## Application building

```
<!DOCTYPE html>

<html>

<head>

  <meta charset="utf-8">

  <script src="https://cdn.jsdelivr.net/npm/@mediapipe/camera_utils/camera_utils.js"
crossorigin="anonymous"></script>

  <script src="https://cdn.jsdelivr.net/npm/@mediapipe/control_utils/control_utils.js"
crossorigin="anonymous"></script>

  <script src="https://cdn.jsdelivr.net/npm/@mediapipe/drawing_utils/drawing_utils.js"
crossorigin="anonymous"></script>

  <script src="https://cdn.jsdelivr.net/npm/@mediapipe/hands/hands.js"
crossorigin="anonymous"></script>

  <script src="https://cdn.jsdelivr.net/npm/@mediapipe/holistic/holistic.js"
crossorigin="anonymous"></script>



  <script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@2.0.0/dist/tf.min.js"></script>

  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>

  <!-- <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css"> -->

  <!-- <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"></script>
-->

  <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css">

  <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/font-awesome/4.7.0/css/font-
awesome.min.css">

  <script src="https://code.jquery.com/jquery-3.4.1.slim.min.js"></script>

  <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js"></script>

  <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js"></script>

  <link rel="stylesheet" href = "css/styles.css">


  <script type="module">
```

```
// Step 1: Load HTML elements to JS Variables

 const videoElement = document.getElementsByClassName('input_video')[0];

 const canvasElement = document.getElementsByClassName('output_canvas')[0];

 const canvasCtx = canvasElement.getContext('2d');

 const labelElement = document.getElementById("label");

 const modelButton = document.getElementById('model-change');

 const backspaceButton = document.getElementById('backspace');

 const modelName = document.getElementById('model-name');

 const mainSentence = document.getElementById('global-sentence');

 const cameraButton = document.getElementById('camera-switch');

 const spaceButton = document.getElementById('space');

 const clearButton = document.getElementById('clear');



// Step 2: Intialize Variables for Model and labels

 let alphabetLabels = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'U', 'V',
'W', 'X', 'Y']

 let gestureLabels = ['eat', 'friend', 'Hello', 'Help me', 'Home', 'how', 'Love You', 'my', 'name', 'No',
'Some', 'Sorry', 'Thanks', 'Understand', 'Yes']

 let emptyLandmark = [[0.0, 0.0], [0.0, 0.0], [0.0, 0.0], [0.0, 0.0], [0.0, 0.0], [0.0, 0.0], [0.0, 0.0], [0.0,
0.0], [0.0, 0.0], [0.0, 0.0], [0.0, 0.0], [0.0, 0.0], [0.0, 0.0], [0.0, 0.0], [0.0, 0.0], [0.0, 0.0], [0.0, 0.0], [0.0,
0.0], [0.0, 0.0], [0.0, 0.0], [0.0, 0.0]]

 var labels;

 let label = '';

 var model;

 let currentModel = "Gesture";

 let predictionArray = [];

 let globalSentence = '';
```

```
// Step 3: Initialize static functions used for calculation
 function get_angles(a,b,c){
   const ang = (Math.atan2(c[1]-b[1], c[0]-b[0]) - Math.atan2(a[1]-b[1], a[0]-b[0])) * (180/Math.PI);
   if(ang<0)
     return 360 + ang;
   else
     return ang;
 }

 function indexOfMax(arr) {
   if (arr.length === 0) {
       return -1;
   }
   var max = arr[0];
   var maxIndex = 0;
   for (var i = 1; i < arr.length; i++) {
       if (arr[i] > max) {
           maxIndex = i;
           max = arr[i];
       }
   }
   return maxIndex;
}

modelButton.addEventListener('click', function(){

 console.log("Button Value: ", modelButton.innerHTML);
 if(modelButton.innerHTML == "Alphabet Model"){

   modelButton.innerHTML = "Gesture Model";
```

```
        let loadedModel = loadGestureModel().then((resolve, reject)=>{   //Handing the promise from
before
          model = resolve;
        }).then(function() {
          console.log("Model: ", model);
          currentModel = "Gesture";
          console.log("Current Model: ", currentModel);
        })
      }

      else{

        modelButton.innerHTML = "Alphabet Model";
        let loadedModel = loadAlphabetModel().then((resolve, reject)=>{   //Handing the promise from
before
          model = resolve;
        }).then(function() {
          console.log("Model: ", model);
          currentModel = "Alphabet";
          console.log("Current Model: ", currentModel);
        })
      }
    });



    function getMainSentence(){
      const mainSentence = document.getElementById('global-sentence');
      return mainSentence;
    }

    spaceButton.addEventListener('click', function(){
      mainSentence.textContent+='\xa0\xa0';
```

```
  globalSentence+='\xa0\xa0';

});

clearButton.addEventListener('click', function(){
  mainSentence.textContent='';
  globalSentence="";

});

backspaceButton.addEventListener('click', function(){

  const ms = getMainSentence();

  console.log("Length: ", ms.textContent);
  if(ms.textContent.length != 0){
    globalSentence=ms.textContent;
    let sentence = globalSentence.trim();
    console.log("Sentece: ", sentence);
    const myArray = sentence.split(" ");
    console.log("my Array: ", myArray);
    console.log("Size: ", myArray.length);
    myArray.pop();

     const newArray =myArray;
    // console.log("new Array: ", newArray);
     let newSentence = "";
    for(const word of newArray){
      newSentence += word + ' ';
    }
    console.log("New Sentence: ", newSentence);
```

```
      mainSentence.textContent = newSentence;
      globalSentence=newSentence;
    }
    else{
      mainSentence.textContent = "Global Sentence"
    }
  });




  // Step 4: Start loading the model asynchronously
    console.log("Start Load");

    async function loadAlphabetModel(){     // Promise of loading model from JSON file
      const model = await tf.loadLayersModel('./models/alphabets/model.json')
      labels = alphabetLabels;
      return model;
    }

    async function loadGestureModel(){     // Promise of loading model from JSON file
      const model = await tf.loadLayersModel('./models/gestures/model.json')
      labels = gestureLabels;
      return model;
    }

    let loadedModel = loadGestureModel().then((resolve, reject)=>{   //Handing the promise from
before
      model = resolve;
    }).then(function() {
```

```
    console.log("Model: ", model)
  });

  console.log("Loaded Model: ", loadedModel);




// Step 5: Async prediction function which predicts the probabilities of classes and return label
  async function predictModel(input){
    console.log("Input is: ", input.arraySync());
    const predictionArr = await model.predict(input);
    const prediction =  indexOfMax(predictionArr.arraySync()[0]);
    console.log("ArgMax: ", prediction);

    return labels[prediction];
  }



// Step 6: The required onResults function for the Mediapipe model. Collect landmarks and
// format them as 3D tensors before passing it to the predct function

  function onAlphabetResults(results) {

    canvasCtx.save();
    canvasCtx.clearRect(0, 0, canvasElement.width, canvasElement.height);
    canvasCtx.drawImage(results.image, 0, 0,
                        canvasElement.width, canvasElement.height);

    if (results.leftHandLandmarks || results.rightHandLandmarks) {
      const landmark_list =[];
      if (results.leftHandLandmarks){
        console.log("Left hand");
        const angle_1 = get_angles((results.leftHandLandmarks[4]['x'],
results.leftHandLandmarks[4]['y']),
                              (results.leftHandLandmarks[0]['x'], results.leftHandLandmarks[0]['y']),
                              (results.leftHandLandmarks[20]['x'], results.leftHandLandmarks[20]['y']));
        const angle_2 = get_angles((results.leftHandLandmarks[8]['x'],
results.leftHandLandmarks[8]['y']),
                              (results.leftHandLandmarks[5]['x'], results.leftHandLandmarks[5]['y']),
                              (results.leftHandLandmarks[10]['x'], results.leftHandLandmarks[10]['y']));

        for(const landmarks of results.leftHandLandmarks){
          const temp = [landmarks.x * 480, landmarks.y * 640]   //ADJUST LATER
          landmark_list.push(temp);
          // landmark_list.push(landmarks.x * 480);
          // landmark_list.push(landmarks.y * 640);
        }
        // landmark_list.push([angle_1, angle_2])
        // landmark_list.push(angle_1);
        // landmark_list.push(angle_2);
        // console.log("Landmark List: ", landmark_list);
      }
      else if(results.rightHandLandmarks){
        console.log("Right hand");
        const angle_1 = get_angles((results.rightHandLandmarks[4]['x'],
results.rightHandLandmarks[4]['y']),
                              (results.rightHandLandmarks[0]['x'], results.rightHandLandmarks[0]['y']),
                              (results.rightHandLandmarks[20]['x'], results.rightHandLandmarks[20]['y']));
        const angle_2 = get_angles((results.rightHandLandmarks[8]['x'],
results.rightHandLandmarks[8]['y']),
                              (results.rightHandLandmarks[5]['x'], results.rightHandLandmarks[5]['y']),
                              (results.rightHandLandmarks[10]['x'], results.rightHandLandmarks[10]['y']));
          for(const landmarks of results.rightHandLandmarks){
            const temp = [landmarks.x * 480, landmarks.y * 640]   //ADJUST LATER
            landmark_list.push(temp);
            // landmark_list.push(landmarks.x * 480);
            // landmark_list.push(landmarks.y * 640);
          }
          // landmark_list.push([angle_1, angle_2])
          // landmark_list.push(angle_1);
          // landmark_list.push(angle_2);
        }

        let predictionMade = predictModel(tf.tensor([landmark_list])).then((resolve, reject)=>{
          label = resolve;
          labelElement.textContent = label;
          console.log("Promise Output: ", resolve, reject);
        }).then(function() {
            console.log("Answer: ", label);

            predictionArray.push(label);
            if(predictionArray.length > 20)
              predictionArray = predictionArray.slice(-20);
            console.log(predictionArray);

            let distinctItems = [...new Set(predictionArray)];

            if(distinctItems.length == 1 && distinctItems.slice(-1)==label &&
predictionArray.length==20){
              globalSentence += label;
              // predictionArray.splice(0, predictionArray.length);
              predictionArray = []
              mainSentence.textContent = globalSentence;
        }
        console.log("Global Sentence: ", globalSentence);

      });

    }

    drawConnectors(canvasCtx, results.leftHandLandmarks, HAND_CONNECTIONS,
                {color: '#858585', lineWidth: 2});  //#CC0000
    drawLandmarks(canvasCtx, results.leftHandLandmarks,
                {color: '#f77979', lineWidth: 1});  //00FF00
    drawConnectors(canvasCtx, results.rightHandLandmarks, HAND_CONNECTIONS,
                {color: '#858585', lineWidth: 2});
    drawLandmarks(canvasCtx, results.rightHandLandmarks,
                {color: '#f0a66e', lineWidth: 1});

    canvasCtx.restore();
  }



//////////////////////////////////////////////////////////////////////

  function onGestureResults(results) {
    canvasCtx.save();
    canvasCtx.clearRect(0, 0, canvasElement.width, canvasElement.height);
    canvasCtx.drawImage(results.image, 0, 0, canvasElement.width, canvasElement.height);

    if (results.leftHandLandmarks || results.rightHandLandmarks) {

      let landmark_list =[];
      let lh = [];
      let rh = []

      if(!results.leftHandLandmarks){
        lh = emptyLandmark;
      }
      else{
        for(const landmarks of results.leftHandLandmarks){
        const temp = [landmarks.x, landmarks.y]   //ADJUST LATER
        lh.push(temp);
        }
      }

      if(!results.rightHandLandmarks){
        rh = emptyLandmark;
      }
      else{
        for(const landmarks of results.rightHandLandmarks){
        const temp = [landmarks.x, landmarks.y]   //ADJUST LATER
        rh.push(temp);
        }
      }

      landmark_list = lh.map(function(e, i) {
        return [e[0], e[1],  rh[i][0], rh[i][1]];
      });

      let predictionMade = predictModel(tf.tensor([landmark_list])).then((resolve, reject)=>{
        label = resolve;
        labelElement.textContent = label;
        console.log("Promise Output: ", resolve, reject);
```

```
          }).then(function() {
            console.log("Answer: ", label);

            predictionArray.push(label);
            if(predictionArray.length > 20)
              predictionArray = predictionArray.slice(-20);
            console.log(predictionArray);

            let distinctItems = [...new Set(predictionArray)];

            if(distinctItems.length == 1 && distinctItems.slice(-1)==label &&
predictionArray.length==20){
              globalSentence += ' ' + label;
              // predictionArray.splice(0, predictionArray.length);
              predictionArray = []
              mainSentence.textContent = globalSentence;
            }
            console.log("Global Sentence: ", globalSentence);

          });

        }

        drawConnectors(canvasCtx, results.leftHandLandmarks, HAND_CONNECTIONS,
                   {color: '#858585', lineWidth: 2});  //#CC0000
        drawLandmarks(canvasCtx, results.leftHandLandmarks,
                   {color: '#f77979', lineWidth: 1});  //00FF00
        drawConnectors(canvasCtx, results.rightHandLandmarks, HAND_CONNECTIONS,
                   {color: '#858585', lineWidth: 2});
        drawLandmarks(canvasCtx, results.rightHandLandmarks,
                   {color: '#f0a66e', lineWidth: 1});
```

```
  canvasCtx.restore();
 }

 // Deciding which Results fucntion to use
 function onResults(results){
  if(currentModel == "Gesture")
    onGestureResults(results);
  else
    onAlphabetResults(results);
 }

// Step 7: Defining Holistic Model for pose detection

 const holistic = new Holistic({locateFile: (file) => {
  return `https://cdn.jsdelivr.net/npm/@mediapipe/holistic/${file}`;
 }});

 holistic.setOptions({
  modelComplexity: 1,
  smoothLandmarks: true,
  enableSegmentation: true,
  smoothSegmentation: true,
  refineFaceLandmarks: true,
  minDetectionConfidence: 0.7,
  minTrackingConfidence: 0.7
 });

 holistic.onResults(onResults);

 const camera = new Camera(videoElement, {
  onFrame: async () => {
          await holistic.send({image: videoElement});
        },
        width: 640,
        height: 480
      });

      camera.start();



      cameraButton.addEventListener('click', function(){
        // const content = cameraButton.textContent;
        if(cameraButton.textContent == 'Camera Off'){
          camera.stop();
          cameraButton.textContent = 'Camera On';
        } else{
          camera.start();
          cameraButton.textContent = 'Camera Off';
        }
      });

  </script>

</head>

<body>
  <section id="nav-bar">
    <nav class="navbar navbar-expand-lg navbar-light bg-dark ">
        <a class="navbar-brand" href="#"><img src="./assets/logo_cropped.png"></a>
        <button class="navbar-toggler" type="button" data-toggle="collapse" data-
target="#navbarNav" aria-expanded="true" aria-label="Toggle navigation" aria-
controls="navbarNav">
          <span class="navbar-toggler-icon"></span>
        </button>
        <div class="navbar-collapse collapse show" id="navbarNav" style="">
          <ul class="navbar-nav ml-auto">
            <li class="nav-item">
              <a class="nav-link" href="./sample.html">TRANSLATE</a>
            </li>
            <li class="nav-item">
              <a class="nav-link" href="./info.html">INFO</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="./aboutus.html">ABOUT US</a>
            </li>
          </ul>
        </div>
      </nav>
</section>

<div class="jumbotron jumbotron-fluid">
  <div class="container">
    <h1 class="display-4">Bridging gaps in communication</h1>
    <p class="lead">Real-Time Communication System Powered by AI for Specially Abled</p>
  </div>
</div>



<div id="content">


<div id="global-content">
  <h2 id="global-sentence">Global Sentence</h2>
</div>
```

```
<div class="container">


  <div id="top-controls">
    <button id="space" class="button-30">Space</button>
    <button id="clear" class="button-30">Clear All</button>
    <button id="backspace" class="button-30">Backspace </button>
  </div>


  <div id="canvas-video">
    <div id="loader"></div>
    <video class="input_video"></video>
    <canvas class="output_canvas" width="640px" height="480px"></canvas>
</div>


<div id="model-controls">
    <button id="model-change" class="button-30">Gesture Model</button>
    <h3 id="label">Label Here.</h3>
    <button class="button-30" id="camera-switch">Camera Off</button>
</div>
</div>




</body>
</html>
```