Project Name: SICSR – Video Library
Group Member: Manasi Bhagat (11)
Amol Borkar (12)
Technology: JavaScript, Html, CSS

Min Requirement: Latest Version of chrome (Version of Chrome support Video tag)

Project Structure (Folder Structure):

Index.html – Html code
Style.css --- CSS code
Index.js --- JavaScript Code

Run the Project:

Right Click on the Project and open in any browser (Recommended to Use Chrome browser)
Note: There are three supported video formats in HTML: MP4, WebM, and OGG.

Git Setting:

1. Create the Repository in GitHub
2. git remote add origin https://github.com/13amol/SICSR-VideoLibrary.git
3. git init
4. git add .
5. git commit –m “login page”
6. git push origin master

Step by Step Details:

We will use the HTML5 video tag with custom controls.

```
. . .
<video controls class="video" id="video" preload="metadata" poster="poster.jpg">
  <source src="video.mp4" type="video/mp4"></source>
</video>
. . .
```

Description:
Controls -- Specifies that video controls should be displayed (such as a play/pause button etc).
Preload – values(auto ,metadata, none) Specifies if and how the author thinks the video should be loaded when the page loads
Poster -- Specifies an image to be shown while the video is downloading, or until the user hits the play button
Src -- Specifies the URL of the video file
Width – Sets the width of the video player
Height – Sets the height of the video player
preload attribute is set to *metadata* which instructs the browser to fetch only the video metadata (such as duration).

Reference: https://www.w3schools.com/tags/tag_video.asp

Stage 1: check the browser supports html5 video And Hide the Native Controls

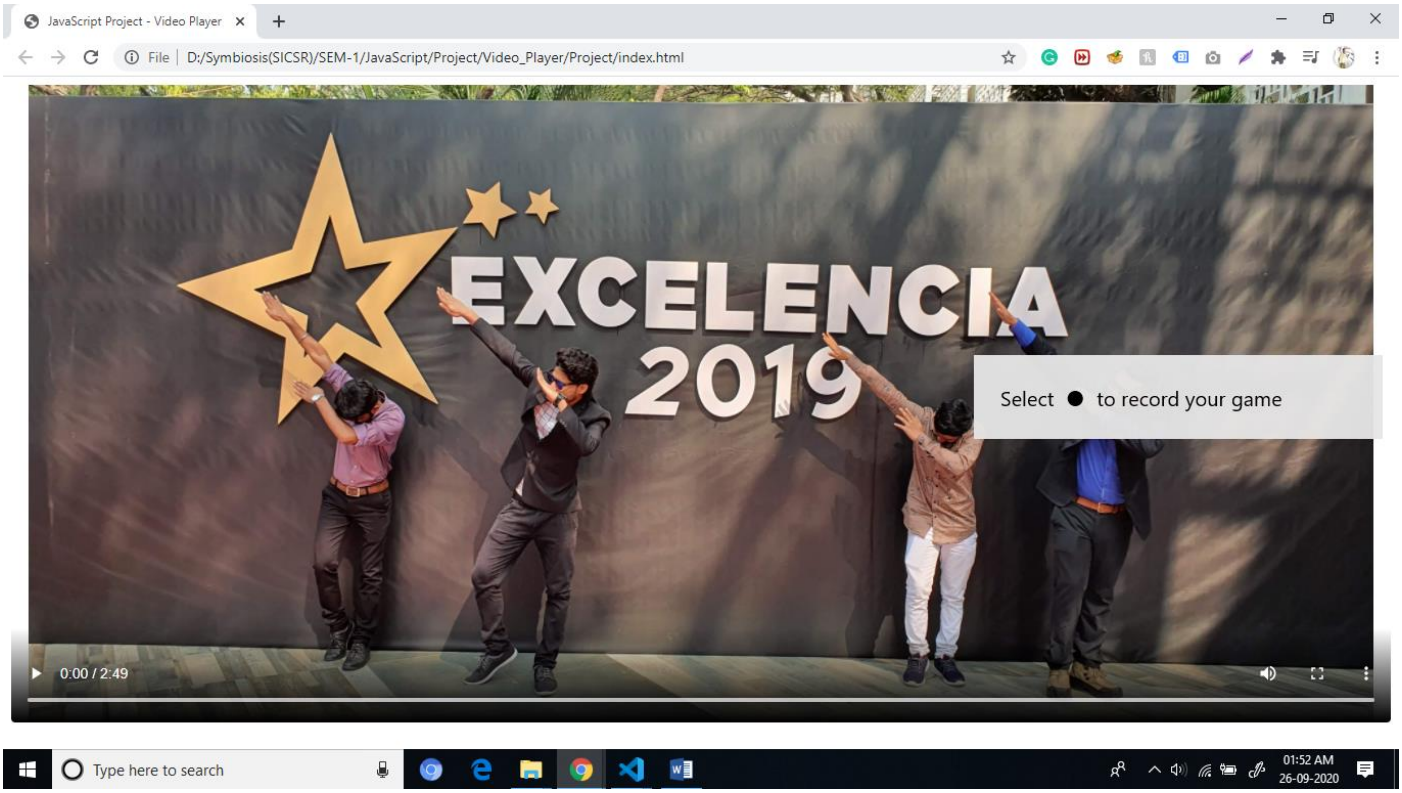
```
/**
const video = document.getElementById('video');
const videoControls = document.getElementById('video-controls');

const videoWorks = !!document.createElement('video').canPlayType;

if (videoWorks) {
  video.controls = false;
  videoControls.classList.remove('hidden');
}
**/
```

The *canPlayType* property is how we are able to detect support for a video format in the browser. To use it, we need to create an instance of the *<video>* element and check if it supports the *canPlayType* method.

Reference: https://www.w3schools.com/tags/ref_av_dom.asp



Stage2: Play/Pause Video Button icon should change on click on the button.

Index.js:
const playButton = document.getElementById('play');

```
function togglePlay() {
  if (video.paused || video.ended) {
    video.play();
  } else {
    video.pause();
  }
}
```

Event Listener that executes the togglePlay function when PlayButton is Clicked.

playButton.addEventListener('click', togglePlay);

HTML code for PlayButton:

```
        <div class="bottom-controls">
          <div class="left-controls">
            <button data-title="Play (k)" id="play">
              <svg class="playback-icons">
                <use href="#play-icon"></use>
                <use class="hidden" href="#pause"></use>
              </svg>
            </button>
```

We use the SVG for Play and Pause the button and link it using #.
But only one display at a time and other one is Hidden.

Now Toggle the Icon depending on the Play and Pause.

Index.js
const playbackIcons = document.querySelectorAll('.playback-icons use');

```
function updatePlayButton() {
  playbackIcons.forEach(icon => icon.classList.toggle('hidden'));
}
```

Add EventListner:

```
video.addEventListener('play', updatePlayButton);
video.addEventListener('pause', updatePlayButton);
```

When the video is played or paused, the **updatePlayButton** function is executed which toggles the **hidden** class on each button. Since we have the hidden class on the pause icon by default, once the video is played, this icon will be displayed and the play icon will be hidden.

One extra thing to do is updating the text in the tooltip that appears when you hover over the play button. It reads **play (k)** by default, but we need to update it so it reads **pause (k)** when the video is playing.

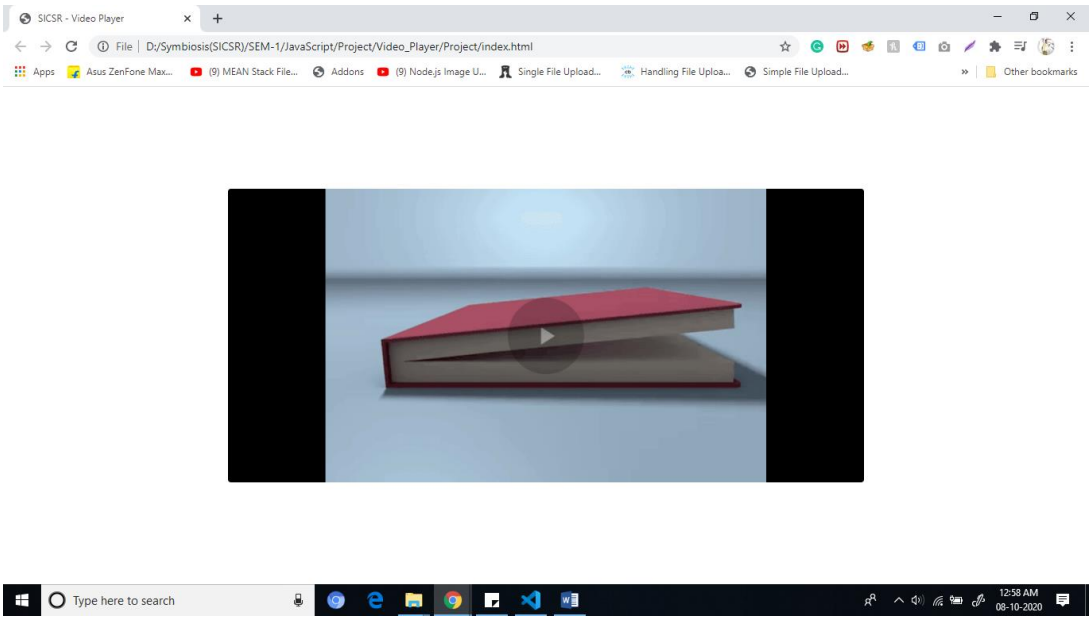
```
function updatePlayButton() {
  playbackIcons.forEach(icon => icon.classList.toggle('hidden'));

  if (video.paused) {
    playButton.setAttribute('data-title', 'Play (k)')
  } else {
    playButton.setAttribute('data-title', 'Pause (k)')
  }
}
```

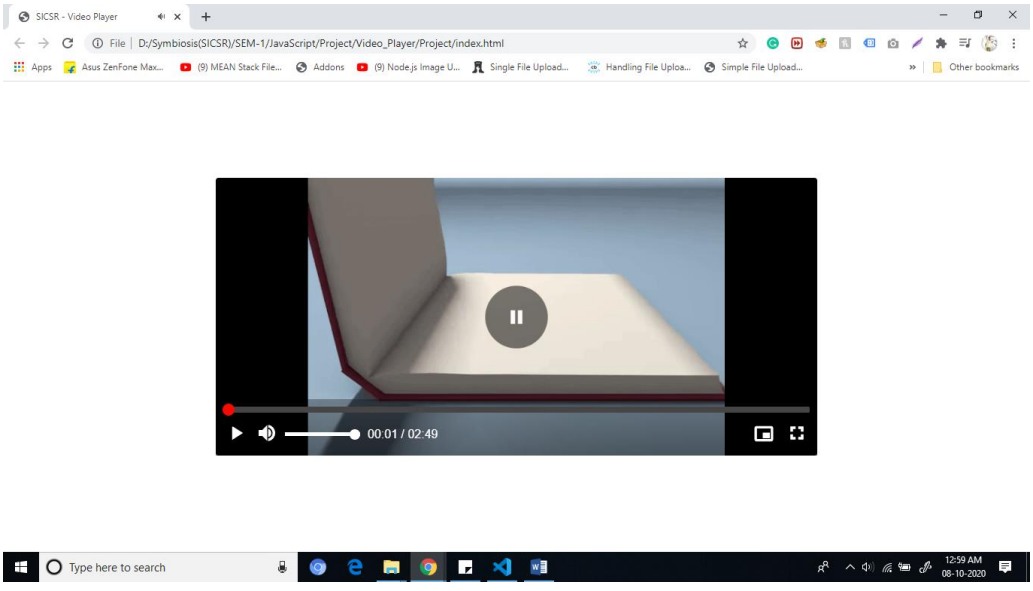
Data-title is CSS part u can check in style.css – button:before , button:hover::before

Screenshot:

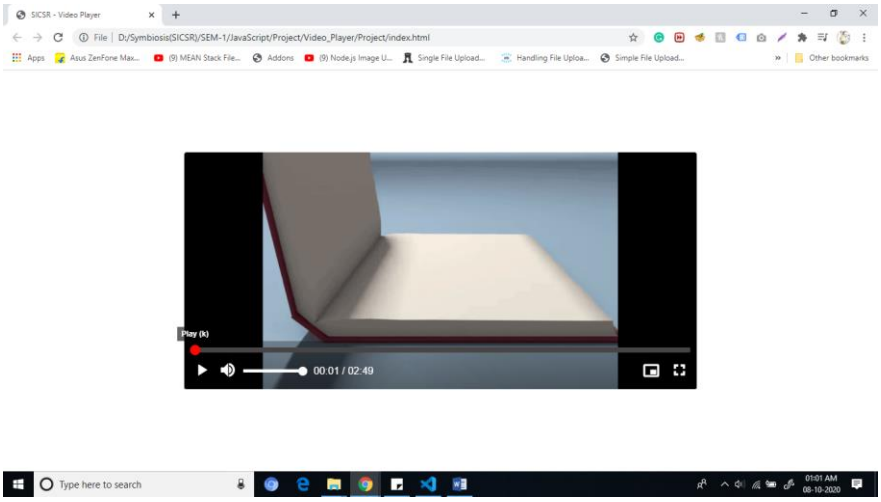
Play Button:



Pause Button:



Tooltip in Play and Pause Button:



Step 3: Show the video duration and time elapsed

1. Add Html code for Video Duration (00:00)

2. Add Controls in Index.js

   const timeElapsed = document.getElementById('time-elapsed');
   const duration = document.getElementById('duration');

   Duration property show the total duration of the video once the page loads.
   Duration property represent in Number of Seconds of the video we need to convert into Minutes and Seconds.
   For this Create the function will take a time in seconds and converts into Minute and Seconds.

```
function formatTime(timeInSeconds) {
  const result = new Date(timeInSeconds * 1000).toISOString().substr(11, 8);

  return {
    minutes: result.substr(3, 2),
    seconds: result.substr(6, 2),
  };
};
```

After Calculating Time then we need to add in video when it's initialized.

```
function initializeVideo() {
  const videoDuration = Math.round(video.duration);
  const time = formatTime(videoDuration);
  duration.innerText = `${time.minutes}:${time.seconds}`;
  duration.setAttribute('datetime', `${time.minutes}m ${time.seconds}s`)
}
```

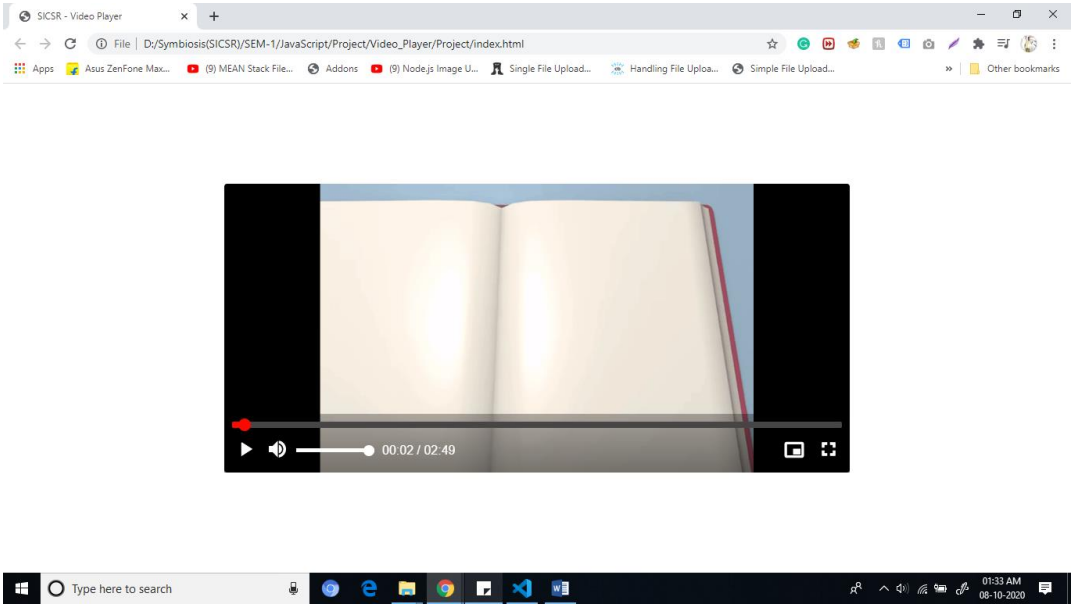
datetime attribute is also updated to a time string that represents the duration of the video. The above code ensures that once the video's *currentTime* is updated by virtue of playing the video, the elapsed time is also updated appropriately.

initializeVideo function to the video's loadedmetadata.

This will cause the video duration to be updated when the video's metadata has been loaded.

video.addEventListener('loadedmetadata', initializeVideo);

Screenshot:
Video Duration/Time Elapsed



Step4: Updating in progress bar

Add HTML code in Index.html

```
. . .
<div class="video-progress">
  <progress id="progress-bar" value="0" min="0"></progress>
  <input class="seek" id="seek" value="0" min="0" type="range" step="1">
  <div class="seek-tooltip" id="seek-tooltip">00:00</div>
</div>
. . .
```

The ***min*** attribute on both is set to zero, and the *value* attribute indicates the current value of both elements. They also need a ***max*** attribute which will be set to the duration of the video in seconds which is from **video.duration** *we will check in the* **initializeVideo** function.

```
const progressBar = document.getElementById('progress-bar');
const seek = document.getElementById('seek');
```

Update code in **initializeVideo** function.

```
seek.setAttribute('max', videoDuration);
  progressBar.setAttribute('max', videoDuration);
```

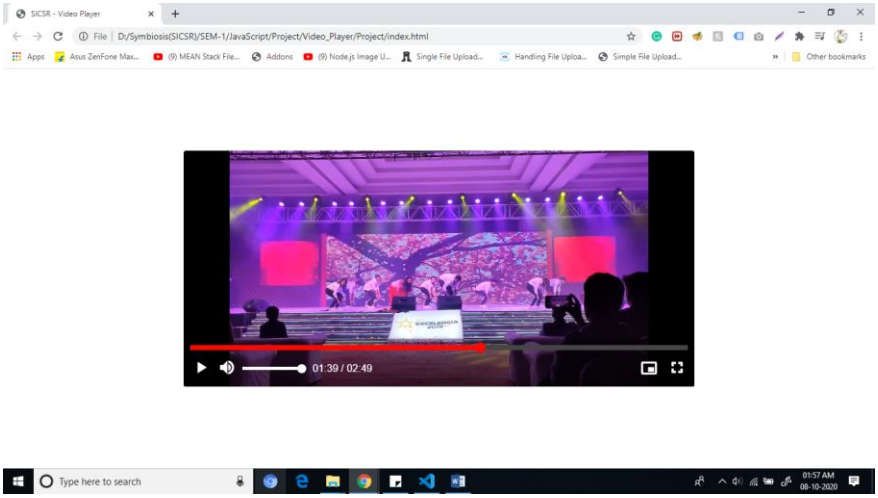
When the video is being played so that the progress bar becomes updated.

```
function updateProgress() {
  seek.value = Math.floor(video.currentTime);
  progressBar.value = Math.floor(video.currentTime);
}

video.addEventListener('timeupdate', updateProgress);
```

Screenshot:

Progress Bar:



Step 5: Skip ahead

Click on the progress bar then jump to the particular point in the video.

```
const seekTooltip = document.getElementById('seek-tooltip');
```

Then add a function to display the timestamp in the tooltip element when the cursor is over the progress bar:

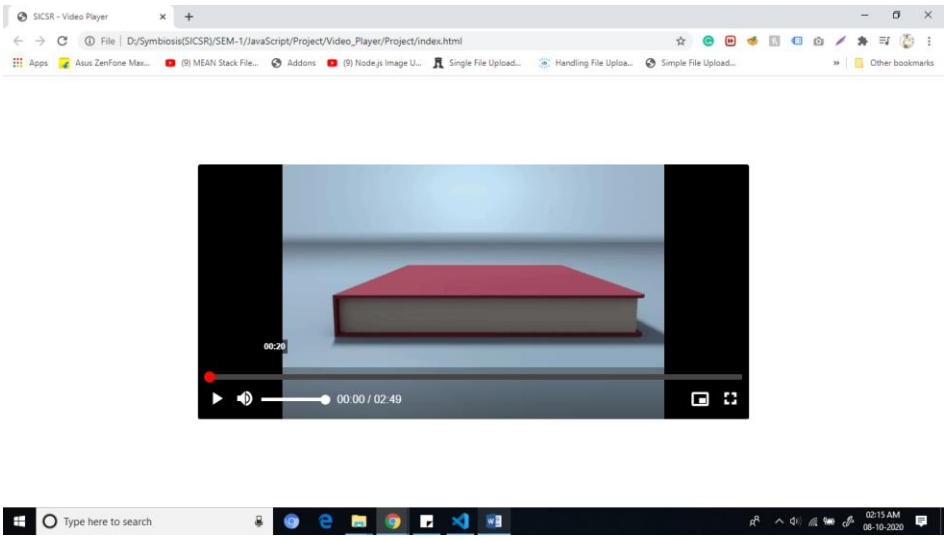
```
// updateSeekTooltip uses the position of the mouse on the progress bar to
function updateSeekTooltip(event) {
  const skipTo = Math.round((event.offsetX / event.target.clientWidth) *
parseInt(event.target.getAttribute('max'), 10));
  seek.setAttribute('data-seek', skipTo)
  const t = formatTime(skipTo);
  seekTooltip.textContent = `${t.minutes}:${t.seconds}`;
  const rect = video.getBoundingClientRect();
  seekTooltip.style.left = `${event.pageX - rect.left}px`;
}
```

seek element to roughly work out where in the range input the user is hovering on, and stores the position in a *data-seek* attribute while updating the tooltip to reflect the timestamp at that position.

seek.addEventListener('mousemove', updateSeekTooltip);

Screenshot:

Skip Ahead:



Once the value of *seek* element is changed either by clicking or dragging the thumb, we want the video to jump to the time set in the *data-seek* attribute. Create a new **skipAhead** function below ***updateSeekTooltip***.

```
// skipAhead jumps to a different point in the video when
// the progress bar is clicked
function skipAhead(event) {
  const skipTo = event.target.dataset.seek ? event.target.dataset.seek : event.target.value;
  video.currentTime = skipTo;
  progressBar.value = skipTo;
  seek.value = skipTo;
}
```

This function will be executed when the value of the *seek* element changes can be monitored using the *input* event. We then get the value of the *data-seek* attribute and check if it exists. If it does, we grab the value and update the video's elapsed time and the progress bar to that value. If the *data-seek* property does not exist (on mobile for example), the value of the *seek* element is used instead.

```
seek.addEventListener('input', skipAhead);
```

Step 6: Volume controls

Add HTML code for Volume Controls in Indedx.html

The first thing we need to do is update the volume of the video when the value of the *volume* element changes. We also need to update the icon to reflect the current volume of the video.
Volume input ranges from 0 to 1, and each step in the input increases the volume by 0.01.

0 is the lowest volume and 1 is the highest.

updateVolume function to update the volume as soon as the volume input is changed.

```
// updateVolume updates the video's volume and disables the muted state if active
function updateVolume() {
  if (video.muted) {
    video.muted = false;
  }
  video.volume = volume.value;
}

volume.addEventListener('input', updateVolume);
```

Now,update the Volume Icon

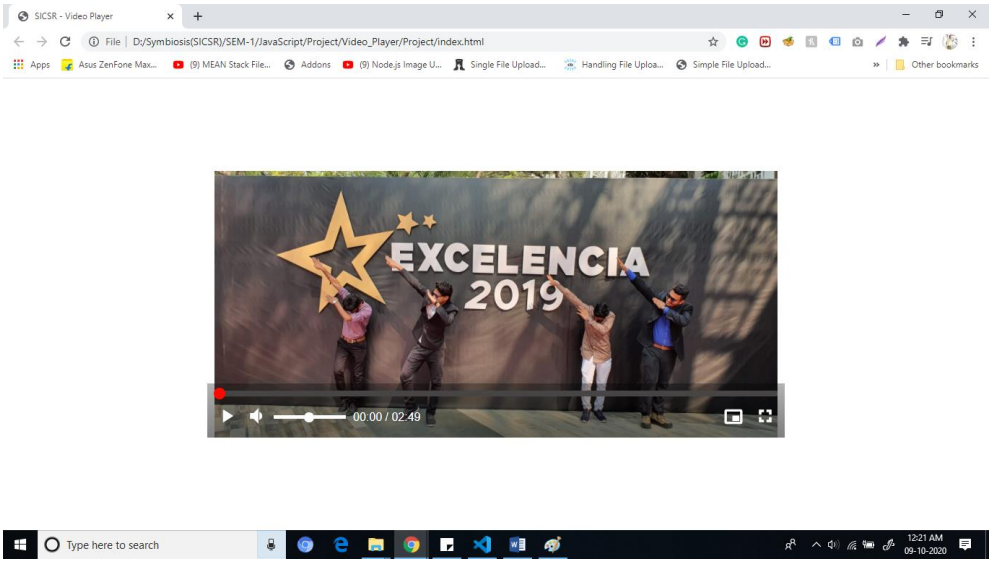
```
// updateVolumeIcon updates the volume icon so that it correctly reflects
function updateVolumeIcon() {
  volumeIcons.forEach(icon => {
    icon.classList.add('hidden');
  });

  volumeButton.setAttribute('data-title', 'Mute (m)')

  if (video.muted || video.volume === 0) {
    volumeMute.classList.remove('hidden');
    volumeButton.setAttribute('data-title', 'Unmute (m)')
  } else if (video.volume > 0 && video.volume <= 0.5) {
    volumeLow.classList.remove('hidden');
  } else {
    volumeHigh.classList.remove('hidden');
  }
}

video.addEventListener('volumechange', updateVolumeIcon);
```

Screenshot:
Increasing/Decreasing Volume



Now, mute and unmute the video by clicking on the volume icon.
Create a new ***toggleMute*** function for this purpose:

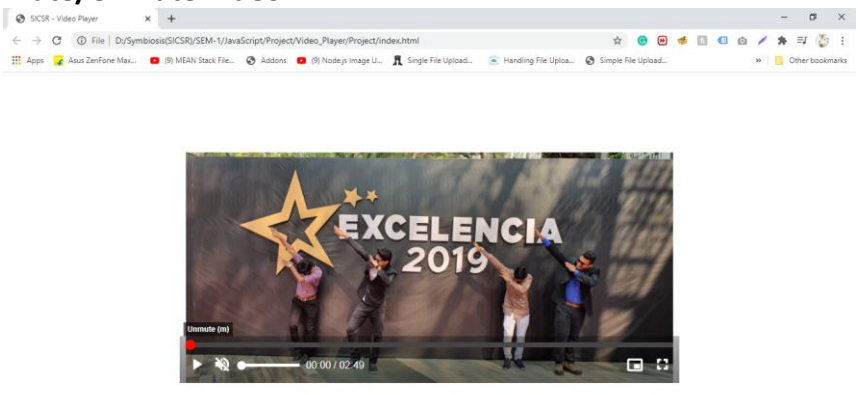
```
// toggleMute mutes or unmutes the video when executed
// When the video is unmuted, the volume is returned to the value
function toggleMute() {
  video.muted = !video.muted;

  if (video.muted) {
    volume.setAttribute('data-volume', volume.value);
    volume.value = 0;
  } else {
    volume.value = volume.dataset.volume;
  }
}

volumeButton.addEventListener('click', toggleMute);
```

When the video is muted, the volume is stored in a *data-volume* attribute on the *volume* element, so that when the video is unmuted, we can restore the state of the volume to its previous value.

Screenshot:
Mute/Unmute Video



Step 7: Full-screen video

For Full Screen Video we have to select the .video-container element and ask the browser to place it (and its descendants) in full-screen.

```
const fullscreenButton = document.getElementById('fullscreen-button');
const videoContainer = document.getElementById('video-container');

// toggleFullScreen toggles the full screen state of the video
// If the browser is currently in fullscreen mode,then it must be exited and vice versa.
function toggleFullScreen() {
  if (document.fullscreenElement) {
    document.exitFullscreen();
  } else {
    videoContainer.requestFullscreen();
  }
}
```

Update the Full Screen Icon

```
const fullscreenIcons = fullscreenButton.querySelectorAll('use');

// updateFullscreenButton changes the icon of the full screen button
function updateFullscreenButton() {
  fullscreenIcons.forEach(icon => icon.classList.toggle('hidden'));

  if (document.fullscreenElement) {
    fullscreenButton.setAttribute('data-title', 'Exit full screen (f)')
  } else {
    fullscreenButton.setAttribute('data-title', 'Full screen (f)')
  }
}

videoContainer.addEventListener('fullscreenchange', updateFullscreenButton);
```

Screenshot:
Full Screen Mode

Step 8: Add Picture-In-Picture support

Allows users to watch videos in a floating window so they can keep an eye on what they're watching while interacting with other sites, or applications.

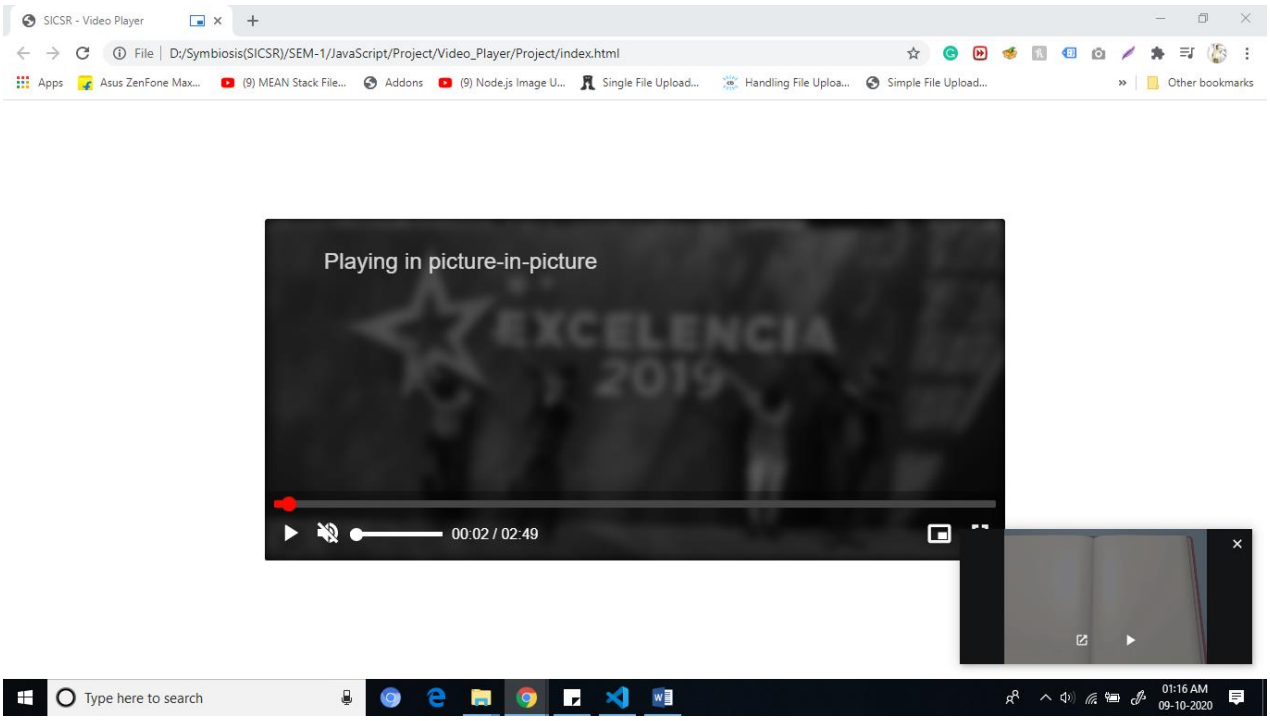
```
const pipButton = document.getElementById('pip-button')

document.addEventListener('DOMContentLoaded', () => {
  if (!('pictureInPictureEnabled' in document)) {
    pipButton.classList.add('hidden');
  }
});

// togglePip toggles Picture-in-Picture mode on the video
async function togglePip() {
  try {
    if (video !== document.pictureInPictureElement) {
      pipButton.disabled = true;
      await video.requestPictureInPicture();
    } else {
      await document.exitPictureInPicture();
    }
  } catch (error) {
    console.error(error)
  } finally {
    pipButton.disabled = false;
  }
}

pipButton.addEventListener('click', togglePip);
```

Screenshot:
PIP Mode



Step 9: Add keyboard shortcuts

Perform specific action when specific key is pressed.

Shortcut we used:

- *k*: Play or pause the video
- *m*: Mute or unmute the video
- *f*: Toggle full screen
- *p*: Toggle Picture-in-Picture mode

Keyup Event -- detect the key that was pressed and run the relevant functions for key.

```
document.addEventListener('keyup', keyboardShortcuts);
```

Future Scope:

- Add support for captions and subtitles.
- Add speed support.
- Add the ability to fast-forward or rewind the video.
- Add ability to choose video resolution (720p, 480p, 360p, 240p).

Read About the:

ForEach Loop
Event Listener
ClassList in JS
setAttribute
substr
toISOString
innerText
FormatTime
Document.fullscreenelement
Document.exitFullscreen
.requestFullscreen
DOMContentLoaded
document.getElementById('pip-button')