StreamShare

A Video Streaming Platform where creators and viewers are much closer.

Project Overview

An encrypted streaming platform where users can choose from a wide range of creators and view their streams by joining rooms based on a token system and interact with other users in the room.

Tech Stack

Frontend

- **Core API: *WebRTC* for real-time video and audio capture and playback.**
- **Framework: *React.js* for building the user interface.**
- **Styling: *Tailwind CSS* for efficient and maintainable styling.**
- **Build Tool: *Vite* for bundling and running the frontend app.**
- **UI Elements: Display user coin balance, "pay" button on creator streams.**

Backend

- **Language: *Node.js* .**
- **Framework: *Express.js* for building the REST API and handling server-side logic.**
- **Real-time Communication: *Socket.IO* for the signaling process to establish *WebRTC* connections and handle real-time chat/notifications.**
- **Database: MongoDB for storing room information, user view history (will store coins field in user documents, and transaction history).**
- **Authentication: JWT (to identify the user making the payment).**

Media Server

- **Core Technology: WebRTC Media Server.**
- **Tools: MediaMTX (for handling the broadcast architecture, receiving one stream from the creator and efficiently fanning it out to many viewers).**

- STUN/TURN Server: Coturn, to facilitate peer connections and ensure streams work even behind complex firewalls.

Features of the App

- Multiple video options for the user on their home screen.
- Users will join a room made by the creator of the stream where they can interact with other users as well as the creator.
- Creator is notified whenever a user enters or leaves their stream for noting view-count.
- A token system is used where user has to use set number of tokens to view a specific stream.

Real-time Streaming (WebRTC)

- Frontend (Creator):

  1. The browser's `RTCPeerConnection` API is used to capture video and audio via `getUserMedia()`.

  2. The frontend establishes a connection with the media server (MediaMTX).

  3.The `RTCPeerConnection` exchanges SDP (Session Description Protocol) offer/answer and ICE candidates with the media server via the Socket.IO signaling server.

  4. The media stream is then sent directly from the creator's browser to the media server.

- Frontend (Viewer):

  1. The browser's `RTCPeerConnection` connects to the media server.

  2. The frontend receives the SDP offer from the media server (via signaling server) and sends back an SDP answer.

  3. The media stream from the creator is received directly from the media server and displayed in the viewer's video player.

Implementation of Coin System

- Backend API for Coin Management:
  - User Coin Balance Endpoint (`GET /api/users/balance`):
    - Protected by JWT authentication.

        - Fetches the coins value from the authenticated user's document in MongoDB and returns it to the frontend.
    - Transaction Endpoint (`POST /api/transactions/stream-join`):
        - Endpoint for handling payments to join a stream.
        - It will be a transactional operation to ensure either both deduction and credit happen, or neither does.
    - Logic:

      1. Receive `userId` (from JWT), `creatorId` , `roomId` , and `amount` from frontend.

      2. Start a MongoDB session/transaction.

      3. Deduct coins from user: Find the user's document and decrement their `coins` field by `amount` . Check if the user has sufficient balance before deducting. If not, no transaction and return error.

      4. Crediting coins to the creator: Find the creator's document and increment their `coins` field by `amount` .

      5. Commit the transaction. If any step fails, go back

      6. Return success/failure to the frontend.

- Frontend Integration:
    - Display Balance: Displays the user's current coin balance in the UI. Update after successful transactions.
        - "Pay" Button: On a creator's stream page, display the required coin amount and a "Pay" button.
        - Joining Logic: When the "Pay" button is clicked:

          1. Makes an API call to `POST /api/transactions/stream-join` endpoint.

          2. If the payment is successful, proceed with the WebRTC signaling to join the stream.

          3. If the payment fails (e.g., insufficient balance), display an appropriate message to the user.

Future goals

- **Making End-To-End Encryption**
- **One-To-One Interactions between users and creators.**
- **Add in-app payment option inorder to acquire new coins.**