Boarding Pass API Integration Guide

The boarding pass integration program is divided into 4 functions :

1. **Loading** : File object loading and filtering into file types images and pdfs

2. **Compression** : if file is single image, compress the image and create a data url.

3. **Pdf-To-Png** : If file is pdf conversion to images and create data urls.

4. **Detection API** request-response

Before the program begins, we will be creating few global variables.

```
let dataUrl =[]
let pdfNum;
const options = {
   maxSizeMB:1,
   maxWidthOrHeight: 1920,
   useWebWorker: true
}
```

- The first function is a simple loading function that acts on element.oninput() or element.onchange on the input element.
- The function works asynchronously as the functions called inside require promises.
- The function segragates objects on the basis of file type, if image then it calles streamer function.
- If the file is pdf, it calls pdfToPng.
- We are emptying the global array dataUrl because we want fresh data on every load event.

```
var loadFile = async function(event) {
      dataUrl = [];
      fileObj = event.target.files[0];
      if((fileObj.type=="image/png")||(fileObj.type=="image/jpg")||(fileObj.type=="image/jpeg"))
streamer(event.target.files[0]);
      else if(fileObj.type=="application/pdf") pdfToPng(input)
      else console.log("none")
}
```

IMPORTANT : the input variable in the above function is the name of the element variable which is attached to loadFile.

Compression

- The second function is streamer which does compression of the file, if the file is an image. For this an external JS file has to be used.
- The function is again an asynchronous function as it awaits compressed files.
- It also calls another function urlMaker which creates the URL for the compressed file.
- The function pushes the created URL in dataUrl global array.

```
<script src="compressor.js"></script>
```

```
function urlMaker(file){
   reader = new FileReader();
   reader.onloadend = function () {
      dataUrl.push(reader.result);
   };
   reader.readAsDataURL(file);
}

async function streamer(file){
               cf = await imageCompression(file, options);
               await urlMaker(cf);
};
```

PDF To PNG

- The third function is that of pdfToPng which converts the given pdf file into corresponding images.
- Its a function which again uses an external library, PDF.js.
- The canvas element created is an invisible element and it is through painting on that canvas are we creating the pngs. Don’t change display to any other value.

```
<script src="https://cdn.jsdelivr.net/npm/pdfjs-dist@2.1.266/build/pdf.min.js"></script>
function pdfToPng(el) {
file = el.files[0]
fileReader = new FileReader();
fileReader.onload = function(ev) {
   pdfjsLib.getDocument(fileReader.result).then(function getPdfHelloWorld(pdf) {
      pdfNum = pdf.numPages;
      for(let i=1;i<=pdfNum;i++){
         pdf.getPage(i).then(function getPageHelloWorld(page) {
         var scale = 1.5;
         var viewport = page.getViewport(scale);
         var canvas = document.createElement('canvas');
         canvas.style.display = "none";
         el.appendChild(canvas);

         var context = canvas.getContext('2d');
         canvas.height = viewport.height;
         canvas.width = viewport.width;
         var task = page.render({
            canvasContext: context,
            viewport: viewport
         })
         task.promise.then(function() {
            iimg = new Image();
            iimg = canvas.toDataURL('image/png');
            dataUrl.push(iimg);
         });
      });
      }
      },
        function(error) console.log(error);
    };
    fileReader.readAsArrayBuffer(file);
}
```

Detection API

- The fourth function is simple asynchronous function that calls the Boarding Pass Detection API, built by our team.
- It is ideally meant to work on a click event, ideally bound to a div/button element.
- The function name is check, and has 4 arguments, which are policy number, flight number, flight operator and data.
- The data is an array object which contains all the data URLs of the images.
- The function inside runs a loop to call the API for all the images present within the pdf document.
- In case of single image, the loop would only run once, as data would be carrying a single element only.
- The API also takes in an additional argument doc_num which keeps an account of page number of the given pdf.

```
async function check(policyNum,flightNum,flightOp,data){
    for(let i=0;i<data.length;i++)
    {
        let data = {
            "encoded_string": data[i],
            "policy_no": policyNum,
            "flight_no": flightNum,
            "flight_op": flightOp,
            "doc_num" : (i+1).toString()
        }

        let response = await fetch('https://39u3b3cep7.execute-api.ap-south-
1.amazonaws.com/prod/checker', {
            method: 'POST',
                  headers: {
                  'Content-Type': 'application/json;charset=utf-8',
                  'authorizationToken' : 'abc123'
                  },
                  mode: 'cors',
                  body: JSON.stringify(data)
                });

        let result = await response.json();
        return result;
    }

}
```

The result of this API would be in the form of a JSON object containing three elements.
1. Document Number : which will be 1 for images, and page-wise for PDFs
2. Score : score that the algorithm has given to that particular image/page number

3. Label : Boarding Pass or ‘Other’, giving a discrete label based on the threshold set by us.

To see the above program in full action, and to understand integration easily, kindly use the following link – https://kriteesh.github.io/tfcheck/index.html