**Discente**: Pedro Henrique Pires

**Disciplina**: Projetos de Arquitetura de Software

**Professor**: Jacson Rodrigues Barbosa

**Curso**: Engenharia de Software

# Detecção e Classificação de Emoções em Falas de Vídeo

## Introdução

As emoções podem afetar consideravelmente o tom de voz de uma pessoa. Quando estamos emocionados, o nosso tom de voz pode variar em diversos aspectos como volume, velocidade, altura e intensidade da voz.

Por exemplo, a raiva pode fazer a voz soar mais alta, ríspida e rápida. A tristeza pode deixar a voz mais baixa, arrastada e monótona. A alegria geralmente faz a voz soar mais animada, com entonação mais alta e ritmada. O medo pode resultar em uma voz tensa, trêmula ou mais baixa.

A voz humana não é apenas uma ferramenta de comunicação linguística, mas também um canal importante para expressar emoções. Segundo um estudo de Scherer et al. (2001), as emoções podem modificar características acústicas da voz, como frequência fundamental (pitch), intensidade e ritmo.

Pesquisas também indicam que a regulação emocional pode afetar o controle motor da fala, influenciando a produção vocal. Em uma pesquisa de Tracy & Matsumoto (2008), foi observado que pessoas comumente ajustam sua produção vocal de acordo com as normas sociais e emocionais. Quando uma pessoa experimenta emoções intensas, como raiva ou felicidade, sua capacidade de controlar o tom e o ritmo da voz pode ser prejudicada, o que resulta em variações audíveis no discurso.

## Problema Proposto

Considerando o contexto acima, o problema que a solução busca resolver é a criação de um software que, por meio do uso de ferramentas de Inteligência Artificial, seja capaz de receber mídias de áudio e vídeo e, baseando-se nas vozes do interlocutor, analise suas emoções.

## Solução Proposta

O software será desenvolvido em FastAPI, utilizando a linguagem de programação Python.

Esse software poderá receber 3 tipos de entradas:

- Áudio gravado pelo usuário diretamente no site
- Arquivo em áudio upado pelo usuário
- Arquivo em vídeo upado pelo usuário

Será utilizada a biblioteca **Librosa**, especialmente útil para extrair características de som como mel-frequency cepstral coefficients (**MFCCs**), comumente usados na análise de emoções. Dependendo do desempenho, podemos utilizar também a biblioteca **OpenSMILE** para a mesma finalidade.

Para aprendizado de máquina, há expectativa de utilizar as bibliotecas **TensorFlow** ou **Keras.** Contudo, dependendo do nível de complexidade da implementação, talvez utilizemos bibliotecas como **pyAudioAnalysis** ou **Vokaturi**, que oferecem implementações de modelos de detecção de emoções já treinados.

Após análise do material disponibilizado pelo usuário, aparecerá um prompt na tela descrevendo as emoções identificadas no arquivo.

# Modelagem Arquitetural Adotada

A abordagem utilizada na documentação será o Modelo C4, criado por Simon Brown. O nome C4 vem das 4 camadas de diagrama que ele propõe.

Esta abordagem possui alguns benefícios como a **simplicidade**, evitando diagramas excessivamente complexos, **hierarquia clara**, propiciando à diversos atores envolvidos a visão adequada do sistema, **padronização** que auxilia a comunicação entre equipes e a **independência de ferramentas**, podendo utilizar UMLs, diagramas desenhados a mão, Structurizr, etc.

## Diagrama de Contexto

Mostra uma visão de alto nível do sistema, incluindo usuários e sistemas externos com os quais interage.

## Diagrama de Contêiner

Divide o sistemas em contêineres e demonstra suas interações

## Diagrama de Componente

Detalha os componentes dentro de um contêiner e suas interações

## Diagrama de Código

**opcional**

Mostra detalhes da implementação no nível de código. Classes, métodos, etc.

# Trabalhos relacionados

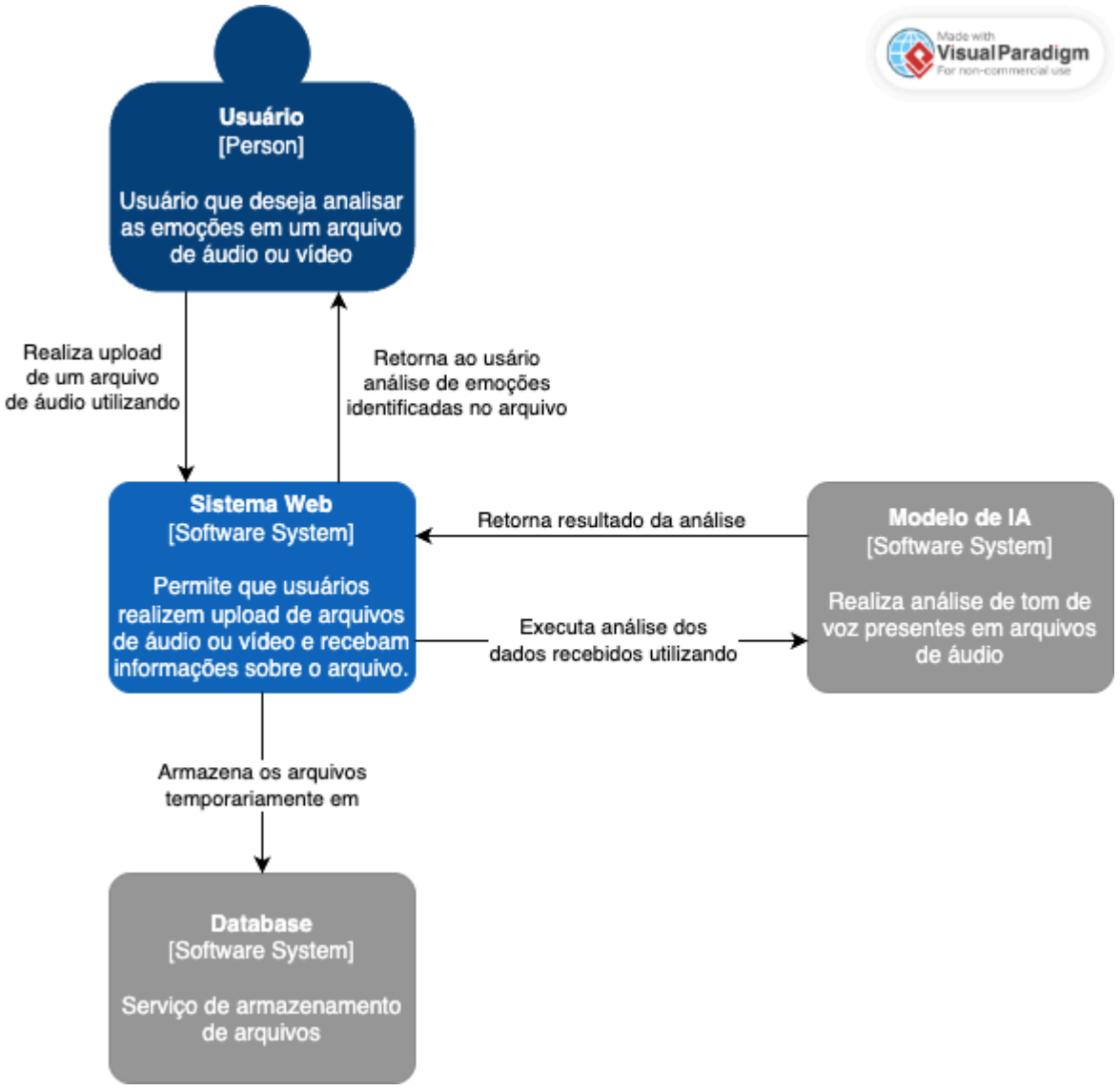
# Diagrama de Contexto

## Descrição

O sistema recebe um áudio ou vídeo de um usuário, processa o som para extrair características relevantes usando **librosa** e realiza a análise emocional.

## Principais atores envolvidos

**Usuário:** Envia arquivos de áudio ou vídeo;
**Sistema de Análise de Emoções (FastAPI):** Processa o arquivo e extrai emoções;
**Serviço de Armazenamento:** Para salvar os arquivos temporariamente. Obs.: Esta implementação é opcional pois os arquivos podem ser escritos em tempo de execução na memória da aplicação e descartado após retorno da análise;
**Modelo de IA:** Analisa o tom de voz e retorna emoções detectadas.



# Diagrama de Conteiner

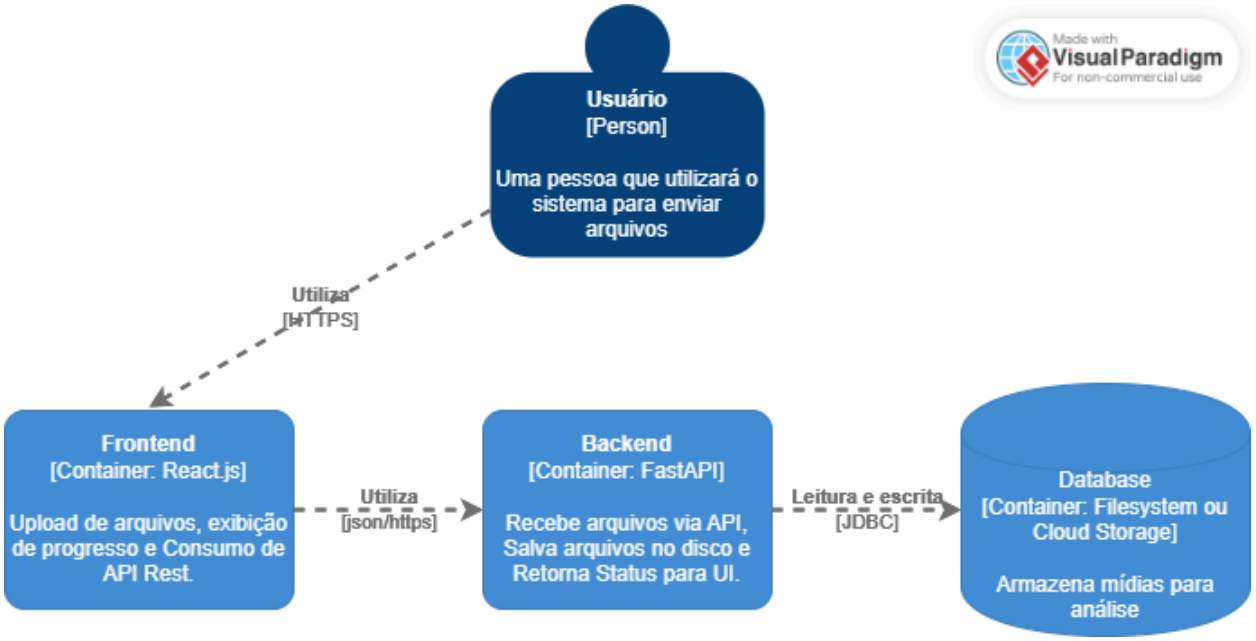
## Componentes do Sistema

A aplicação tem **três grandes blocos** principais:

1. **Frontend (Next.js - React)**
   - Interface gráfica para upload de arquivos.
   - Comunicação com o backend via API REST.
   - Exibição do status do upload (barra de progresso).
2. **Backend (FastAPI - Python)**
   - Processa e valida os arquivos recebidos.
   - Armazena os arquivos no diretório `uploads/`.
   - Retorna respostas HTTP para o frontend.
3. **Armazenamento (Sistema de Arquivos - Local ou S3)**
   - Os arquivos enviados são salvos localmente na pasta `uploads/`.
   - Opcionalmente, pode ser expandido para armazenar os arquivos em um **banco de dados ou serviço de armazenamento na nuvem (AWS S3, Google Cloud Storage, etc.)**.

## Descrição do Diagrama de Contêiner

O diagrama contém **quatro elementos principais**:
1️⃣ **Usuário** → Acessa o frontend para fazer o upload do arquivo.
2️⃣ **Frontend (Next.js - React)** → Interface gráfica que interage com o backend via API REST.
3️⃣ **Backend (FastAPI - Python)** → API que processa o upload e salva os arquivos.
4️⃣ **Armazenamento (Sistema de Arquivos / Cloud Storage)** → Onde os arquivos são armazenados.



## Como esses componentes interagem?

1️⃣ O **Usuário** acessa o frontend (**Next.js**) e faz upload de um arquivo.
2️⃣ O frontend chama a API REST do **backend (FastAPI)** via um `POST /upload/`.
3️⃣ O backend recebe o arquivo, processa e o **salva no sistema de arquivos (uploads/)**.
4️⃣ O backend responde ao frontend, informando o **progresso do upload e status da operação**.

## Benefícios desse modelo

✔ **Separa responsabilidades** → Frontend, Backend e Armazenamento trabalham independentemente.
✔ **Fácil escalabilidade** → Podemos expandir a arquitetura, trocando o sistema de arquivos por um **banco de dados** ou **armazenamento em nuvem**.
✔ **Manutenção simplificada** → O frontend pode evoluir sem impactar diretamente o backend, e vice-versa.