Project Overview Document

1. Introduction

This document provides an overview of the project architecture, including descriptions of major variables, components, pages, database structure, and entry points. It is designed for those with limited experience in front-end development and React. Moreover, this document also goes over the typical user experience from login to sign-out, to provide more about the research tool's usage.

2. Key React Concepts

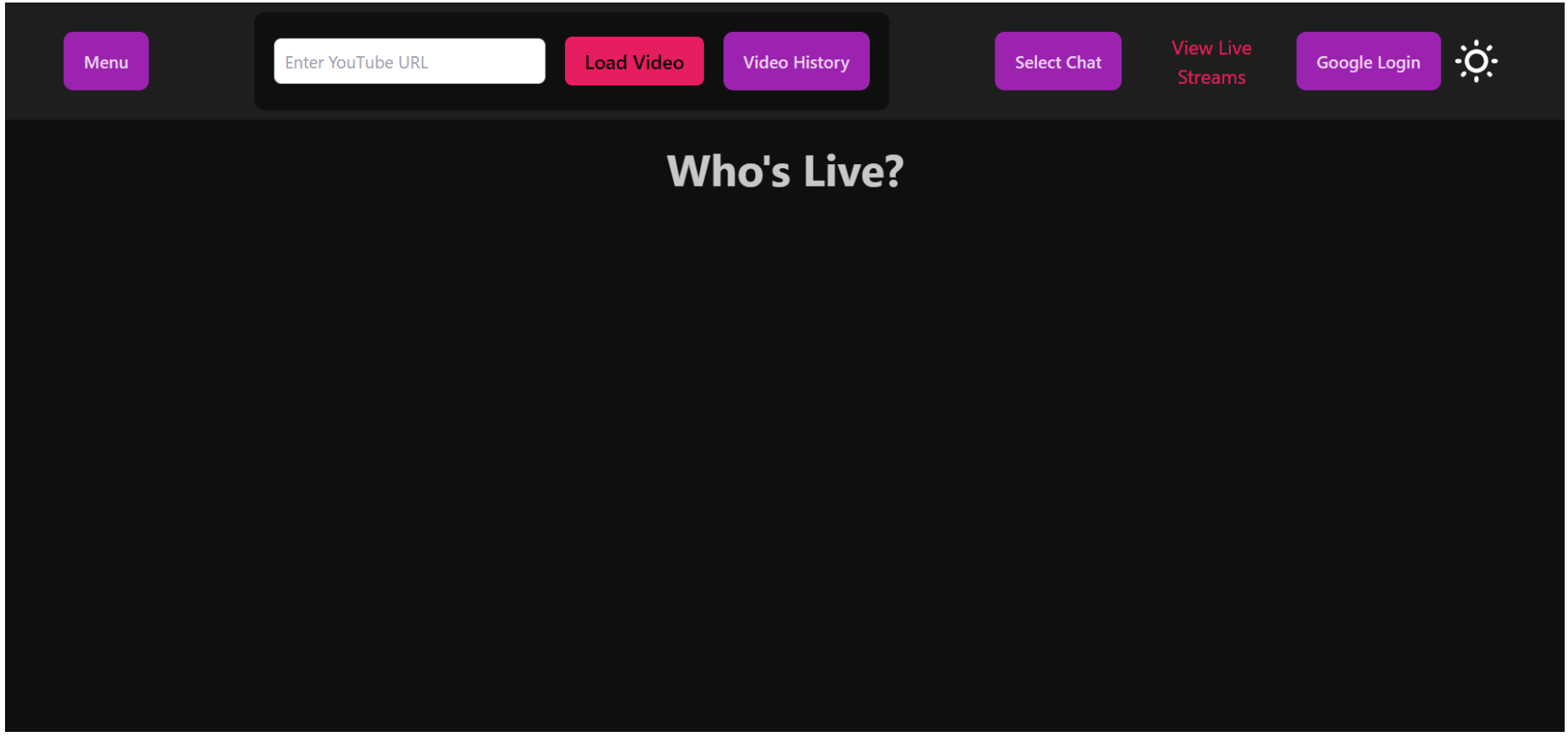
1. **useState Hook**:
    - The `useState` hook is used to create state variables in a functional React component. State variables allow components to maintain and update their data dynamically. For example, `const [count, setCount] = useState(0)` creates a `count` variable with an initial value of `0`, and `setCount` is used to update it.
2. **useEffect Hook**:
    - The `useEffect` hook allows you to run side effects in your components. Side effects include data fetching, subscriptions, or manual DOM updates. It runs after every render by default, but you can control when it runs by passing dependencies. For example, `useEffect(() => { fetchData(); }, [dependency]);` runs `fetchData` whenever `dependency` changes

3. Project Entry Points

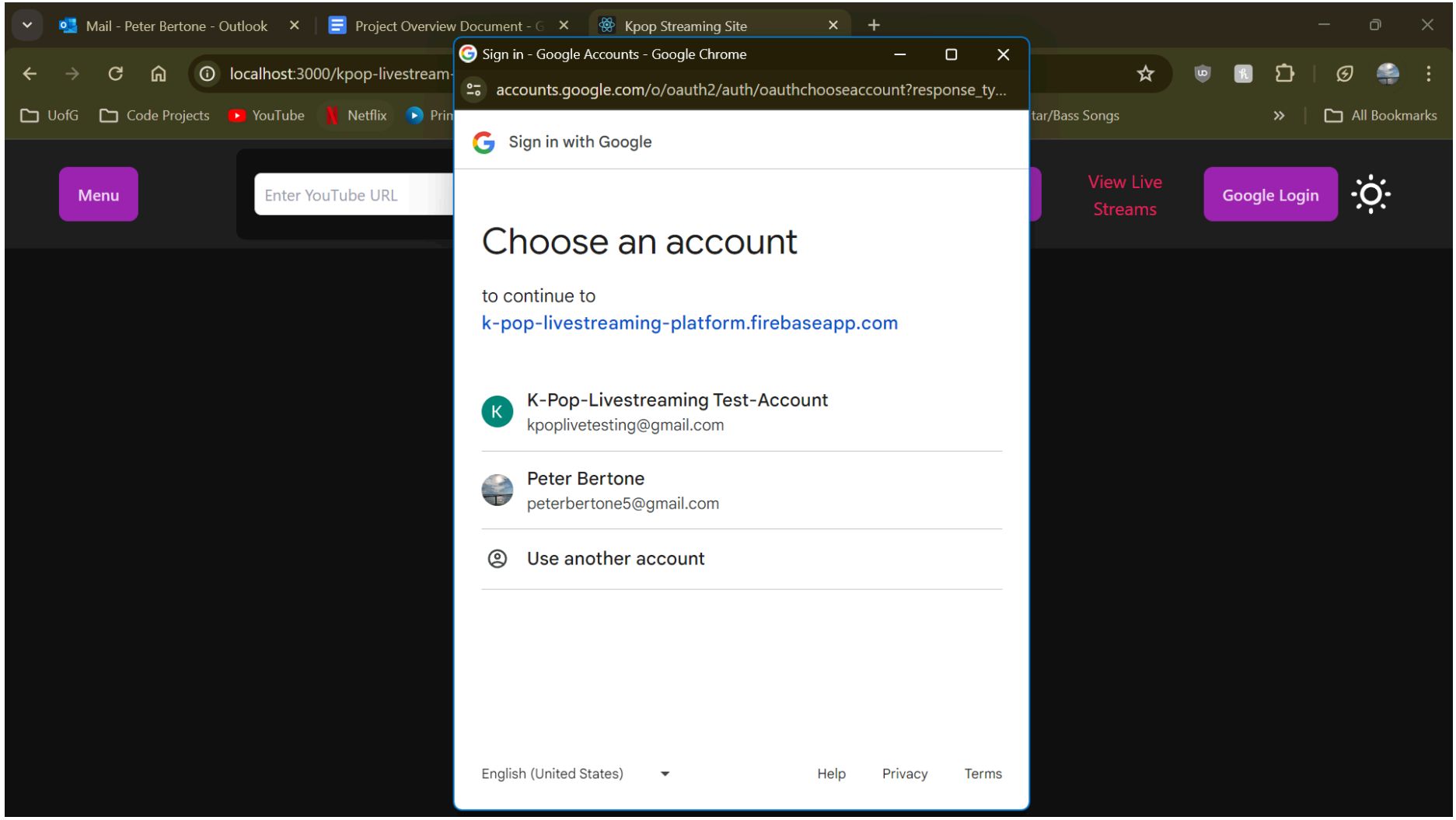
- **`index.js`**: The main entry point of the application. It renders the root React component (`App`) into the DOM.
- **`App.js`**: The root component that wraps the entire application. It manages the routing and sets up global providers (e.g., context providers)

User Experience

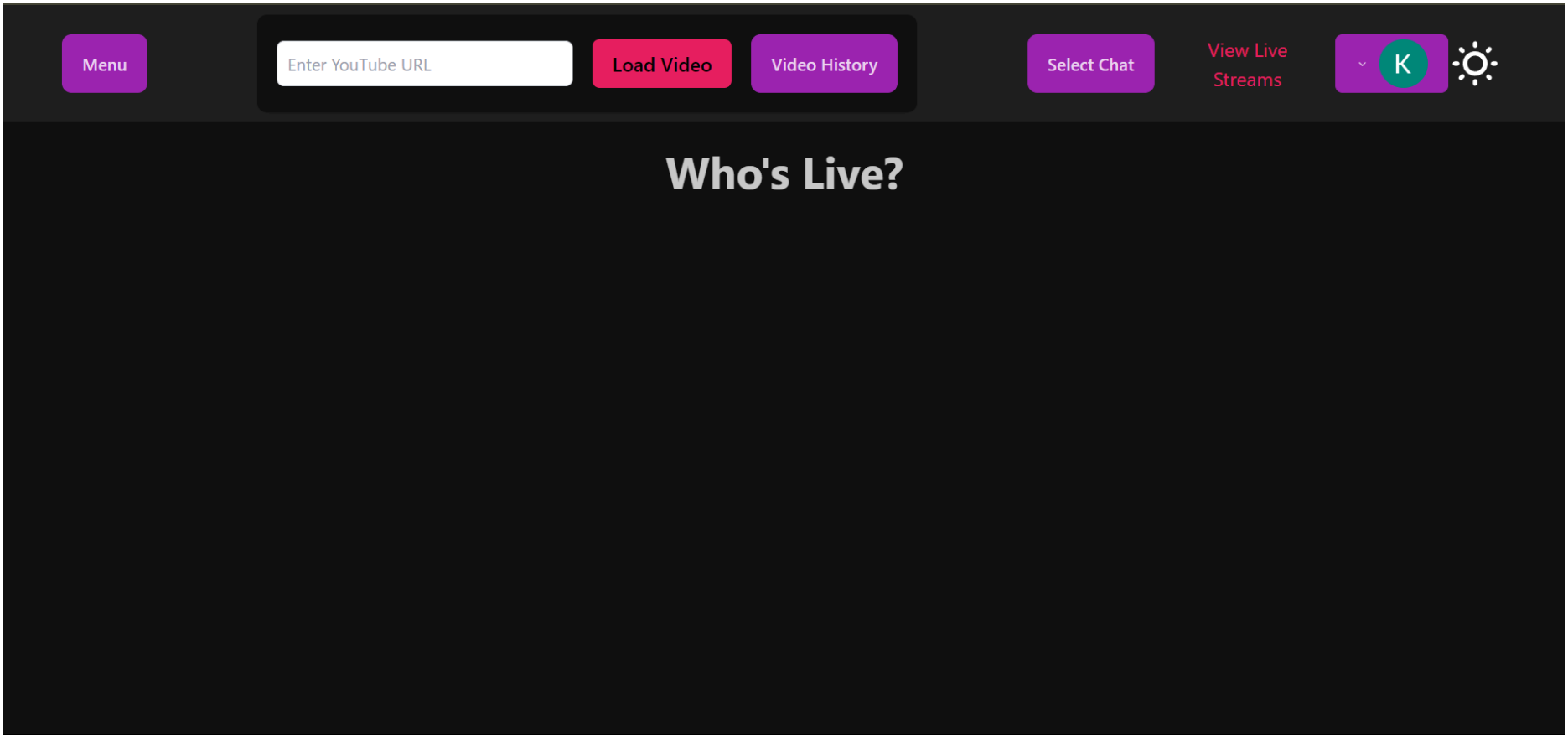
User Login via Google

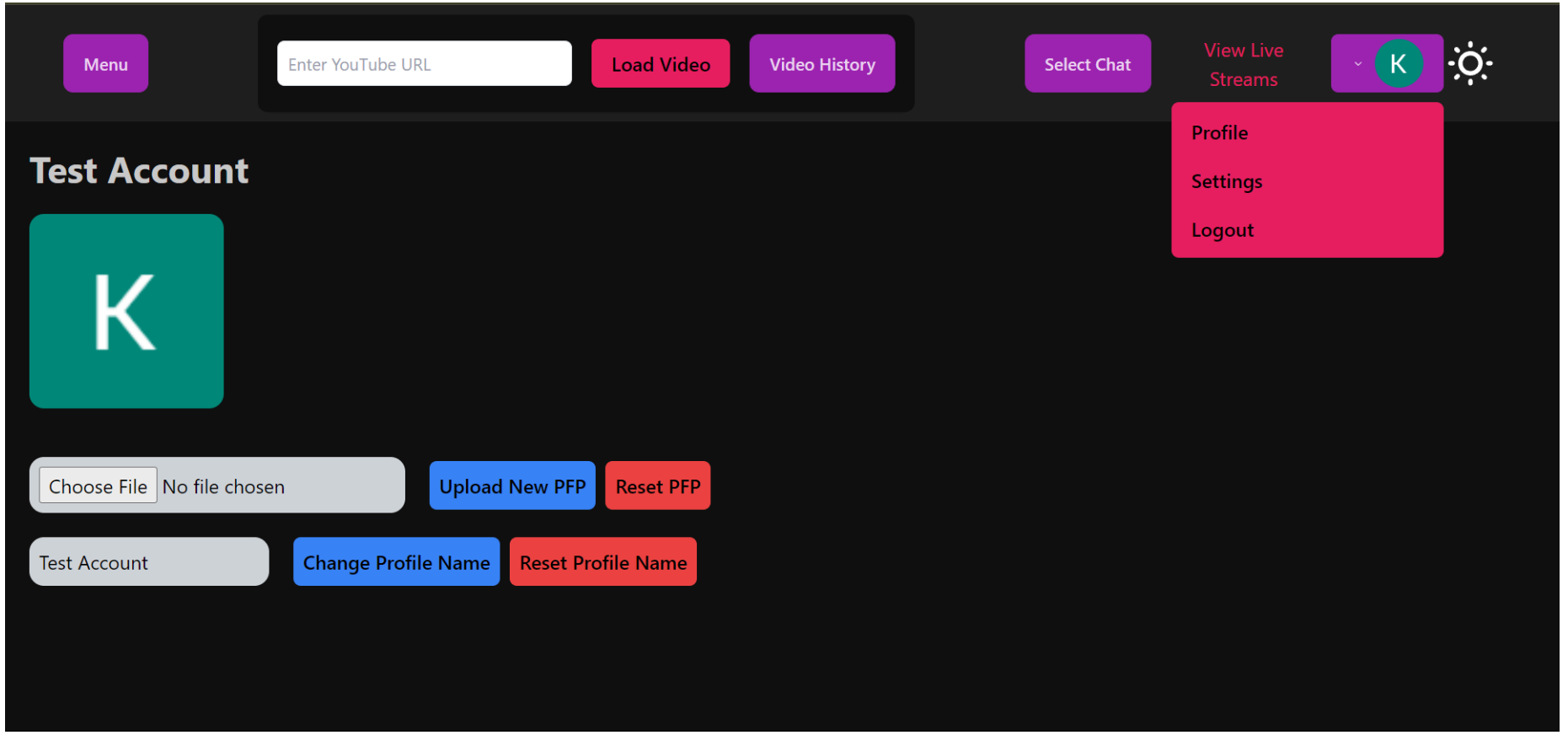


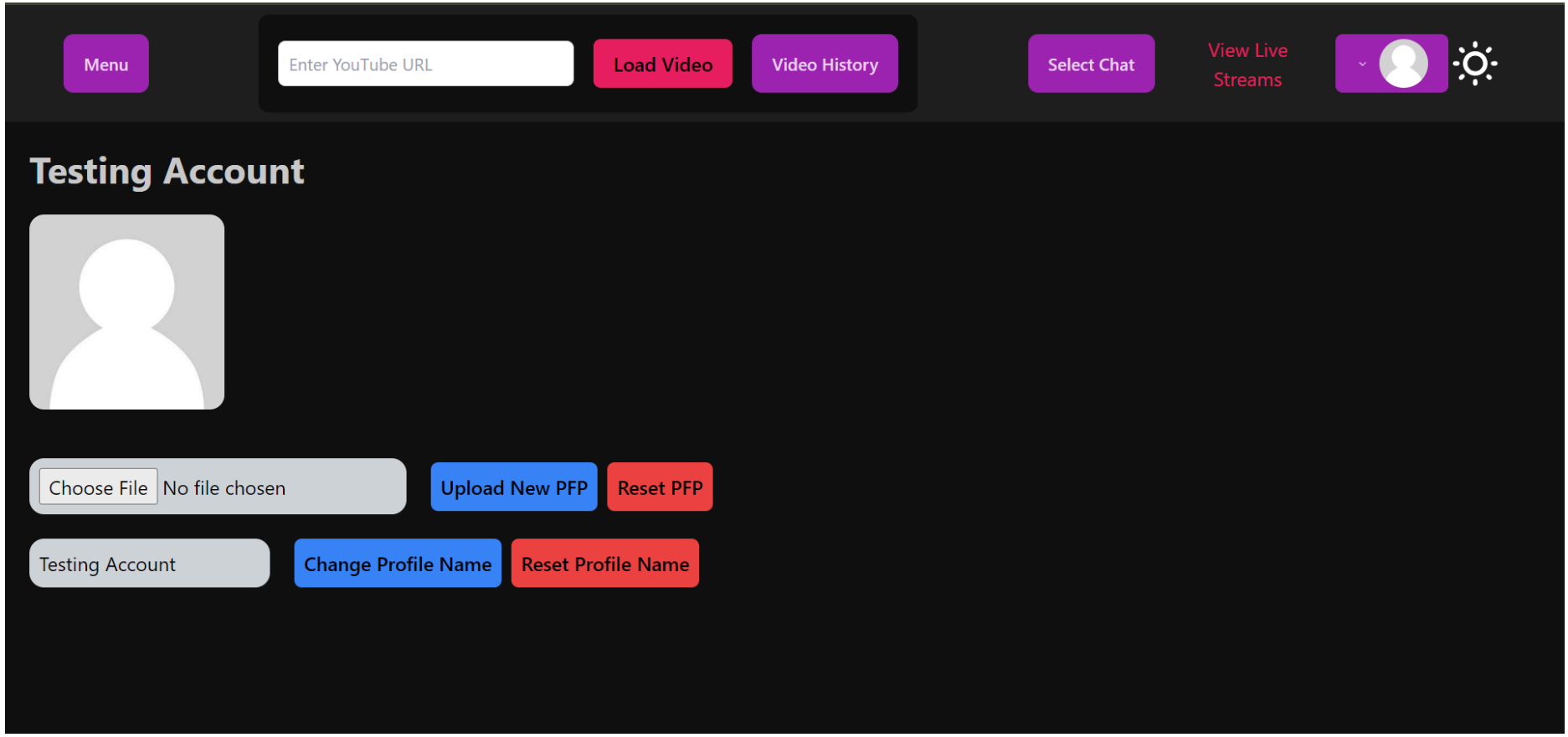
User chooses an account via Google pop-up:



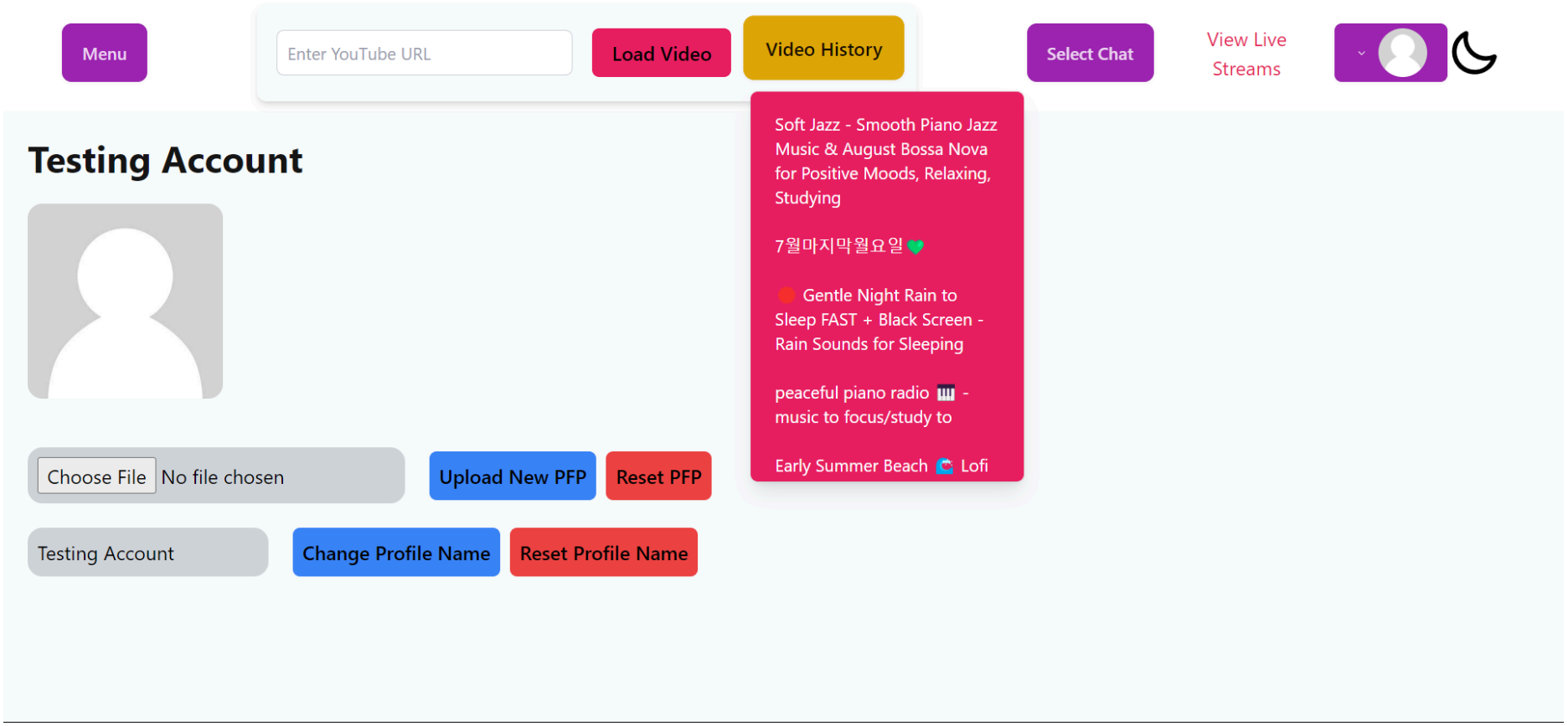
User is now logged in and decides to change their profile picture and name:



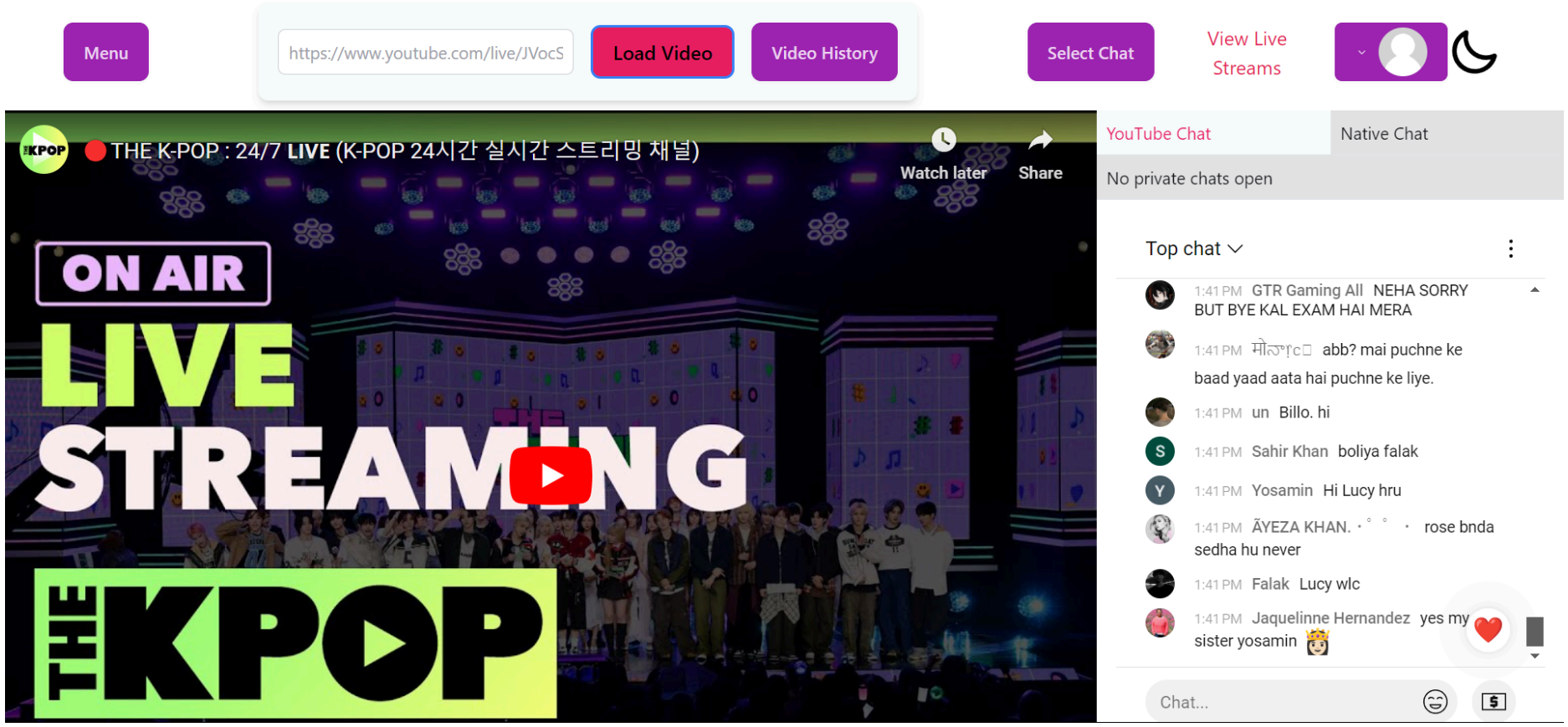

Menu
Enter YouTube URL
Load Video
Video History
Select Chat
View Live Streams
K
Profile
Settings
Logout
Test Account
K
Choose File
No file chosen
Upload New PFP
Reset PFP
Test Account
Change Profile Name
Reset Profile Name

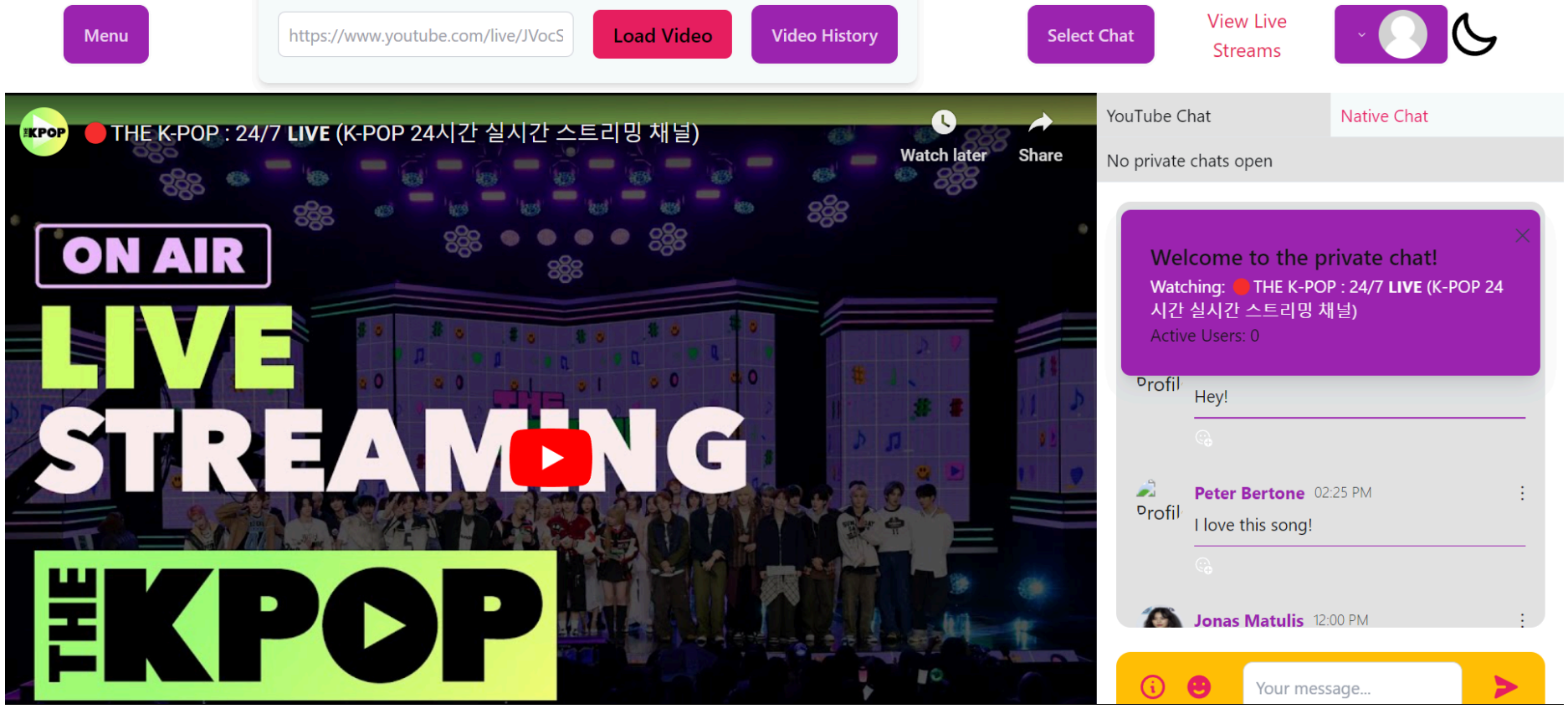

Menu
Enter YouTube URL
Load Video
Video History
Select Chat
View Live Streams
Testing Account
Choose File
No file chosen
Upload New PFP
Reset PFP
Testing Account
Change Profile Name
Reset Profile Name

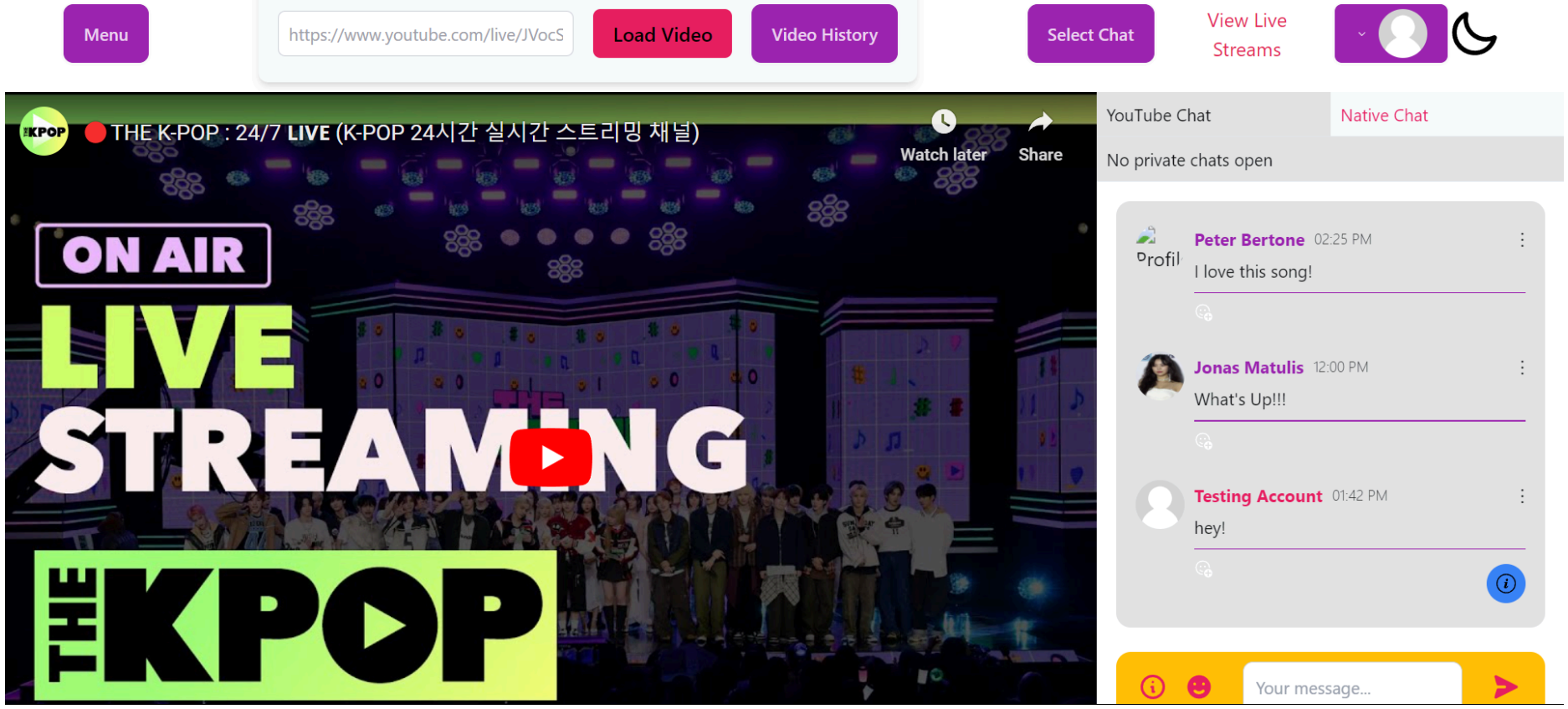
User decides to switch to the light theme, and loads a video from their video history

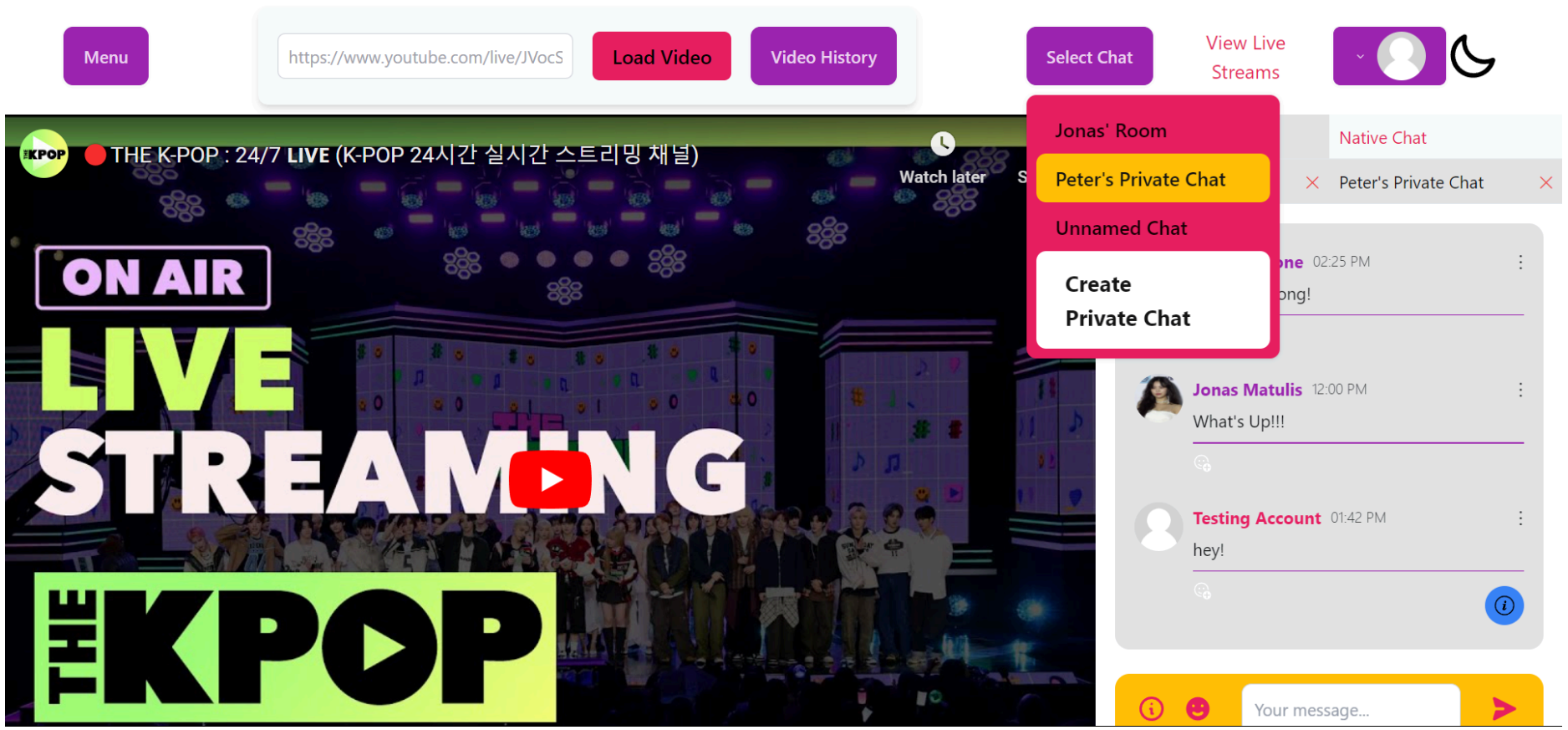


User reads some messages in the YouTube Embedded chat, then decides to chat in the Native chat:

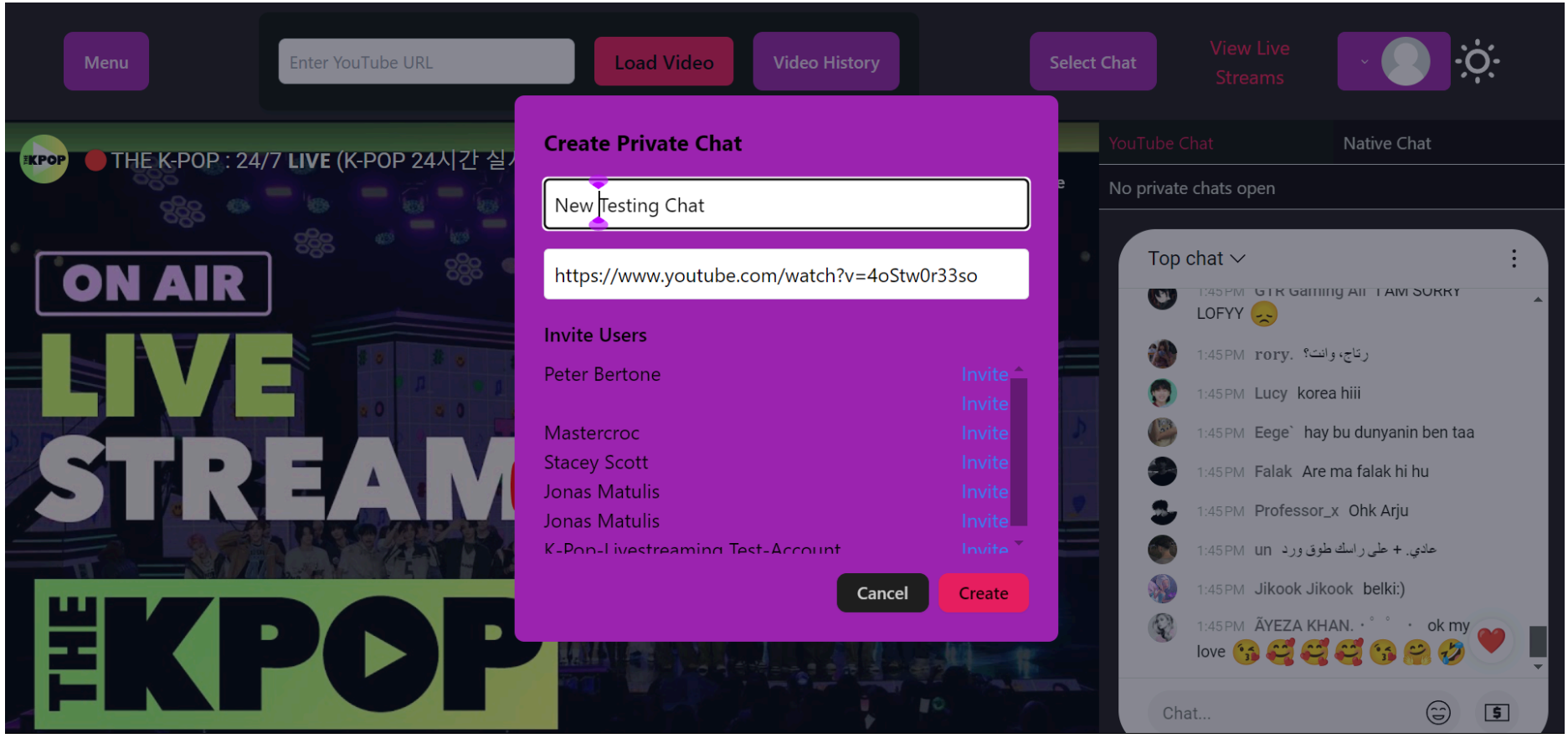



Menu
https://www.youtube.com/live/JVocS
Load Video
Video History
Select Chat
View Live Streams
THE K-POP : 24/7 LIVE (K-POP 24시간 실시간 스트리밍 채널)
Watch later
Share
ON AIR
LIVE STREAMING
THE KPOP
YouTube Chat
Native Chat
No private chats open
Welcome to the private chat!
Watching: THE K-POP : 24/7 LIVE (K-POP 24 시간 실시간 스트리밍 채널)
Active Users: 0
Hey!
Peter Bertone 02:25 PM
I love this song!
Jonas Matulis 12:00 PM
Your message...

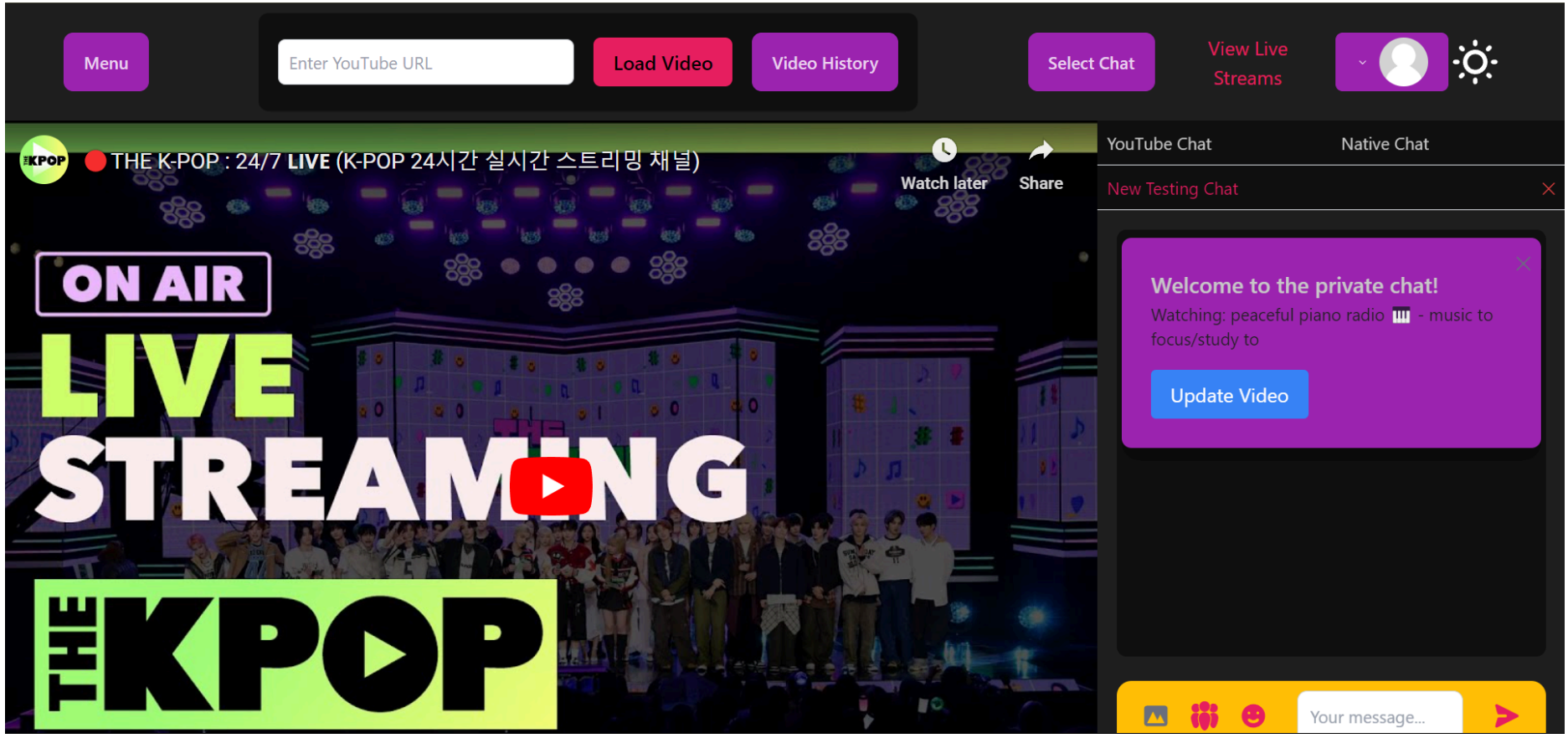

Menu
https://www.youtube.com/live/JVocS
Load Video
Video History
Select Chat
View Live Streams
THE K-POP : 24/7 LIVE (K-POP 24시간 실시간 스트리밍 채널)
Watch later
Share
ON AIR
LIVE STREAMING
THE KPOP
YouTube Chat
Native Chat
No private chats open
Peter Bertone 02:25 PM
I love this song!
Jonas Matulis 12:00 PM
What's Up!!!
Testing Account 01:42 PM
hey!
Your message...


Menu
https://www.youtube.com/live/JVocS
Load Video
Video History
Select Chat
View Live Streams
Jonas' Room
Peter's Private Chat
Unnamed Chat
Create Private Chat
Native Chat
Peter's Private Chat
THE K-POP : 24/7 LIVE (K-POP 24시간 실시간 스트리밍 채널)
Watch later
ON AIR
LIVE STREAMING
THE KPOP
02:25 PM
Jonas Matulis 12:00 PM
What's Up!!!
Testing Account 01:42 PM
hey!
Your message...

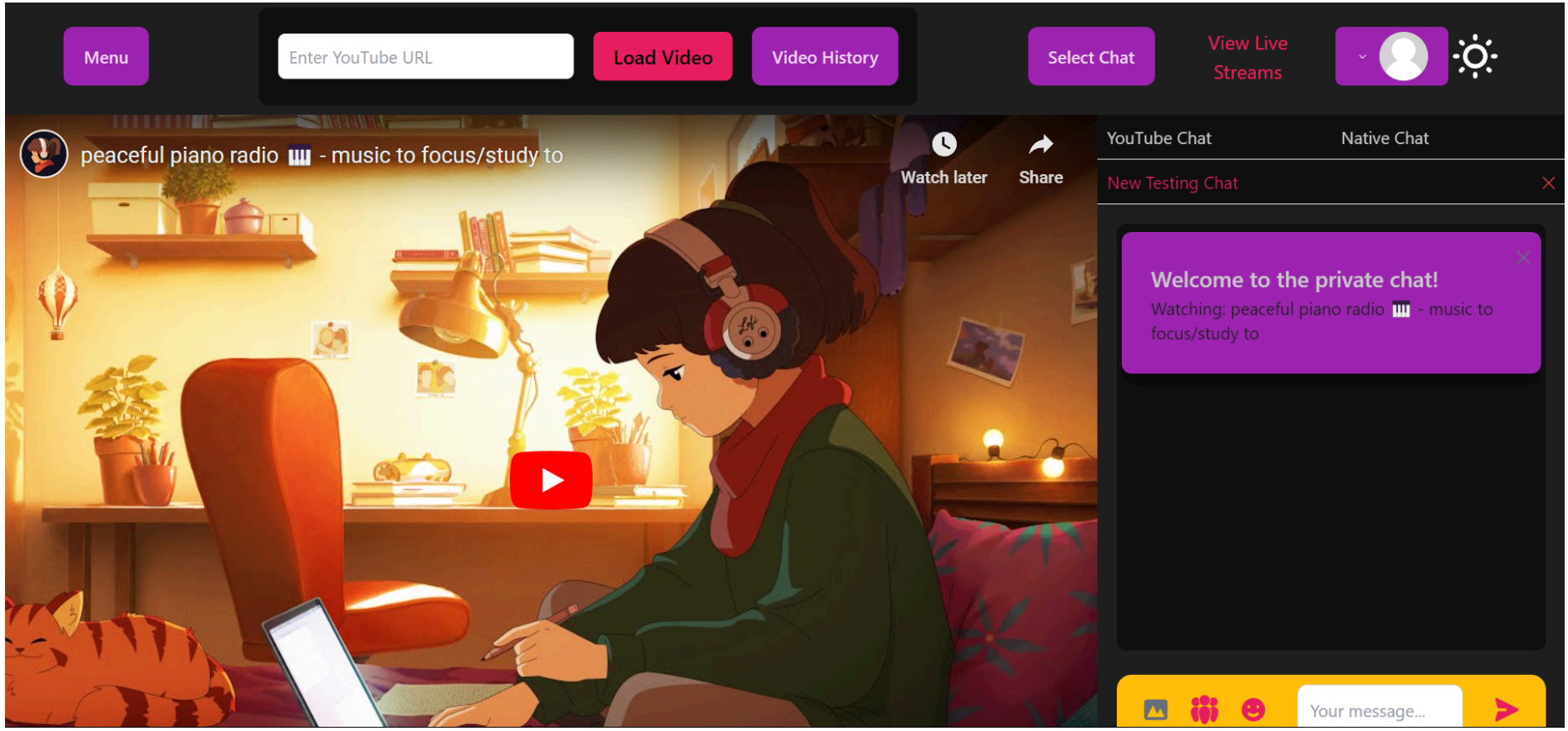
User decides to create a new private chat watch party:



After creating and opening their new chat, the user updates the current video:



After using the private chat with their friends, the user decided to logout, and end the session:



4. Variables

Variable	Type	Description	Scope	Usage
videoId	string	Stores the ID of the current video being played in the VideoPlayer.	Global (Page-Level)	Used in VideoPlayer to dynamically load the correct video based on the videoId.
setVideoId	function	Function to update the videoId.	Global (Page-Level)	Allows components to change the videoId, such as when a new video is selected.
selectedChats	array	Holds the IDs of currently active chat sessions.	SwitchableChat	Enables the user to switch between different chat tabs.
setSelectedChats	function	Function to update the selectedChats.	SwitchableChat	Updates the active chats when a new chat is opened or an existing chat is closed.
isPopupOpen	boolean	Boolean that determines if the mobile menu popup is open or closed.	Header (Mobile)	Toggles the display of the popup menu for mobile users.
activeUsers	array	Stores a list of users who are currently active in the chat or live stream.	Global (Page-Level)	Displays the real-time presence of users in various components like NativeChat.

setActiveUsers	function	Function to update the activeUsers.	Global (Page-Level)	Updates the list of active users whenever there is a change detected (e.g., someone joins or leaves).
privateChats	array	Array containing information about private chat sessions the user is a part of.	Global (Page-Level)	Displays the available private chats in PrivateChatHeader and other related components.
setPrivateChats	function	Function to update the privateChats array.	Global (Page-Level)	Adds or removes private chats from the list based on user actions or new data fetched from the database.
invitations	array	Holds pending invitations to join private chats.	Global (Page-Level)	Allows the user to accept or reject chat invitations through InvitationPopup.
setInvitations	function	Function to update the invitations array.	Global (Page-Level)	Updates the invitations when a new one is received or when an invitation is accepted/rejected.
selectedChatId	string	Holds the ID of the currently selected chat tab.	SwitchableChat	Tracks which chat is currently active to display the correct messages and participants.

setSelectedChatId	function	Function to update the selectedChatId.	SwitchableChat	Changes the active chat tab based on user selection.
isActiveUsersModalOpen	boolean	Determines if the modal showing active users is open.	SwitchableChat	Toggles the active users modal in SwitchableChat.
setIsActiveUsersModalOpen	function	Function to update the isActiveUsersModalOpen state.	SwitchableChat	Opens or closes the active users modal.
isPrivateChatUsersModalOpen	boolean	Determines if the modal showing private chat members is open.	SwitchableChat	Toggles the private chat members modal in SwitchableChat.
setIsPrivateChatUsersModalOpen	function	Function to update the isPrivateChatUsersModalOpen state.	SwitchableChat	Opens or closes the private chat members modal.
privateChatVideoId	string	Stores the video ID for private chats that have an associated video.	SwitchableChat	Allows the VideoPlayer to load the correct video for private chat rooms with video content.
showLoginAlert	boolean	Determines if the login alert is shown when a user attempts to use the native chat without being logged in.	SwitchableChat	Used to prompt the user to log in before accessing the native chat.
setShowLoginAlert	function	Function to update the showLoginAlert state.	SwitchableChat	Toggles the visibility of the login alert.
notification	string	Holds notification messages that are displayed to the user.	Global (Page-Level)	Displays important alerts, such as successful chat creation or new invitations.

setNotification	function	Function to update the notification state.	Global (Page-Level)	Changes the notification message when triggered by specific actions.

5. Major Components

1. Header

- **Purpose**: Manages the navigation and theme controls, and includes subcomponents like `VideoHeader`, `PrivateChatHeader`, and the theme toggle.
- **Technical Details**:
    - **State Management**: The Header component may include state management for toggling between themes (light/dark modes) using `useState` and `useEffect` hooks.
    - **Event Handling**: Handles user interactions like theme switching through event listeners. The theme is typically stored in a global state or directly applied to the `document.documentElement` for persistent changes.
    - **Responsive Design**: Uses Tailwind CSS classes (e.g., `md:hidden`, `flex`, `justify-between`) to hide or show different parts of the header based on screen size. The desktop view shows more options, while the mobile view collapses them into a menu that can be toggled.
    - **Subcomponents**:
        - **VideoHeader**: Displays controls related to the video being played, like changing the video URL or video ID.
        - **PrivateChatHeader**: Shows the active private chats and invitations, allowing users to manage their private chat interactions directly from the header.
        - **Theme Toggle**: Switches between light and dark themes, often using a checkbox input that triggers a state change.

2. VideoPlayer

- **Purpose**: Renders the video using the YouTube embed link, adapting to both mobile and desktop views.
- **Technical Details**:
    - **Props**: Receives the `videoId` prop to construct the YouTube embed URL dynamically.
    - **Responsive Design**: Uses Tailwind CSS classes to adjust the size and positioning of the video player based on the viewport (e.g., `w-full`, `h-[50%]`, `md:h-full`, `md:w-[70%]`).
    - **Embedding YouTube**: The video is embedded using an `<iframe>` element, with attributes like `allowFullScreen`, `allow="accelerometer; autoplay; encrypted-media; gyroscope; picture-in-picture"`, ensuring proper behavior on various devices.
    - **Cross-Platform Compatibility**: Ensures that the video player works seamlessly across different devices by adjusting the dimensions and allowing controls.

3. SwitchableChat

- **Purpose**: A chat component that switches between different chat types (e.g., YouTube chat, native chat, private chat) based on user selection.
- **Technical Details**:
    - **State Management**: Uses `useState` to track the active chat tab (`selectedTab`) and toggle between YouTube chat, native chat, and private chats.
    - **Conditional Rendering**: Renders different chat components (`YouTubeChat`, `NativeChat`, `PrivateChat`) based on the active tab. The selected tab's content is displayed, while the others remain hidden.
    - **Responsive Design**: Adjusts the layout for mobile and desktop using Tailwind CSS classes, positioning the chat below the video on mobile (`h-[50%]`) and beside it on desktop (`w-[30%]`).
    - **Chat Tabs**: Manages the switching between different chats through `PrivateChatTabs`, allowing users to open and close chats, and displays the current active chat.
    - **Integration**: Integrates with `VideoPlayerPage` to handle chat-related actions like invitations, private chat creation, and user interaction.

4. PrivateChat

- **Purpose**: Manages private chat functionality, including chat creation, invitations, and messaging.
- **Technical Details**:
    - **State Management**: Tracks private chat messages, active users, and invitations using `useState` and `useEffect`. Handles chat-specific state like `privateChatId`, `messages`, and `members`.
    - **Fetching Data**: Uses Firebase Firestore or similar backend to fetch private chat messages, invited users, and other related data. This may involve asynchronous operations (`async/await`) to retrieve and display real-time data.
    - **Messaging**: Allows users to send and receive messages within a private chat. Typically involves managing form input, validating message content, and interacting with the backend to store and retrieve messages.
    - **Video Synchronization**: Handles video synchronization between users in the private chat, ensuring everyone is watching the same content. This can involve tracking the video's progress and sending updates to all users.
    - **User Invitations**: Manages the invitation process for private chats, allowing users to invite others, accept invitations, or reject them. Invitations are typically stored in the database, and the state is updated based on user interactions.

5. NativeChat

- **Purpose**: A custom chat system that allows users to communicate without relying on external platforms like YouTube.
- **Technical Details**:
    - **State Management**: Tracks chat messages, active users, and any user interactions within the native chat. This may involve storing data like `messages`, `input`, and `activeUsers`.
    - **Real-Time Communication**: Uses Firebase Firestore or WebSockets to facilitate real-time communication. Messages are sent to the backend, where they are stored and broadcast to all participants in the chat.
    - **User Interface**: Displays a chat interface with messages, timestamps, and user information. Tailwind CSS is used to style the chat, ensuring it adapts to different screen sizes.
    - **User Interaction**: Provides options for users to see who's online, view user profiles, or perform chat actions like replying, deleting, or reporting messages.
    - **Customization**: Unlike external chat platforms, `NativeChat` can be fully customized to fit the specific needs of the app, including the design, features, and integrations with other parts of the app like video synchronization or user management.

6. Major Pages

- `VideoPlayerPage`: The main page where the video player and chat components are displayed. The layout changes based on the device (mobile vs. desktop).
- `AccountPage`: Manages user account settings, including profile picture updates and username changes.

7. Database Structure

- **Firebase Firestore**: The backend database used to store user data, chat data, and live stream information.
    - **Collections**:
        - `users`: Stores user profiles, including usernames, profile pictures, and preferences.
        - `privateChats`: Stores private chat data, including chat names, invited users, and messages.
        - `liveStreams`: Stores data about live streams, including video URLs, titles, and active user counts.
    - **Documents**: Each collection contains multiple documents identified by unique IDs, which represent individual records (e.g., user profiles, chat sessions).
- **Utils:** Utils hold functions that fetch and upload data to the database. There is a util file for each collection.

8. Data Flow

- **User Interaction**: Users interact with the UI (e.g., opening a chat, selecting a video), triggering state changes in components.
- **State Management**: React manages component state, propagating changes to child components and triggering re-renders.
- **Database Interaction**: Data is fetched or updated in Firebase Firestore using Firebase SDK functions. State updates are made based on the data returned from Firestore.

9. Routing and Navigation

- **React Router**: Manages navigation between different pages (`VideoPlayerPage`, `AccountPage`, etc.).
- **Conditional Rendering**: Different components and layouts are rendered based on the route and device type (mobile vs. desktop).

10. Styling

Styling was done using Tailwind CSS and Daisy UI. Tailwind.config file has major positions and themes, more specific style changes can be done directly in the element class names.

- **Tailwind CSS:**
  - **Utility Classes:** Utilizes Tailwind's utility classes for consistent, efficient styling directly in JSX.
  - **Responsive Design:** Adapts layouts for mobile and desktop using responsive classes like `md:hidden` and `lg:flex`.
- **Moveable Components:**
  - **Mobile vs. Desktop:** Components adjust between stacked layouts on mobile and side-by-side layouts on desktop.

11. Technical Considerations and Limitations

- **Database:**
  - **Query Quota:** Limited by a query quota, requiring efficient data management to stay within limits.
  - **Storage Limit:** We are also limited by the amount of data we can store *for free* on Firebase's Firestore
- **Hosting:**
  - **Static Constraints:** Limited to static hosting, so dynamic features must be handled client-side or with external services.
- **User Testing:**
  - **Critical Testing:** It is necessary to ensure the tool is user-friendly and performs well despite technical limitations.
- **In browser translation**
  - This feature involves making the website easily translatable by browsers or adding a built-in translation feature to the website, allowing users to view content in their preferred language.
- **Consolidated log of session activity**
  - This will involve creating a script that gathers user activity data from the database and consolidates it into a format that makes it easier to analyze and gain insights from user interactions across both Private and Native Chats.

12. Instrumentation

Instrumentation tracks user interactions during sessions to analyze engagement with K-pop videos.

- **Session Times:**
  - **Login and Logout Times:** Capture exact timestamps of when users log in and out.
  - **Session Duration:** Measures the total time spent on the platform per session.
- **Video Data:**
  - **Video ID's, URL's, and Titles:** Record identifiers, URLs, and titles for each video accessed. This data provides insights into user preferences and content popularity.
- **Chat Data:**
  - **Emojis:** the emoji's sent and the amount users have reacted with it are saved under each chat message to provide insights into user

13. Links

Live Website - https://s9scott.github.io/kpop-livestream-platform/
GitHub - Link
GitLab - Link
Firebase Console - Link
Research Document - Link
Trello - Link