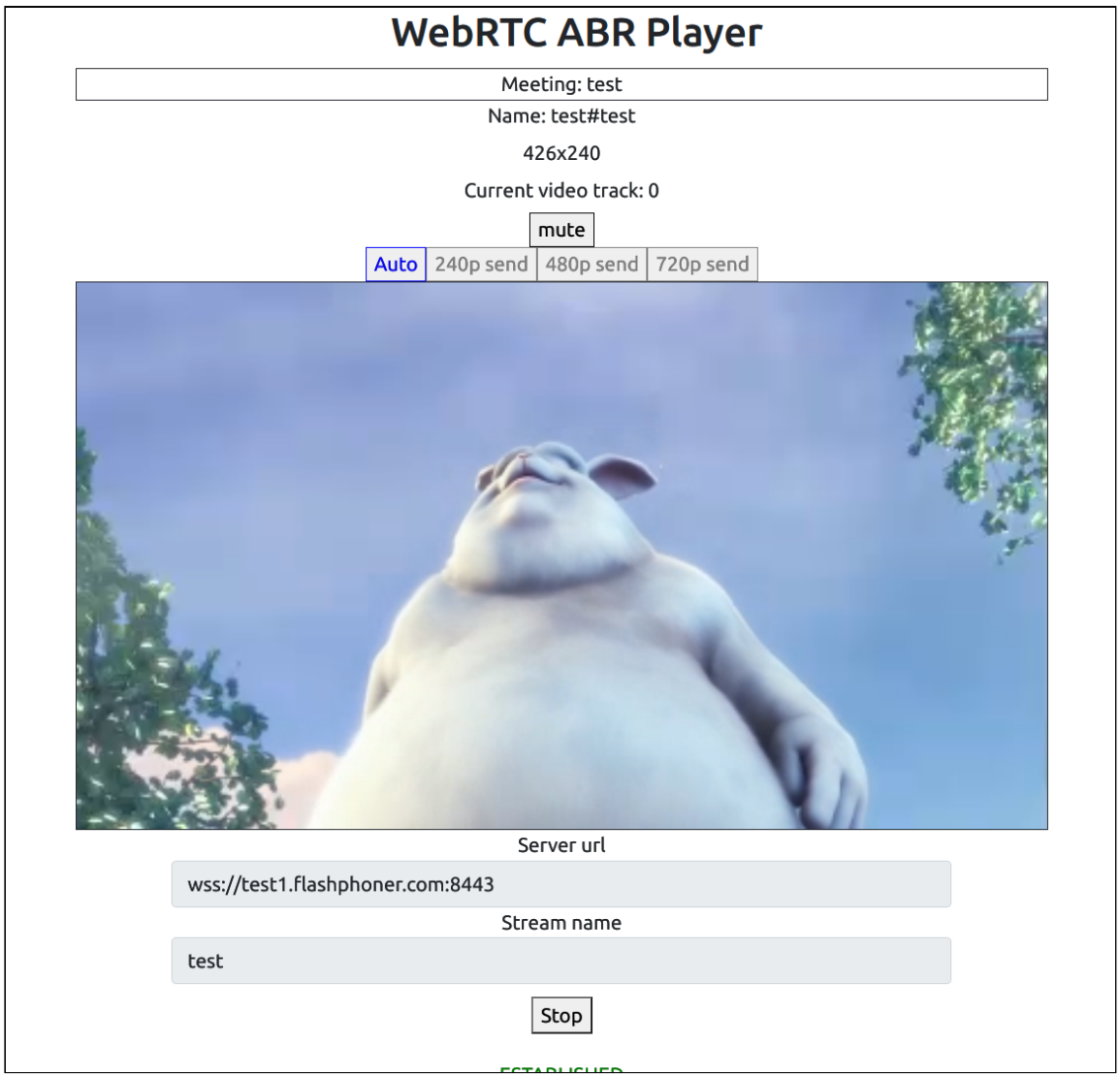
WebRTC ABR Player

Пример демонстрирует проигрывание потока, опубликованного на WCS сервере по WebRTC в нескольких качествах видео.

На скриншотах ниже:

- Server url - Websocket URL WCS сервера
- Stream name - имя потока
- Auto, 240p send, 480p send, 720p send - кнопки переключения принимаемого качества по именам профилей из файла `/usr/local/FlashphonerWebCallServer/conf/wcs_sfu_bridge_profiles.yml`



Обратите внимание, что аудио дорожка проигрывается в отдельном элементе.

Исходный код примера

Исходный код разбит на следующие модули:

- player.html - HTML страница
- player.css - стили HTML страницы
- player.js - основная логика приложения

Анализ исходного кода

Для работы с исходным кодом примера возьмем версию файла player.js, доступную здесь

1. Локальные переменные

Объявление локальных переменных для работы с константами, SFU SDK, для отображения видео и работы с конфигурацией клиента

code

```
const constants = SFU.constants;
const sfu = SFU;
const PRELOADER_URL="../commons/media/silence.mp3";
const playStatus = "playStatus";
const playErrorInfo = "playErrorInfo";
```

2. Объект для хранения текущего состояния проигрывания

Хранит данные Websocket сессии, WebRTC соединения, SFU комнаты и объекта для отображения аудио и видео

code

```
const CurrentState = function() {
    let state = {
        pc: null,
        session: null,
        room: null,
        display: null,
        roomEnded: false,
        set: function(pc, session, room) {
            state.pc = pc;
            state.session = session;
            state.room = room;
            state.roomEnded = false;
        },
        clear: function() {
            state.room = null;
```

```
            state.session = null;
            state.pc = null;
            state.roomEnded = false;
        },
        setRoomEnded: function() {
            state.roomEnded = true;
        },
        isRoomEnded: function() {
            return state.roomEnded;
        },
        isConnected: function() {
            return (state.session && state.session.state() ===
constants.SFU_STATE.CONNECTED);
        },
        isActive: function() {
            return (state.room && !state.roomEnded && state.pc);
        },
        setDisplay: function (display) {
            state.display = display;
        },
        disposeDisplay: function () {
            if (state.display) {
                state.display.stop();
                state.display = null;
            }
        }
    };
    return state;
}
```

3. Инициализация

`init()` code

Функция `init()` вызывается после того, как страница загрузится:

- инициализирует объекты состояния
- инициализирует поля ввода

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

4. Соединение с сервером

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

Функция `connect()` вызывается по нажатию кнопки `Play`:

- создает объект `PeerConnection`
- очищает отображение статуса предыдущей сессии
- настраивает конфигурацию комнаты и создает Websocket сессию
- подписывается на события Websocket сессии

```
const connect = async 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, "");
    try {
        // Connect to the server (room should already exist)
        const session = await sfu.createRoom(roomConfig);
        // Set up session ending events
        session.on(constants.SFU_EVENT.DISCONNECTED, function() {
            onStopClick(state);
            onDisconnected(state);
            setStatus(playStatus, "DISCONNECTED", "green");
        }).on(constants.SFU_EVENT.FAILED, function(e) {
            onStopClick(state);
            onDisconnected(state);
            setStatus(playStatus, "FAILED", "red");
            if (e.status && e.statusText) {
                setStatus(playErrorInfo, e.status + " " + e.statusText,
"red");
            } else if (e.type && e.info) {
                setStatus(playErrorInfo, e.type + ": " + e.info, "red");
            }
        });
        // Connected successfully
        onConnected(state, pc, session);
        setStatus(playStatus, "CONNECTING...", "black");
    } catch(e) {
        onDisconnected(state);
        setStatus(playStatus, "FAILED", "red");
        setStatus(playErrorInfo, e, "red");
    }
}
```

5. Запуск проигрывания при установке соединения

`onConnected()` code

The `onConnected()` function:

- настраивает действия по нажатию кнопки `Stop`
- подписывается на событие `SFU_ROOM_EVENT.PARTICIPANT_LIST` для проверки, опубликован ли поток в SFU комнате
- подписывается на события об ошибках комнаты
- вызывает функцию проигрывания

```
const onConnected = async function(state, pc, session) {
    state.set(pc, session, session.room());
    $("#playBtn").text("Stop").off('click').click(function () {
        onStopClick(state);
    });
    $('#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) {
        onOperationFailed(state);
    }).on(constants.SFU_ROOM_EVENT.ENDED, function () {
        // Publishing is stopped, dispose playback and close connection
        setStatus(playErrorInfo, "ABR stream is stopped", "red");
        state.setRoomEnded();
        onStopClick(state);
    }).on(constants.SFU_ROOM_EVENT.DROPPED, function () {
        // Client dropped from the room, dispose playback and close
connection
        setStatus(playErrorInfo, "Playback is dropped due to network issues",
"red");
        state.setRoomEnded();
        onStopClick(state);
    });
    await playStreams(state);
    // Enable button after starting playback #WCS-3635
    $("#playBtn").prop('disabled', false);
}
```

6. Проигрывание потоков

`playStreams()`, `initRemoteDisplay()`, `SFURoom.join()` code

Функция `playStreams()`:

- инициализирует базовый элемент для отображения входящих медиа потоков
- настраивает ABR для переключения между доступными качествами видео при изменении параметров канала
- настраивает WebRTC соединение в комнате

```
const playStreams = async function (state) {
    try {
        // Create remote display item to show remote streams
        const display = initRemoteDisplay(state.room,
document.getElementById("remoteVideo"), {quality:true, autoAbr: true},
{thresholds: [
            {parameter: "nackCount", maxLeap: 10},
        {parameter: "freezeCount", maxLeap: 10},
        {parameter: "packetsLost", maxLeap: 10}
    ], abrKeepOnGoodQuality: ABR_KEEP_ON_QUALITY, abrTryForUpperQuality:
ABR_TRY_UPPER_QUALITY, interval:
ABR_QUALITY_CHECK_PERIOD},createDefaultMeetingController,
createDefaultMeetingModel, createDefaultMeetingView,
oneToOneParticipantFactory(remoteTrackProvider(state.room)));
        state.setDisplay(display);
        // Start WebRTC negotiation
        await state.room.join(state.pc, null, null, 1);
    } catch(e) {
        if (e.type === constants.SFU_ROOM_EVENT.OPERATION_FAILED) {
            onOperationFailed(state, e);
        } else {
            console.error("Failed to play streams: " + e);
            setStatus(playErrorInfo, e.name, "red");
            onStopClick(state);
        }
    }

}
```

7. Остановка проигрывания

`CurrentState.disposeDisplay()` code

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

8. Действия по нажатию кнопки Play

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

Функция `onStartClick()`:

- проверяет правильность заполнения полей ввода
- перед стартом воспроизведения, в браузере Safari вызывает функцию `playFirstSound()` для автоматического проигрывания аудио
- вызывает функцию `connect()`

```
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. Действия по нажатию кнопки Stop

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

Функция `onStopClick()`:

- останавливает воспроизведение
- разрывает Websocket сессию

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

10. Действия при разрыве Websocket сессии

`onDisconnected()` code

Функция `onDisconnected()`:

- настраивает действия по нажатию кнопки `Play`
- открывает доступ к полям ввода `Server url` и `Stream name`

```
const onDisconnected = function(state) {
    state.clear();
```

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