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 Serie…
> 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

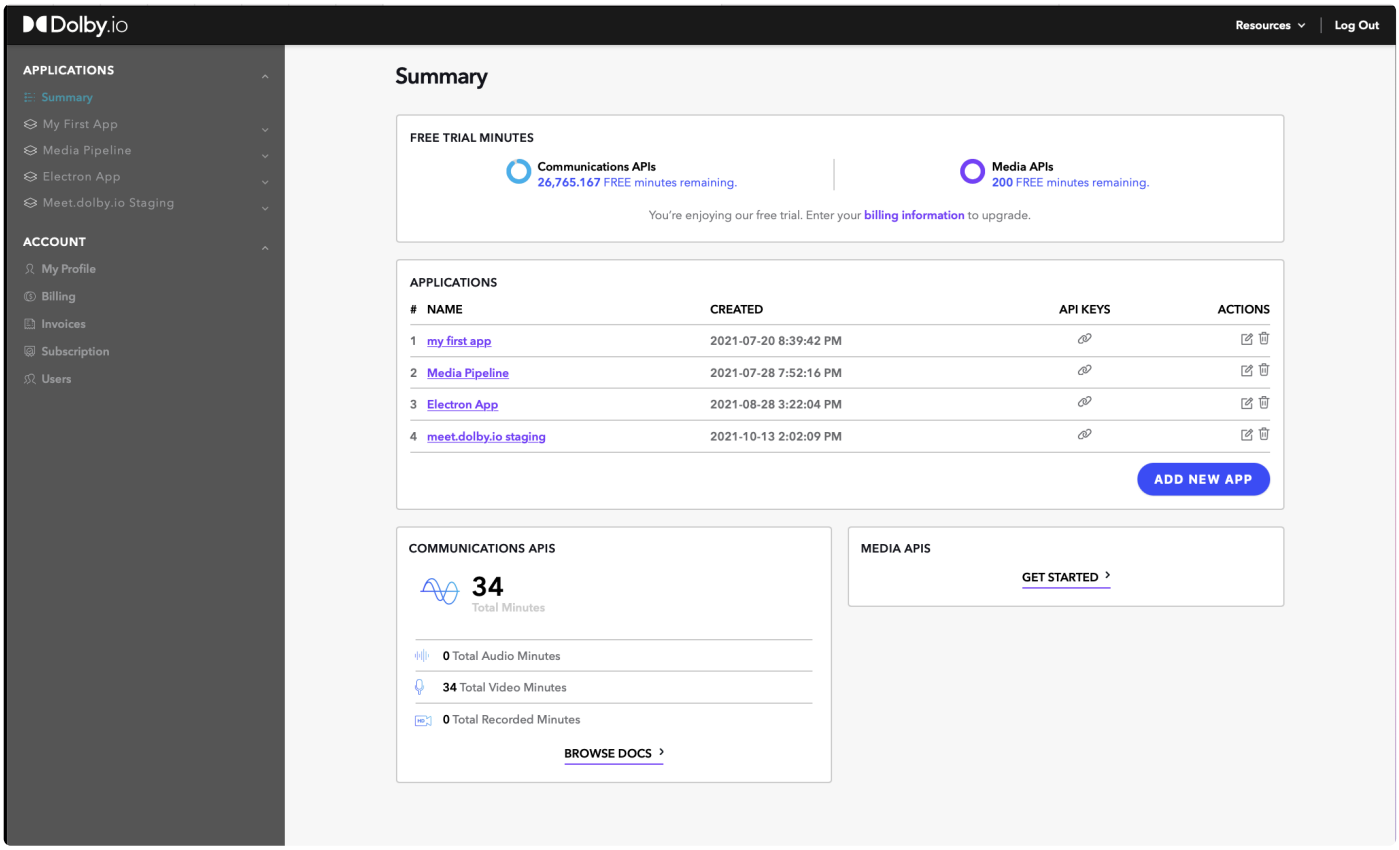
Dolby.io Dashboard

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.

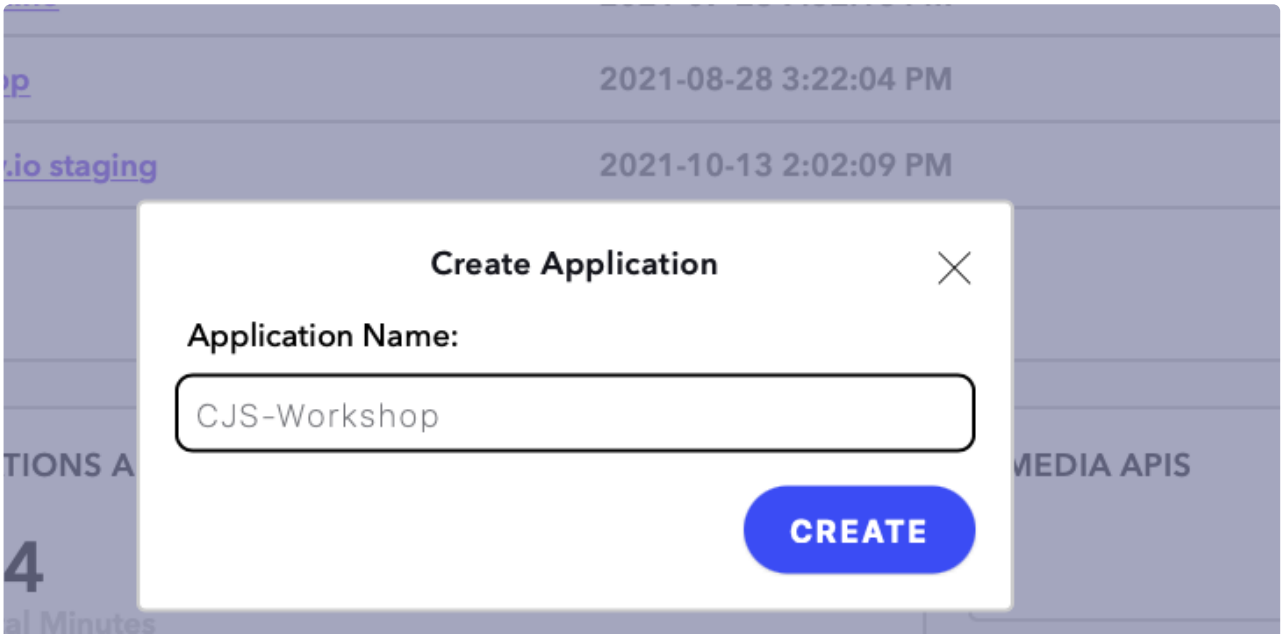


Summary Page

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.



Create Application Dialog

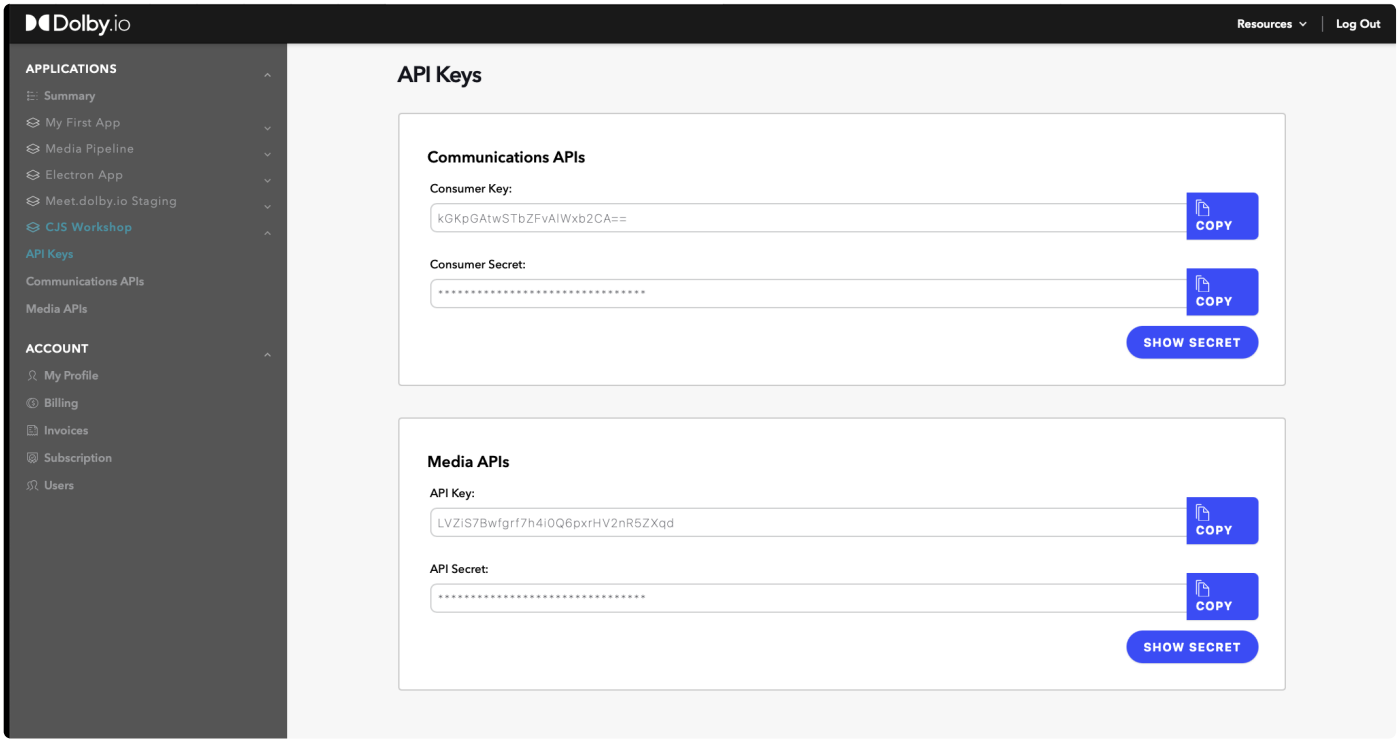
API Keys

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.



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.

```
// Set the consumerKey and consumerSecret variables

const consumerKey = "CONSUMER_KEY";
const consumerSecret = "CONSUMER_SECRET";

$(function() {
    // Initialize the Voxeet SDK
    VoxeetSDK.initialize(consumerKey, consumerSecret);
    logMessage("The Voxeet SDK has been initialized");

});
```

> ⚠

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 initalization 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 deply 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

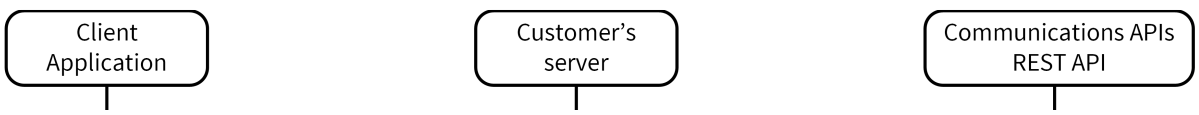
Workshop Code on GitHub

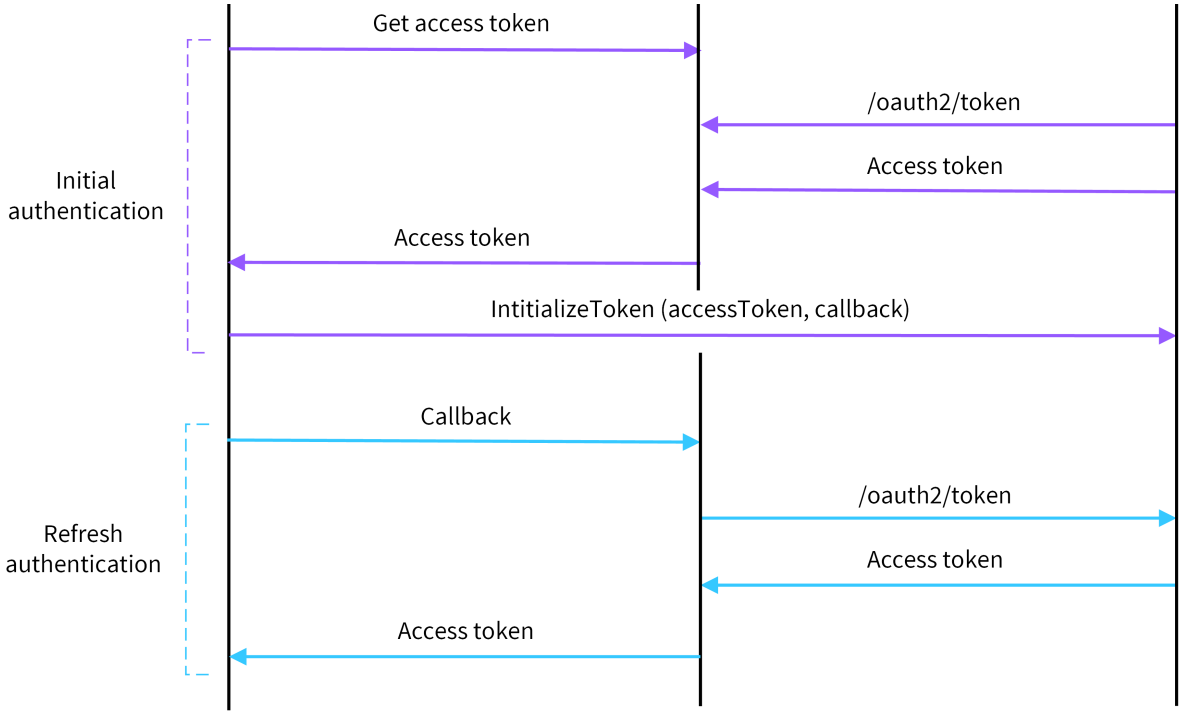
Using a token for initialzation helps in part to secure your application since tokens are only valid for a short period of time. For this workshop, we're demonstrating the use of a simple serverless function that calls our API to generate the token.

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 enpoint 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.





The secure Auth sequence

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

Let's understand Token Initialzation 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 initialzation state.

```
// URL to our Token Server
const tokenServerURL = 'Enter the url to your token server here';


/**   initializeToken authorization flow on script load  **/

(function () {
  try {
    getTokenAndInitalize()
  } catch (e) {
    alert('Something went wrong initalizaton : ' + e);
  }
})();

/** Fetch our token and start initialization of SDK, update UI sources */
async function getTokenAndInitalize() {
  return fetch(tokenServerURL)
    .then((res) => {
      return res.json();
```

```
    })
    .then((result) => {
      VoxeetSDK.initializeToken(result.access_token, refreshToken);
      return result.access_token
    })
    .then((token) => {
      console.info('token received', token);
      initializeConferenceSession()
    })
    .catch((error) => {
      console.error(error);
    });
}

/**  Refresh Token is called when token expiration is 50% completed, this keeps the app in
async function refreshToken() {
  return fetch(tokenServerURL)
    .then((res) => {
      return res.json();
    })
    .then((json) => json.access_token)
    .catch((error) => {
      console.error(error);
    });
}

```

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 stying 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.

```
git clone <your name /organization>/workshop-communications-api-simple-conference

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 Comminications API (Voxeet-web-sdk) , client.js, ui.js and our styles. Add any additional frameworks or libraries to support your user interface.

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>Basic Web Video Conference Application</title>
    <script
      src="https://unpkg.com/@voxeet/voxeet-web-sdk@latest/dist/voxeet-sdk.js"
      type="text/javascript"
    ></script>
    <script src="./ui.js" type="text/javascript"></script>
     <link href="./styles.css" rel="stylesheet">

      <!-- Bootstrap CSS only -->
      <!-- ... -->

  </head>

  <body>
    <div id="app">
      <h1 id="name-message">Logging in...</h1>
      // todo add UI controls and layout
    </div>
    <script type="text/javascript" src="./client.js"></script>
  </body>
</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

```
joinButton.onclick = () => {
  // Default conference parameters
  // See: https://docs.dolby.io/interactivity/docs/js-client-sdk-model-conferenceparamete
  let conferenceParams = {
    liveRecording: true,
    rtcpMode: "average", // worst, average, max
    ttl: 0,
    videoCodec: "H264", // H264, VP8
    dolbyVoice: true
  };

  // See: https://docs.dolby.io/interactivity/docs/js-client-sdk-model-conferenceoptions
  let conferenceOptions = {
    alias: conferenceAliasInput.value,
    params: conferenceParams,
  };

  // 1. Create a conference room with an alias
  VoxeetSDK.conference.create(conferenceOptions)
    .then((conference) => {
      // See: https://docs.dolby.io/interactivity/docs/js-client-sdk-model-joinoptions
      const joinOptions = {
        constraints: {
          audio: false,
          video: true
        },
        simulcast: false,
      };

      // 2. Join the conference
      VoxeetSDK.conference.join(conference, joinOptions)
```

```
            .then((conf) => {
              lblDolbyVoice.innerHTML = `Dolby Voice is ${conf.params.dolbyVoice ? 'On' : 'O
              // configure your ui buttons (start / stop audio / video etc
            })
            .catch((e) => console.log(e));
        })
        .catch((e) => console.log(e));
    };
```

Create a Participant

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

```
// Create a particpant
let participant = { name: randomName, avatarURL:<URL to an image>, externalId:"<some exter
```

```
try {
    await VoxeetSDK.session.open(particpant)
    initUI();
}
```

Leaving the Conference

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

```
leaveButton.onclick = () => {
       VoxeetSDK.conference
           .leave()
           .then(() => {
               joinButton.disabled = false
               leaveButton.disabled = true
           })
           .catch((err) => console.error(err));
   }
```

Event Handling

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

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

```
/* Dolby.io Event handlers */

// When a stream is added to the conference
VoxeetSDK.conference.on('streamAdded', (participant, stream) => {
  if (stream.type === 'ScreenShare') {
    return addScreenShareNode(stream);
  }
  if (stream.getVideoTracks().length) {
    // Only add the video node if there is a video track
    addVideoNode(participant, stream);
  }
  addParticipantNode(participant);
});

```

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

```
// When a stream is updated
VoxeetSDK.conference.on('streamUpdated', (participant, stream) => {
  if (stream.type === 'ScreenShare') return;
  if (stream.getVideoTracks().length) {
    // Only add the video node if there is a video track
    addVideoNode(participant, stream);
  } else {
    removeVideoNode(participant);
  }
});
```

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

```
// When a stream is removed from the conference
VoxeetSDK.conference.on('streamRemoved', (participant, stream) => {
  if (stream.type === 'ScreenShare') {
    return removeScreenShareNode();
  }
  removeVideoNode(participant);
  removeParticipantNode(participant);
});
```

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 afer the promise successfully returns.

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

And set any initial values.

```
const initUI = () => {
  const nameMessage = document.getElementById('name-message');
  const conferenceAliasInput = document.getElementById('alias-input');
  const joinButton = document.getElementById('join-btn');
  const leaveButton = document.getElementById('leave-btn');
  const lblDolbyVoice = document.getElementById('label-dolby-voice');
  const startVideoBtn = document.getElementById('start-video-btn');
  const stopVideoBtn = document.getElementById('stop-video-btn');
  const startAudioBtn = document.getElementById('start-audio-btn');
  const stopAudioBtn = document.getElementById('stop-audio-btn');
  const startScreenShareBtn = document.getElementById('start-screenshare-btn');
  const stopScreenShareBtn = document.getElementById('stop-screenshare-btn');
  const startRecordingBtn = document.getElementById('start-recording-btn');
  const stopRecordingBtn = document.getElementById('stop-recording-btn');

 // Workshop Part three hide html elements / uncomment // showPlayer.style = ""; to show cl
  const showPlayer =  document.getElementById('clip-player-controls');
  showPlayer.style = "";

 // Update the login message with the name of the user
 nameMessage.innerHTML = `You are logged in as ${randomName}`;
 // nameInput.value = randomName;
 joinButton.disabled = false;


...
```

Create Click Handlers for each button action.

```
joinButton.onclick = () => {
   // Default conference parameters
   // See: https://docs.dolby.io/interactivity/docs/js-client-sdk-model-conferenceparamete
   let conferenceParams = {
     liveRecording: true,
     rtcpMode: "average", // worst, average, max
     ttl: 0,
```

```
    videoCodec: "H264", // H264, VP8
    dolbyVoice: true
  };

  // See: https://docs.dolby.io/interactivity/docs/js-client-sdk-model-conferenceoptions
  let conferenceOptions = {
    alias: conferenceAliasInput.value,
    params: conferenceParams,
  };

  // 1. Create a conference room with an alias
  VoxeetSDK.conference.create(conferenceOptions)
    .then((conference) => {
      // See: https://docs.dolby.io/interactivity/docs/js-client-sdk-model-joinoptions
      const joinOptions = {
        constraints: {
          audio: false,
          video: true
        },
        simulcast: false
      };

      // 2. Join the conference
      VoxeetSDK.conference.join(conference, joinOptions)
        .then((conf) => {
          lblDolbyVoice.innerHTML = `Dolby Voice is ${conf.params.dolbyVoice ? 'On' : 'O

          conferenceAliasInput.disabled = true;
          joinButton.disabled = true;
          leaveButton.disabled = false;
          startVideoBtn.disabled = true;
          stopVideoBtn.disabled = false;
          startAudioBtn.disabled = false;
          stopAudioBtn.disabled = true;
          startScreenShareBtn.disabled = false;
          startRecordingBtn.disabled = false;
          startClipBtn.disabled = false;
          playClipBtn.disabled = true;
          pauseClipBtn.disabled = true;
          stopClipBtn.disabled = true;
        })
        .catch((e) => console.log(e));
    })
    .catch((e) => console.log(e));
};

leaveButton.onclick = () => {
  // Leave the conference
  VoxeetSDK.conference.leave()
    .then(() => {
      lblDolbyVoice.innerHTML = '';

      conferenceAliasInput.disabled = false;
```

```
      joinButton.disabled = false;
      leaveButton.disabled = true;
      startVideoBtn.disabled = true;
      stopVideoBtn.disabled = true;
      startAudioBtn.disabled = true;
      stopAudioBtn.disabled = true;
      startScreenShareBtn.disabled = true;
      stopScreenShareBtn.disabled = true;
      startRecordingBtn.disabled = true;
      stopRecordingBtn.disabled = true;
      startClipBtn.disabled = true;
      playClipBtn.disabled = true;
      pauseClipBtn.disabled = true;
      stopClipBtn.disabled = true;
    })
    .catch((e) => console.log(e));
};

startVideoBtn.onclick = () => {
  // Start sharing the video with the other participants
  VoxeetSDK.conference.startVideo(VoxeetSDK.session.participant)
    .then(() => {
      startVideoBtn.disabled = true;
      stopVideoBtn.disabled = false;
    })
    .catch((e) => console.log(e));
};

stopVideoBtn.onclick = () => {
  // Stop sharing the video with the other participants
  VoxeetSDK.conference.stopVideo(VoxeetSDK.session.participant)
    .then(() => {
      stopVideoBtn.disabled = true;
      startVideoBtn.disabled = false;
    })
    .catch((e) => console.log(e));
};

startAudioBtn.onclick = () => {
  // Start sharing the Audio with the other participants
  VoxeetSDK.conference.startAudio(VoxeetSDK.session.participant)
    .then(() => {
      startAudioBtn.disabled = true;
      stopAudioBtn.disabled = false;
    })
    .catch((e) => console.log(e));
};

stopAudioBtn.onclick = () => {
  // Stop sharing the Audio with the other participants
  VoxeetSDK.conference.stopAudio(VoxeetSDK.session.participant)
    .then(() => {
      stopAudioBtn.disabled = true;
```

```
        startAudioBtn.disabled = false;
      })
      .catch((e) => console.log(e));
  };

  startScreenShareBtn.onclick = () => {
    // Start the Screen Sharing with the other participants
    VoxeetSDK.conference.startScreenShare()
      .then(() => {
        startScreenShareBtn.disabled = true;
        stopScreenShareBtn.disabled = false;
      })
      .catch((e) => console.log(e));
  };

  stopScreenShareBtn.onclick = () => {
    // Stop the Screen Sharing
    VoxeetSDK.conference.stopScreenShare()
      .catch((e) => console.log(e));
  };

  startRecordingBtn.onclick = () => {
    let recordStatus = document.getElementById('record-status');
    // Start recording the conference
    VoxeetSDK.recording.start()
      .then(() => {
        recordStatus.innerText = 'Recording...';
        startRecordingBtn.disabled = true;
        stopRecordingBtn.disabled = false;
      })
      .catch((e) => console.log(e));
  };

    stopRecordingBtn.onclick = () => {
    let recordStatus = document.getElementById('record-status');

    // Stop recording the conference
    VoxeetSDK.recording.stop()
      .then(() => {
        recordStatus.innerText = '';
        startRecordingBtn.disabled = false;
        stopRecordingBtn.disabled = true;
      })
      .catch((e) => console.log(e));
  };

}; // end init
```

Add helper functions to manage adding and removing DOM elements

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

```
  let videoNode = document.getElementById('video-' + participant.id);

  if (!videoNode) {
    videoNode = document.createElement('video');

    // add css class to mirror current user's video
    if (participant.id === VoxeetSDK.session.participant.id) {
      videoNode.setAttribute('class', 'video-item flipped-video');
    } else {
      videoNode.setAttribute('class', 'video-item');
    }
    videoNode.setAttribute('id', 'video-' + participant.id);
    videoNode.setAttribute("playsinline", true);
    videoNode.muted = true;
    // videoNode.setAttribute("autoplay", 'autoplay');
    videoNode.autoplay = true;
    // videoNode.controls = true;
    const videoContainer = document.getElementById('video-container');
    videoContainer.appendChild(videoNode);
  }
  navigator.attachMediaStream(videoNode, stream);
};

// Remove the video streem from the web page
const removeVideoNode = (participant) => {
  let videoNode = document.getElementById('video-' + participant.id);
  if (videoNode) {
    videoNode.parentNode.removeChild(videoNode);
  }
};


const createParticpantCard = (participant) => {

  let newCard = `<li class="list-group-item-primary d-flex justify-content-between align-i
  ${participant.info.name}
   <img src="${participant.info.avatarUrl}" class="img-fluid rounded-start my-list" alt="$
</li>`
  return newCard;
}


// Add a new participant to the list
const addParticipantNode = (participant) => {
  // If the participant is the current session user, don't add himself to the list
  if (participant.id === VoxeetSDK.session.participant.id) return;

  let participantNode = document.createElement('p');
  participantNode.setAttribute('id', 'participant-' + participant.id);
  participantNode.innerHTML = createParticpantCard(participant);
  const participantsList = document.getElementById('participants-list');
  participantsList.appendChild(participantNode);
};
```

```

// Remove a participant from the list
const removeParticipantNode = (participant) => {
  let participantNode = document.getElementById('participant-' + participant.id);
  if (participantNode) {
    participantNode.parentNode.removeChild(participantNode);
  }
};

// Add a screen share stream to the web page
const addScreenShareNode = (stream) => {
  let screenShareNode = document.getElementById('screenshare');
  if (screenShareNode) {
    return alert('There is already a participant sharing a screen!');
  }
  screenShareNode = document.createElement('video');
  screenShareNode.setAttribute('class', 'screenshare');
  screenShareNode.setAttribute('id', 'screenshare');
  screenShareNode.autoplay = 'autoplay';
  screenShareNode.controls = true;  // allows PIP and full screen
  navigator.attachMediaStream(screenShareNode, stream);
  const screenShareContainer = document.getElementById('screenshare-container');
  screenShareContainer.appendChild(screenShareNode);
}

// Remove the screen share stream from the web page
const removeScreenShareNode = () => {
  let screenShareNode = document.getElementById('screenshare');
  if (screenShareNode) {
    screenShareNode.parentNode.removeChild(screenShareNode);
  }
  const startScreenShareBtn = document.getElementById('start-screenshare-btn');
  startScreenShareBtn.disabled = false;
  const stopScreenShareBtn = document.getElementById('stop-screenshare-btn');
  stopScreenShareBtn.disabled = true;
}
```

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 conditinally call **addVideoNode.**

```

/* Dolby.io Event handlers */
```

```

// When a stream is added to the conference
VoxeetSDK.conference.on('streamAdded', (participant, stream) => {
  if (stream.type === 'ScreenShare') {
    return addScreenShareNode(stream);
  }
  if (stream.getVideoTracks().length) {
    // Only add the video node if there is a video track
    addVideoNode(participant, stream);
  }
  addParticipantNode(participant);
});

// When a stream is updated
VoxeetSDK.conference.on('streamUpdated', (participant, stream) => {
  if (stream.type === 'ScreenShare') return;
  if (stream.getVideoTracks().length) {
    // Only add the video node if there is a video track
    addVideoNode(participant, stream);
  } else {
    removeVideoNode(participant);
  }
});

```

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**.

```
// Add a video stream to the web page
const addVideoNode = (participant, stream) => {
  let videoNode = document.getElementById('video-' + participant.id);

  if (!videoNode) {
    videoNode = document.createElement('video');

    // add css class to mirror current user's video
    if (participant.id === VoxeetSDK.session.participant.id) {
      videoNode.setAttribute('class', 'video-item flipped-video');
    } else {
      videoNode.setAttribute('class', 'video-item');
    }
    videoNode.setAttribute('id', 'video-' + participant.id);
    videoNode.setAttribute("playsinline", true);
    videoNode.muted = true;
    videoNode.autoplay = true;
    // videoNode.controls = true;  // remove comment to have PIP and fullscreen
    const videoContainer = document.getElementById('video-container');
    videoContainer.appendChild(videoNode);
  }
  navigator.attachMediaStream(videoNode, stream);
};
```

```
```

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)**.

```
// Add a screen share stream to the web page
const addScreenShareNode = (stream) => {
  let screenShareNode = document.getElementById('screenshare');
  if (screenShareNode) {
    return alert('There is already a participant sharing a screen!');
  }
  screenShareNode = document.createElement('video');
  screenShareNode.setAttribute('class', 'screenshare');
  screenShareNode.setAttribute('id', 'screenshare');
  screenShareNode.autoplay = 'autoplay';
  screenShareNode.controls = true;  // allows PIP and full screen
  navigator.attachMediaStream(screenShareNode, stream);
  const screenShareContainer = document.getElementById('screenshare-container');
  screenShareContainer.appendChild(screenShareNode);
}
```

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

```
// Remove the screen share stream from the web page
const removeScreenShareNode = () => {
  let screenShareNode = document.getElementById('screenshare');
  if (screenShareNode) {
    screenShareNode.parentNode.removeChild(screenShareNode);
  }
  const startScreenShareBtn = document.getElementById('start-screenshare-btn');
  startScreenShareBtn.disabled = false;
  const stopScreenShareBtn = document.getElementById('stop-screenshare-btn');
  stopScreenShareBtn.disabled = true;
}
```

Mirror the current particpant'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**.

```
.flipped-video {
    transform: rotateY(180deg);
    -webkit-transform: rotateY(180deg); /* Safari and Chrome */
    -moz-transform: rotateY(180deg); /* Firefox */
}

```

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

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

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

```
/ Add a video stream to the web page
const addVideoNode = (participant, stream) => {
  let videoNode = document.getElementById('video-' + participant.id);

  if (!videoNode) {
    videoNode = document.createElement('video');

    // add css class to mirror current user's video
    if (participant.id === VoxeetSDK.session.participant.id) {
      videoNode.setAttribute('class', 'video-item flipped-video');
    } else {
      videoNode.setAttribute('class', 'video-item');
    }
    videoNode.setAttribute('id', 'video-' + participant.id);
    videoNode.setAttribute("playsinline", true);
    videoNode.muted = true;
    // videoNode.setAttribute("autoplay", 'autoplay');
    videoNode.autoplay = true;
    // videoNode.controls = true;
    const videoContainer = document.getElementById('video-container');
    videoContainer.appendChild(videoNode);
  }
  navigator.attachMediaStream(videoNode, stream);
};
```