main.js

1. Local variables

Declare local variables for constants, SFU SDK, local display and controls

code

```
const constants = SFU.constants;
const sfu = SFU;
let localDisplay;
let cControls;
```

2. Default configuration

Declare default room and publishing configuration which will be used if there was no `config.json` file available

code

With this config client will be preconfigured to connect to localhost over WSS, enter room "ROOM1" with pin "1234" and nickname "Alice". Media section directs client to publish audio and video tracks. Video will have two sub-tracks - high (h) and medium (m).

```
const defaultConfig = {
    room: {
        url: "wss://127.0.0.1:8888",
        name: "ROOM1",
        pin: "1234",
        nickName: "Alice"
    },
    media: {
        audio: {
            tracks: [
                {
                    source: "mic",
                    channels: 1
                }
            ]
        },
        video: {
            tracks: [
                {
                    source: "camera",
                    width: 1280,
                    height: 720,
                    codec: "H264",
```

```
                    encodings: [
                        {rid: "m", active: true, maxBitrate: 300000,
scaleResolutionDownBy: 2},
                        {rid: "h", active: true, maxBitrate: 900000}
                    ]
                }
            ]
        }
    }
};
```

3. Initialization

`init()` code

Init function is called when page is finished loading. The function will load config.json or default config and open entrance modal window.

```
/**
 * load config and show entrance modal
 */
const init = function () {
    //read config
    $.getJSON("config.json", function (config) {
        cControls = createControls(config);
    }).fail(function () {
        //use default config
        cControls = createControls(defaultConfig);
    });
    //open entrance modal
    $('#entranceModal').modal('show');
}
```

4. Connect to the server and create or enter to the room

`connect()` code

Connect function that is called once user clicks Enter in entrance modal window.

```
/**
 * connect to server
 */
async function connect() {
    // hide modal
    $('#entranceModal').modal('hide');
    // disable controls
    cControls.muteInput();
    //create peer connection
    const pc = new RTCPeerConnection();
    //get config object for room creation
    const roomConfig = cControls.roomConfig();
```

```
    //kick off connect to server and local room creation
    try {
        const session = await sfu.createRoom(roomConfig);
        // Now we connected to the server (if no exception was thrown)
        session.on(constants.SFU_EVENT.FAILED, function (e) {
            if (e.status && e.statusText) {
                displayError("CONNECTION FAILED: " + e.status + " " +
e.statusText);
            } else if (e.type && e.info) {
                displayError("CONNECTION FAILED: " + e.info);
            } else {
                displayError("CONNECTION FAILED: " + e);
            }
        }).on(constants.SFU_EVENT.DISCONNECTED, function (e) {
            displayError("DISCONNECTED. Refresh the page to enter the room
again");
        });
        const room = session.room();
        room.on(constants.SFU_ROOM_EVENT.FAILED, function (e) {
            displayError(e);
        }).on(constants.SFU_ROOM_EVENT.OPERATION_FAILED, function (e) {
            displayError(e.operation + " failed: " + e.error);
        })

        // create local display to show local streams
        localDisplay =
initLocalDisplay(document.getElementById("localDisplay"));
        // display audio and video control tables
        await cControls.displayTables();
        cControls.onTrack(async function (s) {
            await publishNewTrack(room, pc, s);
        });
        //create and bind chat to the new room
        const chatDiv = document.getElementById('messages');
        const chatInput = document.getElementById('localMessage');
        const chatButton = document.getElementById('sendMessage');
        createChat(room, chatDiv, chatInput, chatButton);

        //setup remote display for showing remote audio/video tracks
        const remoteDisplay = document.getElementById("display");
        initDefaultRemoteDisplay(room, remoteDisplay, {quality: 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});

        //get configured local video streams
        let streams = cControls.getVideoStreams();
        //combine local video streams with audio streams
        streams.push.apply(streams, cControls.getAudioStreams());

        // Publish preconfigured streams
        publishPreconfiguredStreams(room, pc, streams);
    } catch (e) {
        console.error(e);
```

```
        displayError(e);
    }
}
```

5. connect() function details

Hide modal as we don't need it anymore and mute controls before connection is established

code

```
async function connect() {
    // hide modal
    $('#entranceModal').modal('hide');
    // disable controls
    cControls.muteInput();
    ...
}
```

Create PeerConnection and prepare the room config for the creation of session and room

code

```
async function connect() {
    ...
    //create peer connection
    const pc = new RTCPeerConnection();
    //get config object for room creation
    const roomConfig = cControls.roomConfig();
    ...
}
```

Create session (which will automatically connect to the server)

code

```
async function connect() {
    ...
    //kick off connect to server and local room creation
    try {
        const session = await sfu.createRoom(roomConfig);
        ...
    } catch (e) {
        console.error(e);
        displayError(e);
    }
}
```

Subscribe to session's events

code

```
async function connect() {
    ...
    //kick off connect to server and local room creation
    try {
        ...
        // Now we connected to the server (if no exception was thrown)
        session.on(constants.SFU_EVENT.FAILED, function (e) {
            if (e.status && e.statusText) {
                displayError("CONNECTION FAILED: " + e.status + " " +
e.statusText);
            } else if (e.type && e.info) {
                displayError("CONNECTION FAILED: " + e.info);
            } else {
                displayError("CONNECTION FAILED: " + e);
            }
        }).on(constants.SFU_EVENT.DISCONNECTED, function (e) {
            displayError("DISCONNECTED. Refresh the page to enter the room
again");
        });
        ...
    } catch (e) {
        console.error(e);
        displayError(e);
    }
}
```

Create a room object and subscribe to error events

code

```
async function connect() {
    ...
    //kick off connect to server and local room creation
    try {
        ...
        const room = session.room();
        room.on(constants.SFU_ROOM_EVENT.FAILED, function (e) {
            displayError(e);
        }).on(constants.SFU_ROOM_EVENT.OPERATION_FAILED, function (e) {
            displayError(e.operation + " failed: " + e.error);
        })
        ...
    } catch (e) {
        console.error(e);
        displayError(e);
    }
}
```

Create an object to display local video

code

```
async function connect() {
    ...
    //kick off connect to server and local room creation
```

```
    try {
        ...
        // create local display to show local streams
        localDisplay =
initLocalDisplay(document.getElementById("localDisplay"));
        // display audio and video control tables
        await cControls.displayTables();
        cControls.onTrack(async function (s) {
            await publishNewTrack(room, pc, s);
        });
        ...
    } catch (e) {
        console.error(e);
        displayError(e);
    }
}
```

Initialize chat window

code

```
async function connect() {
    ...
    //kick off connect to server and local room creation
    try {
        ...
        //create and bind chat to the new room
        const chatDiv = document.getElementById('messages');
        const chatInput = document.getElementById('localMessage');
        const chatButton = document.getElementById('sendMessage');
        createChat(room, chatDiv, chatInput, chatButton);
        ...
    } catch (e) {
        console.error(e);
        displayError(e);
    }
}
```

Initialize an object to display other participants tracks

code

```
async function connect() {
    ...
    //kick off connect to server and local room creation
    try {
        ...
        //setup remote display for showing remote audio/video tracks
        const remoteDisplay = document.getElementById("display");
        initDefaultRemoteDisplay(room, remoteDisplay, {quality: 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});
        ...
    } catch (e) {
        console.error(e);
        displayError(e);
    }
}
```

Get preconfigured local media parameters and publish local media streams

code

```
async function connect() {
    ...
    //kick off connect to server and local room creation
    try {
        ...
        //get configured local video streams
        let streams = cControls.getVideoStreams();
        //combine local video streams with audio streams
        streams.push.apply(streams, cControls.getAudioStreams());

        // Publish preconfigured streams
        publishPreconfiguredStreams(room, pc, streams);
    } catch (e) {
        console.error(e);
        displayError(e);
    }
}
```

6. Enter the room and publish a local media tracks

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

```
const publishPreconfiguredStreams = async function (room, pc, streams) {
    try {
        const config = {};
        //add our local streams to the room (to PeerConnection)
        streams.forEach(function (s) {
            let contentType = s.type || s.source;
            //add each track to PeerConnection
            s.stream.getTracks().forEach((track) => {
                config[track.id] = contentType;
                addTrackToPeerConnection(pc, s.stream, track, s.encodings);
                subscribeTrackToEndedEvent(room, track, pc);
            });
            localDisplay.add(s.stream.id, "local", s.stream, contentType);
        });
        //join room
        await room.join(pc, null, config, 10);
        // Enable Delete button for each preconfigured stream #WCS-3689
        streams.forEach(function (s) {
            $('#' + s.stream.id + "-button").prop('disabled', false);
```

```
        });
    } catch (e) {
        onOperationFailed("Failed to publish a preconfigured streams", e);
        // Enable Delete button for each preconfigured stream #WCS-3689
        streams.forEach(function (s) {
            $('#' + s.stream.id + "-button").prop('disabled', false);
        });
    }
}
```

7. Publish an additional local tracks

`publishNewTrack()`, `Room.updateState()` code

```
const publishNewTrack = async function (room, pc, media) {
    try {
        let config = {};
        //add local stream to local display
        let contentType = media.type || media.source;

        localDisplay.add(media.stream.id, "local", media.stream,
contentType);
        //add each track to PeerConnection
        media.stream.getTracks().forEach((track) => {
            config[track.id] = contentType;
            addTrackToPeerConnection(pc, media.stream, track,
media.encodings);
            subscribeTrackToEndedEvent(room, track, pc);
        });
        // Clean error message
        displayError("");
        //kickoff renegotiation
        await room.updateState(config);
        // Enable Delete button for a new stream #WCS-3689
        $('#' + media.stream.id + "-button").prop('disabled', false);
    } catch (e) {
        onOperationFailed("Failed to publish a new track", e);
        // Enable Delete button for a new stream #WCS-3689
        $('#' + media.stream.id + "-button").prop('disabled', false);
    }
}
```

8. Finalizing local track

`subscribeTrackToEndedEvent()` code

This is a helper function that subscribes new local track to "ended" event. Once event fired we remove track from peer connection and kickoff renegotiation.

```
const subscribeTrackToEndedEvent = function (room, track, pc) {
    track.addEventListener("ended", async function () {
```

```
        try {
            //track ended, see if we need to cleanup
            let negotiate = false;
            for (const sender of pc.getSenders()) {
                if (sender.track === track) {
                    pc.removeTrack(sender);
                    //track found, set renegotiation flag
                    negotiate = true;
                    break;
                }
            }
            // Clean error message
            displayError("");
            if (negotiate) {
                //kickoff renegotiation
                await room.updateState();
            }
        } catch (e) {
            onOperationFailed("Failed to update room state", e);
        }
    });
}
```

9. Add new local track to peer connection

`addTrackToPeerConnection()` code

This is a helper function which adds new local track to peer connection.

```
const addTrackToPeerConnection = function(pc, stream, track, encodings) {
    pc.addTransceiver(track, {
        direction: "sendonly",
        streams: [stream],
        sendEncodings: encodings ? encodings : [] //passing encoding types
for video simulcast tracks
    });
}
```