CLUESO TECHNICAL DOCUMENTATION

System Overview

Clueso is an AI-powered screen recording platform that transforms raw screen recordings into professional product demos with AI-generated narration. The system captures screen video, audio, and DOM events, then uses multiple AI models to generate polished narration and insights.

Architecture Summary

The system consists of four primary layers working together:

1. **Chrome Extension** - Captures screen video, microphone audio, and DOM events
2. **Node.js Backend** - Orchestrates data flow, transcription, and storage
3. **Python AI Layer** - Generates professional scripts and audio narration
4. **Next.js Frontend** - Provides user interface and real-time updates

Technology Stack

Frontend Layer

- Next.js 16
- React 19
- TypeScript
- TailwindCSS
- Clerk Authentication
- Socket.IO Client

Backend Layer

- Node.js
- Express 5
- Socket.IO Server
- Multer (file handling)
- Deepgram SDK

AI Processing Layer

- Python
- FastAPI
- Google Gemini 2.5 (script generation)
- Deepgram TTS (text-to-speech)

Data Layer

- Supabase (PostgreSQL database)

- Supabase Storage (media files)

Chrome Extension

- Chrome Manifest V3
- React
- Vite
- MediaRecorder API

AI Models

- Google Gemini - Script generation
- Deepgram - Speech-to-text and text-to-speech
- NVIDIA Qwen - Insight generation

System Architecture

Layer 1: Chrome Extension

Responsibilities:

- Capture screen video stream using MediaRecorder API
- Capture microphone audio stream
- Track DOM events (clicks, typing, scrolls, focus)
- Stream data chunks to backend in real-time
- Manage recording state and user interface

Data Captured:

- Screen video (WebM format)
- Microphone audio (WebM format)
- DOM event metadata including element details (tag, text, selector, position)
- Timing information for all events

Communication:

- Streams chunks to Node.js backend via HTTP POST
- Maintains connection during active recording
- Sends final data package on recording stop

Layer 2: Node.js Backend

Responsibilities:

- Receive video, audio, and event streams from extension
- Upload media files to Supabase Storage

- Transcribe audio using Deepgram API
- Orchestrate AI processing workflow
- Broadcast updates via WebSocket
- Generate AI insights with NVIDIA Qwen
- Manage database operations

API Endpoints:

- Media upload endpoints
- Recording metadata endpoints
- Authentication verification
- Insight generation endpoints

Processing Flow:

1. Receive streamed data from extension
2. Upload raw video and audio to Supabase Storage
3. Broadcast video URL immediately (user can watch)
4. Send audio to Deepgram for transcription
5. Forward transcript, timings, and DOM events to Python AI layer
6. Receive processed audio from Python layer
7. Upload processed audio to Supabase Storage
8. Broadcast processed audio URL via WebSocket
9. Generate insights using NVIDIA Qwen model
10. Store insights in database

WebSocket Events:

- register - Client registers for session updates
- video - Screen recording delivered
- audio - AI-generated narration delivered
- instructions - DOM events (future feature)
- error - Error notifications

Layer 3: Python AI Layer

Responsibilities:

- RAG-based script generation using multiple data sources
- Timing analysis of speech patterns

- DOM event interpretation
- Professional narration generation
- Text-to-speech conversion

FastAPI Endpoints:

- Script generation endpoint
- Audio processing endpoint
- Health check endpoint

AI Processing Pipeline:

Step 1: Data Analysis

- Parse raw transcript with word-level timings
- Identify gaps greater than 200ms between words
- Detect speaking segments versus silence periods
- Calculate speaking rate and pacing metrics

Step 2: Context Building (RAG)

- Analyze DOM event timeline
- Group events into logical steps
- Extract UI element information
- Build comprehensive context of user actions

Step 3: Script Generation

- Send transcript, timings, and DOM events to Gemini AI
- AI removes filler words (um, uh, like, so)
- AI references actual UI elements from DOM events
- AI structures narration around natural pauses
- AI maintains professional tone and clarity

Step 4: Audio Synthesis

- Convert polished script to speech using Deepgram TTS
- Generate natural-sounding narration
- Return audio file to Node.js backend

Transformation Example:

Input Data:

- Transcript: "um so I'm gonna click this button here"

- DOM Event: Click on button with text "Submit" at timestamp T
- Timing: 200ms pause after "here"

Output:

- Script: "I'm clicking the Submit button to save the form data"
- Audio: Natural speech synthesized from script

Layer 4: Next.js Frontend

Responsibilities:

- User authentication and session management
- Recording dashboard and video library
- Video playback interface
- Real-time WebSocket connection for updates
- AI insights display
- User feedback collection

Key Components:

- Authentication (Clerk integration)
- Dashboard with recording list
- Video player with dual audio tracks (original and AI-generated)
- Insight viewer with markdown rendering
- Real-time notification system

Real-Time Updates:

- Connects to Node.js backend via Socket.IO
- Listens for video availability
- Listens for processed audio availability
- Listens for insight generation completion
- Updates UI dynamically without page refresh

Database Schema

Tables

users

- Synced from Clerk authentication system
- Stores user profile information
- Links to recordings via user ID

recordings

- Stores recording metadata
- Contains Supabase Storage paths for media files
- Tracks processing status
- Includes session identifiers
- Links to user who created recording

recording_insights

- Stores AI-generated summaries
- Contains structured markdown content
- Links to parent recording
- Includes generation timestamp
- Cached for performance

feedback

- Stores user feedback on recordings
- Tracks satisfaction ratings
- Contains optional text comments
- Links to recording and user

Storage Structure

Supabase Storage Buckets:

Video files: {userId}/{sessionId}_video.webm

- Original screen recording
- WebM format
- Uploaded immediately after recording

Audio files: {userId}/{sessionId}_audio.webm

- Original microphone audio
- WebM format
- Uploaded with video

Processed audio: {userId}/{sessionId}_processed.mp3

- AI-generated narration
- MP3 format
- Uploaded after AI processing

Security Architecture

Authentication

- Clerk handles all user authentication
- JWT tokens for API authentication
- Session management with automatic expiration
- Secure cookie handling

Data Protection

- Supabase Row Level Security (RLS) policies
- Users can only access their own recordings
- Signed URLs for media file access
- URLs expire after 1 hour
- Extension validates authentication before recording

API Security

- JWT token validation on all protected endpoints
- Rate limiting on API endpoints
- Input validation and sanitization
- Error messages sanitized to prevent information leakage

AI Model Integration

Google Gemini 2.5 Integration

Purpose: Generate professional scripts from raw transcripts

Input Format:

- Raw transcript with word timings
- DOM event timeline
- User action context

Output Format:

- Polished script text
- Structured narration
- Professional tone

Configuration:

- Model: Gemini 2.5
- Temperature: Balanced for consistency

- Context window: Handles long recordings
- Prompt engineering: RAG-based context inclusion

Deepgram Integration

Speech-to-Text:

- Real-time transcription
- Word-level timing information
- High accuracy for technical terms
- Processing time: 2-5 seconds

Text-to-Speech:

- Natural voice synthesis
- Professional tone
- Multiple voice options available
- Processing time: 10-20 seconds

Configuration:

- Model: Nova-2 for transcription
- Voice: Configurable for TTS
- Format: MP3 output

NVIDIA Qwen Integration

Purpose: Generate intelligent insights about recordings

Processing:

- Analyzes complete transcript
- Generates structured summaries
- Provides quality observations
- Creates actionable feedback

Output Format:

- Markdown formatted content
- Structured sections
- Key observations highlighted

Performance:

- Processing time: 2-4 seconds
- Results cached in database

- Fast retrieval on subsequent requests

Data Flow Diagrams

Recording Flow

1. **User Initiates Recording (Extension)**
    - Extension starts countdown
    - MediaRecorder APIs initialized
    - DOM event listeners attached
2. **Active Recording (Extension → Backend)**
    - Video chunks streamed every N seconds
    - Audio chunks streamed every N seconds
    - DOM events sent in batches
    - All data includes session ID and timestamps
3. **Stop Recording (Extension)**
    - Stop all MediaRecorder instances
    - Send final data chunks
    - Redirect user to dashboard
    - Extension cleans up resources
4. **Initial Upload (Backend)**
    - Receive all media chunks
    - Combine chunks into complete files
    - Upload to Supabase Storage
    - Generate signed URLs
    - Broadcast video URL via WebSocket
5. **Transcription (Backend)**
    - Send audio to Deepgram
    - Receive transcript with word timings
    - Store transcript in database
6. **AI Processing (Backend → Python)**
    - Package transcript, timings, DOM events
    - Send to Python FastAPI endpoint
    - Python generates professional script

- Python converts script to audio
- Return processed audio to backend

7. **Final Delivery (Backend → Frontend)**
    - Upload processed audio to Supabase
    - Generate signed URL
    - Broadcast audio URL via WebSocket
    - Trigger insight generation
    - Update database with completion status

Insight Generation Flow

1. **Trigger (Backend)**
    - Recording processing completes
    - Transcript available in database
2. **API Call (Backend → NVIDIA)**
    - Send transcript to NVIDIA Qwen
    - Request structured insight generation
3. **Processing (NVIDIA)**
    - Analyze transcript content
    - Generate summary
    - Create observations
    - Structure output as markdown
4. **Storage (Backend)**
    - Receive insight response
    - Store in recording_insights table
    - Link to parent recording
5. **Delivery (Backend → Frontend)**
    - Broadcast insight availability
    - Frontend fetches from database
    - Display in user interface

Done By: Mohammed Faris Sait