UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE CENTRO DE TECNOLOGIA

INSTITUTO METRÓPOLE DIGITAL

ENGENHARIA DE COMPUTAÇÃO

Aplicação em Internet das Coisas:

Relatório do projeto Easy Heart

Diêgo Henrique

Natal-RN

2025

Sumário

1. Introdução

Atualmente, as doenças cardíacas representam uma das principais preocupações de saúde pública. A detecção precoce de anomalias nos sinais vitais, como batimentos cardíacos, pode salvar vidas ao permitir intervenções rápidas e precisas. No entanto, os dispositivos de monitoramento disponíveis no mercado são frequentemente caros, possuem funcionalidades limitadas ou são inacessíveis para grande parte da população.

Com isso, o projeto **Easy Heart** surge com a proposta de criar uma solução acessível, integrada e de fácil uso para monitoramento contínuo da saúde cardíaca. Utilizando uma tecnologia mais acessível, como o microcontrolador ESP32 e sensores, o sistema visa oferecer funcionalidades que incluem a medição de frequência cardíaca, saturação de oxigênio no sangue (SpO2) (se tivesse o sensor) e pressão arterial (mesmo caso do anterior). Esses dados são analisados por inteligência artificial e sincronizados com a nuvem por meio de uma API.

O objetivo final é demonstrar o projeto **Easy Heart** pode ser uma solução viável para monitoramento da saúde cardíaca em tempo real e com baixo custo.

2. Descrição do sistema

O sistema **Easy Heart** foi projetado para monitorar sinais vitais de forma contínua e realizar análises automatizadas com suporte de inteligência artificial. Seu principal objetivo é oferecer uma solução portátil, acessível e eficiente, permitindo a detecção de anomalias relacionadas à saúde cardíaca.

Componentes do Sistema:

2.1 Hardware Principal:

ESP32: Microcontrolador responsável pelo processamento dos dados coletados e comunicação via Wi-Fi.

Sensores:

- **MAX30102**: Sensor utilizado para medir a frequência cardíaca.
- **Leds**: Leds para sinalização:
  - Amarelo: Leitura
  - Vermelho: Anomalia

2.2 Software e Processamento:

Banco de Dados Local: Usando o sqlite3 para o armazenamento de dados

API: Desenvolvida em **FastAPI (Python)** para sincronizar os dados com a nuvem e realizar análises. Com os seguintes endpoints:

- **(POST) /analisar**: Recebe os dados do usuário, normaliza os batimentos e realiza a análise com o modelo de IA, retornando um diagnóstico ("normal", "suspeito", "anormal").
- **(GET) /ultimos_5_dados**: Retorna os cinco últimos registros armazenados no banco de dados. Especialmente usado na representação de dados na web.
- **(GET) /ultimo_dado**: Recupera o último registro salvo. Especialmente usando para pegar o status da IA para notificar o usuário em caso de anormalidade.
- **(GET) /dados_por_data**: Consulta dados armazenados em um intervalo de datas. Especialmente usado na representação de dados na web.
- **(GET) /dados_anormais**: Lista registros com diagnósticos classificados como "anormal" ou com perda superior a um limite predefinido

2.2.1 Inteligência Artificial:

O modelo de IA é um **autoencoder**, uma rede neural projetada para aprender uma representação compacta dos dados de entrada e, em seguida, reconstruí-los. Essa arquitetura é útil para detecção de anomalias, pois o erro de reconstrução (perda) pode indicar desvios nos padrões esperados.

A arquitetura é composta por:

Encoder: Camadas densas que reduzem gradualmente a dimensionalidade dos dados, extraindo as características mais relevantes.

Decoder: Camadas densas que expandem os dados compactados pelo encoder, reconstruindo-os no formato original.

2.2.2 Processo de Análise

Entrada: Os dados de batimentos cardíacos consistem em uma lista de 141 valores normalizados no intervalo [0,1]. Valores fora desse intervalo são ajustados para garantir a consistência da entrada no modelo.

Reconstrução: Os dados normalizados passam pelo encoder, que reduz suas dimensões e pelo decoder, que tenta reconstruir os dados originais.

A diferença entre os dados de entrada e os reconstruídos é calculada usando a **perda média absoluta** (**mae**).

2.2.3 Diagnóstico:

Baseado na perda de reconstrução:

- **Normal**: Perda < 0,3.
- **Suspeito**: Perda entre 0,3 e 0,4.
- **Anormal**: Perda > 0,4.

O diagnóstico, juntamente com o nível de risco (baixo, médio ou alto), é retornado ao usuário.

2.2.4 Treinamento do Modelo

O modelo foi treinado em um conjunto de dados obtido da plataforma **Kaggle**, utilizando dados de ECGs normalizados. Após o treino salvei os pesos para a implementação.

2.3 Interface Web:

A interface web foi desenvolvida utilizando **Streamlit**, uma biblioteca Python que permite a criação de aplicações interativas e intuitivas. Com ele é possível implementar aplicações simples além disso posso facilitar o acesso aos dados coletados pelo sistema e apresentá-los para análise detalhada das informações.

2.3.1 Visualização dos Últimos Registros:

Exibe os cinco últimos registros armazenados no banco de dados. Além de permitir selecionar um registro específico para visualização de gráficos e métricas detalhadas.

2.3.2 Busca por Intervalo de Datas:

O usuário pode realizar buscas personalizadas especificando um intervalo de datas. Exibe os resultados em uma tabela com destaque para os valores de erro máximo e mínimo.

2.3.3 Exibição de Registros Anormais:

Filtra registros classificados pela IA como "anormais" ou com erro elevado. Apresenta uma visão geral e permite a análise detalhada de cada registro.

2.3.4 Tabela Resumida:

Apresenta os dados em uma tabela formatada com as colunas: ID, Usuário, SpO2, Pressão, Status Local, Diagnóstico IA, Erro (Perda), Data e Hora.

2.3.5 Análise Detalhada:

Apresento o gráfico de linha com os batimentos reais além da Comparação entre os batimentos reais e a reconstrução realizada pela IA.

2.3.6 Bibliotecas usadas:

- **Streamlit**: Criação de uma interface leve e responsiva.
- **Matplotlib**: Geração de gráficos para visualização de batimentos cardíacos e reconstruções.
- **Pandas**: Manipulação e exibição de dados em formato tabular.
- **NumPy**: Processamento dos batimentos cardíacos e cálculos de erro.

3. Requisitos Funcionais e Não Funcionais

3.1 Requisitos Funcionais:

3.1.1 Leitura de Dados do Sensor: O sistema deve ser capaz de ler os valores do sensor de batimentos cardíacos.

3.1.2 Cálculo da Média de Batimentos: O sistema deve calcular a média dos batimentos lidos a partir de 141 amostras e converter esse valor para BPM (batimentos por minuto) apesar de ter enfrentado grande problemas em calcular o bpm.

3.1.3 Análise do Status do Batimento: O sistema deve classificar localmente o status dos batimentos como "Bat. Baixo", "Bat. Alto" ou "Estável", dependendo do valor do BPM calculado.

3.1.4 Envio de Dados para a API: O sistema deve enviar os dados do sensor, incluindo os batimentos, para a API configurada, através de uma requisição HTTP POST.

3.1.5 Verificação do Status na API: O sistema deve verificar periodicamente o último status enviado para a API (via requisição HTTP GET) e atualizar os LEDs de acordo com o diagnóstico retornado.

3.1.6 Controle de LEDs: O sistema deve acionar LEDs para indicar se está lendo os dados do sensor e se os batimentos são anormais.

3.2 Requisitos Não Funcionais:

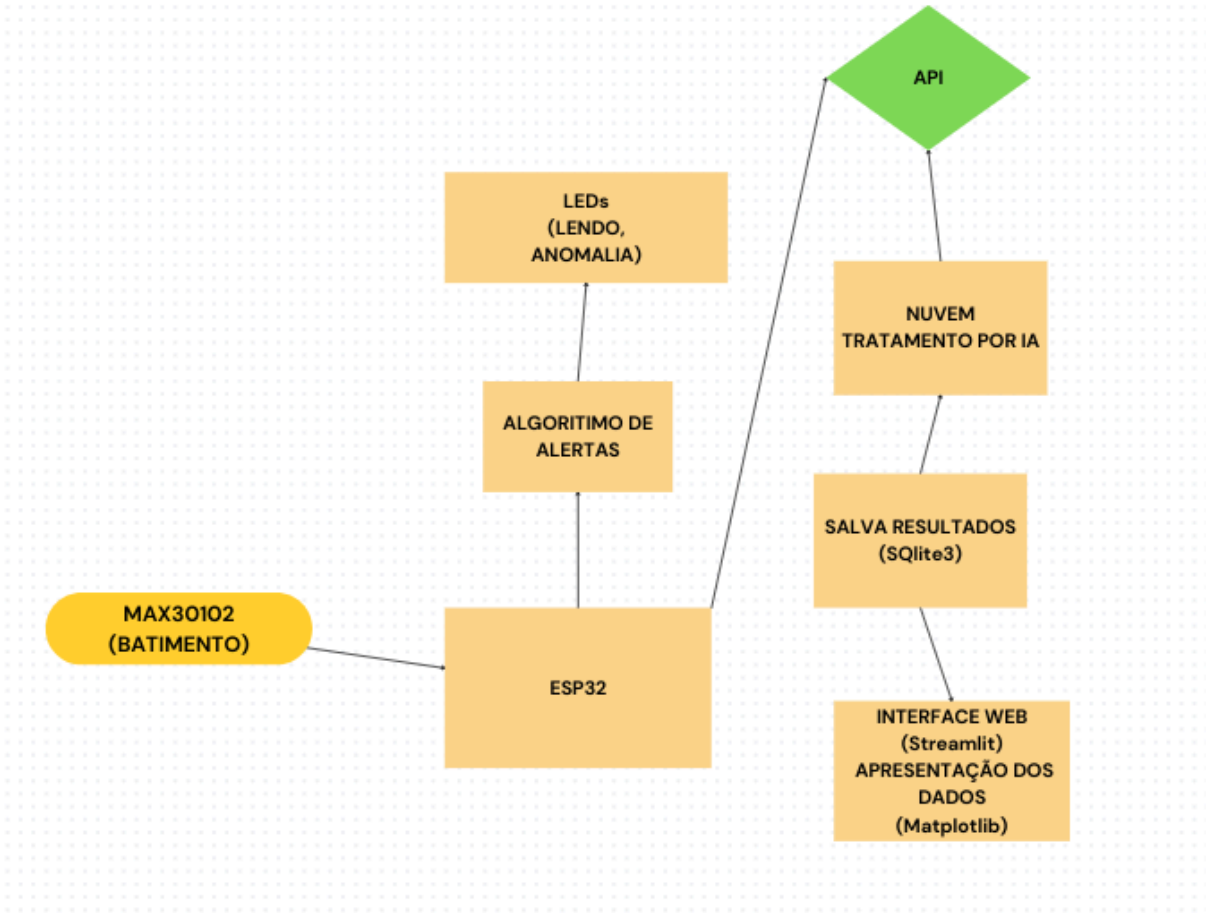
3.2.1 Conectividade Wi-Fi: O sistema deve ser capaz de se conectar a uma rede Wi-Fi específica para o envio de dados API.

3.2.2 Desempenho e Tempo de Resposta: O sistema deve ser capaz de ler e processar os dados do sensor em tempo real, com intervalos curtos entre leituras (definido por DELAY_BETWEEN_READINGS).

3.2.3 Segurança nas Transmissões: Embora o código atual utilize HTTP sem criptografia, um requisito futuro seria garantir que a comunicação com a API seja realizada via HTTPS, garantindo a segurança dos dados transmitidos.

3.2.4 Facilidade de Manutenção: O código deve ser modular, facilitando modificações e manutenção, como a adição de novos sensores ou ajustes nos cálculos.

4. Modelagem



O diagrama do código teve alterações por conta de limitações de hardware, mas está de forma modular então se disponível é possível implementar as propostas discutidas no planejamento do projeto. O Fluxo segue partindo do ESP fazendo a leitura do sensor é acionado o led de leitura, quando temos as 141 amostras enviamos via api para nuvem onde antes de armazenar tratamos os dados por uma IA. Ela fará a reconstrução do ECG e retornando um valor de perda onde com base na perda podemos inferir a anormalidade no ECG. Os dados serão armazenados via sqlite3 e podem ser analisados via web com uma aplicação via streamlit.

5. Implementação

Na implementação, tentei seguir o mais próximo possível do proposto. Comecei pela API, criando os endpoints para o envio de dados e o resgate do último dado. Em seguida, comecei a trabalhar na implementação e nos testes com o sensor (o que demorou bastante, pois nenhum cabo que eu tinha funcionou com o ESP).

Enquanto enfrentava problemas com o ESP, inseri alguns dados manualmente no banco de dados e comecei a implementar a parte da web. Nesse momento, percebi que a IA implementada na API estava retornando valores imprecisos. Foi então que notei que ela estava mal treinada (erro meu, é claro). Com isso, ajustei alguns parâmetros e treinei novamente com 200 épocas, o que resultou em uma acurácia de 96%.

Agora, com a IA funcionando de forma mais precisa, implementei novos endpoints para o retorno de dados voltados ao uso na web. Quando finalmente consegui conectar os ESPs, enfrentei outro problema: a leitura dos batimentos cardíacos não estava sendo realizada. Esse problema só foi resolvido após trocar o pino utilizado.

Depois disso, tentei implementar o status local, mas ele não está retornando o valor esperado (pois preciso dos batimentos por minuto - BPM). No meu caso, como sei que estou detectando um batimento? O sensor que estou utilizando detecta a variação de luz, o que possibilita inferir um gráfico de variações, mas acredito que não seja possível detectar o pulso diretamente com este tipo de sensor.

Implementei os alertas, mas notei algo inconveniente: como a leitura dos dados é feita por um sensor de luz, ele está sempre enviando dados, mesmo que não haja ninguém utilizando o dispositivo, se ele estiver para ele detecta pequenas variações de luz que quando normalizador pode ser bem próximos de um ecg (pois na frequência da coleta geralmente detecta essas variações como pequenos pulsos).

Por fim, depois da implementação da api, do ESP e da interface web foi possível fazer testes e mostrar que ele é eficiente para detectar anomalias mas ainda precisa de refinamentos.

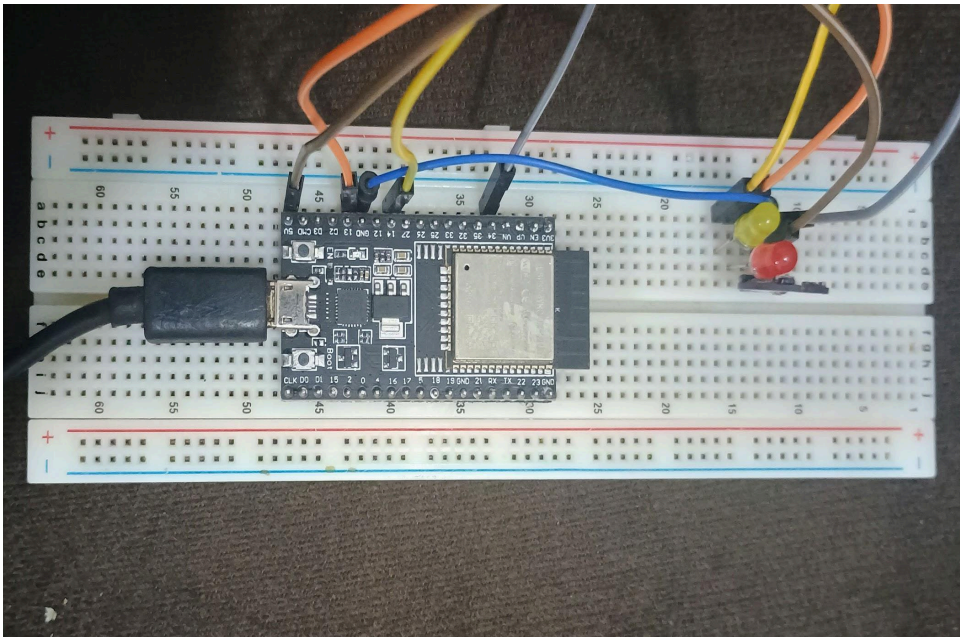


Imagem: Implementação do projeto

6. Conclusão

Apesar das dificuldades, o projeto apresenta potencial, especialmente na integração de monitoramento contínuo de batimentos cardíacos. Contudo, é preciso torná-lo mais robusto e funcional, sendo necessário aprimorar a precisão do sensor e implementar alguma maneira que reduza falsos positivos nos dados gerados. Mesmo assim, é uma base promissora para monitoramento cardíaco acessível.

7. Referências

VIA, Carol Correia. Como utilizar o sensor de batimento cardíaco/monitor de pulso com Arduino. **Blog da Robótica**, 21 ago. 2023. Disponível em: https://www.blogdarobotica.com/2023/08/21/como-utilizar-o-sensor-de-batimento-cardiaco-monitor-de-pulso-com-arduino/. Acesso em: 22 jan. 2025.

NIGAM, Vijeet. Anomaly detection: ECG | Autoencoders. **Kaggle**, 2023. Disponível em: https://www.kaggle.com/code/vijeetnigam26/anomaly-detection-ecg-autoencoders. Acesso em: 22 jan. 2025.