Dhruvil Shah Follow

May 30, 2020 · 11 min read ★ · 38:32

DEEP LEARNING | REACTJS

Facial recognition login system using Deep learning + ReactJS

Building a facial recognition app that can be integrated into any system as a login phase!

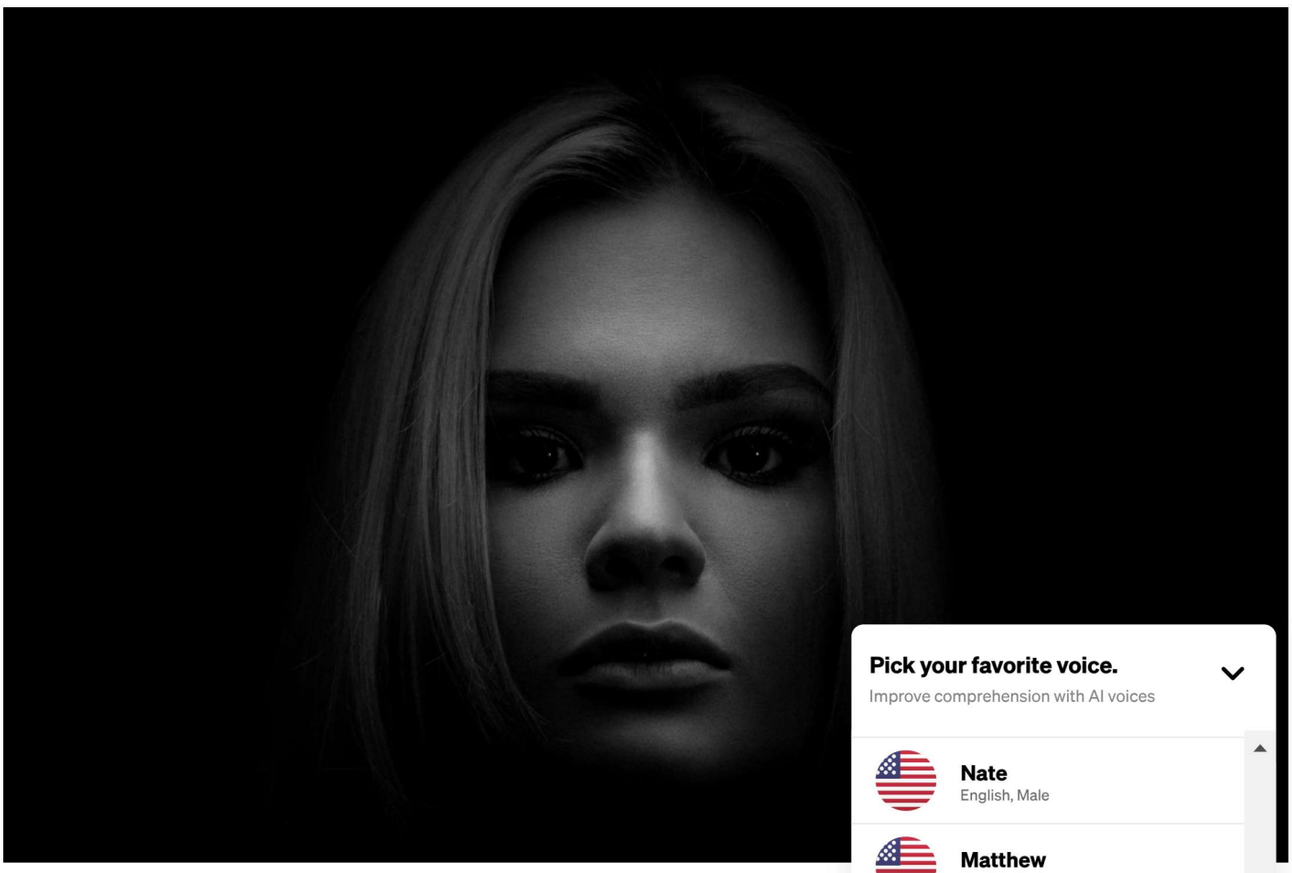
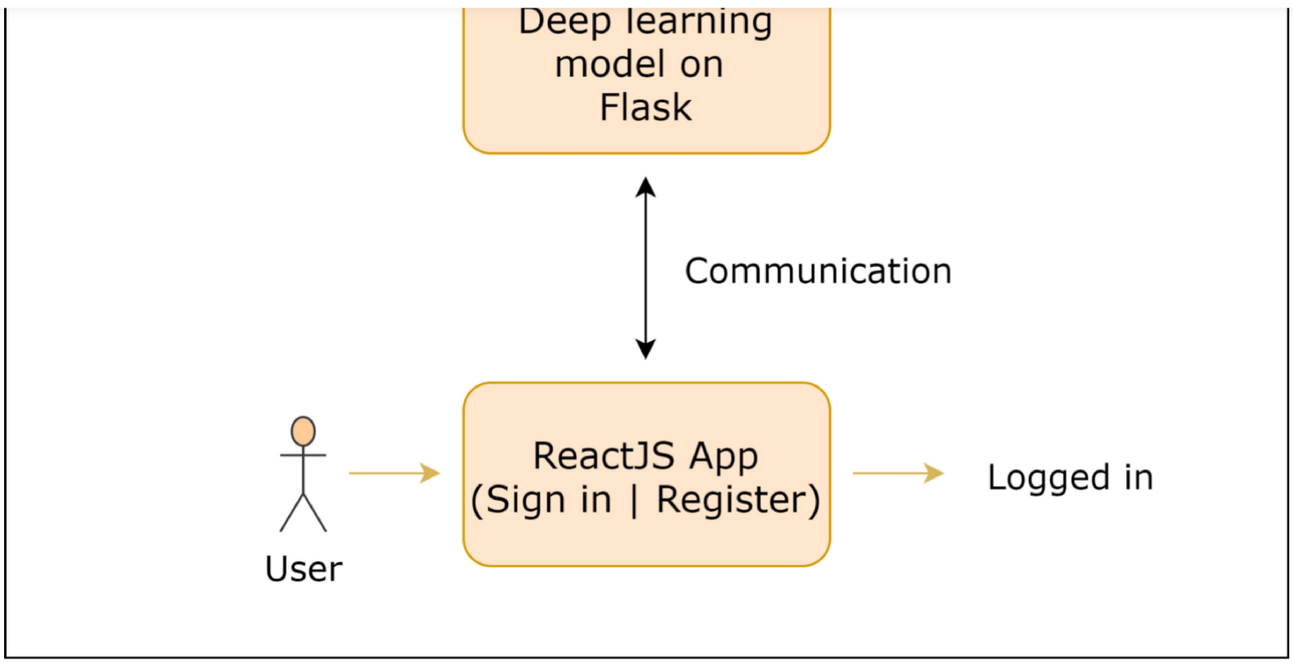


Photo by Andrey Zvyagintsev on Unsplash

You must have seen a facial recognition login system in apps or websites. website developer you may want to add facial recognition as a login phas to build a very easy-to-adapt login system from scratch that you can integ aspects.

We will build this system in 2 conspicuous steps. These steps include crea deep learning model and creating a ReactJS app for login. ReactJS applic deep learning model with the help of the Flask server. It can send the ima return the true identity of the person in the image. These all will be clear coding part and create our login system. Take a look at the below figure t system to be built.



The simple flow of the system

Creating a deep learning model

Facial Recognition and *Facial Verification* are two different things. Facial recognition is concerned with identifying who is the person in the image by detecting the face while facial verification determines if the given two images are of the same person or not. Here, to build our login system we will require a model that can detect faces from the images and convert the *facial details* into a 128-dimensional vector which we can use later for facial recognition or verification purposes.

We can use the FaceNet model provided by Hiroki Taniai so that we do not have to build one from scratch and need to worry about the hyperparameters. This FaceNet model has been trained on the MS-Celeb-1M dataset and expects a color image of size 160x160 pixels. The model can be downloaded from here: Keras FaceNet Pre-Trained Model.

First, let's import important modules for our Flask server that contains our deep learning model.

```
# Import all the necessary files!
import os
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras import Model
from tensorflow.python.keras.backend import set_session
from flask import Flask, request
from flask_cors import CORS
import cv2
import json
import numpy as np
import base64
from datetime import datetime
```


Some important steps for our API to work. You can learn more about why

```
graph = tf.get_default_graph()

app = Flask(__name__)
CORS(app)

sess = tf.Session()
set_session(sess)
```

Let's create a function that converts the given image into a 128-dimensional vector. This function takes the image path and our model as parameters and outputs the vector that contains information of the image.

```
def img_to_encoding(path, model):
  img1 = cv2.imread(path, 1)
  img = img1[...,::-1]
  dim = (160, 160)
  # resize image
  if(img.shape != (160, 160, 3)):
    img = cv2.resize(img, dim, interpolation = cv2.INTER_AREA)
  x_train = np.array([img])
  embedding = model.predict(x_train)
  return embedding
```

Let's create a sample database that stores the identity of the people. You can store some identities beforehand if you want. You can give an image of a person as an argument to the *img_to_encoding* method along with the model and you will get the 128-dimensional vector which you can store as a value to the database dictionary with the person's name being the key.

```
database = {}
database["Klaus"] = img_to_encoding("images/klaus.jpg", model)
database["Levi"] = img_to_encoding("images/captain_levi.jpg", model)
database["Eren"] = img_to_encoding("images/eren.jpg", model)
database["Armin"] = img_to_encoding("images/armin.jpg", model)
```

Let's create a function that will return the identity of the person in the image using the database. It will compute the distance as shown above for all the entries in the database and the new image and find the minimum distance. If the minimum distance is larger than the threshold it will show that the person is not registered otherwise it will return the identity with the minimum distance.

```
def who_is_it(image_path, database, model):
    encoding = img_to_encoding(image_path, model)
    min_dist = 1000
    #Looping over the names and encodings in the database.
    for (name, db_enc) in database.items():
        dist = np.linalg.norm(encoding-db_enc)
        if dist < min_dist:
            min_dist = dist
            identity = name
    if min_dist > 5:
        print("Not in the database.")
    else:
        print ("it's " + str(identity) + ", the distance i
    return min_dist, identity
```


Finally, we want our Flask server to do two tasks: 1) Register the user to t
registered 2) Identify the identity of the user from an image. We will crea
create a route that identifies the user from an image and returns the iden
data of the image from React app captured by the user at a time and it wi
registered or it will send a hint that the user is not registered so that our F
Note: Go here if you want to know the reason behind using the TensorFlow se

```
app.route('/verify', methods=['POST'])
def verify():
    img_data = request.get_json()['image64']
```

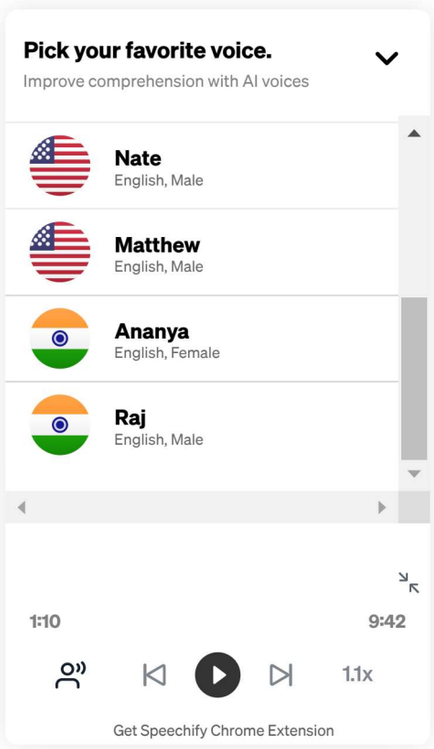

```
    global graph
    with graph.as_default():
        set_session(sess)
        min_dist, identity = who_is_it(path, database, model)
    os.remove(path)
    if min_dist > 5:
        return json.dumps({"identity": 0})
    return json.dumps({"identity": str(identity)})
```

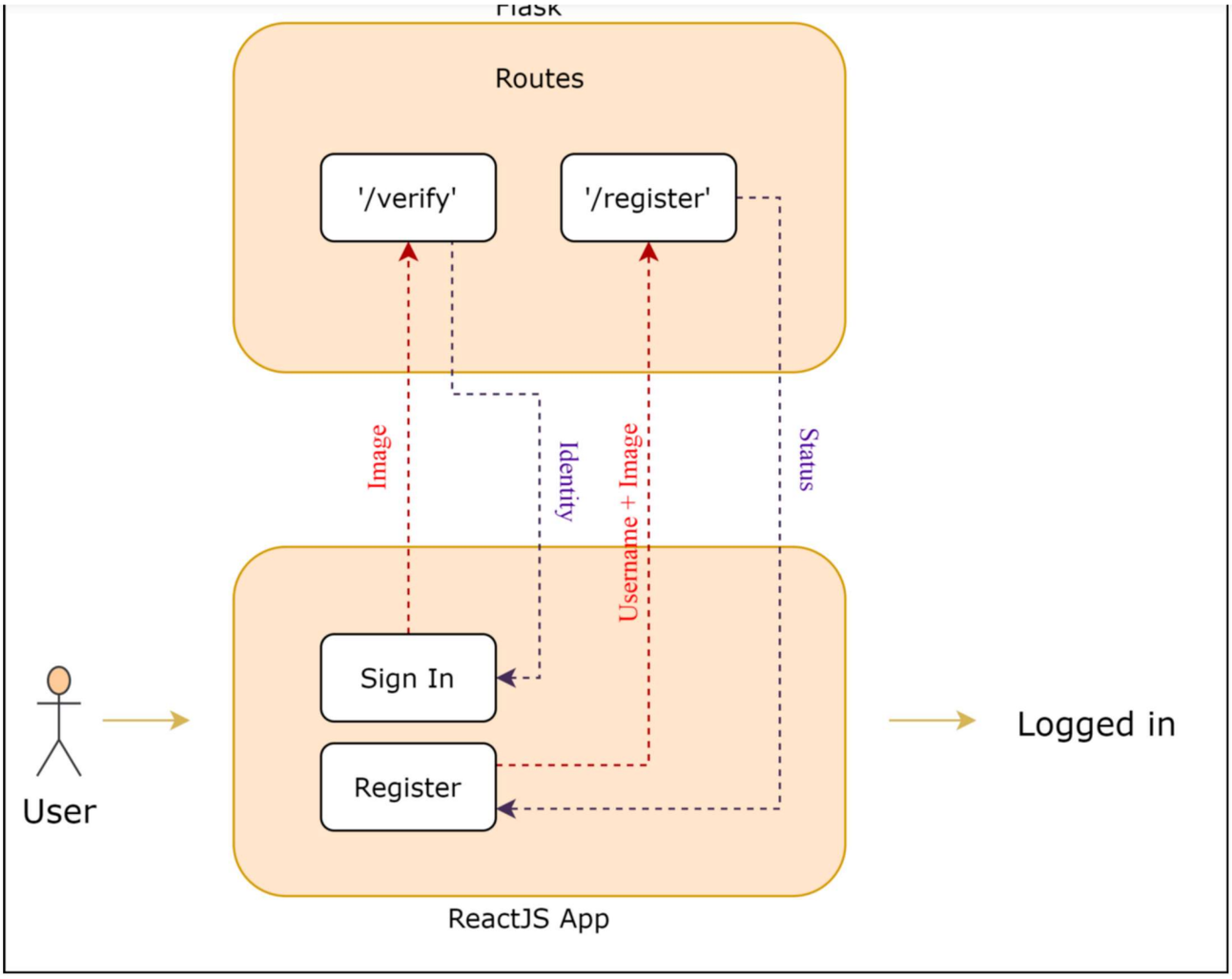
Above we created a temporary image file for our purposes and after checking for the identity of the user we removed it as it is not useful to us. We are naming this file with the help of the timestamp.

Time to create our second route that will register the new person into the database. It will receive the user's name and the base64 data of the user's image from the React app. It will compute the 128-dimensional vector for the image and store it into the database dictionary. If registered successfully it will send the status code of 200 otherwise it will send 500 status code (internal error).

```
@app.route('/register', methods=['POST'])
def register():
    try:
        username = request.get_json()['username']
        img_data = request.get_json()['image64']
        with open('images/'+username+'.jpg', "wb") as fh:
            fh.write(base64.b64decode(img_data[22:]))
        path = 'images/'+username+'.jpg'

global sess
        global graph
        with graph.as_default():
            set_session(sess)
            database[username] = img_to_encoding(path, model)
        return json.dumps({"status": 200})
    except:
        return json.dumps({"status": 500})
```

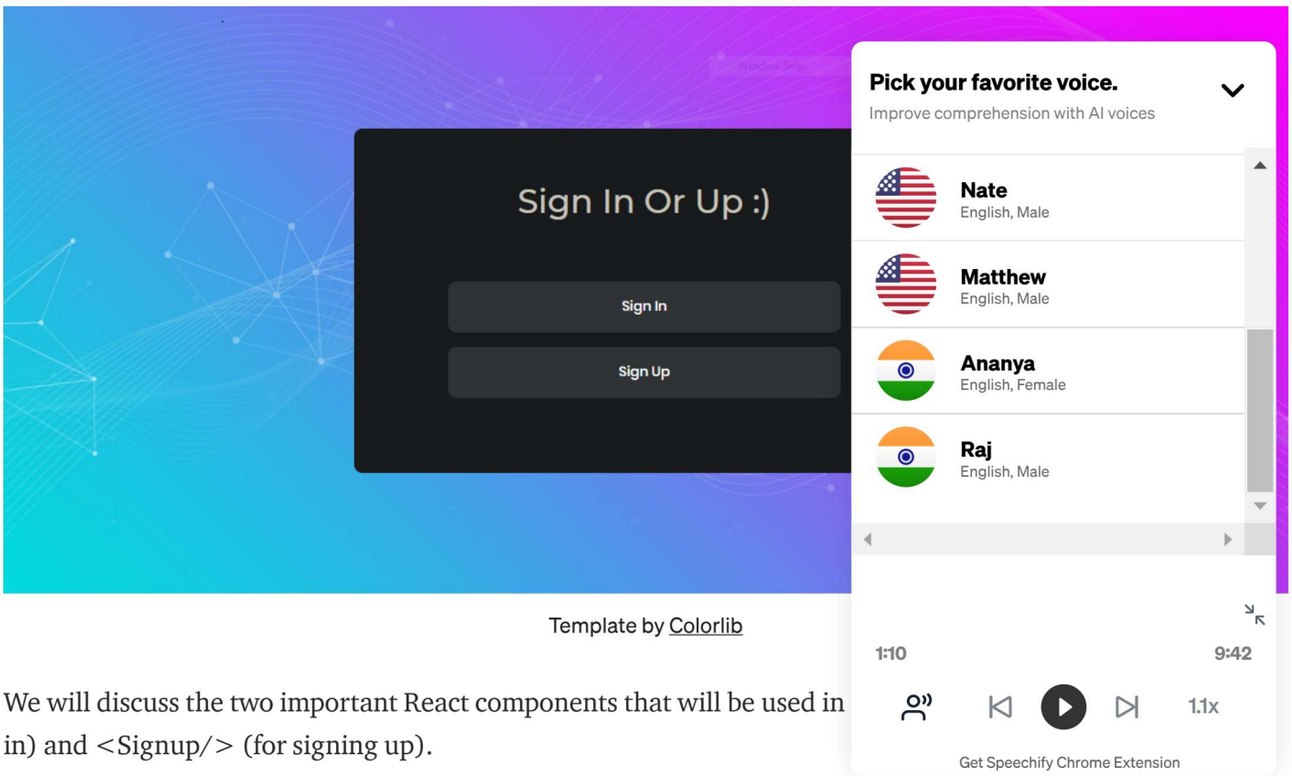
That is all for the deep learning part of the system. There are chances that you are still unclear about how these all will fit into the system we are building. Let's take a look at the flow of the system after creating our Flask server.





Creating ReactJS Application

The front-end coding of the application is out of the scope for this project. Details on the front-end will not be discussed here but you can download the whole code from the project link. The home page will look like this:



Template by Colorlib

We will discuss the two important React components that will be used in in) and <Signup/> (for signing up).

The app.js file (that contains <App/> component) will contain the following code

```
import './css/util.css';
import {Signup} from './signup.js';
import {Verify} from './verify.js';
export class App extends Component {
 constructor(props){
  super(props);
  this.state = {
   signup : false,
   verify : false
  };
  this.backhome = this.backhome.bind(this);
 }
 backhome(){
  this.setState({
   signup: false,
   verify: false
  })
 }
 signup(){
  this.setState({
   signup: true,
   verify: false
  })
 }
 verify(){
  this.setState({
   signup: false,
   verify: true
  })
 }
 render(){
  let home = (
    <div>
     <div className="limiter">
      <div className="container-login100">
       <div className="wrap-login100 p-l-110 p-r-110 p-t-62 p-b-33">
        <form className="login100-form validate-form flex-sb flex-w">
         <span className="login100-form-title p-b-53">
          Sign In Or Up :)
         </span>
         <div className="container-login100-form-btn m-t-17">
          <button onClick={this.verify.bind(this)} className="login100-form-btn">
           Sign In
          </button>
         </div>
         <div className="container-login100-form-btn m-t-17">
          <button onClick={this.signup.bind(this)} className="login100-form-btn">
           Sign Up
          </button>
         </div>
         <div className="w-full text-center p-t-55">
          <span className="txt2">
          </span>
          <a href="#" className="txt2 bo1">
          </a>
         </div>
        </form>
       </div>
      </div>
     </div>
     <div id="dropDownSelect1"></div></div>)

return (
      <div>
      {!this.state.signup && !this.state.verify ? home : '
      {this.state.signup ? <Signup backhome={this.backhome
      {this.state.verify? <Verify backhome={this.backhome}
      </div>
  )
 }
}
export default App;
```

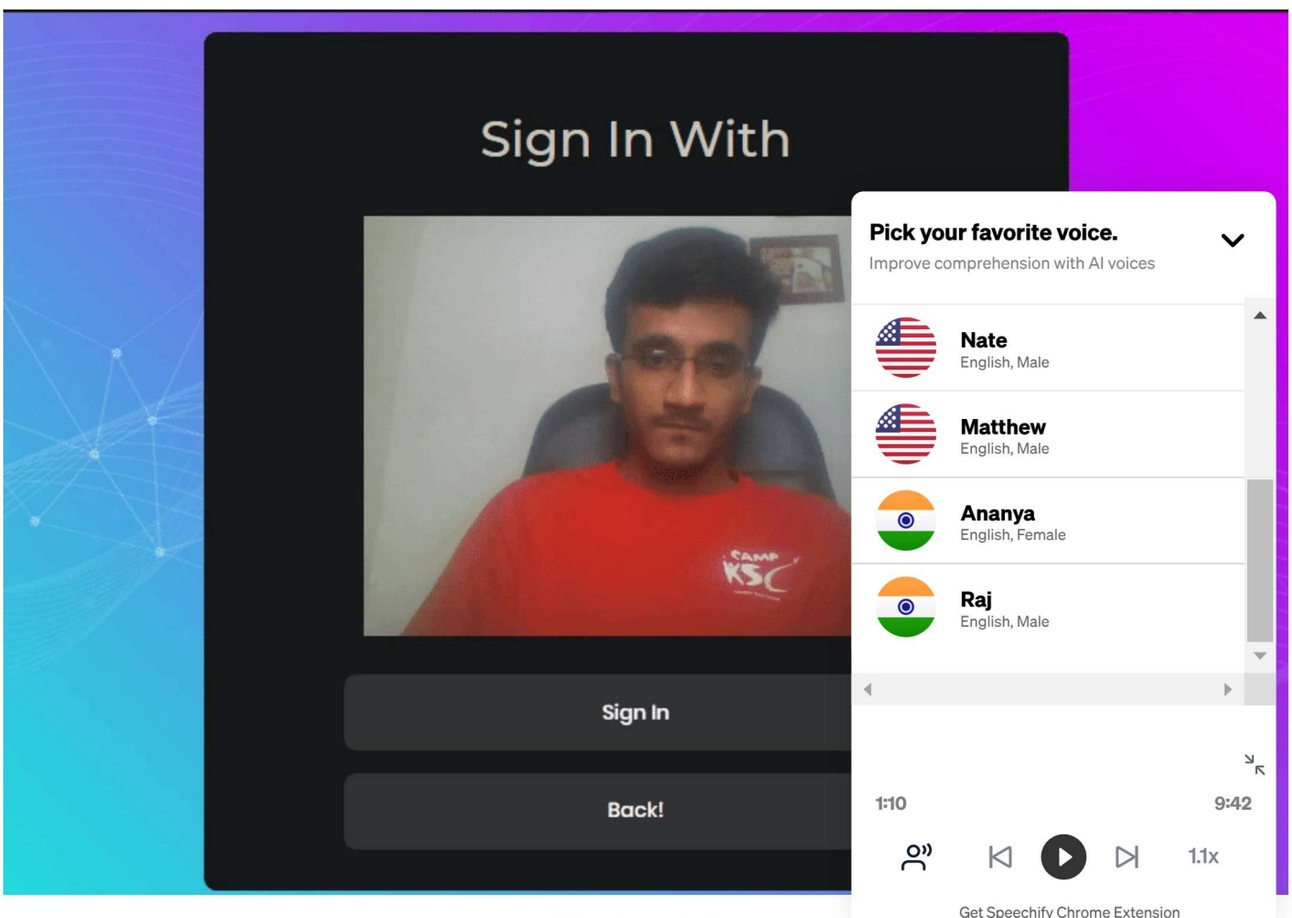
Let's dive into the first component that is <Verify/> (for signing in). Below is the code for the Verify.js file.

```
import React, {Component} from 'react';
import './css/main.css';
import './css/util.css';
import './verify.css';
import Sketch from "react-p5";
const axios = require('axios');
let video;
export class Verify extends Component {
constructor(props){
  super(props);
  this.state = {
   verify : false,
   idenity: ' '
  };
 }

setup(p5='', canvasParentRef='') {
        p5.noCanvas();
        video = p5.createCapture(p5.VIDEO);
    }

//to shut off the camera
stop(){
     const tracks = document.querySelector("video").srcObject.getTracks();
     tracks.forEach(function(track) {
        track.stop();
     });
    }

logout(){
     this.stop();
     this.props.backhome();
    }

setup2(){
     const button = document.getElementById('submit');
        button.addEventListener('click', async event => {
          video.loadPixels();
          console.log(video.canvas);
          const image64 = video.canvas.toDataURL();
          const data = { image64 };
          const options = {
            method: 'POST',
            headers: {
              'Content-Type': 'application/json'
            },
            body: JSON.stringify(data)
          };
          const response = await axios.post('http://localh
{'image64':image64});
          console.log(response.data.identity);
          if(response.data.identity){
           this.stop();
         this.setState({
          verify:true,
          idenity: response.data.identity
         })
          } else {
           this.stop();
           alert("Not a registered user!")
           this.props.backhome();
          }
        });
    }

render(){

let verify = (<div>
     <div className="limiter">
```

```
          <input/>
          <br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/>
          <Sketch setup={this.setup} draw={this.draw}/>
          <div className="container-login100-form-btn m-t-17">
          <button id="submit" onClick={this.setup2.bind(this)} className="login100-form-
btn">
            Sign In
          </button>
          </div>
          <div className="container-login100-form-btn m-t-17">
          <button onClick={this.logout.bind(this)} className="login100-form-btn">
            Back!
          </button>
         </div>
        </div>
       </div>
      </div>
    <div id="dropDownSelect1"></div></div>
   )

  return (<div >
        {this.state.verify? <div><h1>Welcome, {this.state.idenity}.</h1>
         <button onClick={this.props.backhome} className="container-login100-form-
btn">Logout!</button>
        </div> : verify }
       </div>)
  }
  }
  export default Verify;
```

For capturing the image of the user we will use the p5.js library. You can learn about how to use it here or you can directly copy the code and run it as there is no need to learn the entire library just for this particular application. After getting the response from our server we will shut the camera off and render the next component accordingly.

Below is the first output of the <Verify/> component.



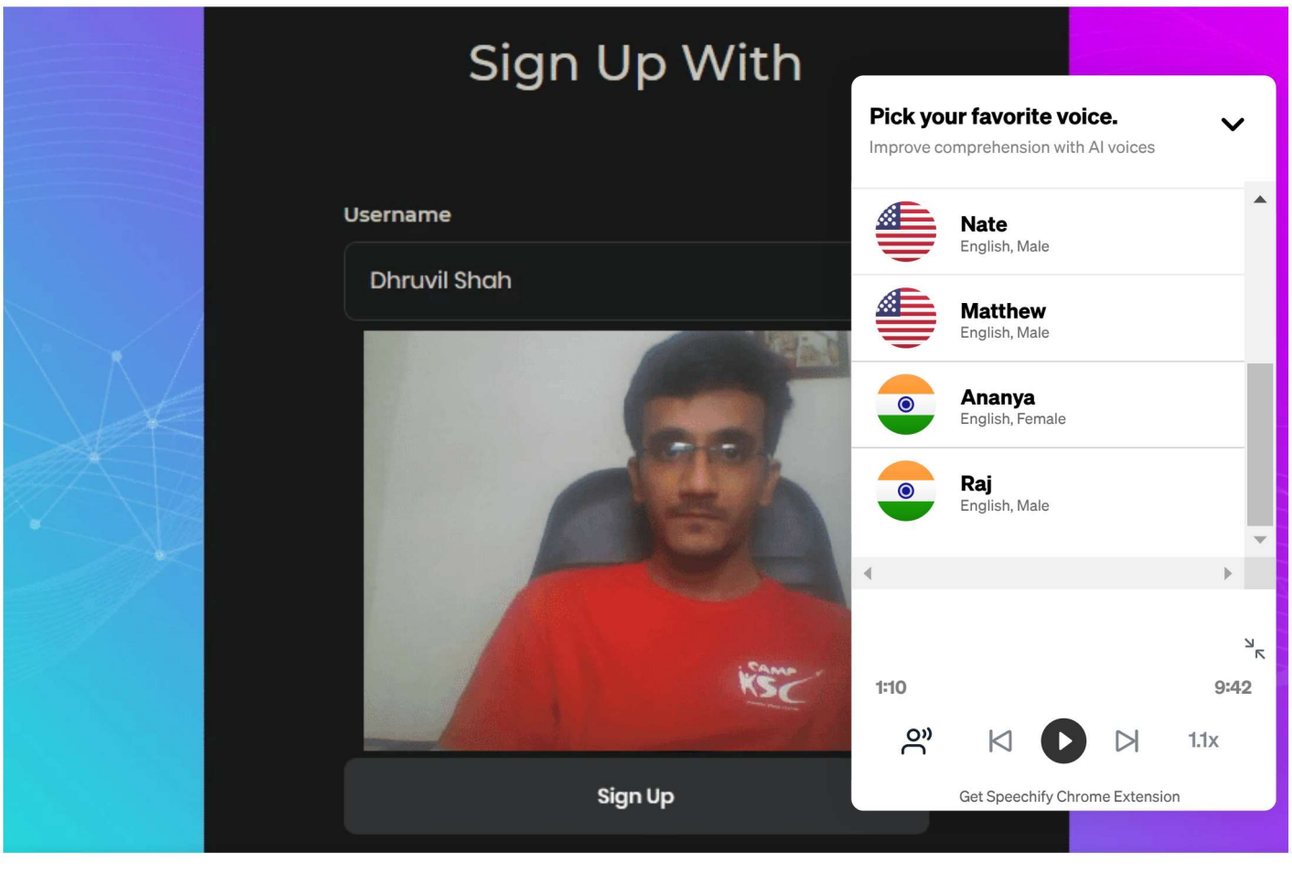
Without registration

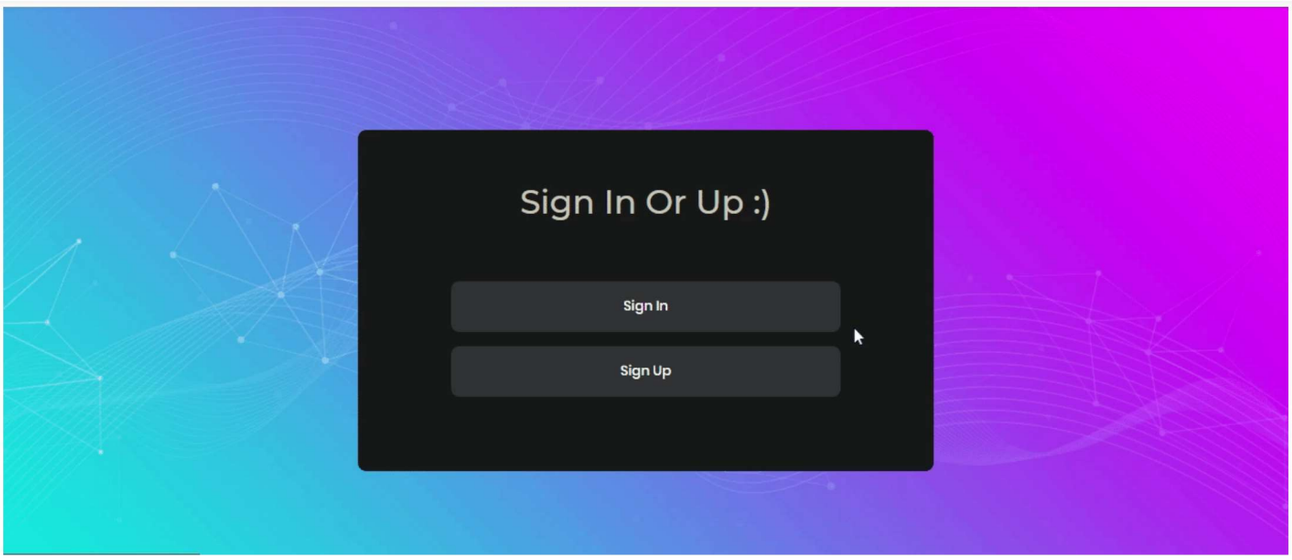
```
import React, {Component} from 'react';
import './css/main.css';
import './css/util.css';
import './signup.css';
import Sketch from "react-p5";
const axios = require('axios');
let video;
export class Signup extends Component {
 constructor(props){
  super(props);
  this.state = {
   signup : true
  };
 }
 setup(p5, canvasParentRef) {
        p5.noCanvas();
        video = p5.createCapture(p5.VIDEO);
        const v = document.querySelector("video");
        let st = "position: absolute; top: 255px;"
        v.setAttribute("style", st);
    }

setup2(){
     const button = document.getElementById('submit');
        button.addEventListener('click', async event => {
          const mood = document.getElementById('mood').value;
          video.loadPixels();
          console.log(video.canvas);
          const image64 = video.canvas.toDataURL();
          const data = { mood, image64 };
          const options = {
            method: 'POST',
            headers: {
              'Content-Type': 'application/json'
            },
            body: JSON.stringify(data)
          };
          console.log(image64);
          const response = await axios.post('http://localhost:5000/register',
{'image64':image64, 'username':mood});
          if(response.status==200){
      const tracks = document.querySelector("video").srcObject.getTracks();
      tracks.forEach(function(track) {
        track.stop();
      });
     };
           this.props.backhome();
          }
        );
    }
    logout(){
     const tracks = document.querySelector("video").srcObj
     tracks.forEach(function(track) {
        track.stop();
     });
     this.props.backhome();
    }

render(){

let signup = (
    <div>
     <div className="limiter">
      <div className="container-login100">
       <div className="wrap-login100 p-l-110 p-r-110 p-t-6
         <span className="login100-form-title p-b-53">
          Sign Up With
         </span>
         <div className="p-t-31 p-b-9">
          <span className="txt1">
           Username
          </span>
```

```
          <input/>
          <br/><br/><br/><br/><br/><br/><br/><br/><br/><br/>
          <br/><br/><br/><br/>

{this.state.signup?<Sketch id="s" setup={this.setup} draw={this.draw}/>:''}

          <div className="container-login100-form-btn m-t-17">
           <button id="submit" onClick={this.setup2.bind(this)} className="login100-
form-btn">
            Sign Up
           </button>
          </div>
          <div className="container-login100-form-btn m-t-17">
           <button onClick={this.logout.bind(this)} className="login100-form-btn">
            Back!
           </button>
          </div>
        </div>
       </div>
      </div>
      <div id="dropDownSelect1"></div>
     </div>
   )
      return (<div >
       { signup }
       </div>
   )
  }
}
export default Signup;
```

When we click on the ‘Sign up’ button on the home page the camera will start and the user will enter the name and click on the ‘Sign up’ button there. This component is responsible for taking our username and image and sending them to the ‘/register’ route of the Flask server we created earlier. The server will encode our image into the 128-dimensional vector and store it the *database* dictionary with our username as key and vector as value. As soon as our React app gets a status of 200 it will render the homepage again and this shows that the user has been successfully registered.

Below is how the output of the above code looks.





Successfully logging in after registration

After the login phase, there could be any page or website you want. For this article, I have kept it simple. After a successful login, the app will simply prompt a welcome message for the user.

You can get all of the code shown above from GitHub. You can also find me on LinkedIn.

Conclusion

We saw how we can integrate Deep learning and ReactJS to build a simple facial recognition login application. This app is supposed to be the login phase of any application or website you want. After the successful login, you can customize the next phase.

We used deep learning to do the face recognition and verification for our application but we had to go through loading the model and managing the images. We can also use pre-programmed libraries. There is a module called *face-api.js* in JavaScript's Node Package Manager (npm) which is implemented on the top of TensorFlow. You can check out this library here.

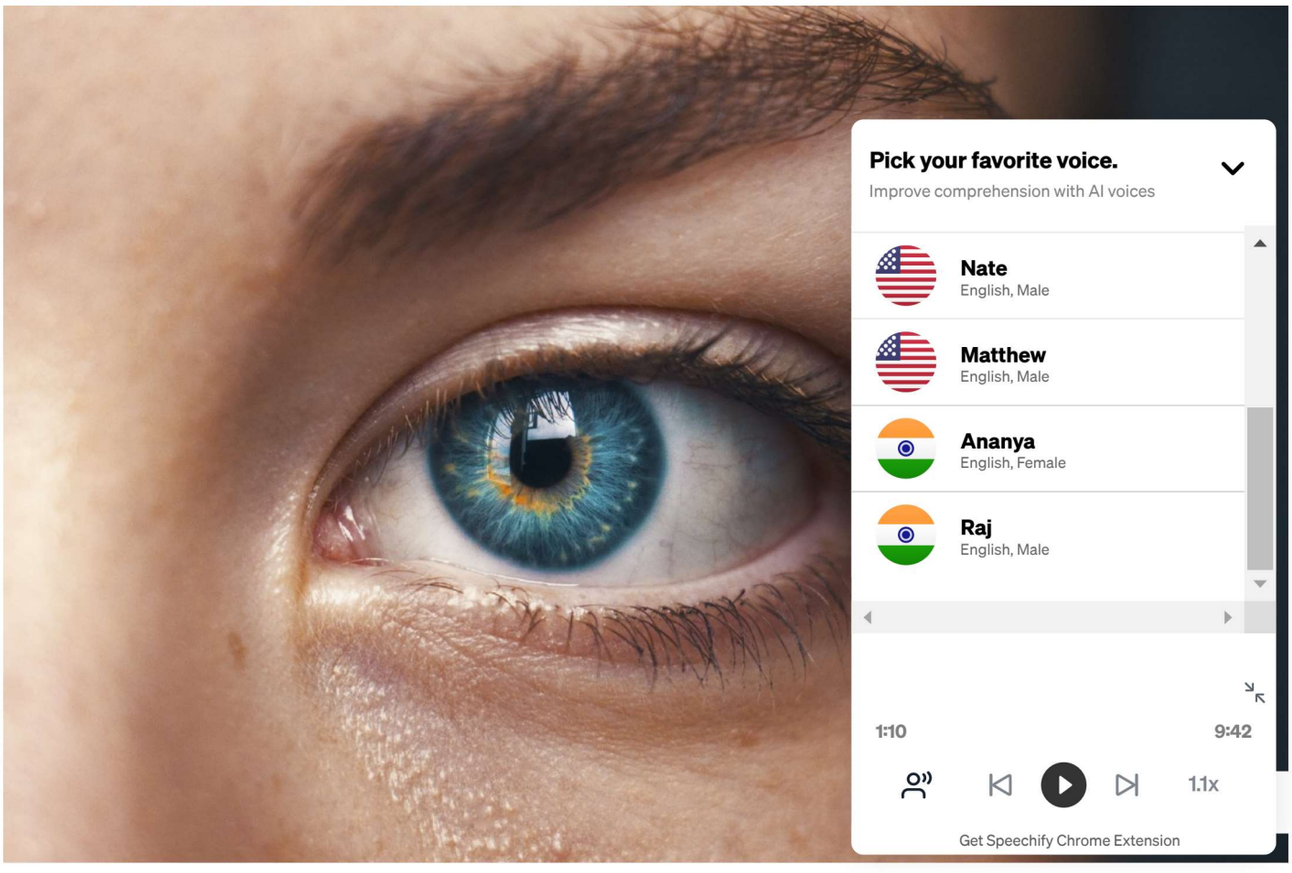


Photo by Amanda Dalbjörn on Unsplash

systems can detect sentiments of the person by looking at the face or detect the gender or even estimate the age of the person.