

Dolby.io Simple Conference Workshop

Introduction

In this workshop we'll walk through building a simple conference app built with Dolby.io

Workshop Prerequisites:

To get started take a moment to sign-up or sign-in at Dolby.io. Login to your GitHub account and Signup for Netlify. Fire up your favorite code editor and terminal.

We will use this GitHub repository as the basis for this workshop:

GitHub - dolbyio-samples/workshop-communications-api-simple-conference: Workshop Series...
GitHub

You'll need to sign up for Dolby.io and have the following stacks and tools available to complete this workshop.

Stacks and Tools

- Stacks
 - [Dolby.io](#)
 - [GitHub Account](#)
 - [Netlify](#)
- Tools
 - Any code editor of your choice - we'll be using VS Code
 - Terminal

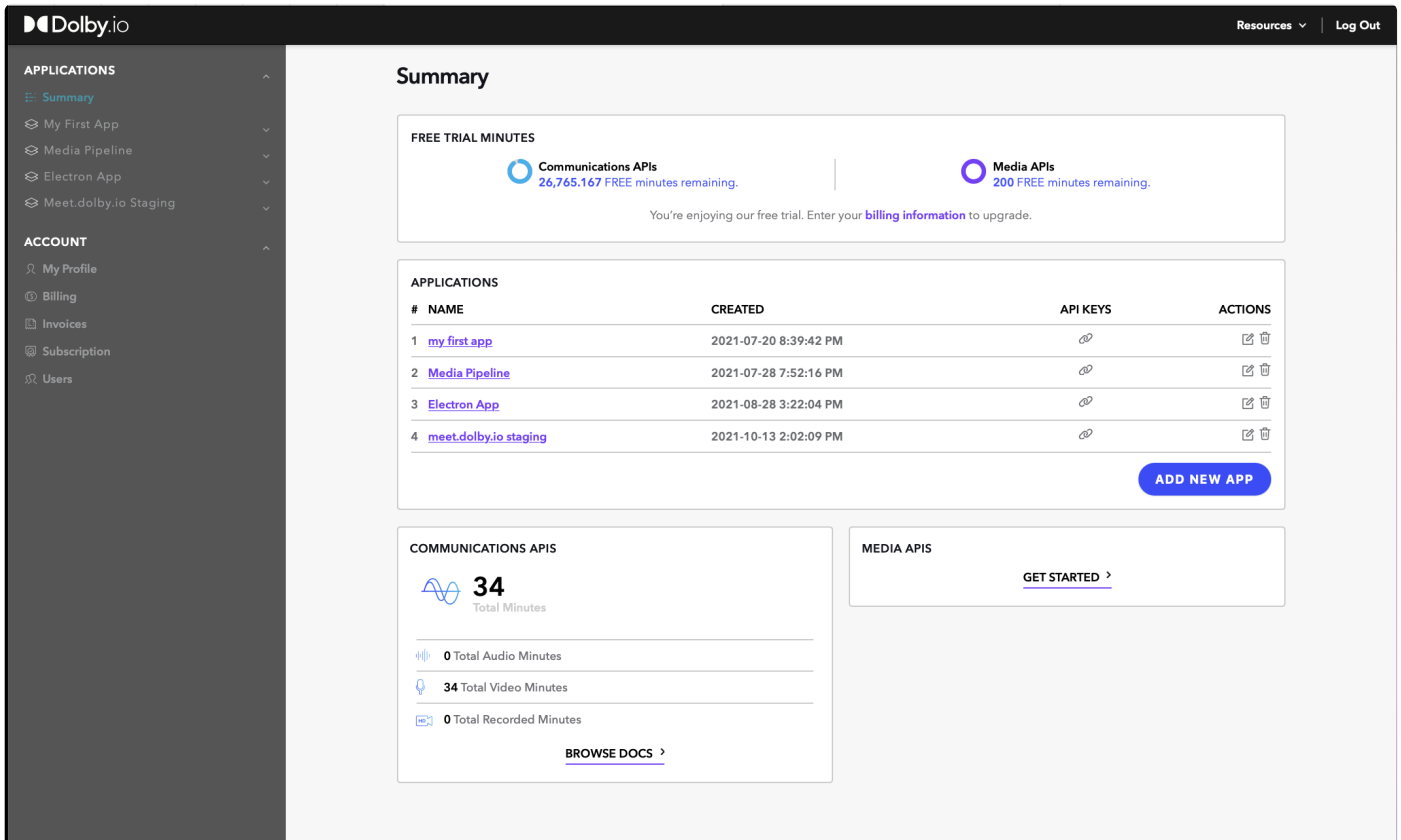
Dolby.io Dashboard

Let's walkthrough the Dashboard

The dashboard summary appears when you first login to Dolby.io

You will initially see an applications list and a starter app called **my first app**. Each app you create will

appear in this list.



The screenshot shows the Dolby.io Summary Page. The left sidebar contains navigation links for APPLICATIONS (Summary, My First App, Media Pipeline, Electron App, Meet.dolby.io Staging) and ACCOUNT (My Profile, Billing, Invoices, Subscription, Users). The main content area is titled 'Summary' and includes a 'FREE TRIAL MINUTES' section with 'Communications APIs' (26,765.167 FREE minutes remaining) and 'Media APIs' (200 FREE minutes remaining). Below this is a table of applications with columns for #, NAME, CREATED, API KEYS, and ACTIONS. The table lists four applications: 'my first app', 'Media Pipeline', 'Electron App', and 'meet.dolby.io staging'. An 'ADD NEW APP' button is at the bottom right of the table. Below the table, there are sections for 'COMMUNICATIONS APIS' (showing 34 Total Minutes) and 'MEDIA APIS' (with a 'GET STARTED' link). A 'BROWSE DOCS' link is also present.

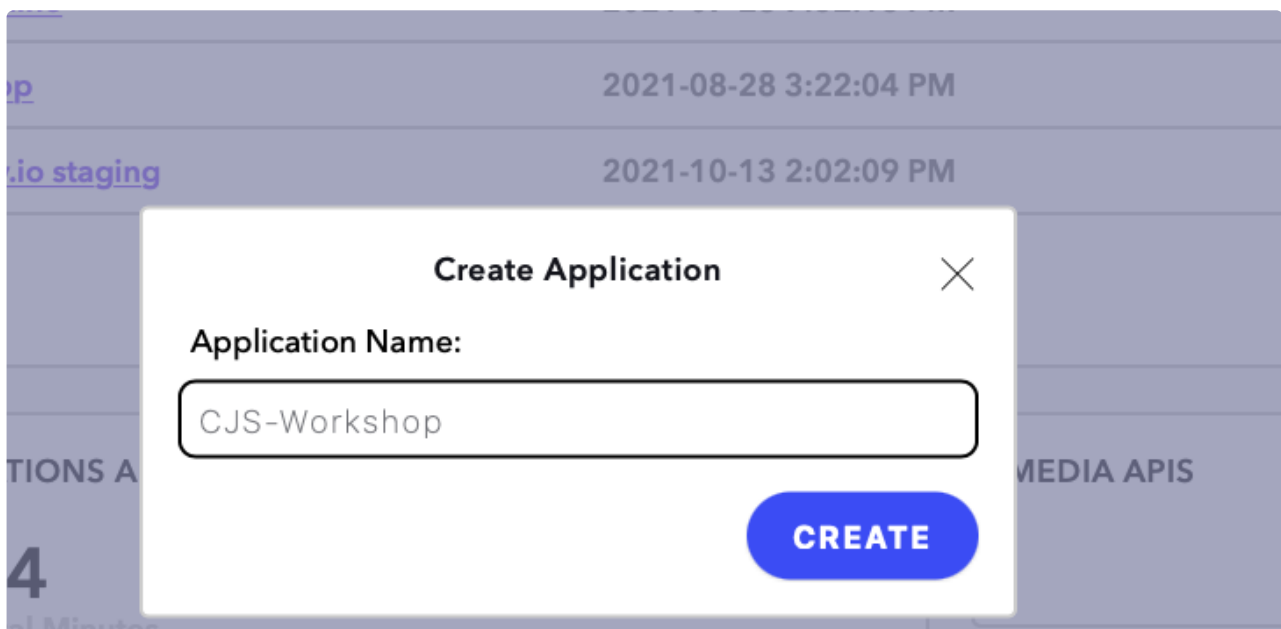
#	NAME	CREATED	API KEYS	ACTIONS
1	my first app	2021-07-20 8:39:42 PM	Link	Edit Delete
2	Media Pipeline	2021-07-28 7:52:16 PM	Link	Edit Delete
3	Electron App	2021-08-28 3:22:04 PM	Link	Edit Delete
4	meet.dolby.io staging	2021-10-13 2:02:09 PM	Link	Edit Delete

Summary Page

Let's create new application by clicking the **New App** button at the bottom of the applications list.

Enter the name of your app in the dialog and after a few moments, a new app will appear on your applications list. Let's click on the link icon to jump to your API Keys.

- ✓ You can click on the link icon to jump to your API keys. Actions like Rename and Delete can also be accomplished on this summary view.



Audio Minutes

Video Minutes

Create Application Dialog

API Keys

API Keys enable authentication with the Dolby.io APIs

Dolby.io offer's two distinct APIs; Communications API's for voice and video conferencing, and Media APIs for advanced processing of media files.

Each of these APIs has a complete set of features and an associated **Key** and **Secret** Pair.



Hint: Hit the show secret button to reveal the Consumer Secret **before** you click the copy button.

The screenshot shows the Dolby.io dashboard with a sidebar on the left containing 'APPLICATIONS' and 'ACCOUNT' sections. The main content area is titled 'API Keys' and displays two API configurations: 'Communications APIs' and 'Media APIs'. Each configuration shows a 'Consumer Key' (or 'API Key') and a 'Consumer Secret' (or 'API Secret'). The 'Consumer Secret' fields are masked with dots. To the right of each key field is a 'COPY' button. Below the 'Consumer Secret' field for 'Communications APIs' is a 'SHOW SECRET' button. The same 'SHOW SECRET' button is located below the 'API Secret' field for 'Media APIs'.

API Type	Key	Secret	Actions
Communications APIs	Consumer Key: kGKpGAtwSTbZFvAIWxb2CA==	Consumer Secret: [masked]	COPY, SHOW SECRET
	Media APIs	API Key: LVZiS7Bwfg7h4i0Q6pxrHV2nR5ZXqd	API Secret: [masked]

Basic Application Flow

Understanding the basic application flow

Common to all Communication API applications is the basic application flow that begins with initializing the SDK, and starting the conference session.

- [Initialize the SDK](#)
- [Starting a Conference](#)
 - Open the Session
 - Create a Participant
 - Join the Session
- [Leaving the Conference](#)
- [Event Handling](#)
- [Managing the UI](#)
- [Adding a participant's video](#)
- [Screenshare](#)
- [Mirror the current participant's video](#)

First, let's cover initialization of the SDK.

Basic Initialization

Basic initialization is useful for rapid prototyping on localhost, it's not secure, nor recommended for production level deployments.

There are two methods to initialize the SDK. You can initialize using a token, which requires a server to return the token to the application.

Alternately, you can initialize with the secrets, which is not secure. In this example below, we are initializing with the secrets.

 For more information, see [Initializing](#) within our website's documentation.

```
1 // Set the consumerKey and consumerSecret variables
2
3 const consumerKey = "CONSUMER_KEY";
4 const consumerSecret = "CONSUMER_SECRET";
5
6 $(function() {
7     // Initialize the Voxeet SDK
8     VoxeetSDK.initialize(consumerKey, consumerSecret);
9     logMessage("The Voxeet SDK has been initialized");
10
11 });
```



WARNING: It is best practice to use the `VoxeetSDK.initializeToken` function to initialize the SDK. Please read the documentation at: <https://docs.dolby.io/communications-apis/docs/initializing-javascript>

Since we recommend using a token to initialize the SDK; In the next section let's look at what's involved to setup a token service to use for token initialization of the SDK.

Token Initialization

Workshop Part One: Understanding authentication and initialization.



Live Demo: **Clone and Deploy your own Token Generator Service.**

Let's follow the link to our workshop on GitHub. Normally we'd ask you to open up your code editor and terminal to clone the project; for this workshop let's we'll use our example project and Netlify to clone and deploy your own token server in a single step.

Follow this GitHub link to instructions and **complete** Part One. When you are done, you'll have deployed a token server, copy and save the URL endpoint of your token server. You should see the actual JSON with the `access_token` value in your browser. Copy the URL from the browser's address bar.

GitHub - dolbyio-samples/workshop-communications-api-simple-conference: Workshop Serie...
[GitHub](#)

[Workshop Code on GitHub](#)

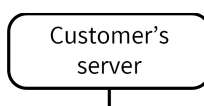
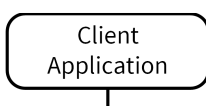
Using a token for initialization helps in part to secure your application since tokens are only valid for a short period of time. For this workshop, we're demonstrating the use of a simple serverless function that calls our API to generate the token.

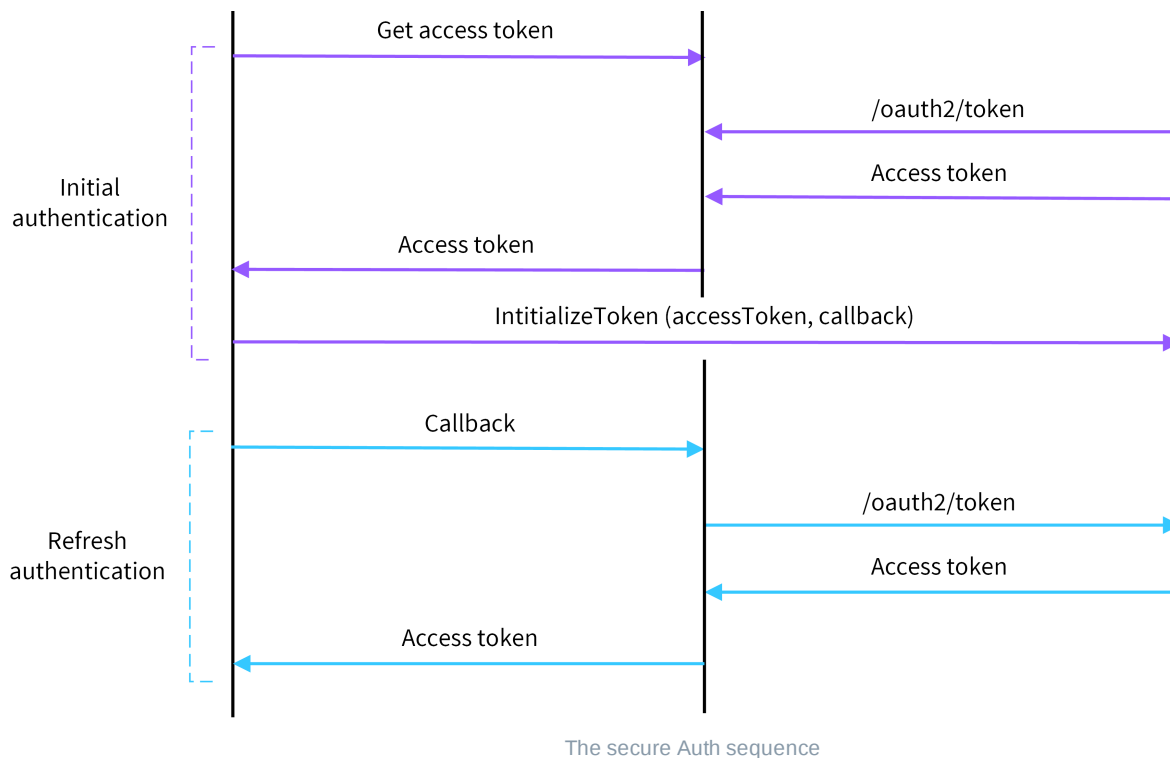
With this example anyone accessing the token url directly would be able to make API calls with your account. So for a production app you'll need to build a more secure solution where your serverless API endpoint is locked down to only permit authenticated requests.

This diagram helps to illustrate the authentication and initialization flow.



For more information, see [Initializing](#) within our website's documentation.





Once you have created a URL endpoint to retrieve a valid token the code to initialize is fairly concise. (End of part one)

Let's understand Token Initialization of the SDK in the client code

In this code example we use a promise to fetch the token and pass the `access_token` from the JSON response to our `VoxeetSDK.initializeToken(result.access_token, refreshToken)` method, the `refreshToken` is a callback method that will get triggered when the token is about to expire so the application will maintain it's initialization state.

```

1 // URL to our Token Server
2 const tokenServerURL = 'Enter the url to your token server here';
3
4
5 /**  initializeToken authorization flow on script load  */
6
7 (function () {
8   try {
9     getTokenAndInitalize()
10  } catch (e) {
11    alert('Something went wrong initalizatton : ' + e);
12  }
13 })();
14
15 /** Fetch our token and start initialization of SDK, update UI sources */
16 async function getTokenAndInitalize() {
17   return fetch(tokenServerURL)
18     .then((res) => {
19     return res.json();
20   })
21 }

```

```

20     })
21     .then((result) => {
22         VoxeetSDK.initializeToken(result.access_token, refreshToken);
23         return result.access_token
24     })
25     .then((token) => {
26         console.info('token received', token);
27         initializeConferenceSession()
28     })
29     .catch((error) => {
30         console.error(error);
31     });
32 }
33
34 /** Refresh Token is called when token expiration is 50% completed, this keeps the app in:
35 async function refreshToken() {
36     return fetch(tokenServerURL)
37         .then((res) => {
38             return res.json();
39         })
40         .then((json) => json.access_token)
41         .catch((error) => {
42             console.error(error);
43         });
44 }
45

```

Starting A Conference

Workshop Part Two: Build the conference app

In our simple conference app, the basic functionality is comprised of just four files located in the **www** directory of our GitHub Project. For the simplicity of this demo, we're using basic HTML, JavaScript, and CSS. We get a little styling help from Bootstrap as well. The code example below is the bare-bones minimum to get a conference going.

Follow GitHub Link below and the instructions for **Part Two** on this repo to clone and deploy the Communications API Simple Conference example app:

workshop-communications-api-simple-conference/README.md at main · dolbyio-samples/wo...
[GitHub](#)


Workshop GitHub Repo

Code walk: After following the Part Two instructions, you should have a working video conference app. Now let's take a wal through the code. Feel free to view the files on GitHub or pull down the project and open it up in you code editor.

```
1 git clone <your name /organization>/workshop-communications-api-simple-conference
2
3 cd workshop-communications-api-simple-conference
```

About the source files:

- ui.js contains code related to the UI management
- client.js contains code related to the conference management
- style.css contains the styles
- index.html contains code related to the main interface

 In the example below we're pointing to the latest version of the SDK, you can, however, target a specific version:

`https://unpkg.com/@voxeet/voxeet-web-sdk@<Version>/dist/voxeet-sdk.js`

in the index.html we include the Communications API (Voxeet-web-sdk) , client.js, ui.js and our styles. Add any additional frameworks or libraries to support your user interface.

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
5     <title>Basic Web Video Conference Application</title>
6     <script
7       src="https://unpkg.com/@voxeet/voxeet-web-sdk@latest/dist/voxeet-sdk.js"
8       type="text/javascript"
9     ></script>
10    <script src="./ui.js" type="text/javascript"></script>
11    <link href="./styles.css" rel="stylesheet">
12
13    <!-- Bootstrap CSS only -->
14    <!-- ... -->
15
16  </head>
17
18  <body>
19    <div id="app">
20      <h1 id="name-message">Logging in...</h1>
21      // todo add UI controls and layout
22    </div>
23    <script type="text/javascript" src="./client.js"></script>
24  </body>
25 </html>
```

Creating the Conference and joining

To allow creating and joining conferences, we will log in with a user name. In this workshop, random user names are assigned.

In this code example when a user clicks the **joinButton** we first provide a configuration object with the **conferenceParams**, setting some basic options and setting such as **liveRecording** and **dolbyVoice** to true.

✓ Learn more about [Dolby Voice](#)

The **conferenceOptions** object to include the name of the conference **alias** (conference name) and the **conferenceParams** object. With this object we create finally create the conference; calling **VoxeetSDK.conference.create(conferenceOptions)** with our option object.

We then take the promise result our **conference** object, and create **JoinOptions** and pass it along with our **conference** object to our next method in the chain; **VoxeetSDK.conference.join(conference, joinOptions)** - You can configure the constraints in the **JoinOptions** to explicitly enable audio and video.

we chain together with our methods in a promise. We set the

```
1  joinButton.onclick = () => {
2    // Default conference parameters
3    // See: https://docs.dolby.io/interactivity/docs/js-client-sdk-model-conferenceparameters
4    let conferenceParams = {
5      liveRecording: true,
6      rtcMode: "average", // worst, average, max
7      ttl: 0,
8      videoCodec: "H264", // H264, VP8
9      dolbyVoice: true
10   };
11
12   // See: https://docs.dolby.io/interactivity/docs/js-client-sdk-model-conferenceoptions
13   let conferenceOptions = {
14     alias: conferenceAliasInput.value,
15     params: conferenceParams,
16   };
17
18   // 1. Create a conference room with an alias
19   VoxeetSDK.conference.create(conferenceOptions)
20     .then((conference) => {
21       // See: https://docs.dolby.io/interactivity/docs/js-client-sdk-model-joinoptions
22       const joinOptions = {
23         constraints: {
24           audio: false,
25           video: true
26         },
27         simulcast: false,
28       };
29
30       // 2. Join the conference
31       VoxeetSDK.conference.join(conference, joinOptions)
32     })
33   }
```

```

33         .then((conf) => {
34             idDolbyVoice.innerHTML = `Dolby Voice is ${conf.params.dolbyVoice ? 'On' : 'Off'}
35         })
36         .catch((e) => console.log(e));
37     })
38     .catch((e) => console.log(e));
39 };

```

Create a Participant

Once we have an active session, we can open the session with our initial participant the current app user. Name is required, avatarURL and externalId is optional.

```

1 // Create a participant
2 let participant = { name: randomName, avatarURL:<URL to an image>, externalId:"<some external id>" };

```

```

1 try {
2     await VoxeetSDK.session.open(participant)
3     initUI();
4 }

```

Leaving the Conference

When a user hits the **leaveButton** we call the **VoxeetSDK.conference.leave()** method and then update the UI accordingly.

```

1 leaveButton.onclick = () => {
2     VoxeetSDK.conference
3     .leave()
4     .then(() => {
5         joinButton.disabled = false
6         leaveButton.disabled = true
7     })
8     .catch((err) => console.error(err));
9 }

```

Event Handling

We can use event handlers to manage the ui state, add and remove video nodes and list participants.

streamAdded in this example we look at the stream type and add a screenshare node or video node and add participant node to the dom through the use of helper functions. [See Adding Participant's Video](#)

```
1 /* Dolby.io Event handlers */
2
3 // When a stream is added to the conference
4 VoxeetSDK.conference.on('streamAdded', (participant, stream) => {
5   if (stream.type === 'ScreenShare') {
6     return addScreenShareNode(stream);
7   }
8   if (stream.getVideoTracks().length) {
9     // Only add the video node if there is a video track
10    addVideoNode(participant, stream);
11  }
12  addParticipantNode(participant);
13 });
14
```

streamUpdated is triggered when there is a change to the number of tracks such as when a participant is added or leaves the conference.

```
1 // When a stream is updated
2 VoxeetSDK.conference.on('streamUpdated', (participant, stream) => {
3   if (stream.type === 'ScreenShare') return;
4   if (stream.getVideoTracks().length) {
5     // Only add the video node if there is a video track
6     addVideoNode(participant, stream);
7   } else {
8     removeVideoNode(participant);
9   }
10 });
```

streamRemoved, as the name implies, is triggered when a stream is removed.

```
1 // When a stream is removed from the conference
2 VoxeetSDK.conference.on('streamRemoved', (participant, stream) => {
3   if (stream.type === 'ScreenShare') {
4     return removeScreenShareNode();
5   }
6   removeVideoNode(participant);
7   removeParticipantNode(participant);
8 });
```

Managing the UI

In this workshop application, user interaction is managed by methods and click handlers in the ui.js script.

- Overview
 - Set up const references to your UI elements.
 - Create Click Handlers for each button action.
 - Add helper functions to manage adding and removing DOM elements

initUI is called from the **client.js** right after the promise successfully returns.

In ui.js Set up const references to your UI elements

And set any initial values.

```
1 const initUI = () => {
2   const nameMessage = document.getElementById('name-message');
3   const conferenceAliasInput = document.getElementById('alias-input');
4   const joinButton = document.getElementById('join-btn');
5   const leaveButton = document.getElementById('leave-btn');
6   const lblDolbyVoice = document.getElementById('label-dolby-voice');
7   const startVideoBtn = document.getElementById('start-video-btn');
8   const stopVideoBtn = document.getElementById('stop-video-btn');
9   const startAudioBtn = document.getElementById('start-audio-btn');
10  const stopAudioBtn = document.getElementById('stop-audio-btn');
11  const startScreenShareBtn = document.getElementById('start-screenshare-btn');
12  const stopScreenShareBtn = document.getElementById('stop-screenshare-btn');
13  const startRecordingBtn = document.getElementById('start-recording-btn');
14  const stopRecordingBtn = document.getElementById('stop-recording-btn');
15
16  // Workshop Part three hide html elements / uncomment // showPlayer.style = ""; to show c
17  const showPlayer = document.getElementById('clip-player-controls');
18  showPlayer.style = "";
19
20  // Update the login message with the name of the user
21  nameMessage.innerHTML = `You are logged in as ${randomName}`;
22  // nameInput.value = randomName;
23  joinButton.disabled = false;
24
25
26 ...
```

Create Click Handlers for each button action.

```
1 joinButton.onclick = () => {
2   // Default conference parameters
3   // See: https://docs.dolby.io/interactivity/docs/js-client-sdk-model-conferenceparameters
4   let conferenceParams = {
5     liveRecording: true,
6     rtcMode: "average", // worst, average, max
7     ttl: 0,
8
```

```

8     videoCodec: "H264", // H264, VP8
9     dolbyVoice: true
10 };
11
12 // See: https://docs.dolby.io/interactivity/docs/js-client-sdk-model-conferenceoptions
13 let conferenceOptions = {
14     alias: conferenceAliasInput.value,
15     params: conferenceParams,
16 };
17
18 // 1. Create a conference room with an alias
19 VoxeetSDK.conference.create(conferenceOptions)
20     .then((conference) => {
21         // See: https://docs.dolby.io/interactivity/docs/js-client-sdk-model-joinoptions
22         const joinOptions = {
23             constraints: {
24                 audio: false,
25                 video: true
26             },
27             simulcast: false
28         };
29
30         // 2. Join the conference
31         VoxeetSDK.conference.join(conference, joinOptions)
32             .then((conf) => {
33                 lblDolbyVoice.innerHTML = `Dolby Voice is ${conf.params.dolbyVoice ? 'On' : 'Off'}`;
34
35                 conferenceAliasInput.disabled = true;
36                 joinButton.disabled = true;
37                 leaveButton.disabled = false;
38                 startVideoBtn.disabled = true;
39                 stopVideoBtn.disabled = false;
40                 startAudioBtn.disabled = false;
41                 stopAudioBtn.disabled = true;
42                 startScreenShareBtn.disabled = false;
43                 startRecordingBtn.disabled = false;
44                 startClipBtn.disabled = false;
45                 playClipBtn.disabled = true;
46                 pauseClipBtn.disabled = true;
47                 stopClipBtn.disabled = true;
48             })
49             .catch((e) => console.log(e));
50     })
51     .catch((e) => console.log(e));
52 };
53
54 leaveButton.onclick = () => {
55     // Leave the conference
56     VoxeetSDK.conference.leave()
57         .then(() => {
58             lblDolbyVoice.innerHTML = '';
59
60             conferenceAliasInput.disabled = false;
61

```

```

61     joinButton.disabled = false;
62     leaveButton.disabled = true;
63     startVideoBtn.disabled = true;
64     stopVideoBtn.disabled = true;
65     startAudioBtn.disabled = true;
66     stopAudioBtn.disabled = true;
67     startScreenShareBtn.disabled = true;
68     stopScreenShareBtn.disabled = true;
69     startRecordingBtn.disabled = true;
70     stopRecordingBtn.disabled = true;
71     startClipBtn.disabled = true;
72     playClipBtn.disabled = true;
73     pauseClipBtn.disabled = true;
74     stopClipBtn.disabled = true;
75 })
76 .catch((e) => console.log(e));
77 };
78
79 startVideoBtn.onclick = () => {
80     // Start sharing the video with the other participants
81     VoxeetSDK.conference.startVideo(VoxeetSDK.session.participant)
82     .then(() => {
83         startVideoBtn.disabled = true;
84         stopVideoBtn.disabled = false;
85     })
86     .catch((e) => console.log(e));
87 };
88
89 stopVideoBtn.onclick = () => {
90     // Stop sharing the video with the other participants
91     VoxeetSDK.conference.stopVideo(VoxeetSDK.session.participant)
92     .then(() => {
93         stopVideoBtn.disabled = true;
94         startVideoBtn.disabled = false;
95     })
96     .catch((e) => console.log(e));
97 };
98
99 startAudioBtn.onclick = () => {
100     // Start sharing the Audio with the other participants
101     VoxeetSDK.conference.startAudio(VoxeetSDK.session.participant)
102     .then(() => {
103         startAudioBtn.disabled = true;
104         stopAudioBtn.disabled = false;
105     })
106     .catch((e) => console.log(e));
107 };
108
109 stopAudioBtn.onclick = () => {
110     // Stop sharing the Audio with the other participants
111     VoxeetSDK.conference.stopAudio(VoxeetSDK.session.participant)
112     .then(() => {
113         stopAudioBtn.disabled = true;

```

```

114         startAudioBtn.disabled = false;
115     })
116     .catch((e) => console.log(e));
117 };
118
119 startScreenShareBtn.onclick = () => {
120     // Start the Screen Sharing with the other participants
121     VoxeetSDK.conference.startScreenShare()
122     .then(() => {
123         startScreenShareBtn.disabled = true;
124         stopScreenShareBtn.disabled = false;
125     })
126     .catch((e) => console.log(e));
127 };
128
129 stopScreenShareBtn.onclick = () => {
130     // Stop the Screen Sharing
131     VoxeetSDK.conference.stopScreenShare()
132     .catch((e) => console.log(e));
133 };
134
135 startRecordingBtn.onclick = () => {
136     let recordStatus = document.getElementById('record-status');
137     // Start recording the conference
138     VoxeetSDK.recording.start()
139     .then(() => {
140         recordStatus.innerText = 'Recording...';
141         startRecordingBtn.disabled = true;
142         stopRecordingBtn.disabled = false;
143     })
144     .catch((e) => console.log(e));
145 };
146
147 stopRecordingBtn.onclick = () => {
148     let recordStatus = document.getElementById('record-status');
149
150     // Stop recording the conference
151     VoxeetSDK.recording.stop()
152     .then(() => {
153         recordStatus.innerText = '';
154         startRecordingBtn.disabled = false;
155         stopRecordingBtn.disabled = true;
156     })
157     .catch((e) => console.log(e));
158 };
159
160 }; // end init

```

Add helper functions to manage adding and removing DOM elements

```

1 // Add a video stream to the web page
2 const addVideoNode = (participant, stream) => {
3

```

```

3   let videoNode = document.getElementById('video-' + participant.id);
4
5   if (!videoNode) {
6     videoNode = document.createElement('video');
7
8     // add css class to mirror current user's video
9     if (participant.id === VoxeetSDK.session.participant.id) {
10      videoNode.setAttribute('class', 'video-item flipped-video');
11    } else {
12      videoNode.setAttribute('class', 'video-item');
13    }
14    videoNode.setAttribute('id', 'video-' + participant.id);
15    videoNode.setAttribute("playsinline", true);
16    videoNode.muted = true;
17    // videoNode.setAttribute("autoplay", 'autoplay');
18    videoNode.autoplay = true;
19    // videoNode.controls = true;
20    const videoContainer = document.getElementById('video-container');
21    videoContainer.appendChild(videoNode);
22  }
23  navigator.attachMediaStream(videoNode, stream);
24 };
25
26 // Remove the video stream from the web page
27 const removeVideoNode = (participant) => {
28   let videoNode = document.getElementById('video-' + participant.id);
29   if (videoNode) {
30     videoNode.parentNode.removeChild(videoNode);
31   }
32 };
33
34
35 const createParticipantCard = (participant) => {
36
37   let newCard = `<li class="list-group-item-primary d-flex justify-content-between align-i
38   ${participant.info.name}
39   `
41   return newCard;
42 }
43
44
45 // Add a new participant to the list
46 const addParticipantNode = (participant) => {
47   // If the participant is the current session user, don't add himself to the list
48   if (participant.id === VoxeetSDK.session.participant.id) return;
49
50   let participantNode = document.createElement('p');
51   participantNode.setAttribute('id', 'participant-' + participant.id);
52   participantNode.innerHTML = createParticipantCard(participant);
53   const participantsList = document.getElementById('participants-list');
54   participantsList.appendChild(participantNode);
55 };
56

```



```

57 // Remove a participant from the list
58 const removeParticipantNode = (participant) => {
59   let participantNode = document.getElementById('participant-' + participant.id);
60   if (participantNode) {
61     participantNode.parentNode.removeChild(participantNode);
62   }
63 };
64
65 // Add a screen share stream to the web page
66 const addScreenShareNode = (stream) => {
67   let screenShareNode = document.getElementById('screenshare');
68   if (screenShareNode) {
69     return alert('There is already a participant sharing a screen!');
70   }
71   screenShareNode = document.createElement('video');
72   screenShareNode.setAttribute('class', 'screenshare');
73   screenShareNode.setAttribute('id', 'screenshare');
74   screenShareNode.autoplay = 'autoplay';
75   screenShareNode.controls = true; // allows PIP and full screen
76   navigator.attachMediaStream(screenShareNode, stream);
77   const screenShareContainer = document.getElementById('screenshare-container');
78   screenShareContainer.appendChild(screenShareNode);
79 }
80
81 // Remove the screen share stream from the web page
82 const removeScreenShareNode = () => {
83   let screenShareNode = document.getElementById('screenshare');
84   if (screenShareNode) {
85     screenShareNode.parentNode.removeChild(screenShareNode);
86   }
87   const startScreenShareBtn = document.getElementById('start-screenshare-btn');
88   startScreenShareBtn.disabled = false;
89   const stopScreenShareBtn = document.getElementById('stop-screenshare-btn');
90   stopScreenShareBtn.disabled = true;
91 }

```

Adding a participant's video

In **client.js**, the Dolby.io Event Handlers **VoxeetSDK.conference.on('streamAdded', (participant, stream)** and **VoxeetSDK.conference.on('streamUpdated', (participant, stream)** both check the **stream.type** and then conditionally call **addVideoNode**.

```

1
2 /* Dolby.io Event handlers */

```

```

3
4 // When a stream is added to the conference
5 VoxeetSDK.conference.on('streamAdded', (participant, stream) => {
6   if (stream.type === 'ScreenShare') {
7     return addScreenShareNode(stream);
8   }
9   if (stream.getVideoTracks().length) {
10    // Only add the video node if there is a video track
11    addVideoNode(participant, stream);
12  }
13  addParticipantNode(participant);
14 });
15
16 // When a stream is updated
17 VoxeetSDK.conference.on('streamUpdated', (participant, stream) => {
18   if (stream.type === 'ScreenShare') return;
19   if (stream.getVideoTracks().length) {
20    // Only add the video node if there is a video track
21    addVideoNode(participant, stream);
22   } else {
23     removeVideoNode(participant);
24   }
25 });
26

```

AddVideoNode(participant, stream) checks to see if there's an existing video node with the id of **video-
<participant-id>** if it's not already there it creates a **videoNode** and sets its attributes. The appends the node to the **videoContainer**.

```

1 // Add a video stream to the web page
2 const addVideoNode = (participant, stream) => {
3   let videoNode = document.getElementById('video-' + participant.id);
4
5   if (!videoNode) {
6     videoNode = document.createElement('video');
7
8     // add css class to mirror current user's video
9     if (participant.id === VoxeetSDK.session.participant.id) {
10      videoNode.setAttribute('class', 'video-item flipped-video');
11    } else {
12      videoNode.setAttribute('class', 'video-item');
13    }
14    videoNode.setAttribute('id', 'video-' + participant.id);
15    videoNode.setAttribute("playsinline", true);
16    videoNode.muted = true;
17    videoNode.autoplay = true;
18    // videoNode.controls = true; // remove comment to have PIP and fullscreen
19    const videoContainer = document.getElementById('video-container');
20    videoContainer.appendChild(videoNode);
21  }
22  navigator.attachMediaStream(videoNode, stream);
23 };

```

Screenshare

Adding Screensharing functionality

Adding a screenshare is similar to adding a video node.

addScreenShareNode(stream) creates a **video** element and sets the id and attributes, the calls **navigator.attachMediaStream(screenShareNode)**, and adds that node to the **screenshare-container** via **appendChild(screenShareNode)**.

```
1 // Add a screen share stream to the web page
2 const addScreenShareNode = (stream) => {
3   let screenShareNode = document.getElementById('screenshare');
4   if (screenShareNode) {
5     return alert('There is already a participant sharing a screen!');
6   }
7   screenShareNode = document.createElement('video');
8   screenShareNode.setAttribute('class', 'screenshare');
9   screenShareNode.setAttribute('id', 'screenshare');
10  screenShareNode.autoplay = 'autoplay';
11  screenShareNode.controls = true; // allows PIP and full screen
12  navigator.attachMediaStream(screenShareNode, stream);
13  const screenShareContainer = document.getElementById('screenshare-container');
14  screenShareContainer.appendChild(screenShareNode);
15 }
```

RemoveShareNode simply removes the screenhare DOM element and updates the ui buttons.

```
1 // Remove the screen share stream from the web page
2 const removeScreenShareNode = () => {
3   let screenShareNode = document.getElementById('screenshare');
4   if (screenShareNode) {
5     screenShareNode.parentNode.removeChild(screenShareNode);
6   }
7   const startScreenShareBtn = document.getElementById('start-screenshare-btn');
8   startScreenShareBtn.disabled = false;
9   const stopScreenShareBtn = document.getElementById('stop-screenshare-btn');
10  stopScreenShareBtn.disabled = true;
11 }
```

Mirror the current participant's video

Flip and mirror the video using CSS styles.

Most video applications present a mirror image of the current users video.

In **styles.css** we create CSS transform style called **flipped-video**.

```
1 .flipped-video {
2     transform: rotateY(180deg);
3     -webkit-transform: rotateY(180deg); /* Safari and Chrome */
4     -moz-transform: rotateY(180deg); /* Firefox */
5 }
6
```

We will conditionally add this the style when we call the **addVideoNode** in **ui.js**

if (participant.id === VoxeetSDK.session.participant.id) checks the participant against the session.participant.id which is the current participant.

if that condition is true we add the class to the video node's attributes. **videoNode.setAttribute('class', 'video-item flipped-video');**

```
1 / Add a video stream to the web page
2 const addVideoNode = (participant, stream) => {
3     let videoNode = document.getElementById('video-' + participant.id);
4
5     if (!videoNode) {
6         videoNode = document.createElement('video');
7
8         // add css class to mirror current user's video
9         if (participant.id === VoxeetSDK.session.participant.id) {
10             videoNode.setAttribute('class', 'video-item flipped-video');
11         } else {
12             videoNode.setAttribute('class', 'video-item');
13         }
14         videoNode.setAttribute('id', 'video-' + participant.id);
15         videoNode.setAttribute("playsinline", true);
16         videoNode.muted = true;
17         // videoNode.setAttribute("autoplay", 'autoplay');
18         videoNode.autoplay = true;
19         // videoNode.controls = true;
20         const videoContainer = document.getElementById('video-container');
21         videoContainer.appendChild(videoNode);
22     }
23     navigator.attachMediaStream(videoNode, stream);
24 };
```