**KYC App**

**Usage:**

This is an example app for KYC verification. The user is asked to enter their first name, last name, and date of birth. Then the user must scan their photo ID. This will only work if the ID contains a photo of the user. Once a photo ID has been detected by the camera, the outline will turn from red to green. The user can only scan their ID if the outline is green. After scanning, all the text is extracted from the image. Finally, each of the user input gets verified if it matches the scanned text. (See end of document).

**How it works:**

This app uses OpenCV.js for computer vision and tesseract.js for OCR. All other services I have found are either paid or only offer trail subscriptions.

The constructor of 'register.page.ts' gets the height and width of the screen so that the camera preview can have the correct dimensions. OpenCV and Tesseract are initialized and roi is also set accordingly:

Text

Description automatically generated

The user can input text using an HTML form in 'register.page.html'. The form data is dynamically stored in the 'userDetails' object in 'register.page.ts'. 'userDetails' contains the values of FirstName, LastName, and DateOfBirth. It also contains a boolean called verified for each field:

Text

Description automatically generated

Submitting the form calls the 'logForm()' function. This function makes sure that all the fields are not empty. Once this is verified, the camera is opened with 'openCamera()'.

The CameraPreview plugin from '@ionic-native/camera-preview' is user for the camera. This plugin allows custom camera overlays. In this case, the overlay contains four semitransparent divs part of the 'inner' class and one transparent div with a bounding box. [ngStyle] is used to set the position and dimensions of the top, left, right, bottom, and box divs:

Text

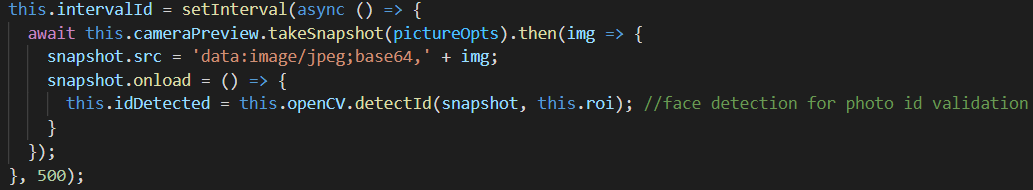
Description automatically generated

This binds the style of the divs to the 'cutout' object in 'register.page.ts'. The color of the bounding box is also dynamic based on if an ID has been detected by the camera. The dimension and postion values can be changed based on preference:

Text

Description automatically generated

If the camera is opened successfully, 'detectId()' is called. This function takes a snapshot of the camera preview window on a set interval of 500ms. Every interval, the snapshot is passed to 'openCV.detectId()'. This function also takes in a 'roi' (region of interest) object. This object contains the relative positions and dimensions of the bounding box div. These values allow OpenCV to crop the image so that only part of the snapshot inside the bounding box is visible. If 'openCV.detectId()' returns true, the boolean 'idDetected' becomes true and the bounding box outline turns from red to green. Now the user can click on the scan button:

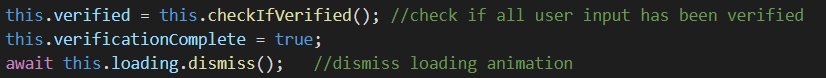


Scanning calls the 'capture()' function. This function stops the setInterval() loop and takes a picture of the camera preview. When the image has been captured, a loading animation starts and 'getVectors()' is called. This function uses OpenCV to get all of the text images from the original captured image. Once this is done, it performs OCR on all the images using tesseract.js:

Text

Description automatically generated

The user input is checked against the extracted text and verified accordingly. Once all user input has been checked, the loading animation is dismissed and 'verficationComplete' becomes true. This allows 'register.page.html' to display the verification results to the user:



**OpenCV:**

All opencv functions are in 'opencv-processing.service.ts'. This service uses the opencv.js library located in 'assets/lib'. At the top of the file, a variable called 'cv' is declared. This can be used by the service because opencv is included in the head of 'index.html':

Text

Description automatically generated

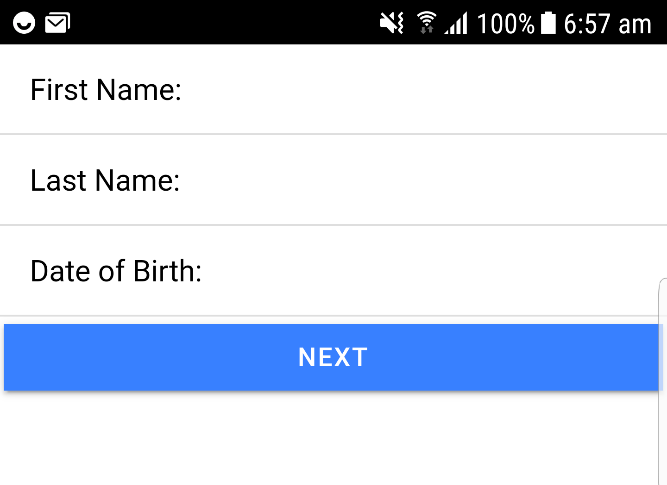
When OpenCV is initialized, a cascade classifier file is created from the data in 'assets/classifier/haarcascade\_frontalface\_default.xml'. The classifiers are used for face detection to make sure that the user is scanning a photo ID.

There are two main functions in the service: 'detectId()' and 'detectText()'. 'detectId()' takes an image and roi object as arguments. 'roi' contains the relative dimensions of the bounding box div in the camera preview. These dimensions are used in cv.roi() to crop the image. After the image is cropped, the cascade classifiers are used to detect faces. If a face is detected, the function returns true.

'detectText()' also takes an image and roi object as arguments. The image is cropped and resized for faster processing. The image mat vector is transformed to isolate the parts of the image with text. These parts are cropped into individual image vectors containing each line of text. At the end, the image vectors are returned.

**Tesseract.js**

All tesseract.js functions are in 'tesseract-ocr.service.ts'. These functions are used for text recognition in images. The 'loadWorker()' function creates a tesseract worker from local storage. The worker, core, and language paths are inside of the assets folder. 'assets/lang-data' contains the training data for English. Once the worker has been initialized, the app is able to recognize text using 'recognizeImage()'. This function takes a base64 image string as an argument.

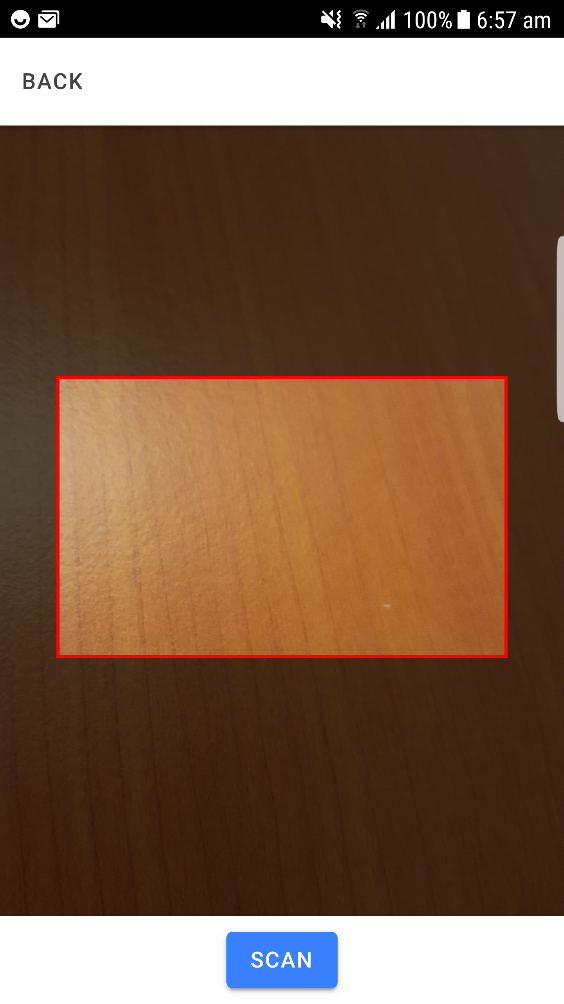


**Figure 1**

Graphical user interface, text, application, email

Description automatically generated

**Figure 3**



**Figure 2**