Universität St.Gallen

Dri-Age

Age Verification based on ID-Cards

Report

7,793,1.00 SHSG Summer School

Dominik Buchegger	(17-611-658)
Mario Mühlematter	(17-603-440)
Patric Naville	(14-612-204)
Jonathan Mösli	(21-610-324)
Luca Grand	(18-610-659)

St. Gallen/ 09. September 2022

Background:

The original case prompt was as follows:

- Build a mobile or web application
- The application should detect the date of birth from an image of the ID card (manually uploaded or taken with the camera)
- An administrator should be able to configure up to two limits (e.g., 16 and 18 years) and the application shall visualize the respective limit after scanning

Nowadays, when buying alcohol and cigarettes, entering a club, or buying a drink at a bar, age verification is done manually. Employees check the age on an ID or equivalent and then compare the photo with the face of the customer. This process is slow and prone to errors. Busy nights can get hectic, and employees can be tempted to reduce the precision of their verification process. This again opens a bigger margin for errors and compromises youth protection. To speed up the process and at the same time reduce the error margin, an app that checks the age of guests and customers can be a handy tool.

Key concept:

The User enters two age limits (e.g., 16, 18) and then proceeds to scan the ID of a customer or upload a photo. The program in the background then calculates the current age of the customer and compares it to the predefined age limits. Dri-Age then visualizes which age limits the customer fulfils.

Technical:

Our python program is at the core of our application. It brings all external libraries together, executes the age calculation and verifies the required age limits. The eye of our product is the OCR-engine Tesseract that reads the code on the identification documents' machine-readable zone (MRZ) – that has been extracted by image manipulation – and returns its code in form of a string. Our python program then parses the string to assign day, month, and year of birth to their own variable. A module – Datetime – is then used to determine the current date. Afterwards, the date of birth is subtracted from the current date to determine the age of the customer. This age is then compared to the user-defined age limits and gives a True/False output for each of the defined age limits. To make the customer experience as smooth as possible, we used the Kivy framework to create an interface which is more intuitive than a terminal. The goal was to keep things simple. There are few buttons and one text box to define custom age limits. At the click of a button, we offer the ages 16 and 18 as pre-set options. Once the user has defined their required age limit configuration, they can proceed to the next page with a camera to take pictures of the identification documents or to upload a file. By pressing the button with a camera icon, the device's camera is triggered. The program takes the image and calculates the output according to the process explained above. The output is then shown with a see-through overlay, which is green for fulfilled age limits and red for age limits that were not fulfilled. If the age limit is set to 16 and 18 and the person whose ID is scanned is 17 years old, half the screen will be covered by a green overlay and the other half by a red overlay. The overlays are also labelled with their corresponding age limit. If the program does not recognize a MRZ-code due to a bad image, nothing happens and the user can re-scan the ID.

Problems:

In the following section, we highlight some problems we faced during development and how we overcame said obstacles.

1. Data, OCR and MRZ:

The initial challenge for our team was to select the data we needed for our age verification process. After some research, we decided to use the purpose-built and internationally standardised code in the machine-readable zone [MRZ] of ID documents. The standard is set out in ICAO Doc 9303. The challenge was to select the right parts of the pictures. We did not solve this problem from scratch, instead we found an existing solution called detect-MRZ. This saved us a huge amount of time.

2. Tesseract on mobile phones:

The second challenge we faced was loading our app on a mobile phone. There is a good argument for using kivy to develop applications because it allows deploying the same code on different operating systems. After building the application for a desktop operation system (windows & mac) we were confident that we can also convert it to a mobile app. However after some time we realized that tesseract uses an engine outside of python for its functionality and that it will for now not be possible to install tesseract on a mobile phone.

Because tesseract is lacking a so called recipe that allows building it into a mobile application we decided on running a server containing a small dockerized flask application where we can send images to and receive the recognized string back. It was helpful that we could use the same code on the server and after installing a few programs and setting corresponding firewall rules the program was up and running on a google cloud products virtual machine. After programming the mobile application – essentially a shortened version of the desktop app – we tried to store and open an image in the format needed for the flask server: bytes in BufferedStream format. Because ios made it so difficult to store files we decided to use the captured image immediately for the POST request to the server and finally achieved building a working app by using several functions to convert the image to the needed type.

Potential future features and implementations

Depending on additional implemented features, this code could prove more useful for our clients and / or become useful in other areas as well.

The first additional feature that we would implement is to save the scanned and analysed data for statistical analysis, e.g., displaying the number of people inside a dance club. Depending on legal clarification, the saved data could also include the names and sex of the checked people, which could allow for gender ratio management during an event or improve data for targeted marketing with return-customer analyses and loyalty programs.

With certain improvements, the code could also be implemented to further increase security in age verification for autonomous services. The self-check-out in supermarkets, for example, would not require a human to verify the age of a customer buying alcohol or tobacco anymore. With the age verification of our application and additional facial recognition, the self-check-out machine could determine the buyer's age reliably on its own. The same use case is also applicable to e-commerce, e-gaming, and accessing or registering for age-restricted websites.

Acknowledgements:

Our solution utilizes open-sourced code and libraries. The most important ones are *Tesseract, OpenCV, Kivy, tessdata_ocrb and detect-MRZ.* A big thank-you goes out to their creators for sharing them.