FotoFaces - A Face Repository Verification System

Filipe Gonçalves
DETI
University of Aveiro
Aveiro, Portugal
filipeg@ua.pt

Gonçalo Machado
DETI
University of Aveiro
Aveiro, Portugal
goncalofmachado@ua.pt

João Borges
DETI
University of Aveiro
Aveiro, Portugal
borgesjps@ua.pt

Pedro Fontes Lopes
DETI
University of Aveiro
Aveiro, Portugal
pdfl@ua.pt

Vicente Samuel
DETI
University of Aveiro
Aveiro, Portugal
vicente.costa@ua.pt

***Abstract*—Most modern companies maintain an active directory of users, some also have records of their photos, as such updating and validating this photos is a time consuming task, especially in large companies. Our system makes this process easier for both ends, the company and their associates. Building a mobile application automates the process of sending or hand delivering a photo. Creating a back-end service allows the company to tackle the problem of manually validating photos. By conjugating both this modules we create a system capable of fully automating this process in a safe way. Companies can now rest assured that every photo taken by a fellow associate respects their standards, and maintains their real identity.**

I. Introduction

The University of Aveiro[3] has an information system, which saves all the faces from students, professors, researchers and administration staff. This personal information is used by all services in the university, which helps in many tasks, namely by creating a profile page in a web portal, available to the public.

Until very recently, the Human Resources at University of Aveiro were responsible for the whole process of updating a photo. This includes receiving a photo from a user, validating the new photo, verifying the identity of both photos and then updating the photo in the information system. At the moment a system as deployed in a testing environment were only some members of the academic community are allowed to use. This system is already incorporated with idp.ua.pt[4]. The scope of our project resides in making a more refined and complete version of the already deployed service. This project will allow every member at the University of Aveiro to update his photo without direct interaction of the Human Resources department.

As such, we felt the need to create three layers: FotoFaces API, Mobile App and Database. The FotoFaces API will have the main algorithms to validate a photo, while the Mobile App will have a few more not to overwork the API, like Live Detection. The Database will contain all the information used by the users in the Mobile App.

II. Requirements

To attend to this problem, this system needs to work properly with many users at the same time, be intuitive and available at any time and to any user. It also shouldn't frequently crash and it should be maintained proper documentation and infrastructure.

In the functional requirements the mobile app must allow the user to authenticate with a username and a password or by Single Sign-On[2], check the user's current photo, check the properties needed for a photo to be valid, let the user choose between picking a photo from the gallery or take a new live one and allow the user to retreat at any step he is in. It also needs to send a photo and the user ID to the FotoFaces API, receive a JSON from the FotoFaces API with the validation properties, check the validity of a photo (based on the properties recieved) and show the user if the chosen photo is valid.

Lastly, for the FotoFaces API, it must be able to receive a photo and an user ID, get the user old photo from the database and verify if the person in the receiving photo and in the old photo are the same person. It also must detect a series of properties from the new photo, send the detected properties in a JSON format to the user and allow plugins to be added or removed for detection of more or less properties.

III. Architecture

As we see in the Fig. 1 the system is centered in the FotoFaces API. The API waits for a message from the Mobile App with a photo to be analysed, named candidate, and the ID of the user that requested it. Then it proceeds to get the old photo of that user in the database, named reference, to get everything it needs to analyse the candidate photo. After the analysis, the API sends the resulted properties to the Mobile App to do their evaluation. If the photo is accepted by the Mobile App, then it updates in the database.

The FotoFaces API will have a plugin architecture to adapt the needs of the entity that will use this system, like the University of Aveiro, schools or others. This way, these entities can remove or add another algorithm, which will be transformed into an independent plugin, without changing most of the API.

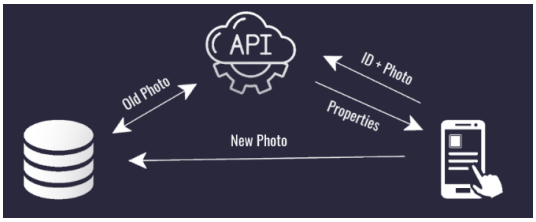

Fig. 1. Architecture Of The App

IV. Database

Our Database was created using a python package, sqlite3[5]. The table can be seen in Fig. 2, as it is a simple SQL table with five atributes and a primary key, id.

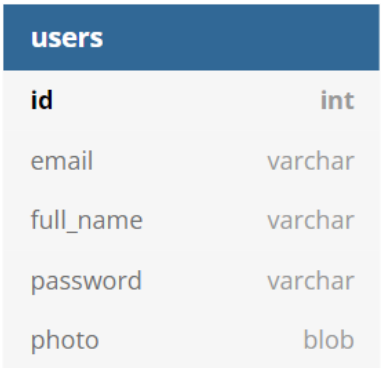

Fig. 2. Database Table

V. Mobile Application

The Mobile App was made with React-Native[14], firstly with a template for the design[1] and later a hand-made style, shown in the Fig. 3.

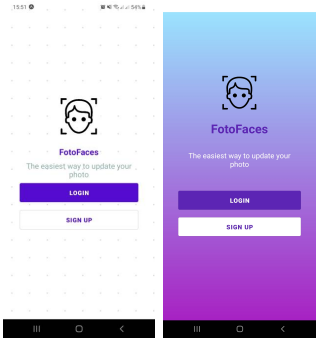

Fig. 3. Start Screen (design template at the left and hand-made style at the right)

It has 6 main screens, shown in the next figures, each representing a step to complete the update the photo. It also has to validate it, by its properties given by the FotoFaces API (represented in Fig. 5, where the crosses are the properties in which it failed to pass), with our sense of rigor and usability.

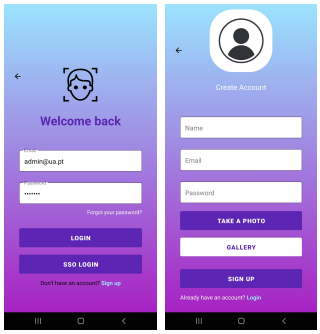

Fig. 4. Login Screen (at the left) and Register Screen (at the right)

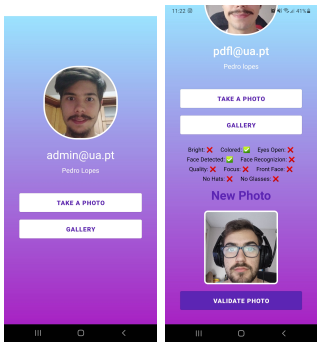

Fig. 5. Main Screen before and after adding a photo for update, with the properties of the photo

A. API Connection

The Mobile App will connect to the database when: a user signs in (by getting the user data, if exists), signs up (by sending the data, if does not exists) or accepts the valid photo for update (by sending the photo and updating in the respective user). For the FotoFaces API, it will connect when a user validates a photo, during registration or in the main page.

B. Taking a Photo with Live Detection

The Live Detection algorithm is used when a user chooses to take a new photo in the main screen or the register screen. The camera option will take the user to an interface, based on a project done by Osama Qarem[17],where he has to put his face inside a frame and do some verification steps, like winking an eye, as shown in the next figure.

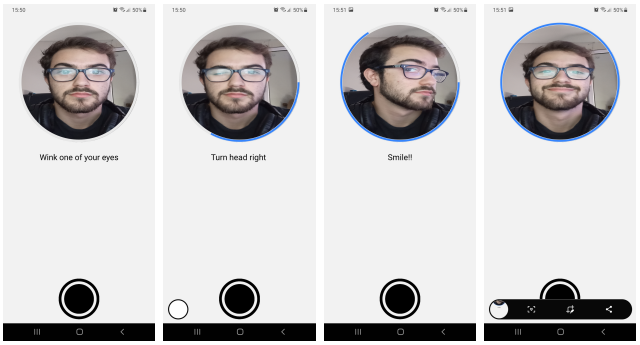

Fig. 6. Take a photo process with all steps

We use an expo package called FaceDetector[7] which uses functions of Google Mobile Vision framework[19] to detect the faces on images and gives an array that contains information about the face, e.g. the coordinates of the center of the nose, the winking probability, etc.. The FaceDetector package is usually used along with the Camera package also from expo[6], where we can define the properties of the FaceDetector detection, as for example the minimum detection interval, which defines in what space of time it should return a new array of the properties of the face. By analysing that array, we can confirm if the user is smiling or not, by checking the value of the key "smilingProbability", if the number is bigger, it means that the user is most likely smiling.

VI. FotoFaces API

The FotoFaces is an API for photo analysis and processing. It was built in Flask[16] and receives a photo, candidate, to be analysed and the id of the user that wants to update the picture. For the main process, the candidate photo is analyzed and verified if the photo is not grey scale and has a face. Next, it crops the image to a size where only the face is shown, it calls the Database API to get the old user photo, reference, and sends this information into every plugin. The results are sent to the Mobile App as a set of properties to be verified there.

A. Plugin Architecture

Plugin Architecture[8] enables our application to become more flexible and versatile enough to adapt to different needs.

Since our application relies on plugins, we can assure that we are only evaluating essential parameters. By eliminating plugins that do not possess relevance to the acceptance criteria of the candidate image, we are creating a much lighter environment. On the other hand, adding more parameters to the acceptance criteria will result in a heavier environment.

Each plugin must comply to a certain structure, and the execution is ensured by our core application, although the developer is in charge of the execution flow in each plugin. The core application also ensures that some predefined criteria are met before instantiating and running this algorithms, for example, a photo with multiple faces or without a face will not invoke the rest of the plugins.

B. Algorithms

The algorithms implemented in this API were based in the algorithms of the similar existing system in the Aveiro's University for teachers and from a master degrees thesis 19/20, as well as new algorithms implemented by the elements of the group.

1) Face Detection: The Face Detection plugin converts the image into grey scale, and using the face detector algorithm from dlib[12], it gets all possible matches in a dlib rectangle.

Then, it verifies if each rectangle is a good match, when is either not None or all the area is in the photo, and converts all of them into an opencv[18] array ([x y width height]).To choose the best rectangle, we calculate their areas and the one with the biggest one is selected. After, it needs to convert the raw shape of the face, created by using the dlib face predictor with the 68 facial landmarks to detect each part of the face, into a numpy[15] format (shape).

In the end, it returns the shape, rectangle selected in opencv format and the raw shape.

In the Fig. 7 is shown the results of this algorithm.

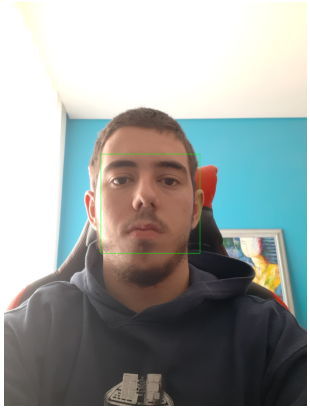

Fig. 7. Example of the result face detection algorithm

2) Brightness: The Brightness plugin uses the Cropping function to cut the person's face in the photo with a more tight result, so the background captured is the smallest possible. Then it converts the image colour from BGR (Blue, Green, Red) to HSV (Hue, Saturation, Value) and separates it in its components. Finally, to calculate the brightness of the person's face, it calculates the mean value of the third component, the Value.

In the Fig. 8 is shown an example of a photo with acceptable brightness and another one without. The average brightness of the photo can be above 0, being the higher the value the brightest is the photo.

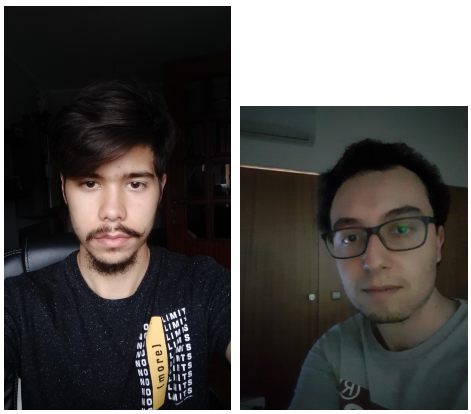

Fig. 8. Brightness plugin accepting and failing photo

3) Cropping: The Cropping function is used to cut a photo in a format defined as an argument (alpha). It calculates the best location in the image due to the argument shape.

In the Fig. 9 is shown an example of the results of cropping function with the default alpha parameter 0.95.

Fig. 9. Example of the result of the cropping algorithm

4) Open Eyes: The Open Eyes Plugin uses the argument shape to calculate the distance between the eyelid using the landmarks 37 to 42 (left eye) and 43 48 (right eye), returning the average of this two distances.

In the Fig. 10 the first photo is accepted by this algorithm, however the second one is refused. This algorithm can return values between 0.10 and 0.50, being the higher the value the more open the eyes of the person are.

Fig. 10. Eyes open plugin accepting and failing photo

5) Face Recognition: The Face Recognition plugin uses the face detection function on the reference image to get the referenced raw shape. It uses the dlib face recognition model v1 to convert each face, reference and candidate, into 128D vectors, with the dlib get face chip function, which is a machine learning algorithm that maps human faces into vectors, where pictures of the same person are near each other and different people are far apart.

In the Fig. 12 the Face Recognition plugin accepts the second photo based on the first one, while in the Fig. 11 it doesn't accept the photo. This algorithm can return values between 0 and 1, being the lower the value the more similar are the person in the photos.

6) Glasses: The Glasses plugin gets the values of the nose, which are the face landmarks 29 to 36, cutting a square of the image with the nose within. Applies a blur using the gaussian blur function from opencv for accurate detection, and paints the bridges of the glasses as strips of white pixels, using the canny function from opencv. Then, takes the transpose of the

Fig. 11. Face Recognition plugin accepting photo

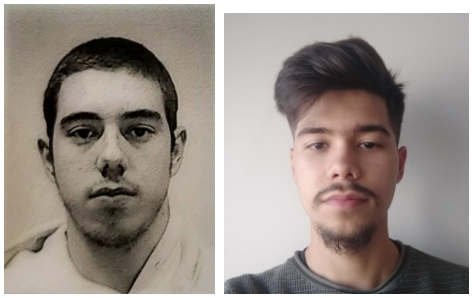

Fig. 12. Face Recognition plugin refusing photo

image to get column vectors and takes a vector along the center of the nose, checks if there are any white pixels in the image, and if yes then returns "True", else returns "False".

In the Fig. 13 the algorithm detects the glasses in the first photo, while in the second one the algorithm doesn't detect. This algorithm in the case of detecting the glasses returns the string "true" and, in the case it doesn't, returns the string "false".

Fig. 13. Glasses plugin recognizing glasses and not

This algorithm is also explained in the website referenced in [20].

7) Hats: The Hats plugin is a machine learning prediction test, which cuts the image between the ears, nose and eyebrows to detect the possible object like a hat, bonnet... Then it resizes the image to the preferred size, converts it to BGR, normalizes the image channel values and loads a json[9] model to detect the hat. Finally, it compiles the model, using the

library keras[10], and returns a prediction with the image test data.

In the Fig. 14 the algorithm detects the hat in the first photo, while in the second one the algorithm doesn't detect. This algorithm in the case of detecting the hat returns the string "true" and, in the case it doesn't, returns the string "false".

Fig. 14. Hats plugin recognizing hat and not

8) Head Position: The Head Position plugin uses the opencv solvePnP function to estimate the orientation of the face, converting the rotation vector to rotation matrix using opencv function rodrigues.

It then stacks concatenate the rotation matrix with the translation vector (column wise) to get a projection matrix. The projection matrix is decomposed to euler angles using opencv, being separated to its components (pitch), converting them from radians to degrees and to their absolute value.

Finally it returns an array with the pitch roll yaw metrics of the face.

In the Fig. 15 the algorithm accept the photo while in the second one it refuses. The values returned by this algorithm can be above 0, being the lower the value the more the person is faced at the camera.

Fig. 15. Head Position plugin accepting and failing photo

9) Image Quality: The Image Quality plugin converts the image from BGR to grey scale and calculates the BRISQUE score by using the opencv function QualityBRIQUE_compute with the brisque model and brisque range yml files.

In the Fig. 16 the algorithm accept the photo while in the second one it refuses. The values returned by this algorithm can be above 0, being the lower the value the higher the quality of the photo.

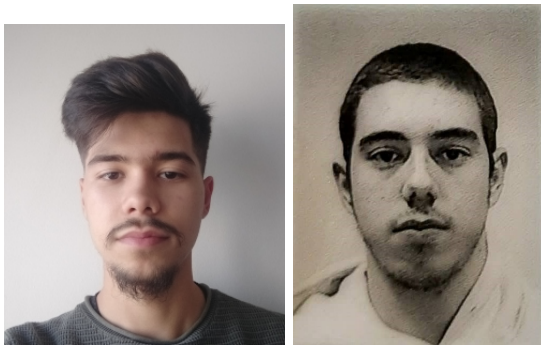

Fig. 16. Image Quality plugin accepting and failing photo

10) Sunglasses: The Sunglasses plugin converts the image to HSV and cuts the eyes of the image.

Then it compares the brightness of the eyes with the brightness of a skin reference: if the eyes aren't detected, it does a machine learning prediction test and gets the eyes of the image again, resizes the image to the preferred size and converts it to BGR.

Finally, it normalises the image channel values and loads a json model to detect the sunglasses, compiles the model with the library keras and does a prediction with the image test data.

In the Fig. 17 the algorithm detects the sunglasses in the first photo, while in the second one the algorithm doesn't detect. This algorithm in the case of detecting the sunglasses returns the string "true" and, in the case it doesn't, returns the string "false".

Fig. 17. Sunglasses plugin recognizing sunglasses and not

11) Focus / Gaze: The Focus plugin, also referred as Gaze plugin, gets the values of the landmarks of the left eye and filters the image by those references, converting the eye to grey scale and using opencv bilateral filter to blur the eye without damaging the edges (eyelids).

It also uses the opencv erode function to apply erosion (just like soil erosion) on the edges of the eye and the threshold

function to convert the eye to black and white image, by converting all pixels above the threshold to white and to black otherwise.

Then chooses the optimal threshold with the otsu algorithm, and uses the function findCounters to denote the eye, returning all necessary contours (points) of the eyes boundaries (removes redundant points) and the hierarchy of the nested contour. The contour with the greatest area is chosen and it proceeds to get the coordinates of the centroid of that contour using the function moments form the opencv library.

Afterwards, it calculates the distance ratio between the left point of the eye (37 landmark) and right side (38 landmark) with center in the centroid. The same is done for the right eye and it returns the average of both distance ratio or, in case of not having result of both eyes, returns just one ratio.

Fig. 18. Gaze plugin accepting and failing photo

In the Fig. 19 the algorithm accept the photo while in the second one it refuses. The values returned by this algorithm can range between 50 and 100, being the higher the value the better the person is looking at the camera.

VII. Results

A. Mobile App

When the application is opened, the user is prompted to login or register; if it's the first use of the app, the register page asks for a name, mail, password and a photo, which can be taken from the gallery or with the camera. After registering we can login into the app, using our email and password; the main screen will show the old photo and in what way the user wants to update his photo, from the gallery, or take a new one. Finally, the user can choose to update this new photo, or take another one.

When choosing to select a photo from the gallery, it will open the file manager of the device and let the user pick a photo and cut it in the way it wants. Contrary, the camera option will take to an interface where it will be tested the Liveness of the picture; after the steps are done, the user is prompted to take the photo.

When validating a photo, the properties of the image, which will be returned after requesting them to the FotoFaces, will be verified and if the photo is not valid, it will display all the criteria that it passed and which ones didn't, so the user knows what to do in the next take. We can see two examples in the next figure, Fig. 18.

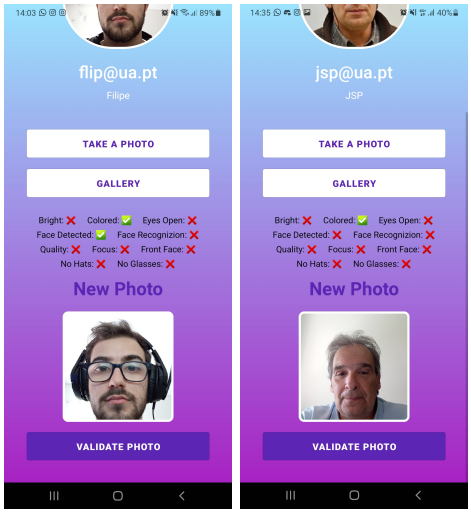

Fig. 19. Invalid Photos properties checklist

The accepted values for each property verified can be seen in the Table 1.

Brightness	Colored Picture	Eyes Open	Face Recog.
≥ 100	true	≥ 0.20	≤ 0.60

Face Detection	Quality	Focus	Head Pose
true	≤ 36	≥ 70	$[\leq 20, \leq 20, \leq 20]$

Sunglasses	Hats
false	false

TABLE I
PROPERTY VALUES ACCEPTED AS A VALID PHOTO

B. Usability Test

While creating the design of the application, and as developers of the Mobile App, we thought it was best to create an Usability Test, and know the feedback from people who would use the App, mainly students.

As for the flow of the tasks, the feedback was really good, with everyone realizing quickly how to use the application and update their photo, as can be seen in Fig. 20.

The tasks used can be seen in the Fig. 21. The users were not obliged to do all of the Photo Validation Tasks, but were greatly encouraged to do all the principal Tasks.

However, some people said that the task didn't work the way they where expecting to, as we can see in Fig. 22: the error messages were sometimes out of place, and there were situations where it would load the validation request infinitely, or the buttons for "Yes" and "No" in the last step of validation were, for some cellphones, out of place. Moreover, it was not possible to utilize the application using the "eduroam" Wi-fi

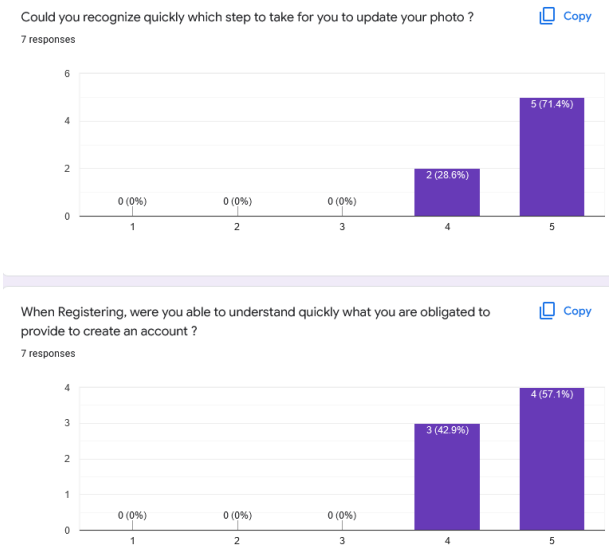

Fig. 20. Usability Test design questions

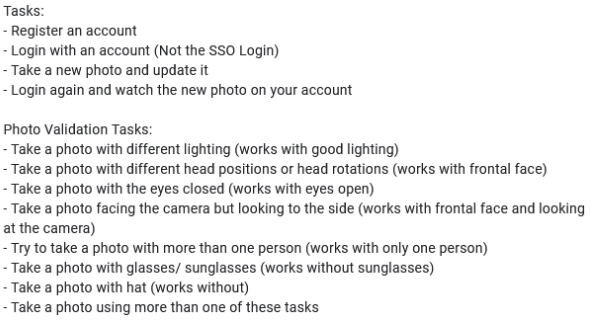

Fig. 21. Usability Test Tasks

connection, making it a bit confusing to some users why the application was not working.

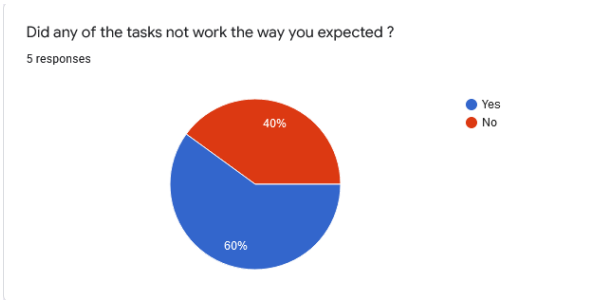

Fig. 22. Usability Test task question

We tried to use the feedback the best way we could, and we had some changes in design because of it: while before the properties check only gave the user a single error message with the first property it didn't validate, now it gives the list of all properties and which it passed and which it didn't, like said before. We also changed part of the taking a photo interface, so that the steps were more explicit and that in the end, the user would be indicated to take a photo by pressing the button; the button to take the photo would also only appear after the user completed all the steps.

VIII. Conclusion

A. Final Results

Resulting of this project we have a functional Mobile App, that contains Live Detection and the possibility to update a photo. We also have a FotoFaces API that, given a picture, returns us the properties the photo has. Finally, we have a database system to store all the information used.

In the next figures we can see the image and the properties of said image.

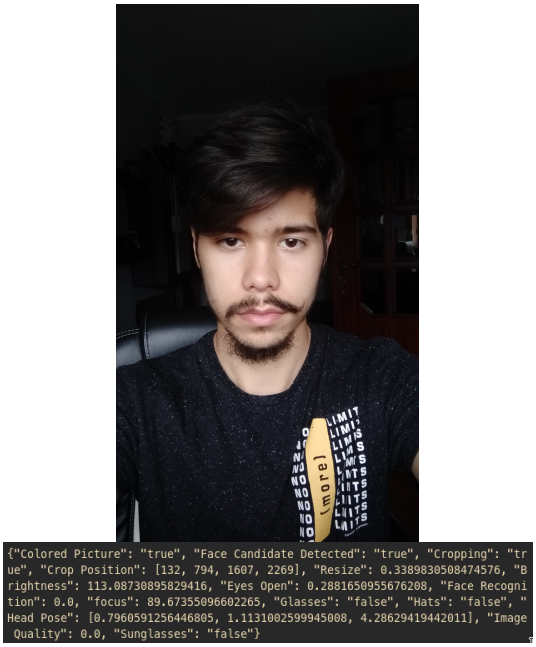

Fig. 23. Image and their properties

The FotoFaces API has much more potential, both in adding/removing plugins for better validation, and we are looking forward to see this product being used in bigger scenarios like in a school, where every person can update their photo in the school system, or even in the citizen card system.

B. Limitations

We had some problems exchanging requests between the APIs and Mobile App, which firstly was resolved with a Kafka implementation, but in the end, HTTP messaging using Flask was better for scalability, availability and usability, so we changed to this new method.

We had problems recognizing if a user was really taking a photo or if the photo was of a physical image. We solved this by adding Live Detection to the Mobile App.

We also had to initialize a docker whenever we had to communicate with the FotoFaces API, so we used Microsoft

Azure cloud deployment[13] to put the docker online and be able to do requests to the API.

C. Future Work

As for future work, we have some ideas of what to improve, implement or remove.

We would like to do some improvements in the Live Detection section of the app, as it still lets someone take a photo of a video of a user doing all the steps of it. As a solution, we could implement a machine learning algorithm to verify the change in the color pigments in the user face along the time; we could also verify with the depth and distance from the user face to the phone, by using infrared implemented in the phone software, but that would limit the scope of use of our application; we could also try to verify the border of the phone around the face, and if it exists it would not let the user proceed with the steps.

Another improvement would be in the Hat Detection plugin, as if a person wears a really small hat it will pass by as false by our algorithm. Also the Gaze Detection Plugin detects where the focus of the person is by the white area of the eye, so, while it is obvious when a person is looking to the right or to the left, since we can clearly see the white area of our eye, it's not the same when we look up or down, because there's a tendency of closing your eyes when looking up or down (specially down).

We also want to implement a better Image Quality algorithm, since we were able to find options with better results, but not within the reasonable time of implementation in this project, after a functional, yet worse, solution.

While HTTP messaging is very useful, it is also not secure, which means that we would also like to implement HTTPS in the FotoFaces API and the database calls using traefik[11].

While the design is very pleasing there are still some problems and as such we also wanted to improve and correct these issues, making it a very usable App. Moreover, both the Mobile App and FotoFaces API code can and should be refactored as there are simpler ways to do some tasks.

Lastly, we had some limitations while trying to implement the Single Sign-On of the university. While the process is relatively simple, it involves some bureaucracy and administration, and because we needed the SSO in the React-Native Mobile App, we needed help with the implementation.

References

[1] Tony Arrived. *React Native Simple Login Template*. URL: https://github.com/venits/react-native-simple-login-template-typescript.

[2] auth0. *Single Sign-On*. URL: https://auth0.com/docs/authenticate/single-sign-on.

[3] Universidade de Aveiro. *Universidade de Aveiro*. URL: https://www.ua.pt.

[4] University of Aveiro. *id.ua.pt*. URL: https://id.ua.pt.

[5] SQLite Consortium. *SQLite*. URL: https://www.sqlite.org/index.html.

[6] Expo Documentation. *Camera*. URL: https://docs.expo.dev/versions/latest/sdk/camera/.

[7] Expo Documentation. *FaceDetector*. URL: https://docs.expo.dev/versions/latest/sdk/facedetector/.

[8] Omar Elgabry. *Plug-in Architecture*. URL: https://medium.com/omarelgabrys-blog/plug-in-architecture-dec207291800.

[9] Json. *JSON*. URL: https://www.json.org/json-en.html.

[10] Keras. *Keras*. URL: https://keras.io/.

[11] Traefik Labs. *Traefik*. URL: https://traefik.io/.

[12] Siddharth Mandgi. *Glasses Detection - OpenCV, DLIB & Edge Detection*. URL: https://medium.com/mlearning-ai/glasses-detection-opencv-dlib-bf4cd50856da.

[13] Microsoft. *Cloud Services of Azure*. URL: https://azure.microsoft.com/pt-pt/services/cloud-services/.

[14] React Native. *React Native*. URL: https://reactnative.dev.

[15] NumPy. *NumPy*. URL: https://numpy.org/.

[16] Pallets. *Flask*. URL: https://flask.palletsprojects.com/en/2.1.x/.

[17] Osama Qarem. *Intro to Liveness Detection with React Native*. URL: https://osamaqarem.com/blog/intro-to-liveness-detection-with-react-native.

[18] OpenCV team. *OpenCV*. URL: https://opencv.org/.

[19] Google Mobile Vision. *Mobile Vision*. URL: https://developers.google.com/vision.

[20] Tianxing Wu. *Realtime glasses detection*. URL: https://github.com/TianxingWu/realtime-glasses-detection.