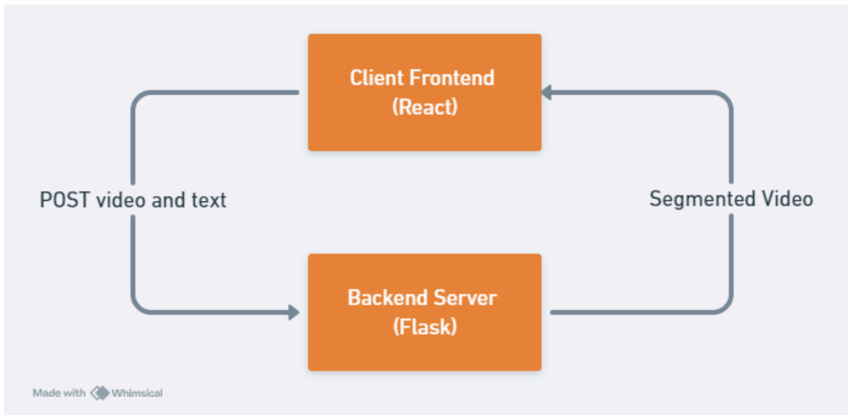
System Architecture Document

1. High-Level Architecture

The system follows a client-server architecture, ensuring clear separation of concerns between the frontend and the backend. Here's a detailed breakdown:

- Client (ReactJS):
  - The client is responsible for user interactions. This includes accepting user inputs such as a video file and a corresponding text instruction.
  - It sends these inputs to the backend via a POST request. After receiving the processed segmented video, it displays the video output to the user.
- Server (Flask):
  - The backend is implemented using Flask, a lightweight Python web framework. The server handles the core business logic of video object segmentation.
  - The backend runs the pre-trained Swin UNetR model to segment the objects from the video based on the user's input (both video and text).
  - The server sends the segmented video back to the client after processing.

Flowchart of Client-Server Interaction:



2. Frontend-Backend Interaction

The interaction between the frontend and backend is simplified through a REST API endpoint that facilitates video segmentation. Here's a breakdown of the process:

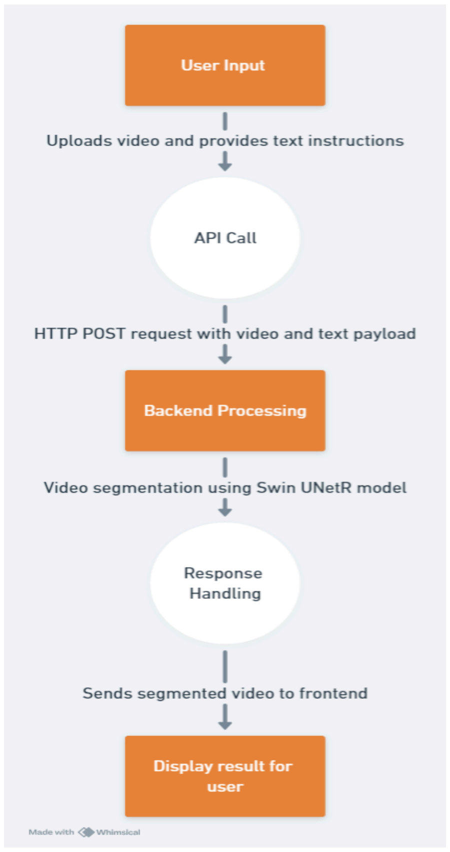
- API Endpoint:
  - URL: /api/segment-video
  - Method: POST
- Payload:
  - video: A video file (in MP4 format, with a maximum size of 100MB).

    - text: The description that guides the segmentation task.
- Response:
    - The backend processes the video and returns a segmented video file (also in MP4 format).

Detailed Flow:

1. User Interaction:
    - The user uploads a video and provides text instructions on the frontend interface (React).
2. API Call:
    - The frontend sends an HTTP POST request to the backend's /api/segment-video endpoint, carrying the video and text as part of the payload.
3. Backend Processing:
    - The backend processes the request, performs video segmentation using the Swin UNetR model, and returns the result.
4. Response Handling:
    - Once the segmented video is ready, the backend sends it back to the frontend. The frontend displays the result for the user.

API Interaction Flow:

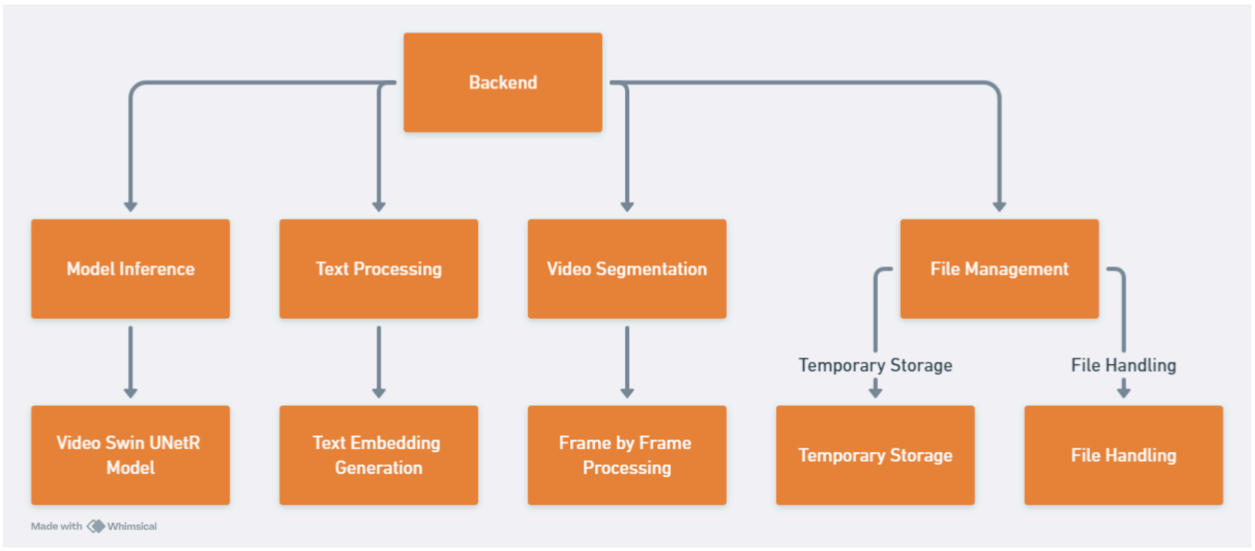

User Input
Uploads video and provides text instructions
API Call
HTTP POST request with video and text payload
Backend Processing
Video segmentation using Swin UNetR model
Response Handling
Sends segmented video to frontend
Display result for user
Made with Whimsical

3. Backend Subsystems

The backend is composed of several distinct subsystems, each serving a crucial role in video segmentation. These include model inference, text processing, video segmentation, and file management.

- Model Inference:
    - Video Swin UNetR Model: The system loads a pre-trained Swin UNetR model to perform video object segmentation. The model is loaded once during startup and is reused across requests to optimize performance.
    - This subsystem accepts both video frames and text embeddings as input and outputs a segmented video.
- Text Processing:
    - The text provided by the user is processed to extract meaningful embeddings, which help guide the segmentation task. Techniques such as tokenization or embedding generation are used to represent the text in a numerical format that the model can use.
- Video Segmentation:
    - The video is processed frame by frame. For each frame, the Swin UNetR model predicts which parts of the frame correspond to the object described by the user. The segmented portions of the video are then stitched back together into the final output.
- File Management:
    - Temporary Storage: During the processing phase, the video file is stored temporarily in the server's file system or memory to facilitate fast access.
    - File Handling: Once the segmentation is complete, the backend sends the segmented video file as part of an HTTP response to the frontend.

Backend Subsystems Flow:



4. Deployment Architecture

The deployment of the system leverages modern practices like containerization, reverse-proxying, and distributed hosting to ensure scalability and reliability.

- Frontend:
    - The React frontend can be hosted on services like Vercel or Netlify, which are optimized for serving static assets and providing high availability.
- Backend:
    - The Flask backend is deployed using Gunicorn, a Python WSGI HTTP Server that is capable of handling multiple requests simultaneously. It is further reverse-proxied by Nginx to ensure efficient load balancing and security.
    - Nginx serves as the gateway for incoming HTTP requests, routing them to the appropriate Gunicorn server running the Flask application.
- Containers:
    - The entire application is containerized using Docker, ensuring consistent environments across development, testing, and production. Docker simplifies deployment by packaging the application along with all its dependencies.
    - Docker Compose is used to orchestrate both the frontend and backend services, simplifying the deployment process.

Explanation of Key Concepts:

- ReactJS Frontend: This handles all user-facing interactions. It is responsible for sending the user inputs (video and text) to the backend and receiving the segmented video.
- Flask Backend: Manages the machine learning model, processes the video and text inputs, and sends the segmented video back to the frontend.
- Swin UNetR Model: This model performs the actual segmentation task. It takes the video and text as inputs and outputs the segmented video.
- Deployment: The use of Docker, Nginx, and Gunicorn ensures that the system can handle multiple concurrent users and can be deployed in different environments with minimal changes to the codebase.

This document provides a detailed overview of the system's architecture and its components. The flowcharts help visualize how different parts of the system interact, making it easier to understand the flow of data and control within the application.