display.js - video and audio capturing and displaying

The functions to create and destroy HTML5 tags to capture and display video and audio are moved to `display.js` module

Analyzing the source code

To analyze the source code take the version available here

Local video capturing and displaying

1. Initialization

`initLocalDisplay()` code

The `initLocalDisplay()` returns the object to work with HTML5 tags to capture and display local video and audio

```
const initLocalDisplay = function(localDisplayElement){
    const localDisplayDiv = localDisplayElement;
    const localDisplays = {};

    const removeLocalDisplay = function(id) {
        ...
    }

    const getAudioContainer = function() {
        ...
    };

    const add = function(id, name, stream) {
        ...
    }

    const stop = function () {
        ...
    }

    const audioStateText = function (stream) {
        ...
    }

    return {
        add: add,
        stop: stop
```

```
    }
}
```

2. Adding HTML tags to capture and display local video/audio

2.1. ADD AUDIO TRACK TO HTML5 VIDEO TAG

`add()` code

Where:

- audio track is added to video tag
- `onended` event handler is added to audio track

```
if (stream.getAudioTracks().length > 0) {
    let videoElement = getAudioContainer();
    if (videoElement) {
        let track = stream.getAudioTracks()[0];
        videoElement.video.srcObject.addTrack(track);
        videoElement.audioStateDisplay.innerHTML = audioStateText(stream);
        track.addEventListener("ended", function() {
            videoElement.video.srcObject.removeTrack(track);
            videoElement.audioStateDisplay.innerHTML = "No audio";
            //check video element has no tracks left
            for (const [key, vTrack] of
Object.entries(videoElement.video.srcObject.getTracks())) {
                if (vTrack.readyState !== "ended") {
                    return;
                }
            }
            removeLocalDisplay(videoElement.id);
        });
        return;
    }
}
```

2.2. CONTAINER CREATION TO DISPLAY LOCAL VIDEO

`add()` code

Where:

- container `div` tag to display local video is created
- `div` tag to display video information is created

```
const coreDisplay = document.createElement('div');
coreDisplay.setAttribute("class","text-center");
coreDisplay.setAttribute("style","width: auto; height: auto;");
coreDisplay.id = stream.id;
const streamNameDisplay = document.createElement("div");
streamNameDisplay.innerHTML = "Name: " + name;
streamNameDisplay.setAttribute("class","text-center");
streamNameDisplay.setAttribute("style","width: auto; height: auto;");
coreDisplay.appendChild(streamNameDisplay);
```

2.3. BUTTON CREATION TO MUTE/UNMUTE LOCAL AUDIO

`add()` code

Where:

- button to mute/unmute local audio is created
- `click` event handler for the button is added

```
const audioStateDisplay = document.createElement("button");
audioStateDisplay.innerHTML = audioStateText(stream);
audioStateDisplay.addEventListener('click', function(){
    if (stream.getAudioTracks().length > 0) {
        stream.getAudioTracks()[0].enabled = !(stream.getAudioTracks()
[0].enabled);
        audioStateDisplay.innerHTML = audioStateText(stream);
    }
});
coreDisplay.appendChild(audioStateDisplay);
```

2.4. TAG CREATION TO DISPLAY LOCAL VIDEO

`add()` code

Where:

- contaner tag which can be resized to a parent node is created
- HTML5 `video` tag is created (considering Safari publishing)

```
const streamDisplay = document.createElement('div');
streamDisplay.id = "stream-" + id;
streamDisplay.setAttribute("class","text-center");
streamDisplay.setAttribute("style","width: auto; height: auto;");
coreDisplay.appendChild(streamDisplay);
const video = document.createElement("video");
video.muted = true;
if(Browser().isSafariWebRTC()) {
    video.setAttribute("playsinline", "");
    video.setAttribute("webkit-playsinline", "");
}
streamDisplay.appendChild(video);
video.srcObject = stream;
```

2.5. VIDEO TAG EVENT HANDLERS CREATION

`add()` code

Where:

- local video playback is started
- `onended` event handler is set up for video track

- `onresize` event handler is set up for local video to adjust video displaying size to the container dimensions

```
video.onloadedmetadata = function (e) {
    video.play();
};
stream.getTracks().forEach(function(track){
    track.addEventListener("ended", function() {
        video.srcObject.removeTrack(track);
        //check video element has no tracks left
        for (const [key, vTrack] of
Object.entries(video.srcObject.getTracks())) {
            if (vTrack.readyState !== "ended") {
                return;
            }
        }
        removeLocalDisplay(id);
    });
});
video.addEventListener('resize', function (event) {
    streamNameDisplay.innerHTML = "Name: " + name + "<br/>Max.resolution: " +
video.videoWidth + "x" + video.videoHeight;
    resizeVideo(event.target);
});
```

2.6. VIDEO CONTAINER ADDITION TO HTML PAGE

`add()` code

```
localDisplays[id] = coreDisplay;
localDisplayDiv.appendChild(coreDisplay);
return coreDisplay;
```

3. Stop video and audio capturing

`stop()` code

```
const stop = function () {
    for (const [key, value] of Object.entries(localDisplays)) {
        removeLocalDisplay(value.id);
    }
}
```

Room streams published displaying

1. Initialization

`initRemoteDisplay()` code

The `initRemoteDisplay()` function returns the object to work with HTML5 tags to display remote video and audio streams

```
const initRemoteDisplay = function(mainDiv, room, peerConnection) {
    const constants = SFU.constants;
    const remoteParticipants = {};
    ...
    const createRemoteDisplay = function(id, name, mainDiv) {
        ...
    }

    const stop = function() {
        ...
    }

    peerConnection.ontrack = ({transceiver}) => {
        ...
    }

    return {
        stop: stop
    }
}
```

2. Room events handling

2.1. ADD_TRACKS

`initRemoteDisplay()` code

Where:

- a new participant is added to participants list
- tracks quality information is added to tracks list
- elements to display remote audion and video are created

```
room.on(constants.SFU_ROOM_EVENT.ADD_TRACKS, function(e) {
    console.log("Received ADD_TRACKS");
    let participant = remoteParticipants[e.info.nickName];
    if (!participant) {
        participant = {};
        participant.nickName = e.info.nickName;
        participant.tracks = [];
        participant.displays = [];
        remoteParticipants[participant.nickName] = participant;
    }
    participant.tracks.push.apply(participant.tracks, e.info.info);
    for (const pTrack of e.info.info) {
        ...
        let display = createRemoteDisplay(participant.nickName,
participant.nickName, mainDiv);
        participant.displays.push(display);
        if (pTrack.type === "VIDEO") {
            display.videoMid = pTrack.mid;
            display.setTrackInfo(pTrack);
        } else if (pTrack.type === "AUDIO") {
            display.audioMid = pTrack.mid;
```

```
        }
    }
    ...
});
```

2.2. REMOVE_TRACKS

`initRemoteDisplay()` code

Where:

- video elements are removed
- tracks data are dleteted from tracks list

```
}).on(constants.SFU_ROOM_EVENT.REMOVE_TRACKS, function(e) {
    console.log("Received REMOVE_TRACKS");
    const participant = remoteParticipants[e.info.nickName];
    if (!participant) {
        return;
    }
    for (const rTrack of e.info.info) {
        for (let i = 0; i < participant.tracks.length; i++) {
            if (rTrack.mid === participant.tracks[i].mid) {
                participant.tracks.splice(i, 1);
                break;
            }
        }
        for (let i = 0; i < participant.displays.length; i++) {
            let found = false;
            const display = participant.displays[i];
            if (display.audioMid === rTrack.mid) {
                display.setAudio(null);
                found = true;
            } else if (display.videoMid === rTrack.mid) {
                display.setVideo(null);
                found = true;
            }
            if (found) {
                if (!display.hasAudio() && !display.hasVideo()) {
                    display.dispose();
                    participant.displays.splice(i, 1);
                }
                break;
            }
        }
    }
});
```

2.3. LEFT

`initRemoteDisplay()` code

Where:

- participant is removed from participants list

- video elements are removed

```
}).on(constants.SFU_ROOM_EVENT.LEFT, function(e) {
    console.log("Received LEFT");
    let participant = remoteParticipants[e.name];
    if (!participant) {
        return;
    }
    participant.displays.forEach(function(display){
        display.dispose();
    })
    delete remoteParticipants[e.name];
});
```

2.4. TRACK_QUALITY_STATE

`initRemoteDisplay()` code

Where:

- track quality data are updated

```
}).on(constants.SFU_ROOM_EVENT.TRACK_QUALITY_STATE, function(e){
    console.log("Received track quality state");
    const participant = remoteParticipants[e.info.nickName];
    if (!participant) {
        return;
    }

    for (const rTrack of e.info.tracks) {
        const mid = rTrack.mid;
        for (let i = 0; i < participant.displays.length; i++) {
            const display = participant.displays[i];
            if (display.videoMid === mid) {
                display.updateQualityInfo(rTrack.quality);
                break;
            }
        }
    }
});
```

3. Tags to display remote video creation

3.1. CONTAINER DIV CREATION

`createRemoteDisplay()` code

Where:

- container `div` tag for participants streams is created
- child container `div` tag for a certain stream is created
- container `div` tag for quality switch buttons is created

```
const cell = document.createElement("div");
cell.setAttribute("class", "text-center");
cell.id = id;
mainDiv.appendChild(cell);
const streamNameDisplay = document.createElement("div");
streamNameDisplay.innerHTML = "Published by: " + name;
streamNameDisplay.setAttribute("style","width:auto; height:auto;");
streamNameDisplay.setAttribute("class","text-center");
cell.appendChild(streamNameDisplay);
const qualityDisplay = document.createElement("div");
qualityDisplay.setAttribute("style","width:auto; height:auto;");
qualityDisplay.setAttribute("class","text-center");
cell.appendChild(qualityDisplay);

let qualityDivs = [];

const rootDisplay = document.createElement("div");
rootDisplay.setAttribute("style","width:auto; height:auto;");
rootDisplay.setAttribute("class","text-center");
cell.appendChild(rootDisplay);
const streamDisplay = document.createElement("div");
streamDisplay.setAttribute("style","width:auto; height:auto;");
streamDisplay.setAttribute("class","text-center");
rootDisplay.appendChild(streamDisplay);
```

3.2. VIDEO TAG ADDITION

`setVideo()` code

```
setVideo: function(stream) {
    if (video) {
        video.remove();
    }

    if (stream == null) {
        video = null;
        this.videoMid = undefined;
        qualityDivs.forEach(function(div) {
            div.remove();
        });
        qualityDivs = [];
        return;
    }
    video = document.createElement("video");
    video.muted = true;
    if(Browser().isSafariWebRTC()) {
        video.setAttribute("playsinline", "");
        video.setAttribute("webkit-playsinline", "");
    }
    streamDisplay.appendChild(video);
    video.srcObject = stream;
    video.onloadedmetadata = function (e) {
        video.play().then(function() {
            video.muted = false;
        });
    };
```

```
    video.addEventListener("resize", function (event) {
        streamNameDisplay.innerHTML = "Published by: " + name + "<br/>Current
resolution: " + video.videoWidth + "x" + video.videoHeight;
        resizeVideo(event.target);
    });
},
```

3.3. AUDIO TAG ADDITION

`setAudio()` code

```
setAudio: function(stream) {
    if (audio) {
        audio.remove();
    }
    if (!stream) {
        audio = null;
        this.audioMid = undefined;
        return;
    }
    audio = document.createElement("audio");
    audio.controls = "controls";
    audio.muted = true;
    audio.autoplay = true;
    cell.appendChild(audio);
    audio.srcObject = stream;
    audio.onloadedmetadata = function (e) {
        audio.play().then(function() {
            audio.muted = false;
        });
    };
},
```

3.4. SETTING UP QUALITY SWITCHER

`setTrackInfo()` code

```
setTrackInfo: function(trackInfo) {
    if (trackInfo && trackInfo.quality) {
        for (let i = 0; i < trackInfo.quality.length; i++) {
            const qualityDiv = document.createElement("button");
            qualityDivs.push(qualityDiv);
            qualityDiv.innerText = trackInfo.quality[i];
            qualityDiv.setAttribute("style", "display:inline-block; border:
solid; border-width: 1px");
            qualityDiv.style.color = "red";
            qualityDiv.addEventListener('click', function(){
                console.log("Clicked on quality " + trackInfo.quality[i] + "
trackId " + trackInfo.id);
                if (qualityDiv.style.color === "red") {
                    return;
                }
                for (let c = 0; c < qualityDivs.length; c++) {
                    if (qualityDivs[c].style.color !== "red") {
                        qualityDivs[c].style.color = "gray";
                    }
```

```
                }
                qualityDiv.style.color = "blue";
                room.changeQuality(trackInfo.id, trackInfo.quality[i]);
            });
            qualityDisplay.appendChild(qualityDiv);
        }
    }
},
```

3.5. REMOVING THE CONTAINER

`dispose()` code

```
dispose: function() {
    cell.remove();
},
```

4. Subscription to `PeerConnection.ontrack` event

`PeerConnection.ontrack()`, `setAudio()`, `setVideo()` code

Where:

- video or audio tag addition function is called when track is received

```
peerConnection.ontrack = ({transceiver}) => {
    let rParticipant;
    console.log("Attach remote track " + transceiver.receiver.track.id + "
kind " + transceiver.receiver.track.kind + " mid " + transceiver.mid);
    for (const [nickName, participant] of Object.entries(remoteParticipants))
{
        for (const pTrack of participant.tracks) {
            console.log("Participant " + participant.nickName + " track " +
pTrack.id + " mid " + pTrack.mid);
            if (pTrack.mid === transceiver.mid) {
                rParticipant = participant;
                break;
            }
        }
        if (rParticipant) {
            break;
        }
    }
    if (rParticipant) {
        for (const display of rParticipant.displays) {
            if (transceiver.receiver.track.kind === "video") {
                if (display.videoMid === transceiver.mid) {
                    let stream = new MediaStream();
                    stream.addTrack(transceiver.receiver.track);
                    display.setVideo(stream);
                    break;
                }
            } else if (transceiver.receiver.track.kind === "audio") {
                if (display.audioMid === transceiver.mid) {
                    let stream = new MediaStream();
```

```
                    stream.addTrack(transceiver.receiver.track);
                    display.setAudio(stream);
                    break;
                }
            }
        }
    } else {
        console.warn("Failed to find participant for track " +
transceiver.receiver.track.id);
    }
}
```

5. Playback stopping

`stop()` code

```
const stop = function() {
    for (const [nickName, participant] of Object.entries(remoteParticipants))
{
        participant.displays.forEach(function(display){
            display.dispose();
        });
        delete remoteParticipants[nickName];
    }
}
```