LiveSitter: RTSP Livestream with Dynamic Overlays

This application allows users to view RTSP streams through a web interface and add custom overlays (text or images) in real-time. The system converts RTSP streams to HLS format for web browser compatibility and provides a complete CRUD interface for managing overlays.

Features

- RTSP Stream Playback: View any RTSP camera stream directly in your browser
- Dynamic Overlays: Add text or images on top of your video stream
- Real-time Management: Create, update, delete and position overlays while streaming
- Responsive Design: Works on desktop and mobile devices
- HLS Format Conversion: Uses FFmpeg to convert RTSP to browser-compatible HLS format

Setup Instructions

Prerequisites

- Node.js (v14+) and npm
- Python (v3.6+)
- FFmpeg
- MongoDB (optional: falls back to in-memory database if unavailable)

Backend Setup

- Navigate to the backend directory:
- Create a Python virtual environment
- Activate the virtual environment:
- Windows: activate

Install required dependencies:

pip install -r requirements.text

Start the Flask application:

python app.py

Frontend Setup

- Navigate to the frontend directory:
  - cd frontend
- Install required dependencies:
  - npm install
- Start the React development server:
  - npm start

The application should now be accessible at http://localhost:3000

Starting the RTSP Stream

- To convert an RTSP stream to HLS format for web viewing:
- Open a new terminal window
- Navigate to the backend directory
  - cd backend
- Run the FFmpeg script with your RTSP URL:
- for example: *ffmpeg -i "rtsp://rtspstream:TC_eE1VyDSC6kqzAQME_V@zephyr.rtsp.stream/traffic" -c copy -f hls -hls_time 2 -hls_list_size 5 -hls_flags delete_segments static_hls/stream.m3u8*

User Guide

Viewing the Stream

- Open your browser and navigate to http://localhost:3000
- The video player will automatically load the stream
- Use the standard video controls to play, pause, or adjust volume
- Managing Overlays
- Creating Overlays
- Click the "Add Overlay" button in the sidebar
-
- A modal will appear with the following options:
-
- Name: A name for your overlay (e.g., "Company Logo")
- Type: Choose between "Text" or "Image URL"
- Content: Enter text or an image URL depending on your selection
- X Position: Horizontal position (0-100%)
- Y Position: Vertical position (0-100%)
- Width: Width of the overlay (0-100%)
- Height: Height of the overlay (0-100%)
- Z-Index: Stacking order (higher numbers appear on top)
- Visible: Toggle visibility
- Click "Create" to add the overlay to your stream

Editing Overlays

- In the sidebar, find the overlay you want to edit

- Click the "Edit" button next to the overlay
- Make your changes in the modal that appears
- Click "Update" to save your changes
- Deleting Overlays
- In the sidebar, find the overlay you want to delete
- Click the "Delete" button next to the overlay
- Confirm the deletion when prompted

API Documentation

Base URL

All API endpoints are relative to:

http://localhost:5000/api

Health Check

GET /health

Response:


```
{
  "ok": true
}
```


Get ALL Overlays

response:


```
[
  {
    "_id": "6172a3d90c3c233f2ab5bf93",
    "name": "Company Logo",
    "type": "image",
    "content": "https://example.com/logo.png",
    "x": 90,
    "y": 10,
    "width": 20,
```

```
    "height": 10,
    "zIndex": 10,
    "visible": true
  },
  {
    "_id": "6172a3f10c3c233f2ab5bf94",
    "name": "Welcome Text",
    "type": "text",
    "content": "Welcome to the livestream!",
    "x": 50,
    "y": 50,
    "width": 60,
    "height": 15,
    "zIndex": 20,
    "visible": true
  }
]
```


GET /overlays/:id


```
{
  "_id": "6172a3d90c3c233f2ab5bf93",
  "name": "Company Logo",
  "type": "image",
  "content": "https://example.com/logo.png",
  "x": 90,
  "y": 10,
  "width": 20,
```

```
  "height": 10,
  "zIndex": 10,
  "visible": true
}
```

Technical Details

Architecture

This application uses:

- Backend: Python Flask API with MongoDB (optional)
- Frontend: React.js with modern hooks and components
- Video Conversion: FFmpeg for RTSP to HLS transcoding
- Video Playback: HLS.js for browser-based HLS stream playback

Directory Structure

```
livesitter-task/
├── backend/              # Flask API
│   ├── app.py            # Main Flask application
│   ├── models.py         # Data models
│   ├── start_ffmpeg.sh   # RTSP to HLS conversion script
│   └── static_hls/       # HLS output directory
│
└── frontend/             # React frontend
    ├── public/           # Static assets
    └── src/              # React source code
        ├── components/   # React components
        │   ├── HLSPlayer.jsx        # Video player component
        │   ├── OverlayEditor.jsx    # Overlay management component
        │   └── OverlayItem.jsx      # Individual overlay rendering
        ├── App.js        # Main application component
```

```
└── index.js      # Application entry point
```

Troubleshooting

- Common Issues
- Stream Not Loading
- Ensure FFmpeg is running with the correct RTSP URL
- Check if the RTSP stream is accessible from your server
- Verify that the backend Flask server is running on port 5000
- Check browser console for any JavaScript errors
- Overlays Not Appearing
- Verify that the overlay is set to "Visible: true"
- Check that the overlay coordinates (x, y) are within the visible area (0-100%)
- Ensure the overlay content is valid (for image URLs, check if they load correctly)
- API Connection Errors
- Verify that both frontend and backend servers are running
- Check if CORS is properly configured in the Flask application
- Make sure the REACT_APP_API_URL environment variable is set correctly (defaults to http://localhost:5000)

Acknowledgements

- HLS.js for HLS video playback
- FFmpeg for video transcoding
- React.js for the frontend framework
- Flask for the backend API