main.js - main application logic

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

3. Initialization

init() code

Init function that is called when page is finished loading. The function will load config.json or default config, create local display 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);
    });
    //create local display to show local streams
    localDisplay = initLocalDisplay(document.getElementById("localDisplay"));
    //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
 */
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();
    roomConfig.pc = pc;
    //kick off connect to server and local room creation
    const session = sfu.createRoom(roomConfig);
    session.on(constants.SFU_EVENT.CONNECTED, function(room) {
        //connected to server
        const chatDiv = document.getElementById('messages');
        const chatInput = document.getElementById('localMessage');
        const chatButton = document.getElementById('sendMessage');
        //create and bind chat to the new room
        createChat(room, chatDiv, chatInput, chatButton);

        room.on(constants.SFU_ROOM_EVENT.FAILED, function(e) {
            const errField = document.getElementById("errorMsg");
            errField.style.color = "red";
            errField.innerText = e;
        }).on(constants.SFU_ROOM_EVENT.OPERATION_FAILED, function (e) {
            const errField = document.getElementById("errorMsg");
            errField.style.color = "red";
            errField.innerText = e.operation + " failed: " + e.error;
        })
        //setup remote display for showing remote audio/video tracks
        const remoteDisplay = document.getElementById("display");
        initRemoteDisplay(room, remoteDisplay, pc);

        //get configured local video streams
        let streams = cControls.getVideoStreams();
        //combine local video streams with audio streams
        streams.push.apply(streams, cControls.getAudioStreams());
        //add our local streams to the room (to PeerConnection)
        streams.forEach(function (s) {
            //add local stream to local display
            localDisplay.add(s.stream.id, "local", s.stream);
            //add each track to PeerConnection
            s.stream.getTracks().forEach((track) => {
                addTrackToPeerConnection(pc, s.stream, track, s.encodings);
                subscribeTrackToEndedEvent(room, track, pc);
            });
        });
        //add callback for the new local stream to the local controls
        cControls.onTrack(function (s) {
            //add local stream to local display
            localDisplay.add(s.stream.id, "local", s.stream);
            //add each track to PeerConnection
            s.stream.getTracks().forEach((track) => {
                addTrackToPeerConnection(pc, s.stream, track, s.encodings);
                subscribeTrackToEndedEvent(room, track, pc);
            });
            //kickoff renegotiation
            room.updateState();
        });
        //join room
        room.join();
    });
}
```

5. connect() function details

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

code

```
//hide modal
$('#entranceModal').modal('hide');
//disable controls
cControls.muteInput();
```

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

code

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

Create session (which will automatically connect to the server)

code

```
const session = sfu.createRoom(roomConfig);
```

Subscribe to session's "CONNECTED" event

code

```
session.on(constants.SFU_EVENT.CONNECTED, function(room) {
```

Once we are connected initialize room chat

code

```
//connected to server
const chatDiv = document.getElementById('messages');
const chatInput = document.getElementById('localMessage');
const chatButton = document.getElementById('sendMessage');
//create and bind chat to the new room
createChat(room, chatDiv, chatInput, chatButton);
```

Subscribe to room error events

code

```
room.on(constants.SFU_ROOM_EVENT.FAILED, function(e) {
    const errField = document.getElementById("errorMsg");
    errField.style.color = "red";
    errField.innerText = e;
}).on(constants.SFU_ROOM_EVENT.OPERATION_FAILED, function (e) {
    const errField = document.getElementById("errorMsg");
    errField.style.color = "red";
    errField.innerText = e.operation + " failed: " + e.error;
})
```

Initialize remote display

code

```
//setup remote display for showing remote audio/video tracks
const remoteDisplay = document.getElementById("display");
initRemoteDisplay(room, remoteDisplay, pc);
```

Get preconfigured local media from controls

code

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

Add each stream to local display (so we can see it on page) and peer connection

code

```
        //add our local streams to the room (to PeerConnection)
        streams.forEach(function (s) {
            //add local stream to local display
            localDisplay.add(s.stream.id, "local", s.stream);
            //add each track to PeerConnection
            s.stream.getTracks().forEach((track) => {
                if (s.source === "screen") {
                    config[track.id] = s.source;
                }
                addTrackToPeerConnection(pc, s.stream, track, s.encodings);
                subscribeTrackToEndedEvent(room, track, pc);
            });
        });
```

Add listener to controls so we know if user adds new local streams. Once we have a new stream we will need to add it to local display, add it to peer connection and kickoff renegotiation

code

```
        //add callback for the new local stream to the local controls
        cControls.onTrack(function (s) {
            let config = {};
            //add local stream to local display
            localDisplay.add(s.stream.id, "local", s.stream);
            //add each track to PeerConnection
            s.stream.getTracks().forEach((track) => {
                if (s.source === "screen") {
                    config[track.id] = s.source;
                }
                addTrackToPeerConnection(pc, s.stream, track, s.encodings);
                subscribeTrackToEndedEvent(room, track, pc);
            });
            //kickoff renegotiation
            room.updateState(config);
        });
```

WebRTC negotiation in the room

code

```
//join room
room.join(pc, null, config);
```

6. 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", function() {
        //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;
            }
        }
        if (negotiate) {
            //kickoff renegotiation
            room.updateState();
        }
    });
};
```

7. 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
    });
}
```