Estudo e desenvolvimento de sistema de monitoramento de alertas IoT

Resumo: Com a adição de dispositivos inteligentes a internet, vê-se também um aumento na quantidade de alertas disparados, porém o formato e forma de transmissão varia de acordo com o dispositivo. Tendo isso em vista implementou-se um sistema de monitoramento para receber alertas, vindos de diversos protocolos de comunicação como LoRa, MQTT e HTTP. O Resultado obtido foi um sistema dividido em 3 camadas diferentes, embarcado, *backend* e *frontend* explicados ao longo do trabalho.

Palavras-chave: Sistemas de Gerenciamento de Eventos e Alertas, Internet das coisas, Redes de computadores, computação em nuvem.

INTRODUÇÃO

Internet of things (IoT), é uma área do conhecimento que tem evoluído rapidamente nos últimos 20 anos (WANG et al., 2021), podendo adicionar a rede bilhões ou trilhões de novos dispositivos nos próximos anos (CHEN et al., 2014).

Visto essa enorme quantidade de dispositivos *IoT* a serem desenvolvidos nos próximos anos, faz-se necessário a implementação de uma plataforma que receba mensagens dos mesmos, processe-as e gere alertas. Observando que tais dispositivos podem utilizar diversos protocolos de comunicação tais como: LoRa, LoRaWAN, MQTT, HTTP, entre outros, é preciso planejar bem a arquitetura do sistema e implementar softwares intermediários, *Middleware*, que saibam como enviar corretamente os dados para a plataforma de alertas que fornecerá uma interface única de comunicação.

Logo o objetivo deste trabalho é estudar e desenvolver uma plataforma para monitorar, processar e gerar notificações a partir de alertas disparados por dispositivos *IoT*, transmitidos utilizando uma rede como LoRa ou internet, sendo que tais alertas podem ser imagens ou texto.

MATERIAL E MÉTODOS

Conforme explicado na Introdução, nesse trabalho será desenvolvido uma plataforma para monitorar, processar e gerar notificações, que vieram de dispositivos *IoT* que se comunicam utilizando diversos protocolos HTTP, MQTT e LoRa.

Para atender a esse objetivo será necessário a implementação de diversos elementos de software e hardware compondo as diversas camadas do sistema, sendo elas:

- **Camada de coleta**: implementar dispositivos capazes de mandar texto ou imagem utilizando protocolos HTTP, MQTT e LoRa.
- **Camada de processamento**: implementar um sistema *backend* que fornece uma API REST tanto para que os dispositivos embarcados como para o frontend.
- **Camada de alerta**: implementar uma aplicação web que mostra notificações geradas a partir dos alertas enviados pelos sistemas embarcados, bem como uma interface no telegram que informa os alertas.

Cada uma dessas camadas possui suas tecnologias específicas que foram utilizadas para o desenvolvimento da aplicação, entre elas se destacam:

- **MQTT**: Message Queuing Telemetry Transport, é um protocolo de camada de aplicação que utiliza o conceito de Publicar/Assinar para trocar mensagens, fazendo o uso de tópicos (MLADENOV et al., 2017).
- **HTTP**: Assim como o MQTT, é um protocolo que atua na última camada do modelo OSI, a camada de aplicação. Projetado para transferência de documentos no formato HTML entre navegadores no lado do cliente e servidores, porém atualmente podendo ser usado para enviar documentos no formato XML, JSON, entre outros. O protocolo segue o formato cliente-servidor onde o cliente faz uma requisição e aguarda a resposta do servidor, não mantendo estado entre as requisições de um mesmo cliente.
- **TypeScript**: é um *superset* de JavaScript, ou seja todo código JavaScript é código Typescript mas nem todo código Typescript é JavaScript (BIERMAN; ABADI; TORGERSEN, 2014) , definido pela Microsoft em 2012 de forma publica no github, com o objetivo de auxiliar os programadores com um sistema de tipos

(BIERMAN; ABADI; TORGERSEN, 2014), não tão estrito quanto linguagens com tipos mais fortes como C++, JAVA e Rust, porém com tipos opcionais e flexíveis, apenas para realizar algumas verificações em tempo de compilação.

- **NestJS**: um *framework* para criação de aplicações escritas em JavaScript ou Typescript que executam em um ambiente NodeJs. Construído em cima de outro *framework* minimalista Express, fornece diversas abstrações de alto nível gerando assim uma melhor experiência para o desenvolvedor. Porém a maior vantagem no uso desse *framework* é a definição de uma arquitetura para desenvolvimento, gerando assim um código mais simples de testar, com menor acoplamento e com maior facilidade de manter.
- **NextJS**: Nextjs é um framework desenvolvido pela vercel, com o objetivo de facilitar alguns pontos do desenvolvimento em react e adicionar funcionalidades prontas para o uso do desenvolvedor (DINKU, 2022).
- **Material UI**: uma biblioteca de componentes gráficos construída em cima do react (NGUYEN, 2022), com o objetivo de prover componentes tais como botões, tabelas, alertas, entre outros, prontos para serem utilizados.

Utilizando-se essas tecnologias, pode-se implementar as 3 partes do sistema, sendo elas sistemas embarcados, *backend* e *frontend*, *explicados na próxima seção.*

RESULTADOS E DISCUSSÃO

Entre os dispositivos embarcados implementados, está um que utiliza um sensor ultrassonico e um ESP32-CAM com o protocolo HTTP para enviar imagens com alertas para API, seguindo o seguinte fluxo:

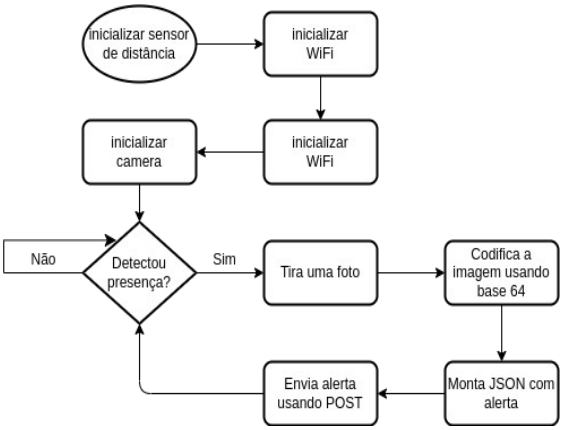

Figura 1. Fluxograma do dispositivo embarcado que envia imagens via HTTP.

Juntando os elementos de *hardware* em uma *protoboard*, teve como resultado o seguinte prototipo:

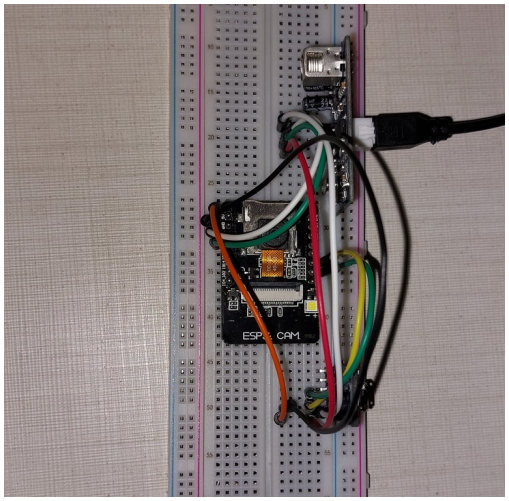

Figura 2. Protótipo utilizando esp32-cam.

Também foram implementados dispositivos embarcados utilizando um esp32 que enviam alertas textuais, sem imagem, para um software intermediário chamado *broker* utilizando MQTT.

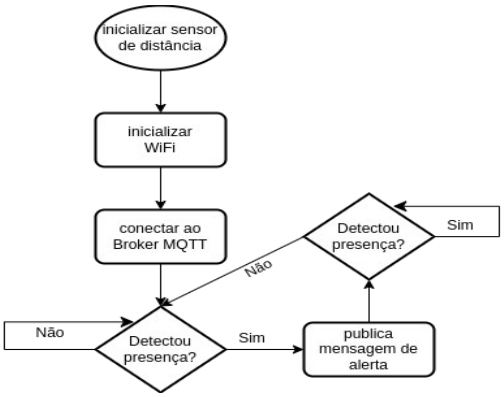

Figura 3. Fluxograma do dispositivo embarcado que envia imagens via MQTT.

E o prototipo montado, ficou:

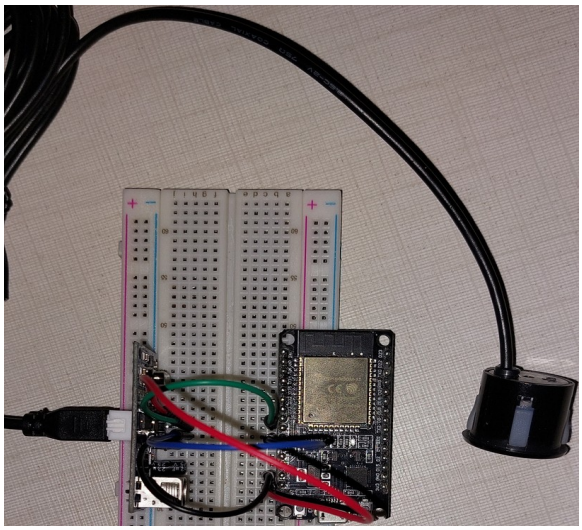

Figura 4. Fluxograma do dispositivo embarcado que envia imagens via MQTT.

Além dos dispositivos embarcados, responsáveis por atender os requisitos da camada de coleta, também foi implementado na camada de processamento uma aplicação *backend*, feita utilizando NestJS, explicado previamente no documento com o objetivo de armazenar e processar os alertas da camada de coleta e enviar alertas para o telegram e o *frontend*.

Por fim na terceira camada, a camada de alerta, foi implementado uma aplicação web utilizando o *framework* NextJS em conjunto com Material UI, possibilitando ao usuário final listar e cadastrar novos dispositivos:

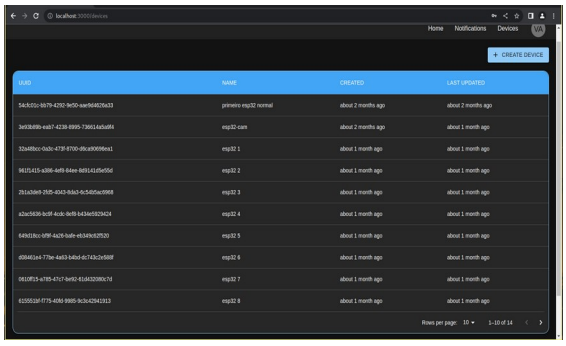

Figura 5. Tela com a listando todos os dispositivos.

E também visualizar de maneira intuitiva as notificações geradas pelos mesmos.

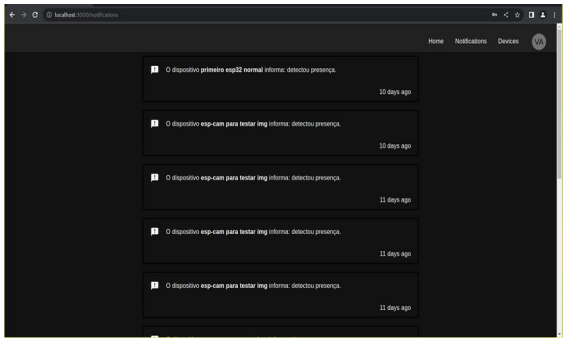

Figura 6. Tela com a lista de notificações.

E ao clicar em uma notificação específica, pode obter mais informações da mesma, tal como o momento em que foi gerada e caso seja um alerta com image, a imagem correspondente:

Figura 7. Tela com a lista de notificações.

CONCLUSÃO

Nesse artigo é apresentado o desenvolvimento de um sistema de monitoramento que recebe alertas de dispositivos *IoT*. Apresentando todas as camadas da aplicação, bem como as tecnologias que dão base a plataforma.

AGRADECIMENTOS

Agradeço a todos que possibilitaram a implementação desse trabalho, permitindo a conclusão do curso.

REFERÊNCIAS

BIERMAN, G.; ABADI, M.; TORGERSEN, M. Understanding typescript. In: SPRINGER. ECOOP 2014–Object-Oriented Programming: 28th European Conference, Uppsala, Sweden, July 28–August 1, 2014. Proceedings 28. [S.l.], 2014. p. 257–281.

CHEN, S. et al. A vision of iot: Applications, challenges, and opportunities with china perspective. IEEE Internet of Things Journal, v. 1, n. 4, p. 349–359, 2014.

DINKU, Z. React. js vs. next. js. 2022.

MLADENOV, K. et al. Formal verification of the implementation of the mqtt protocol in iot devices. SNE Master Research Projects 2016–2017, 2017.

NGUYEN, N. Creating a modern web user interface using react and typescript. 2022.

WANG, J. et al. The evolution of the internet of things (iot) over the past 20 years. Computers & Industrial Engineering, v. 155, p. 107174, 2021.