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 display.js module version availablehere

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
- click event handler for the audio mute/unmute button is added

```
        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) + " " + type;
                videoElement.audioStateDisplay.addEventListener("click", function() {
                    onMuteClick(videoElement.audioStateDisplay, stream, type);
                });
                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 = createContainer(null);
coreDisplay.id = stream.id;
const publisherNameDisplay = createInfoDisplay(coreDisplay, name + " " + type);
```

2.3. Button creation to mute/unmute local audio

add() code

Where:

- button to mute/unmute local audio is created

```
const audioStateDisplay = document.createElement("button");
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 = createContainer(coreDisplay);
streamDisplay.id = "stream-" + id;
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);
            });
        });
        if (stream.getVideoTracks().length > 0) {
            // Resize only if video displayed
            video.addEventListener('resize', function (event) {
                publisherNameDisplay.innerHTML = name + " " + type + " " + video.videoWidth + "x" + video.
videoHeight;
                resizeVideo(event.target);
            });
        } else {
            // Hide audio only container
            hideItem(streamDisplay);
            // Set up mute button for audio only stream
            audioStateDisplay.innerHTML = audioStateText(stream) + " " + type;
            audioStateDisplay.addEventListener("click", function() {
                onMuteClick(audioStateDisplay, stream, type);
            });
        }
```

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(options) {
    const constants = SFU.constants;
    const remoteParticipants = {};
    // Validate options first
    if (!options.div) {
        throw new Error("Main div to place all the media tag is not defined");
    }
    if (!options.room) {
        throw new Error("Room is not defined");
    }
    if (!options.peerConnection) {
        throw new Error("PeerConnection is not defined");
    }

    let mainDiv = options.div;
    let room = options.room;
    let peerConnection = options.peerConnection;
    let displayOptions = options.displayOptions || {publisher: true, quality: true, type: true};
    ...
    const createRemoteDisplay = function(id, name, mainDiv, displayOptions) {
        ...
    }

    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 createDisplay = true;
            for (let i = 0; i < participant.displays.length; i++) {
                let display = participant.displays[i];
                if (pTrack.type === "VIDEO") {
                    if (display.hasVideo()) {
                        continue;
                    }
                    display.videoMid = pTrack.mid;
                    display.setTrackInfo(pTrack);
                    createDisplay = false;
                    break;
                } else if (pTrack.type === "AUDIO") {
                    if (display.hasAudio()) {
                        continue;
                    }
                    display.audioMid = pTrack.mid;
                    display.setTrackInfo(pTrack);
                    createDisplay = false;
                    break;
                }
            }
            if (!createDisplay) {
                continue;
            }
            let display = createRemoteDisplay(participant.nickName, participant.nickName, mainDiv,
displayOptions);
            participant.displays.push(display);
            if (pTrack.type === "VIDEO") {
                display.videoMid = pTrack.mid;
                display.setTrackInfo(pTrack);
            } else if (pTrack.type === "AUDIO") {
                display.audioMid = pTrack.mid;
                display.setTrackInfo(pTrack);
            }
        }
        ...
    });
```

2.2. REMOVE_TRACKS

initRemoteDisplay() code

Where:

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

```
room.on(constants.SFU_ROOM_EVENT.ADD_TRACKS, function(e) {
    ...
}).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

```
room.on(constants.SFU_ROOM_EVENT.ADD_TRACKS, function(e) {
    ...
}).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

```
room.on(constants.SFU_ROOM_EVENT.ADD_TRACKS, function(e) {
    ...
}).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:

- display parameters are set up
- container `div` tag for participants streams is created
- child containerdiv 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);
        let publisherNameDisplay;
        let currentQualityDisplay;
        let videoTypeDisplay;
        let abrQualityCheckPeriod = ABR_QUALITY_CHECK_PERIOD;
        let abrKeepOnGoodQuality = ABR_KEEP_ON_QUALITY;
        let abrTryForUpperQuality = ABR_TRY_UPPER_QUALITY;
        if (displayOptions.abrQualityCheckPeriod !== undefined) {
            abrQualityCheckPeriod = displayOptions.abrQualityCheckPeriod;
        }
        if (displayOptions.abrKeepOnGoodQuality !== undefined) {
            abrKeepOnGoodQuality = displayOptions.abrKeepOnGoodQuality;
        }
        if (displayOptions.abrTryForUpperQuality !== undefined) {
            abrTryForUpperQuality = displayOptions.abrTryForUpperQuality;
        }
        if (!displayOptions.abr) {
            abrQualityCheckPeriod = 0;
            abrKeepOnGoodQuality = 0;
            abrTryForUpperQuality = 0;
        }
        if (displayOptions.publisher) {
            publisherNameDisplay = createInfoDisplay(cell, "Published by: " + name);
        }
        if (displayOptions.quality) {
            currentQualityDisplay = createInfoDisplay(cell, "");
        }
        if (displayOptions.type) {
            videoTypeDisplay = createInfoDisplay(cell, "");
        }
        const qualitySwitchDisplay = createInfoDisplay(cell, "");

        let qualityDivs = [];
        let contentType = "";

        const rootDisplay = createContainer(cell);
        const streamDisplay = createContainer(rootDisplay);
        const audioDisplay = createContainer(rootDisplay);
        const audioTypeDisplay = createInfoDisplay(audioDisplay);
        const audioTrackDisplay = createContainer(audioDisplay);
        const audioStateButton = AudioStateButton();

        hideItem(streamDisplay);
        hideItem(audioDisplay);
        hideItem(publisherNameDisplay);
        hideItem(currentQualityDisplay);
        hideItem(videoTypeDisplay);
        hideItem(qualitySwitchDisplay);
```

3.2. ABR initialization

createRemoteDisplay() code

Stream playback quality parameters are set to switch automatically to appropriate quality

```
        const abr = ABR(abrQualityCheckPeriod, [
            {parameter: "nackCount", maxLeap: 10},
            {parameter: "freezeCount", maxLeap: 10},
            {parameter: "packetsLost", maxLeap: 10}
        ], abrKeepOnGoodQuality, abrTryForUpperQuality);
```

3.3. 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;
                }
                showItem(streamDisplay);
                video = document.createElement("video");
                video.controls = "controls";
                video.muted = true;
                video.autoplay = true;
                if (Browser().isSafariWebRTC()) {
                    video.setAttribute("playsinline", "");
                    video.setAttribute("webkit-playsinline", "");
                    this.setWebkitEventHandlers(video);
                } else {
                    this.setEventHandlers(video);
                }
                streamDisplay.appendChild(video);
                video.srcObject = stream;
                this.setResizeHandler(video);
                abr.start();
            },
```

3.4. Audio tag addition

setAudio() code

```
            setAudio: function(stream) {
                if (audio) {
                    audio.remove();
                }
                if (!stream) {
                    audio = null;
                    this.audioMid = undefined;
                    return;
                }
                showItem(audioDisplay);
                audio = document.createElement("audio");
                audio.controls = "controls";
                audio.muted = true;
                audio.autoplay = true;
                if (Browser().isSafariWebRTC()) {
                    audio.setAttribute("playsinline", "");
                    audio.setAttribute("webkit-playsinline", "");
                    this.setWebkitEventHandlers(audio);
                } else {
                    this.setEventHandlers(audio);
                }
                audioTrackDisplay.appendChild(audio);
                audioStateButton.makeButton(audioTypeDisplay, audio);
                audio.srcObject = stream;
                audio.onloadedmetadata = function (e) {
                    audio.play().then(function() {
                        if (Browser().isSafariWebRTC() && Browser().isiOS()) {
                            console.warn("Audio track should be manually unmuted in iOS Safari");
                        } else {
                            audio.muted = false;
                            audioStateButton.setButtonState();
                        }
                    });
                };
            },
```

3.5. Audio and video tags event handlers set up

setResizeHandler(), setEventHandlers(), setWebkitEventHandlers() code

```
            setEventHandlers: function(video) {
                // Ignore play/pause button
                video.addEventListener("pause", function () {
                    console.log("Media paused by click, continue...");
                    video.play();
                });
            },
            setWebkitEventHandlers: function(video) {
                let needRestart = false;
                let isFullscreen = false;
                // Use webkitbeginfullscreen event to detect full screen mode in iOS Safari
                video.addEventListener("webkitbeginfullscreen", function () {
                    isFullscreen = true;
                });
                video.addEventListener("pause", function () {
                    if (needRestart) {
                        console.log("Media paused after fullscreen, continue...");
                        video.play();
                        needRestart = false;
                    } else {
                        console.log("Media paused by click, continue...");
                        video.play();
                    }
                });
                video.addEventListener("webkitendfullscreen", function () {
                    video.play();
                    needRestart = true;
                    isFullscreen = false;
                });
            },
```

3.6. Adding track info to ABR

setVideoABRTrack() code

```
            setVideoABRTrack: function(track) {
                abr.setTrack(track);
            },
```

3.7. Setting up quality switcher

setTrackInfo() code

Where:

- quality switch buttons are set up
- tracks published info is added to ABR
- tags to display a current playback quality and audio/video source are shown or hidden

```
            setTrackInfo: function(trackInfo) {
                if (trackInfo) {
                    if (trackInfo.quality) {
                        showItem(qualitySwitchDisplay);
                        if (abr.isEnabled()) {
                            const autoDiv = createQualityButton("Auto", qualityDivs, qualitySwitchDisplay);
                            autoDiv.style.color = QUALITY_COLORS.SELECTED;
                            autoDiv.addEventListener('click', function() {
                                setQualityButtonsColor(qualityDivs);
                                autoDiv.style.color = QUALITY_COLORS.SELECTED;
                                abr.setAuto();
                            });
                        }
                        for (let i = 0; i < trackInfo.quality.length; i++) {
                            abr.addQuality(trackInfo.quality[i]);
                            const qualityDiv = createQualityButton(trackInfo.quality[i], qualityDivs,
qualitySwitchDisplay);
                            qualityDiv.addEventListener('click', function() {
                                console.log("Clicked on quality " + trackInfo.quality[i] + " trackId " +
trackInfo.id);
                                if (qualityDiv.style.color === QUALITY_COLORS.UNAVAILABLE) {
                                    return;
                                }
                                setQualityButtonsColor(qualityDivs);
                                qualityDiv.style.color = QUALITY_COLORS.SELECTED;
                                abr.setManual();
                                abr.setQuality(trackInfo.quality[i]);
                            });
                        }
                    } else {
                        hideItem(qualitySwitchDisplay);
                    }
                    if (trackInfo.type) {
                        contentType = trackInfo.contentType || "";
                        if (trackInfo.type == "VIDEO" && displayOptions.type && contentType !== "") {
                            showItem(videoTypeDisplay);
                            videoTypeDisplay.innerHTML = contentType;
                        }
                        if (trackInfo.type == "AUDIO") {
                            audioStateButton.setContentType(contentType);
                        }
                    }
                }
            },
```

3.8. Adding available tracks quality info to ABR

updateQualityInfo() code

```
            updateQualityInfo: function(videoQuality) {
                showItem(qualitySwitchDisplay);
                for (const qualityInfo of videoQuality) {
                    let qualityColor = QUALITY_COLORS.UNAVAILABLE;
                    if (qualityInfo.available === true) {
                        qualityColor = QUALITY_COLORS.AVAILABLE;
                    }
                    for (const qualityDiv of qualityDivs) {
                        if (qualityDiv.innerText === qualityInfo.quality){
                            qualityDiv.style.color = qualityColor;
                            break;
                        }
                    }
                    abr.setQualityAvailable(qualityInfo.quality, qualityInfo.available);
                }
            },
```

3.9. Removing the container

dispose() code

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

4. Subscription to ontrack PeerConnection event

PeerConnection.ontrack(), setAudio(), setVideo(), setVideoABRTrack() 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.setVideoABRTrack(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];
    }
}
```