 KB | - |
| <input type="checkbox"/> user_roles | ★ | 0 | InnoDB | utf8mb4_general_ci | 32.0 KB | - |
| 7 tabelas | Soma | 11 | InnoDB | utf8mb4_general_ci | 224.0 KB | 0 Bytes |

- Métodos de Inserção, Seleção, Edição e Exclusão dos Dados

No sistema, são implementados métodos para realizar as operações básicas de inserção, seleção, edição e exclusão dos dados no banco de dados. Esses métodos são utilizados pelos controladores para interagir com o banco de dados de forma eficiente e segura.

- Mapa conceitual:



Atuador: ESP32

Sensor ultrassônico de presença

LEDs

O funcionamento do código e do dispositivo envolve o acionamento do sensor de presença e conforme a proximidade registrada, os LEDs acendem ou apagam

- Código utilizado:

```

while True:
    # Fazendo o sensor funcionar:
    trig.value(0)
    time.sleep_us(2)
    trig.value(1)
    time.sleep_us(10)
    trig.value(0)

    # Pegando o valor:
    x=machine.time_pulse_us(echo,1)
    x = (0.034*x)/2

    print(f"Distância: {x:.3} cm")

    # Leds:
    if flag == 1:
        desligando()
        flag = 0

    if x <= 300 and x > 225:
        leds[0].on()
        flag = 1

    elif x <= 225 and x > 150:
        leds[1].on()
        flag = 1

    elif x <= 150 and x > 75:
        leds[2].on()
        flag = 1

    elif x <= 75 and x > 2:
        leds[3].on()
        flag = 1

```

- Tópicos MQTT utilizados

Ultrasonic/info: Recebe os dados do sensor ultrassônico

Led/command: Atualiza o status do led (ligado/desligado)

- Comunicação IoT com Flask-MQTT

O Flask-MQTT é uma extensão do Flask que permite a integração de um broker MQTT (Message Queuing Telemetry Transport) para a comunicação IoT. O MQTT é um protocolo de mensagens leve e eficiente, amplamente utilizado em aplicações de IoT para o envio e recebimento de dados.

Na comunicação IoT com o Flask-MQTT, a aplicação Flask atua como um cliente MQTT, capaz de se conectar a um broker MQTT e trocar mensagens com dispositivos IoT.

- Publicação de Mensagens

A aplicação Flask utiliza tópicos específicos para publicar mensagens no broker MQTT, solicitando ações ao hardware. Essas mensagens contém comandos de acionamento para os atuadores ou solicitações de informações para os sensores. Por exemplo, a aplicação Flask pode publicar uma mensagem no tópico "atuador/ligar" para solicitar o acionamento de um atuador específico.

- Recebimento de Mensagens

Além da publicação de mensagens, a aplicação Flask também é capaz de receber mensagens do broker MQTT. Essas mensagens podem conter dados provenientes dos sensores, no caso, a distância relativa de um objeto por meio do sensor ultrassônico.

- Integração de Web e Banco de Dados

A integração entre a aplicação Flask, o banco de dados e o hardware proporciona uma solução completa para a comunicação IoT. A aplicação Flask utiliza os modelos SQLAlchemy para inserir, selecionar, editar e excluir os dados do banco de dados.

TDEs

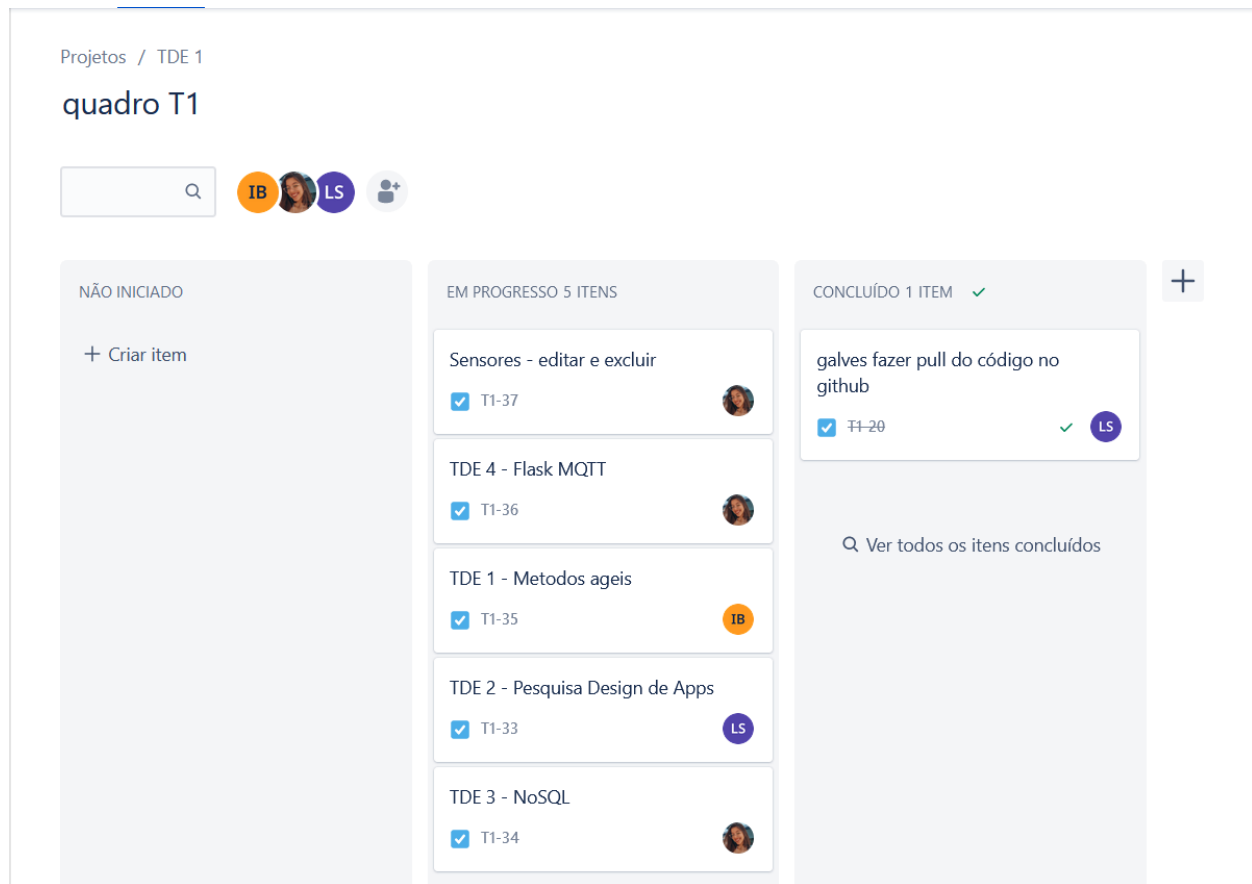
TDE 1 – Metodologias Ágeis

Metodologia Escolhida:

Kanban

O Kanban é um método de gerenciamento de projetos criado no Japão na década de 1940. Seu objetivo é melhorar a eficiência do processo produtivo e reduzir o desperdício de tempo e materiais por meio da organização dos processos de trabalho.

Baseia-se em quatro princípios: processo visual, limitar o trabalho em andamento, priorização e melhoria contínua.



- Reuniões

30/03/2023

Ideia do projeto: Faixa de pedestre com sensor de luz.

Nome do projeto: Lumen Pass.

O Framework utilizado será o **Bootstrap**, pois ele facilita o desenvolvimento do projeto evitando a necessidade de reescrever linhas de código, além de permitir a customização conforme a necessidade da aplicação.

Projeto foi estruturado de forma que a arquitetura mvc fosse evidenciada. Também separamos a aplicação em módulos utilizando o Blueprint, as views foram renderizadas, rotas definidas e css e html organizados.

08/04/2023

Dados a serem armazenados:

- Ação

- Dispositivo
- Read
- Roles
- users
- user roles
- Sensors

Por que?

- Os dados coletados serão utilizados para determinar o melhor tempo de abertura de um semáforo para determinados períodos do dia ou a necessidade de mais faixas de pedestre em ruas próximas para desconcentrar o fluxo de pessoas de apenas uma faixa.

12/04/2023

Finalização do desenvolvimento da aplicação web. Foram ajustados o design do site, logo foi criada, botões organizados e direcionados para cada view. O app foi conectado ao banco de dados.

20/05/2023

Desenvolvimento das modelagens lógica e conceitual da aplicação. Também foram finalizadas as funcionalidades de cadastro, exclusão e edição dos dados pertinentes.

25/05/2023

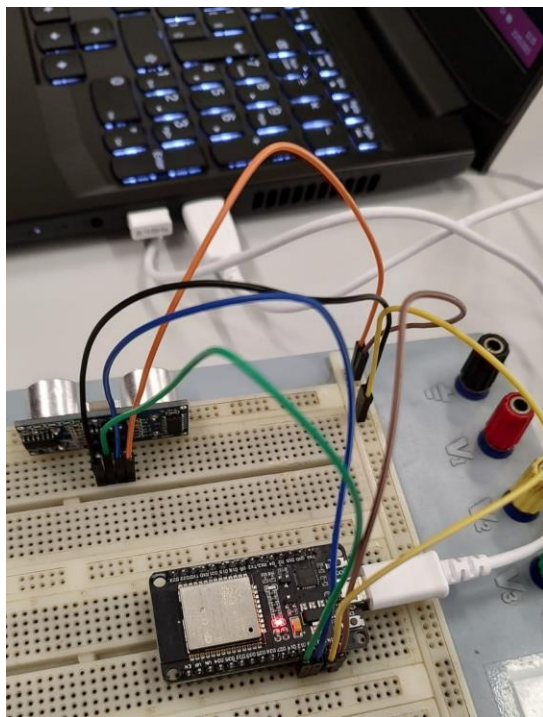
Desenvolvimento do circuito do projeto no simulador Wokwi.

Componentes utilizados:

- ESP32
- Sensor de distância ultrassônico
- LEDs

01/06/2023

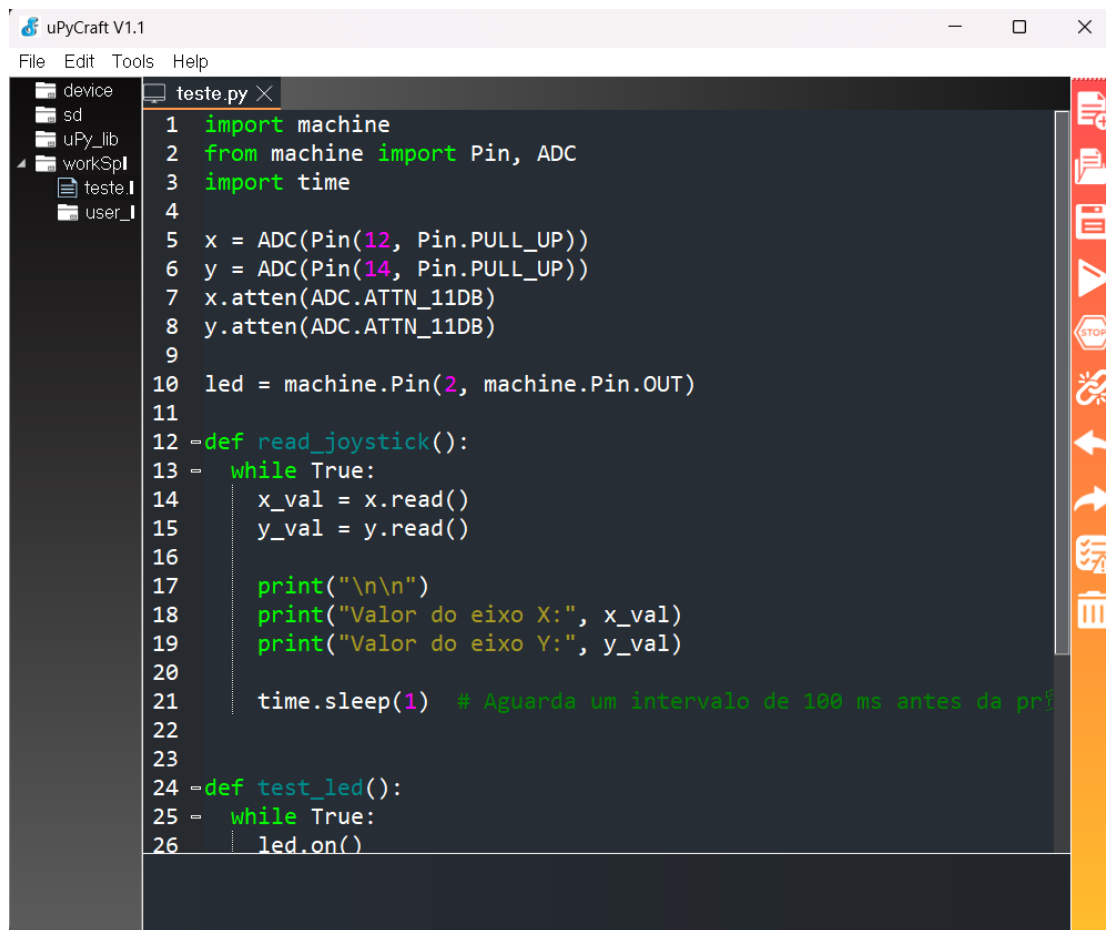
Início do desenvolvimento do circuito do projeto físico (não foi possível completa-lo pela falta de LEDs).



08/06/2023

Fim do desenvolvimento do circuito.

Código em python utilizado:



```
1 import machine
2 from machine import Pin, ADC
3 import time
4
5 x = ADC(Pin(12, Pin.PULL_UP))
6 y = ADC(Pin(14, Pin.PULL_UP))
7 x.atten(ADC.ATTN_11DB)
8 y.atten(ADC.ATTN_11DB)
9
10 led = machine.Pin(2, machine.Pin.OUT)
11
12 def read_joystick():
13     while True:
14         x_val = x.read()
15         y_val = y.read()
16
17         print("\n\n")
18         print("Valor do eixo X:", x_val)
19         print("Valor do eixo Y:", y_val)
20
21         time.sleep(1) # Aguarda um intervalo de 100 ms antes da pr
22
23
24 def test_led():
25     while True:
26         led.on()
```

TDE 2 – Design de Aplicações

- Sobre as escolhas de frameworks em Design no geral:

Os frameworks de design front-end oferecem modelos pré-construídos que permitem aos desenvolvedores criar rapidamente as estruturas básicas de suas interfaces. Esses modelos fornecem uma base sólida para começar e ajudam a manter a consistência visual em todo o aplicativo. Esses frameworks oferecem uma variedade de bibliotecas de componentes reutilizáveis. Essas bibliotecas contêm elementos de interface de usuário, como botões, formulários, barras de navegação e muito mais. Com a reutilização desses componentes, os desenvolvedores podem economizar tempo e esforço na criação de elementos comuns e se concentrar em aspectos mais exclusivos do aplicativo.

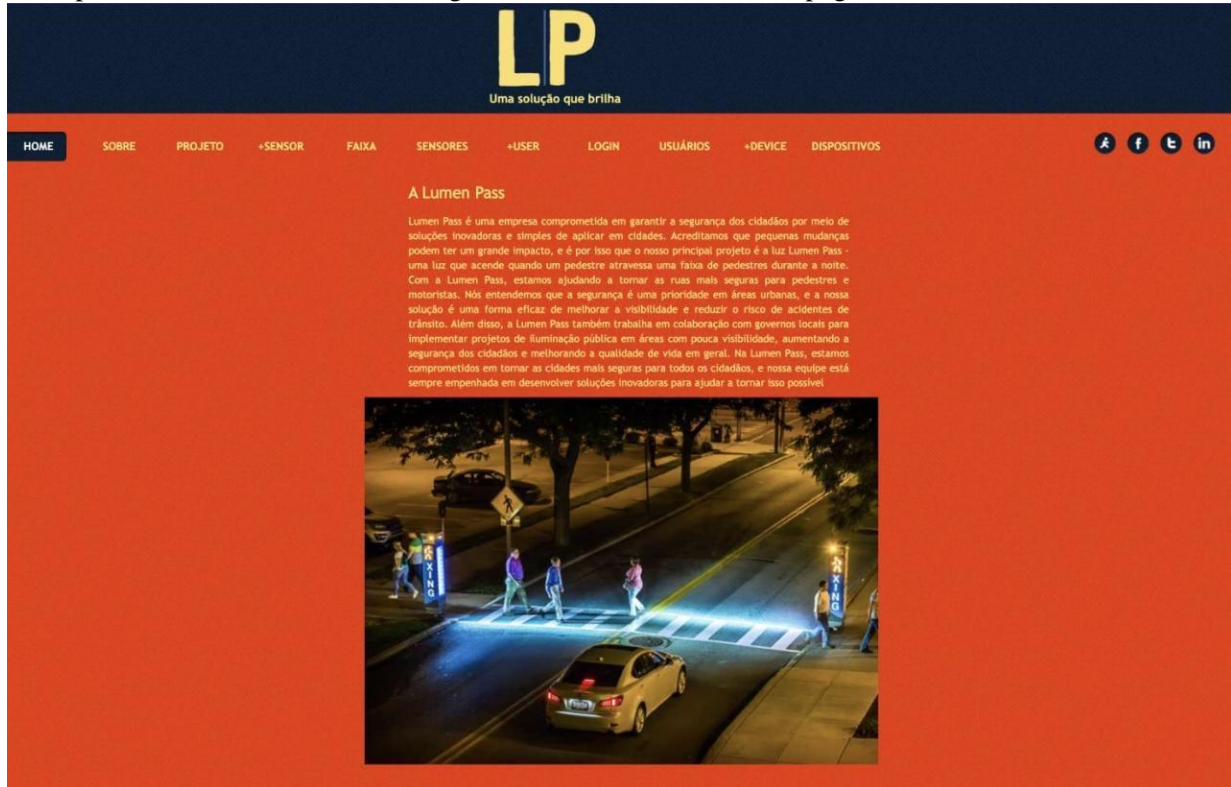
Os frameworks de design front-end são projetados para serem compatíveis com uma ampla gama de navegadores e versões. Isso garante que o aplicativo funcione de forma consistente e tenha uma boa aparência em diferentes plataformas.

- Escolher um framework de design front-end apropriado para o projeto:

Para a utilização do projeto, implementamos o framework Flask do Python. Com ele, conseguimos fazer a conexão entre o HTML, CSS, JavaScript, MYSQL entre outras linguagens e utilidades. O Flask possui um sistema de roteamento que torna possível criar rotas para exibir páginas HTML, processar formulários, lidar com solicitações AJAX e muito mais, chamadas “Templates”. Isso permite separar a lógica de apresentação do código Python, facilitando a criação de páginas dinâmicas. Os templates podem

conter trechos de código Python, estruturas de repetição, condições e placeholders que são preenchidos com dados dinâmicos durante a renderização.

Exemplo de Design (nossa página de Home):



Página de cadastro dos usuários:

[+SENSOR](#)[FAIXA](#)[SENSORES](#)[+USER](#)[LOGIN](#)[USUÁRIOS](#)

CADASTRO

Cadastro do usuário

Nome

Email

lcd@lcd

Senha

....

CPF

Role

Admin ▾

ENVIAR

Página de cadastro dos sensores:

```

1  {% extends "base.html" %}
2  {% block title %} Cadastro {% endblock %}
3  {% block content %}
4
5  <div class="page-wrap">
6  <div class="wrap3">
7      <div class="leftcol">
8          <div class="panel">
9              <div class="title">
10                 <h1>Cadastro de infos dos Sensores: </h1>
11             </div>
12             <div class="panel">
13
14
15
16                 <form action="sensors/registrar_sensor" method="POST">
17                     <div class="contact-form mar-top30">
18                         <label> <span>Nome do Sensor:</span>
19                         <input type="text" class="input_text" name="nome" id="nome"/>
20                         </label>
21                         <label> <span>Descrição: </span>
22                         <input type="text" class="input_text" name="descricao" id="descricao"/>
23
24                         <label> <span>tipo:</span>
25                         <input type="text" class="input_text" name="tipo" id="tipo"/>
26
27                         <!-- <h2>limite de proximidade: </h2> -->
28                         <label> <span>limite de proximidade:</span>
29                         <input type="text" class="input_text" name="limite_proximidade" id="limite_proximidade"/>
30
31
32                         <button type="submit" class="button">Enviar</button>
33                     </div>
34                 </form>
35                 <div class="clearing"></div>
36             </div>
37         </div>
38     </div>
39
40
41     <div class="clearing"></div>
42 </div> |
43 </div><!--page-wrap-End-->
44
45
46
47 {% endblock %}
48 {% block class4 %} class = 'active' {% endblock %}

```


Página

de

cadastro:

The screenshot shows a web application with a dark blue header containing navigation links: HOME, SOBRE, PROJETO, +SENSOR (highlighted), FAIXA, SENSORES, +USER, LOGIN, USUÁRIOS, +DEVICE, and DISPOSITIVOS. Below the header, the page title is 'CADASTRO DE INFOS DOS SENSORES:'. The main content area is white and contains a form with four input fields: 'Nome do Sensor:', 'Descrição:', 'tipo:', and 'limite de proximidade:'. Each field has a light blue border and a light blue placeholder text. Below the fields is a blue button labeled 'ENVIAR'.

Código do flask renderizando as páginas e fazendo as conexões:

```
1 from flask import request, render_template, Blueprint, redirect, url_for, flash
2 from models import *
3 from flask_login import login_user, login_required, logout_user, current_user, login_manager
4 # aqui tudo é renderizado
5 render = Blueprint("render", __name__, template_folder="./views/", static_folder='./static/', root_path=".")
6
7
8 @render.route("/")
9 def default_index():
10     return render_template("home.html")
11
12 @render.route("/home")
13 def ret_home():
14     return render_template("home.html")
15
16 @render.route("/pag_cadastro_sensor")
17 def pag_cad_sens():
18     return render_template("sensors/cadastro_sensor.html")
19
20 @render.route("/pag_cadastro_user")
21 def pag_cad_user():
22     return render_template("user/cadastro_user.html")
23
24 @render.route("/pag_login")
25 def pag_log():
26     return render_template("auth/login.html")
27
28 @render.route("/list_users", methods = ["get"])
29 @login_required
30 def listar_users():
31     id_role = current_user.id_role
32
33     if id_role == 1:
34         users = User.get_users()
35         return render_template("/user/list_users.html", users = users)
36     return redirect(url_for('render.pag_log'))
37
```

TDE 3 - NoSQL para IoT

Introdução

Este relatório apresenta uma pesquisa sobre bancos de dados NoSQL e sua utilização em aplicações de IoT (Internet das Coisas). O objetivo é fornecer uma compreensão abrangente da definição dos bancos de dados NoSQL, suas principais diferenças em relação aos bancos de dados relacionais, vantagens e desvantagens, tipos de bancos de dados NoSQL, bem como a utilização desses bancos de dados em aplicações de IoT. Além disso, serão discutidos os desafios e limitações associados ao uso de bancos de dados NoSQL em aplicações de IoT.

Definição dos bancos de dados NoSQL

Os bancos de dados NoSQL são sistemas de gerenciamento de bancos de dados projetados para lidar com dados não estruturados e sem um esquema fixo. Eles são chamados de "não relacionais" porque não utilizam o modelo de dados relacional tradicional baseado em tabelas, filas e linhas. Em vez disso, eles adotam uma abordagem mais flexível e escalável para armazenar e recuperar dados.

Principais diferenças entre bancos de dados NoSQL e relacionais

Os bancos de dados NoSQL diferem dos bancos de dados relacionais em vários aspectos. Enquanto os bancos de dados relacionais possuem um esquema fixo e estruturado, os bancos de dados NoSQL são flexíveis e permitem a inclusão de novos tipos de dados sem a necessidade de modificar a estrutura existente. Além disso, os bancos de dados NoSQL são projetados para escalabilidade horizontal, podendo lidar com grandes volumes de dados distribuídos em vários servidores. Em contraste, os bancos de dados relacionais têm uma abordagem de escalabilidade vertical, dependendo de hardware mais poderoso. Outra diferença significativa é que os bancos de dados relacionais garantem a consistência imediata dos dados, enquanto os bancos de dados NoSQL podem oferecer diferentes níveis de consistência, priorizando a disponibilidade e a tolerância a falhas.

Vantagens dos bancos de dados NoSQL

Os bancos de dados NoSQL apresentam várias vantagens em relação aos bancos de dados relacionais. Sua flexibilidade permite armazenar e processar diferentes tipos de dados, como documentos, gráficos, colunas e pares chave-valor, adaptando-se a diferentes tipos de aplicativos. Além disso, eles são altamente escaláveis, permitindo o gerenciamento de grandes volumes de dados e cargas de trabalho distribuídas. Os bancos de dados NoSQL também podem fornecer um desempenho superior em determinados cenários, pois não possuem as restrições de integridade referencial presentes nos bancos de dados relacionais.

Desvantagens dos bancos de dados NoSQL

Apesar de suas vantagens, os bancos de dados NoSQL também apresentam algumas desvantagens. Em alguns casos, a consistência eventual pode resultar em atrasos na propagação de alterações entre os nós do sistema, levando a resultados inconsistentes durante operações de leitura. Além disso, os bancos de dados NoSQL podem ter um suporte limitado a consultas complexas e ad-hoc, dificultando a realização de análises complexas nos dados armazenados. O uso de bancos de dados NoSQL também requer uma curva de aprendizado maior para desenvolvedores acostumados aos bancos de dados relacionais tradicionais.

Tipos de bancos de dados NoSQL

Existem diferentes tipos de bancos de dados NoSQL, cada um adequado para diferentes necessidades de aplicativos. Os principais tipos incluem:

Banco de dados de documentos: Esses bancos de dados armazenam dados em formato de documentos, geralmente usando formatos como JSON ou XML. Exemplos populares incluem MongoDB e Couchbase.

Banco de dados de chave-valor: Esses bancos de dados armazenam dados como pares de chave e valor, permitindo uma recuperação rápida dos dados por meio de chaves. Exemplos populares incluem Redis e Riak.

Banco de dados de colunas: Esses bancos de dados armazenam dados em colunas, em vez de linhas, permitindo uma recuperação eficiente de conjuntos específicos de dados. Exemplos populares incluem Cassandra e HBase.

Banco de dados de grafos: Esses bancos de dados armazenam dados em forma de nós e arestas, permitindo a modelagem e consulta de relacionamentos complexos. Exemplos populares incluem Neo4j e Amazon Neptune.

Utilização de bancos de dados NoSQL em aplicações de IoT

Os bancos de dados NoSQL são amplamente utilizados em aplicações de IoT devido às suas características de escalabilidade, flexibilidade e desempenho. As aplicações de IoT lidam com grandes volumes de dados gerados por dispositivos conectados, como sensores e dispositivos inteligentes. Os bancos de dados NoSQL são capazes de lidar com esses dados não estruturados e sem um esquema fixo, permitindo a rápida ingestão, armazenamento e processamento dos dados gerados pela IoT.

Os bancos de dados NoSQL são especialmente adequados para cenários de IoT nos quais os requisitos de escalabilidade e flexibilidade são fundamentais. Eles permitem que os desenvolvedores armazenem e acessem facilmente diferentes tipos de dados gerados pela IoT, como dados de sensores, logs, informações de localização e dados de interação do usuário.

Desafios e limitações na utilização de bancos de dados NoSQL em aplicações de IoT: Embora os bancos de dados NoSQL ofereçam benefícios para aplicações de IoT, também apresentam desafios e limitações específicas. Alguns desses desafios incluem:

Consistência e integridade dos dados: Devido à natureza distribuída dos bancos de dados NoSQL, garantir a consistência e integridade dos dados pode ser um desafio. A replicação e a sincronização de dados entre os nós podem levar a problemas de consistência, exigindo uma cuidadosa modelagem e implementação das aplicações de IoT.

Segurança: O armazenamento e processamento de dados sensíveis em bancos de dados NoSQL podem apresentar desafios de segurança. É necessário adotar medidas adequadas para proteger os dados contra acesso não autorizado e ataques cibernéticos.

Modelagem de dados: A flexibilidade dos bancos de dados NoSQL pode levar a uma maior complexidade na modelagem de dados em comparação com os bancos de dados relacionais. A definição adequada dos esquemas de dados e a escolha do tipo de banco de dados NoSQL mais adequado para o caso de uso de IoT são essenciais para garantir um desempenho eficiente e uma boa escalabilidade.

Conclusão

Os bancos de dados NoSQL oferecem uma alternativa flexível e escalável aos bancos de dados relacionais tradicionais, sendo amplamente utilizados em aplicações de IoT devido à sua capacidade de lidar com grandes volumes de dados não estruturados e distribuídos. No entanto, é importante considerar os desafios e limitações específicos ao utilizar bancos de dados NoSQL em cenários de IoT, como a consistência dos dados, a segurança e a modelagem adequada dos dados. Com uma abordagem adequada, os bancos de dados NoSQL podem ser uma solução eficaz para lidar com as demandas de dados gerados pela IoT.

TDE 4 - Realizando a integração do Hardware, MQTT e Flask

A A integração de hardware, MQTT e Flask na comunicação IoT é uma área de pesquisa em constante crescimento, impulsionada pelo avanço da Internet das Coisas (IoT) e suas aplicações em diversos setores. Nesta pesquisa, exploraremos brevemente os conceitos-chave relacionados a essa integração e destacaremos algumas vantagens e casos de uso dessa abordagem.

- MQTT (Message Queuing Telemetry Transport)

O MQTT é um protocolo de mensagens leve e eficiente projetado para comunicações IoT. Ele utiliza um modelo de publicação/assinatura, no qual os dispositivos enviam mensagens para tópicos específicos e outros dispositivos interessados nesses tópicos podem se inscrever para

receber essas mensagens. O MQTT é amplamente adotado devido à sua baixa sobrecarga de dados e suporte para conexões instáveis, sendo ideal para dispositivos com recursos limitados.

- Flask

Flask é um framework web leve e flexível em Python. Ele oferece recursos para o desenvolvimento de aplicativos web e APIs, permitindo a criação de interfaces de usuário interativas e o processamento de solicitações HTTP. Sua modularidade e simplicidade tornam o Flask uma escolha popular para desenvolvedores.

- Integração de Hardware e MQTT

A integração de hardware com MQTT envolve conectar dispositivos físicos (sensores, atuadores, placas microcontroladoras, etc.) a um broker MQTT, que atua como intermediário para a troca de mensagens. Os dispositivos podem publicar mensagens em tópicos MQTT, transmitindo dados coletados pelos sensores ou solicitações de ações aos atuadores. Da mesma forma, eles podem se inscrever em tópicos MQTT para receber comandos ou informações relevantes.

- Integração de MQTT e Flask

A integração de MQTT e Flask permite que a aplicação Flask atue como um cliente MQTT, recebendo e enviando mensagens por meio do broker MQTT. Essa integração permite que a aplicação Flask processe e armazene os dados provenientes dos dispositivos IoT, bem como publique mensagens para controlar os atuadores. Por exemplo, a aplicação Flask pode receber leituras de sensores MQTT, atualizar um banco de dados e fornecer informações atualizadas aos usuários por meio de uma interface web

Vantagens e Casos de Uso

A integração de hardware, MQTT e Flask traz diversas vantagens, incluindo:

Conectividade eficiente: O uso do protocolo MQTT permite uma comunicação eficiente entre dispositivos IoT e a aplicação Flask, reduzindo a sobrecarga de dados e garantindo uma transferência de informações confiável.

Flexibilidade: A abordagem permite a conexão de uma ampla variedade de dispositivos e sensores, tornando possível criar soluções IoT abrangentes.

Monitoramento e controle em tempo real: A comunicação em tempo real entre os dispositivos IoT e a aplicação Flask permite monitorar e controlar dispositivos remotamente, em tempo real, o que é útil em cenários como automação residencial, gerenciamento de energia e monitoramento ambiental.

Alguns casos de uso práticos incluem:

Monitoramento de qualidade do ar: Sensores de qualidade do ar podem enviar leituras para a aplicação Flask via MQTT, permitindo o monitoramento contínuo da qualidade do ar em uma área específica.

Automação residencial: A integração de dispositivos domésticos (como iluminação, termostatos, fechaduras inteligentes) com MQTT e Flask permite o controle centralizado e inteligente desses dispositivos por meio de uma interface web.

Agricultura de precisão: Sensores agrícolas podem coletar dados sobre umidade do solo, temperatura, umidade, entre outros, que são enviados para a aplicação Flask via MQTT. Esses dados podem ser processados e analisados para otimizar a irrigação e o cultivo de plantações.

- Fotos do projeto em versão beta



- Código mqtt

```
import RPi.GPIO as GPIO
import time
import paho.mqtt.client as mqtt
```

```
# Ultrasonic pin numbers
trigPin1 = 19
```

```
echoPin1 = 21
trigPin2 = 32
echoPin2 = 35

# MQTT broker information
mqttServer = "192.168.1.4"
mqttPort = 1883
clientId = "clientId"

# WiFi information
ssid = "KIMI"
password = "260531942"

# LED pin numbers
led1 = 2
led2 = 4
led3 = 5
led4 = 18

# Initialize GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setup(trigPin1, GPIO.OUT)
GPIO.setup(echoPin1, GPIO.IN)
GPIO.setup(trigPin2, GPIO.OUT)
GPIO.setup(echoPin2, GPIO.IN)
GPIO.setup(led1, GPIO.OUT)
GPIO.setup(led2, GPIO.OUT)
GPIO.setup(led3, GPIO.OUT)
GPIO.setup(led4, GPIO.OUT)

# Create MQTT client
client = mqtt.Client(clientId)

def on_connect(client, userdata, flags, rc):
    print("Connected to MQTT broker")
    client.subscribe("Led/command")

def on_message(client, userdata, msg):
    if msg.topic == "Led/command":
        message = msg.payload.decode()
        if message == "ligar":
            GPIO.output(led1, GPIO.HIGH)
            GPIO.output(led2, GPIO.HIGH)
```

```
        GPIO.output(led3, GPIO.HIGH)
        GPIO.output(led4, GPIO.HIGH)
    elif message == "desligar":
        GPIO.output(led1, GPIO.LOW)
        GPIO.output(led2, GPIO.LOW)
        GPIO.output(led3, GPIO.LOW)
        GPIO.output(led4, GPIO.LOW)
```

```
client.on_connect = on_connect
client.on_message = on_message
```

```
# Connect to WiFi
```

```
def wifiConnect():
```

```
    print("Connecting to WiFi...")
    GPIO.output(led1, GPIO.HIGH)
    GPIO.output(led2, GPIO.HIGH)
    GPIO.output(led3, GPIO.HIGH)
    GPIO.output(led4, GPIO.HIGH)
    time.sleep(1)
    GPIO.output(led1, GPIO.LOW)
    GPIO.output(led2, GPIO.LOW)
    GPIO.output(led3, GPIO.LOW)
    GPIO.output(led4, GPIO.LOW)
```

```
# Connect to MQTT broker
```

```
def mqttConnect():
```

```
    print("Connecting to MQTT broker...")
    while not client.is_connected():
        try:
            client.connect(mqttServer, mqttPort)
            client.loop_start()
        except mqtt.ConnectionError:
            print("MQTT connection failed. Retrying in 5 seconds...")
            time.sleep(5)
```

```
def measure_distance(trigPin, echoPin):
```

```
    GPIO.output(trigPin, GPIO.LOW)
    time.sleep(0.2)
    GPIO.output(trigPin, GPIO.HIGH)
    time.sleep(0.00001)
    GPIO.output(trigPin, GPIO.LOW)
```

```
pulse_start = time.time()
```

```

pulse_end = time.time()

while GPIO.input(echoPin) == 0:
    pulse_start = time.time()

while GPIO.input(echoPin) == 1:
    pulse_end = time.time()

pulse_duration = pulse_end - pulse_start
distance = pulse_duration * 34300 / 2 # Speed of sound = 34300 cm/s
return distance

def main():
    # Connect to WiFi
    wifiConnect()
    print("WiFi connected")
    print("IP address:", GPIO.HIGH)
    print("MAC address:", GPIO.HIGH)

    # Connect to MQTT broker
    mqttConnect()

    try:
        while True:
            # Measure distance from ultrasonic sensors
            distance1 = measure_distance(trigPin1, echoPin1)
            distance2 = measure_distance(trigPin2, echoPin2)

            print("Distancia 1:", distance1, "cm")
            print("Distancia 2:", distance2, "cm")

            # Publish distances to MQTT topics
            client.publish("Ultrasonic/info", str(distance1))
            client.publish("Ultrasonic2/info", str(distance2))

            time.sleep(1)

    except KeyboardInterrupt:
        print("Exiting...")
        client.loop_stop()
        GPIO.cleanup()

if __name__ == '__main__':

```

main()