

The face recognition transaction system

**## Overview**

This is my attempt to make a Face Recognition System for facilitating secure transactions. The application is designed such that the user had to first register and before doing any transaction, he/she had to update his/her profile picture. The system will use that same face for the reference and while making the transaction, it detects the person in front of the camera and then matches the face (in front of camera) with that referenced (registered user's) image.

The transaction proceeds when the face is detected, recognised and matched with the registered user otherwise the transaction is terminated.

<!-- Images -->

![1](https://user-images.githubusercontent.com/89896247/170789925-ef0790e4-1957-4929-958c-f586a68a4c57.png)

![2](https://user-images.githubusercontent.com/89896247/170857746-29996904-c9ce-48dc-8508-71af81249171.png)

![3](https://user-images.githubusercontent.com/89896247/170787852-3d658936-3230-465a-bd2d-6b7769032d7a.jpg)

![4](https://user-images.githubusercontent.com/89896247/170789941-9c635da1-70f6-4285-801c-d0acbe3a06aa.jpg)

![5](https://user-images.githubusercontent.com/89896247/170789944-2179a986-6772-4b25-a257-ea4e9d277d14.png)

**## How to use**

\* **### The Installation**

To get started, you just need a browser.

1. Clone this repository and open it in your favourite IDE or in terminal (Command Prompt in Windows).<br/>

If Node.js is already installed on your system then you can skip the just below step.

2. Install [Node.js](https://nodejs.org/en/) and along with this, [npm](https://www.npmjs.com/) will automatically get installed.

3. Run `npm install` in the terminal. This will install all the dependencies related to this project.

4. Run `node server.js` in the terminal. It will start the server, locally at port 3000.

5. Type `http://localhost:3000/` in the browser's url.

6. The application is live now.

7. To stop the application, just stop the server by `Ctrl+C` in the terminal.

\* **### While using the application**

1. Before using you need to register in the application by clicking the `Sign Up` button. And, then login.

2. Now, upload your profile picture. You can navigate there by clicking the round-shaped image aside of Log Out button.

2. Select the transaction type and fill the transaction details. Click the `Next` button.

3. The application will start loading the camera and the ML models.

4. Click `Start Verification` to verify and authenticate yourself.

5. If you were verified successfully then you will be ask for initiating payment. Click `Pay` button to initiate. Now, you will be seeing the SUCCESS message on the screen.

6. If you were not verified, the application will reject the payment and you will be seeing the FAILED message on the screen.

**## Technology Stack and Documentation**

This project is possible with:

\* Frontend: [HTML](https://developer.mozilla.org/en-US/docs/Web/HTML), [CSS](https://developer.mozilla.org/en-US/docs/Web/CSS), [Bootstrap](https://getbootstrap.com/)

\* Backend: [Vanilla JavaScript](https://developer.mozilla.org/en-US/docs/Web/JavaScript), [Node.js](https://nodejs.org/en/) [(Express)](https://expressjs.com/)

\* Storing and Managing Data : [Google Firebase](https://firebase.google.com/)

\* Machine Learning: [face-api.js](https://justadudewhohacks.github.io/face-api.js/docs/index.html)

**## More on Machine Learning Models**

\* **### [face-api.js](**https://justadudewhohacks.github.io/face-api.js/docs/index.html**)**

JavaScript API for face detection and face recognition in the browser implemented on top of the tensorflow.js core API (tensorflow/tfjs-core)

\* **### Face Detection**

This application is using SSD  (Single Shot Multibox Detector) based on Mobilenet V1 neural network for face detection. Such a network is trained to generate a very accurate and almost unique 68 points face-landmark detection vector, given that, the images of faces which are fed to the network are properly aligned and cropped.

\* **### Face Recognition**

Then, for face recognition, a ResNet-34 like architecture is implemented to compute a face descriptor (a feature vector with 128 values) from any given face image, which is used to describe the characteristics of a person's face. It can determine the similarity of two arbitrary faces by comparing their face descriptors using Euclidean Distance or any other classifier.