Checkpoint 2

Grupo 3:

Mateus Zanetti Camargo Penteado - 11219202
Érika Hortência Pereira Cardoso - 10696830

Gitlab:

[ICMC-SSC0952 / 2022 / giotgrad03 · GitLab]

Aplicação

A aplicação foi desenvolvida utilizando o framework React, conforme foi especificado no checkpoint 1, o que já foi desenvolvido no momento é a homepage que o usuário verá ao acessar a aplicação, onde o mesmo navegará para conseguir visualizar as imagens capturadas pelo ESP32 e gerenciá-las.

A homepage pode ser visualizada abaixo:

Broker

O ESP32-CAM permanece em stand-by, conectado à rede WiFi, aguardando sinal do sensor PIR indicando que houve detecção de movimento.

```
//Credenciais de rede vão aqui
const char* ssid = "NOME_DA_SUA_REDE";
const char* senha = "SENHA_DA_SUA_REDE";

WiFi.begin(ssid, senha);
```

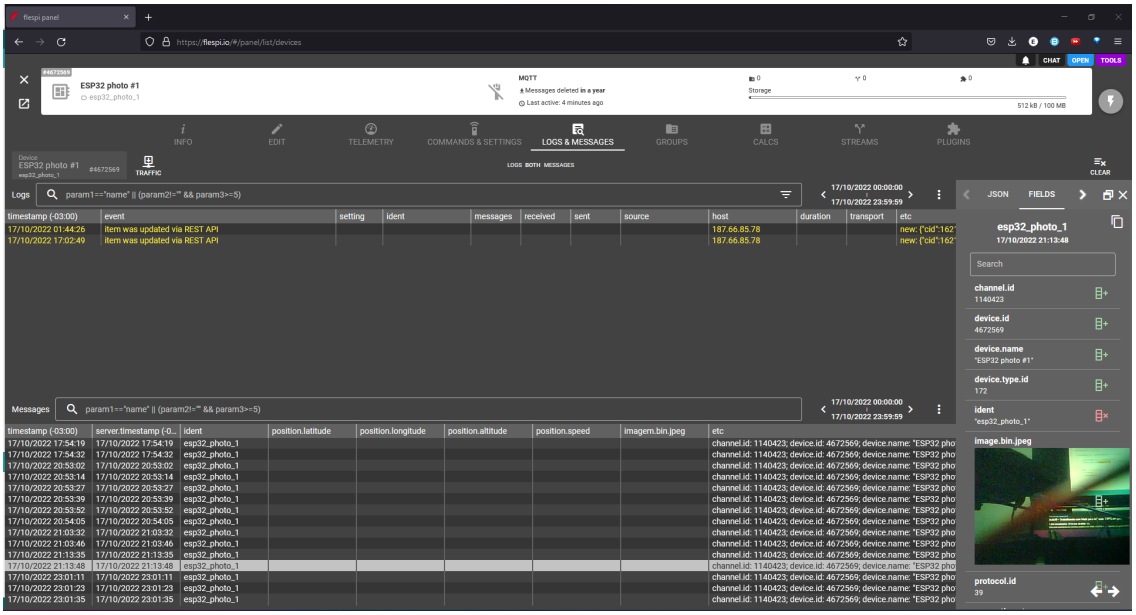
```
//Configuração do broker MQTT
const char* mqtt_servidor = SERVIDOR;
const int mqtt_porta = PORTA;
const char* mqtt_usuario = "TOKEN_MQTT";
const char* mqtt_senha = "SENHA_MQTT";

void reconectar() {
  while (!client.connected()) {
    Serial.print("Conectando ao MQTT ...");
    if (client.connect("ESP32Client", mqtt_usuario, mqtt_senha)) {
      Serial.println("conectado");
    } else {
      Serial.print("falha, rc=");
      Serial.print(client.state());
      Serial.println(" tentando novamente em 5 segundos");
      delay(5000);
    }
  }
}
```

```
esp_sleep_enable_ext0_wakeup(GPIO_NUM_13, 0);
```

Quando isso acontece, a câmera é acionada e um registro fotográfico é feito. O registro é publicado no tópico e pode ser acessado pela aplicação pela API REST.

```
void publicarFoto(String data) {
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  boolean Status = client.publish_P( mqtt_topico, (const uint8_t*)
data.c_str(), data.length(), true);
  Serial.println(String(Status ? "Sucesso!" : "Erro") );
}
```



Armazenamento

O armazenamento foi desenvolvido utilizando MySQL, conforme especificado no checkpoint 1, no momento possuímos no nosso banco de dados uma tabela (“tbl_registers”) que armazena os dados referentes à foto tirada pelo sensor, onde é armazenado um id único, um nome único, a foto em si, o DateTime da foto e o tipo da foto (que futuramente representará se a foto foi acionada pelo sensor ou se o usuário que pediu manualmente para a placa tirar uma foto).
Desta maneira a nossa tabela possui a estrutura conforme o modelo relacional abaixo:

tbl_registers	
int_register_id	integer
chr_register_name	varchar(256)
blob_register_image	MEDIUMBLOB
int_register_type	integer
dte_register_creation_time	timestamp

Micro-Serviço

Os micro-serviços foram desenvolvidos utilizando .NET Core em C#, conforme especificado no checkpoint 1, foi desenvolvido as operações de CRUD (create, read, update, delete) básicas na nossa tabela “tbl_registers” como pode ser observada no Swagger de nossa API, conforme mostra a foto abaixo: