WebRTC ABR Player

The example shows how a stream published to WCS server may be played in a number of video qualities via WebRTC.

On the screenshot below:

- `Server url` - WCS server websocket URL
- `Stream name` - stream name to play
- `h`, `s`, `m send` - quality switching buttons named by quality profiles from `/usr/local/FlashphonerWebCallServer/conf/wcs_sfu_bridge_profiles.yml` file

.

> **Attention**
>
> Audio track is playing separately in `audio` page element

Example source code

The source code consists of the following modules:

- `player.html` - HTML page
- `player.css` - HTML page styles
- `player.js` - main application logic

Analyzing the code

To analyze the example source code, take the file `player.js` version available here

1. Local variables

Local variables declaration to work with constants, SFU SDK, to display video and to work with client configuration

code

```
const constants = SFU.constants;
const sfu = SFU;
```

```
const PRELOADER_URL="../commons/media/silence.mp3"
```

2. Object to store current publishing/playback state

The object should keep Websocket session data, WebRTC connection data, room data and object to display tracks data

code

```
const CurrentState = function() {
    let state = {
        pc: null,
        session: null,
        room: null,
        remoteDisplay: null,
        set: function(pc, session, room) {
            state.pc = pc;
            state.session = session;
            state.room = room;
        },
        clear: function() {
            state.room = null;
            state.session = null;
            state.pc = null;
        },
        setDisplay: function(display) {
            state.remoteDisplay = display;
        },
        disposeDisplay: function() {
            if (state.remoteDisplay) {
                state.remoteDisplay.stop();
                state.remoteDisplay = null;
            }
        }
    };
    return state;
}
```

3. Initialization

`init()` code

The `init()` function is called on page load and:

- initializes state objects
- initializes input fields

```
const init = function() {
    $("#playBtn").prop('disabled', true);
    $("#url").prop('disabled', true);
    $("#streamName").prop('disabled', true);
    onDisconnected(CurrentState());
```

```
    $("#url").val(setURL());
}
```

4. Establishing server connection

`connect()`, `SFU.createRoom()` code

The `connect()` function is called by `Play` button click and does the following:

- creates `PeerConnection` object
- cleans the previous session state displayed
- sets up room configuration and creates Websocket session
- subscribes to Websocket session events

```
const connect = function(state) {
    // Create peer connection
    let pc = new RTCPeerConnection();
    // Create a config to connect to SFU room
    const roomConfig = {
        // Server websocket URL
        url: $("#url").val(),
        // Use stream name as room name to play ABR
        roomName: $("#streamName").val(),
        // Make a random participant name from stream name
        nickname: "Player-" + $("#streamName").val() + "-" + createUUID(4),
        // Set room pin
        pin: 123456
    }
    // Clean state display items
    setStatus("playStatus", "");
    setStatus("playErrorInfo", "");
    // Connect to the server (room should already exist)
    const session = sfu.createRoom(roomConfig);
    session.on(constants.SFU_EVENT.CONNECTED, function() {
        state.set(pc, session, session.room());
        onConnected(state);
        setStatus("playStatus", "CONNECTING...", "black");
    }).on(constants.SFU_EVENT.DISCONNECTED, function() {
        state.clear();
        onDisconnected(state);
        setStatus("playStatus", "DISCONNECTED", "green");
    }).on(constants.SFU_EVENT.FAILED, function(e) {
        state.clear();
        onDisconnected(state);
        setStatus("playStatus", "FAILED", "red");
        setStatus("playErrorInfo", e.status + " " + e.statusText, "red");
    });
}
```

5. Playback start after session establishing

`onConnected()` code

The `onConnected()` function:

- sets up `Stop` button click actions
- subscribes to `SFU_ROOM_EVENT.PARTICIPANT_LIST` event to check if the stream is published in the room
- subscribes to room error events
- calls playback function

```
const onConnected = function(state) {
    $("#playBtn").text("Stop").off('click').click(function () {
        onStopClick(state);
    }).prop('disabled', false);
    $('#url').prop('disabled', true);
    $("#streamName").prop('disabled', true);
    // Add room event handling
    state.room.on(constants.SFU_ROOM_EVENT.PARTICIPANT_LIST, function(e) {
        // If the room is empty, the stream is not published yet
        if(!e.participants || e.participants.length === 0) {
            setStatus("playErrorInfo", "ABR stream is not published", "red");
            onStopClick(state);
        }
        else {
            setStatus("playStatus", "ESTABLISHED", "green");
            $("#placeholder").hide();
        }
    }).on(constants.SFU_ROOM_EVENT.FAILED, function(e) {
        // Display error state
        setStatus("playErrorInfo", e, "red");
    }).on(constants.SFU_ROOM_EVENT.OPERATION_FAILED, function (e) {
        // Display the operation failed
        setStatus("playErrorInfo", e.operation + " failed: " + e.error,
"red");
    }).on(constants.SFU_ROOM_EVENT.ENDED, function () {
        // Publishing is stopped, dispose playback and close connection
        setStatus("playErrorInfo", "ABR stream is stopped", "red");
        onStopClick(state);
    });
    playStreams(state);
}
```

6. Streams playback

`playStreams()`, `Room.join()` code

The `playStreams()` function:

- initializes a base container tag to display incoming media streams
- negotiates WebRTC connection

```
const playStreams = function(state) {
    // Create remote display item to show remote streams
    state.setDisplay(initRemoteDisplay({
        div: document.getElementById("remoteVideo"),
        room: state.room,
        peerConnection: state.pc,
        displayOptions: {
            publisher: false,
            quality: true
        }
    }));
    state.room.join(state.pc);
}
```

7. Playback stopping

`stopStreams()`, `CurrentState.disposeDisplay()` code

```
const stopStreams = function(state) {
    state.disposeDisplay();
}
```

8. `Play` click action

`onStartClick()`, `playFirstSound()`, `connect()` code

The `onStartClick()` function:

- validates input fields
- in Safari browser, calls `playFirstSound()` before playback to automatically play incoming audio
- calls `connect()` function

```
const onStartClick = function(state) {
    if (validateForm("connectionForm")) {
        $("#playBtn").prop('disabled', true);
        if (Browser().isSafariWebRTC()) {
            playFirstSound(document.getElementById("main"),
PRELOADER_URL).then(function () {
                connect(state);
            });
        } else {
            connect(state);
        }
    }
}
```

9. `Stop` click actions

`onStopClick()`, `Session.disconnect()` code

The `onStopClick()` function:

- stops playback
- disconnects Websocket session

```
const onStopClick = function(state) {
    $("#playBtn").prop('disabled', true);
    stopStreams(state);
    state.session.disconnect();
}
```

10. Websocket session disconnection actions

`onDisconnected()` code

The `onDisconnected()` function:

- sets up `Play` click actions
- enables `Server url` and `Room name` fields access

```
const onDisconnected = function(state) {
    $("#placeholder").show();
    $("#playBtn").text("Play").off('click').click(function () {
        onStartClick(state);
    }).prop('disabled', false);
    $('#url').prop('disabled', false);
    $("#streamName").prop('disabled', false);
}
```