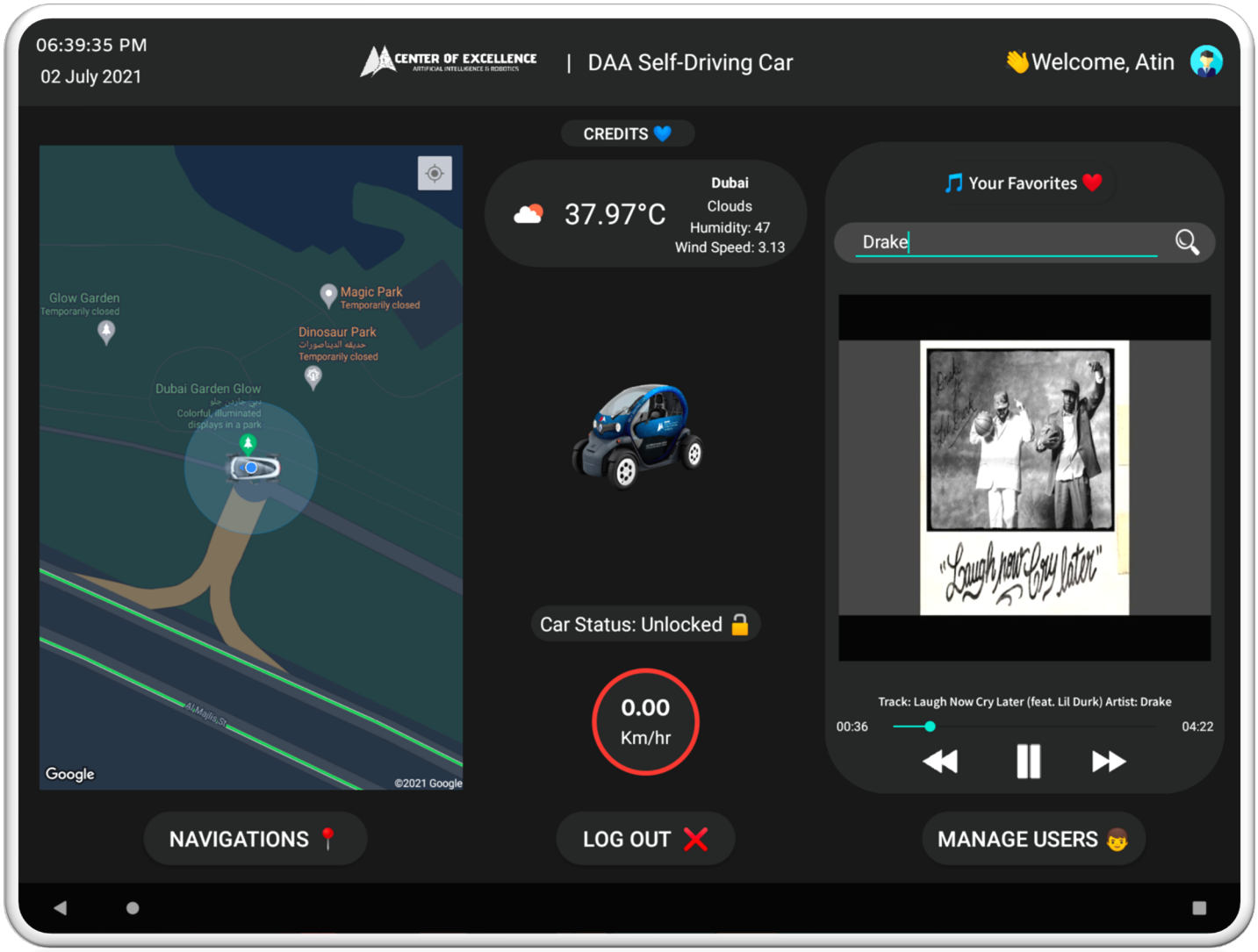
Introducing The DAA Dashboard

In Collaboration with
Center of Excellence Artificial Intelligence and Robotics

Table of Contents

CENTER OF EXCELLENCE
ARTIFICIAL INTELLIGENCE & ROBOTICS

DAA Dashboard



This entire project was built using Android Studio for the Frontend and Python for the Backend. And is open source, available at https://github.com/crypto-code/Car-Dashboard-Studio.

CENTER OF EXCELLENCE
ARTIFICIAL INTELLIGENCE & ROBOTICS

Acknowledgement

Mr. Sreejit Chakrabarty
GEMS Education

Mr.Ram Kumar
GEMS Education

CENTER OF EXCELLENCE
ARTIFICIAL INTELLIGENCE & ROBOTICS

Meet the Team

Rahul Arepaka
Project Lead and Designer
1st Year - MU

Sanjay Pramod
Programmer and Designer
1st Year - NTU

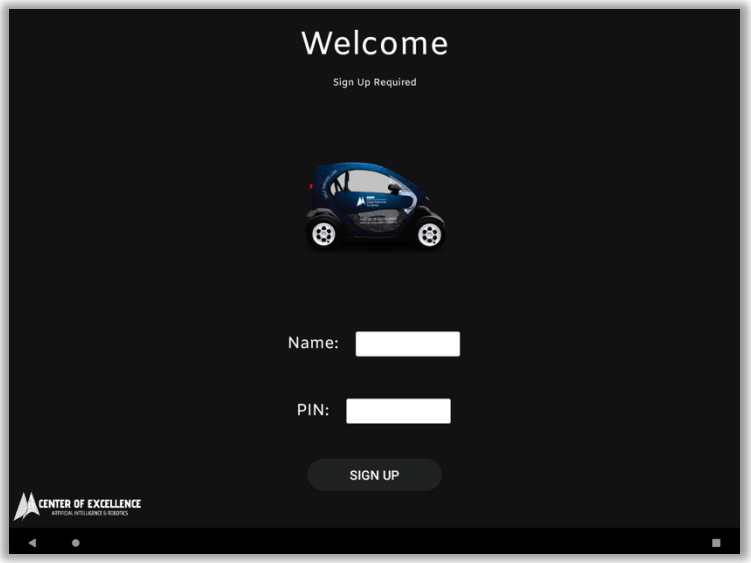
Ethan Hadimani
3D Mount Designer
1st Year - ASU

Atin Sakkeer
Lead Programmer
1st Year - NUS

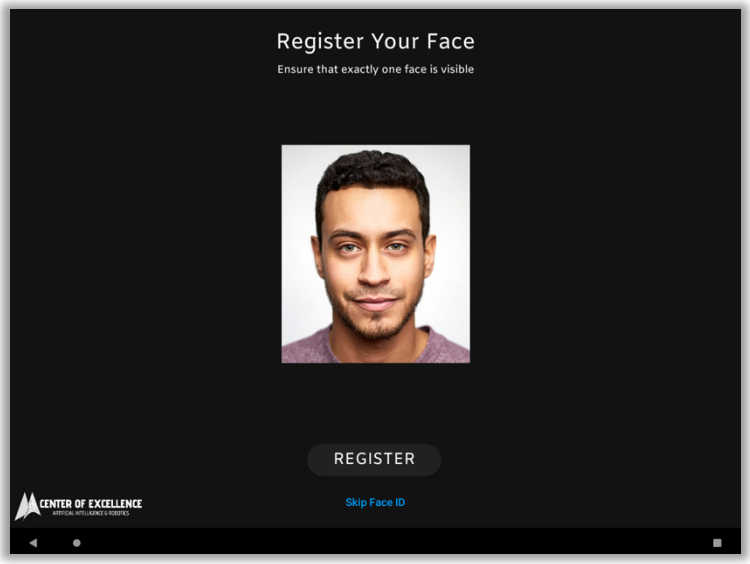
CENTER OF EXCELLENCE
ARTIFICIAL INTELLIGENCE & ROBOTICS

Dashboard Layout

Welcome Screen



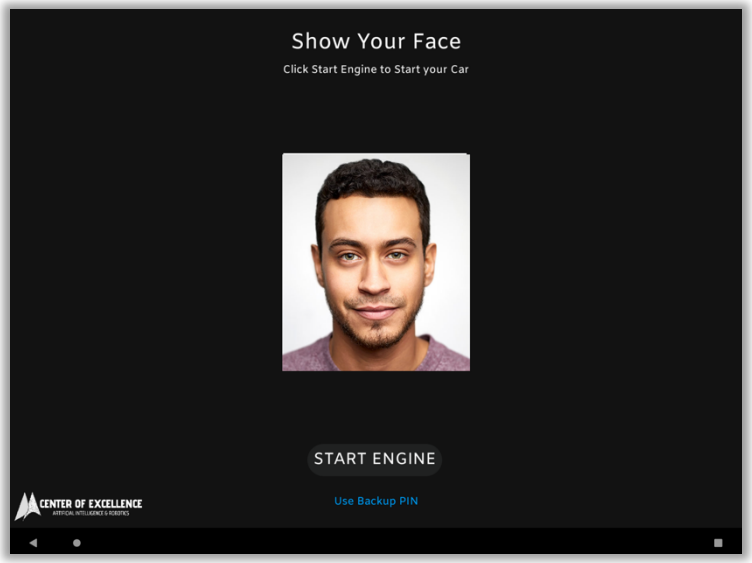
Sign Up Screen



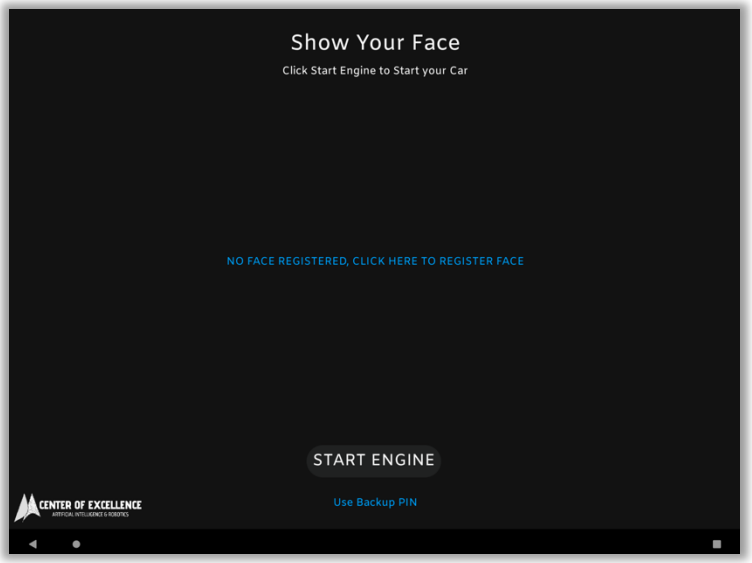
Register Face Screen

CENTER OF EXCELLENCE
ARTIFICIAL INTELLIGENCE & ROBOTICS

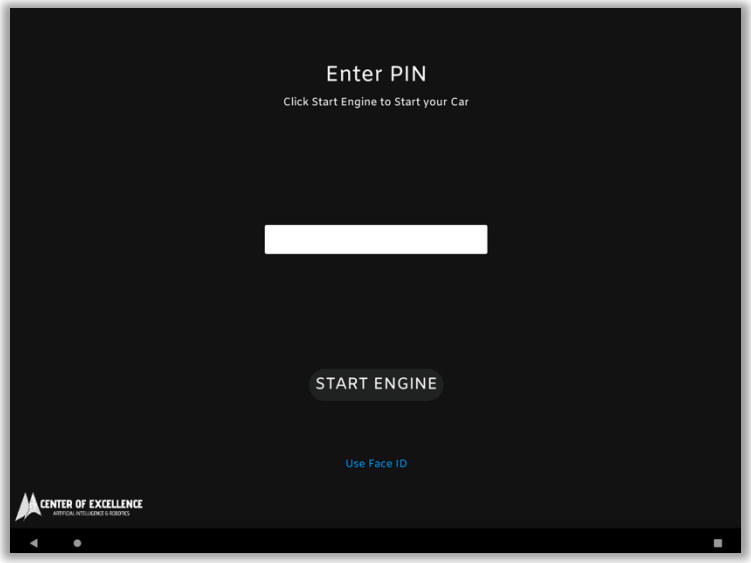
Dashboard Layout



Login Screen
(With Face Registered)



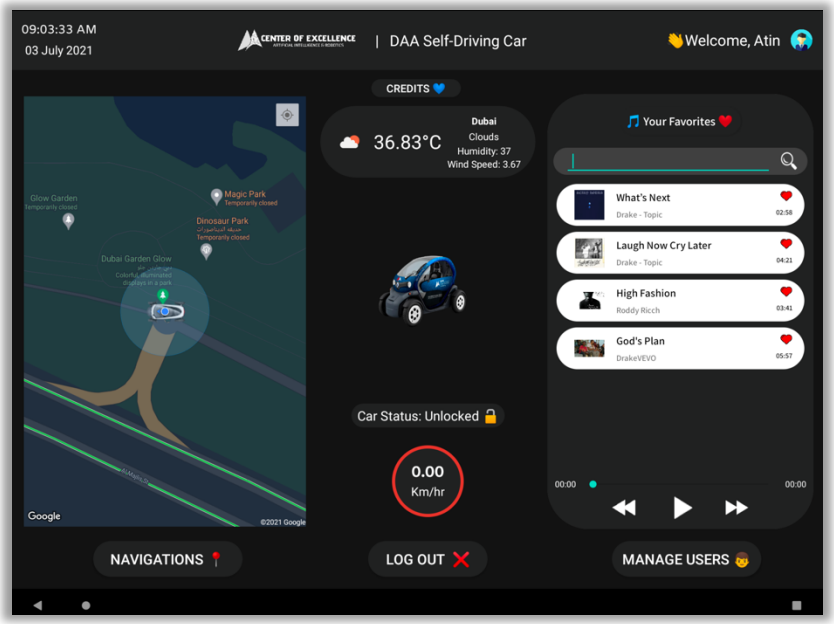
Login Screen
(Without Face Registered)



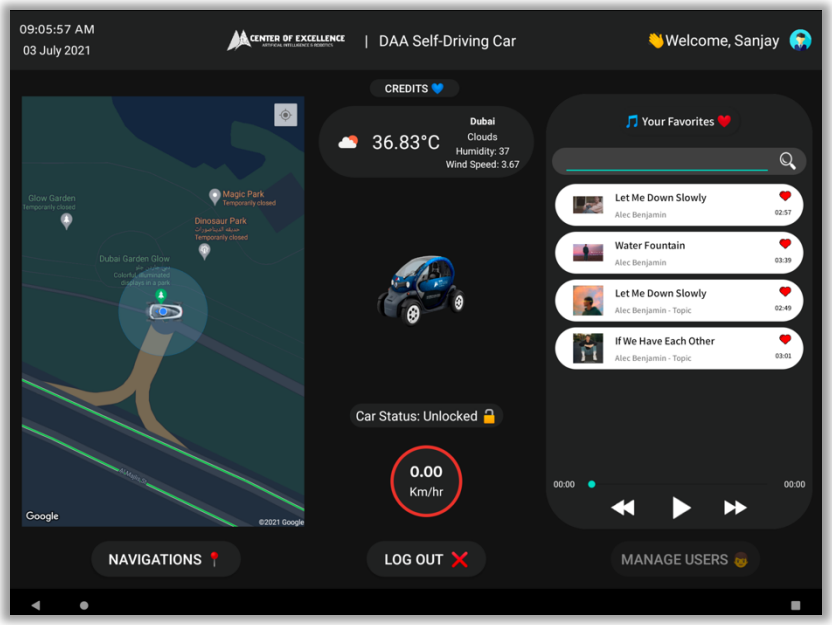
Login Screen
(Using Pin)

CENTER OF EXCELLENCE
ARTIFICIAL INTELLIGENCE & ROBOTICS

Dashboard Layout



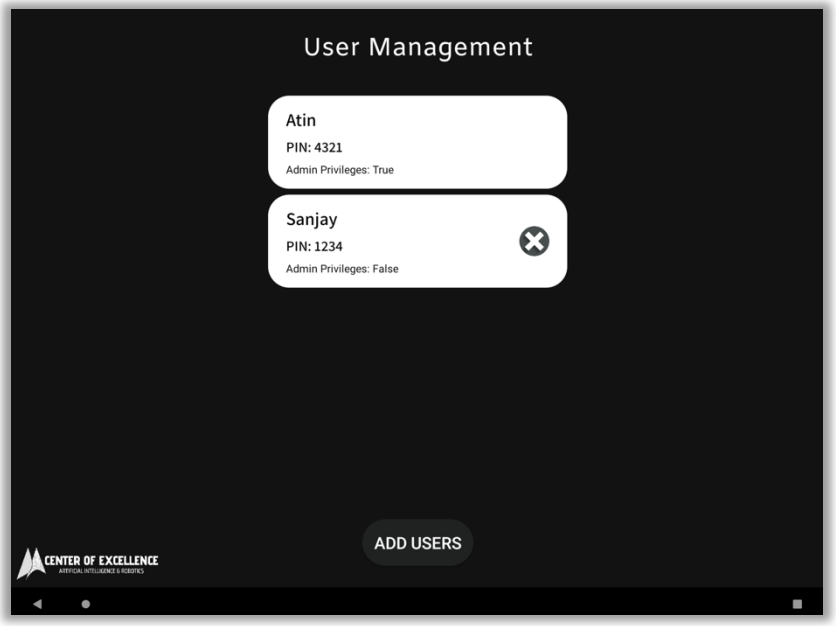
Main Screen (For Admin)



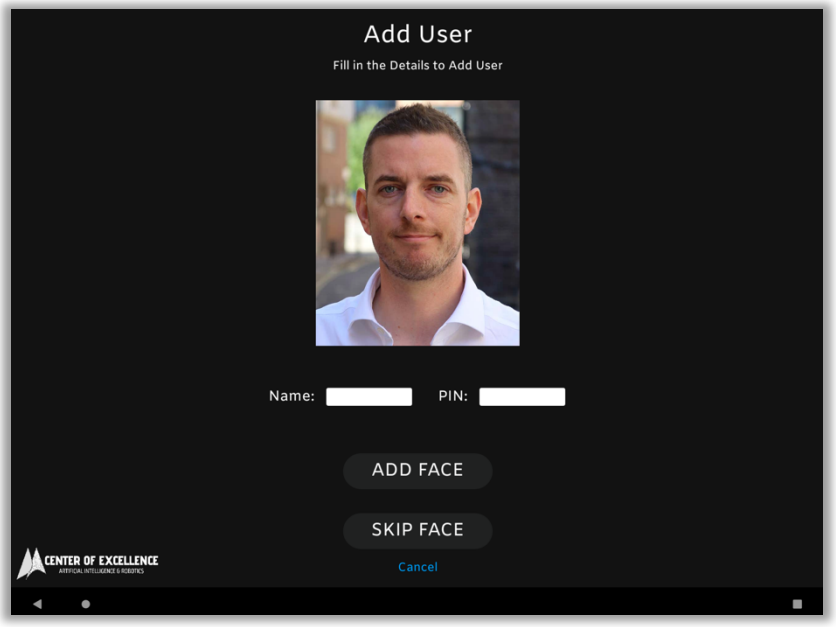
Main Screen (For Non-Admin)

CENTER OF EXCELLENCE
ARTIFICIAL INTELLIGENCE & ROBOTICS

Dashboard Layout

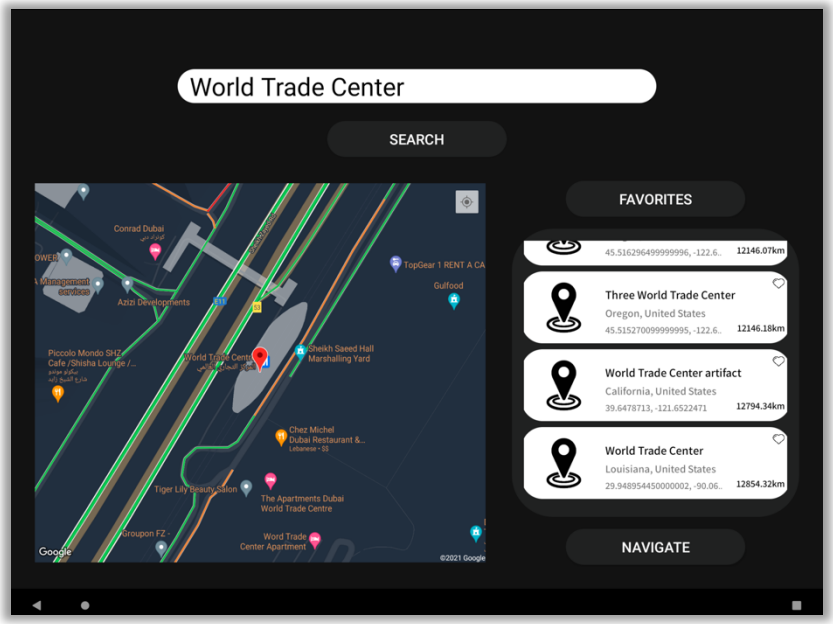


User Management Screen



Add User Screen

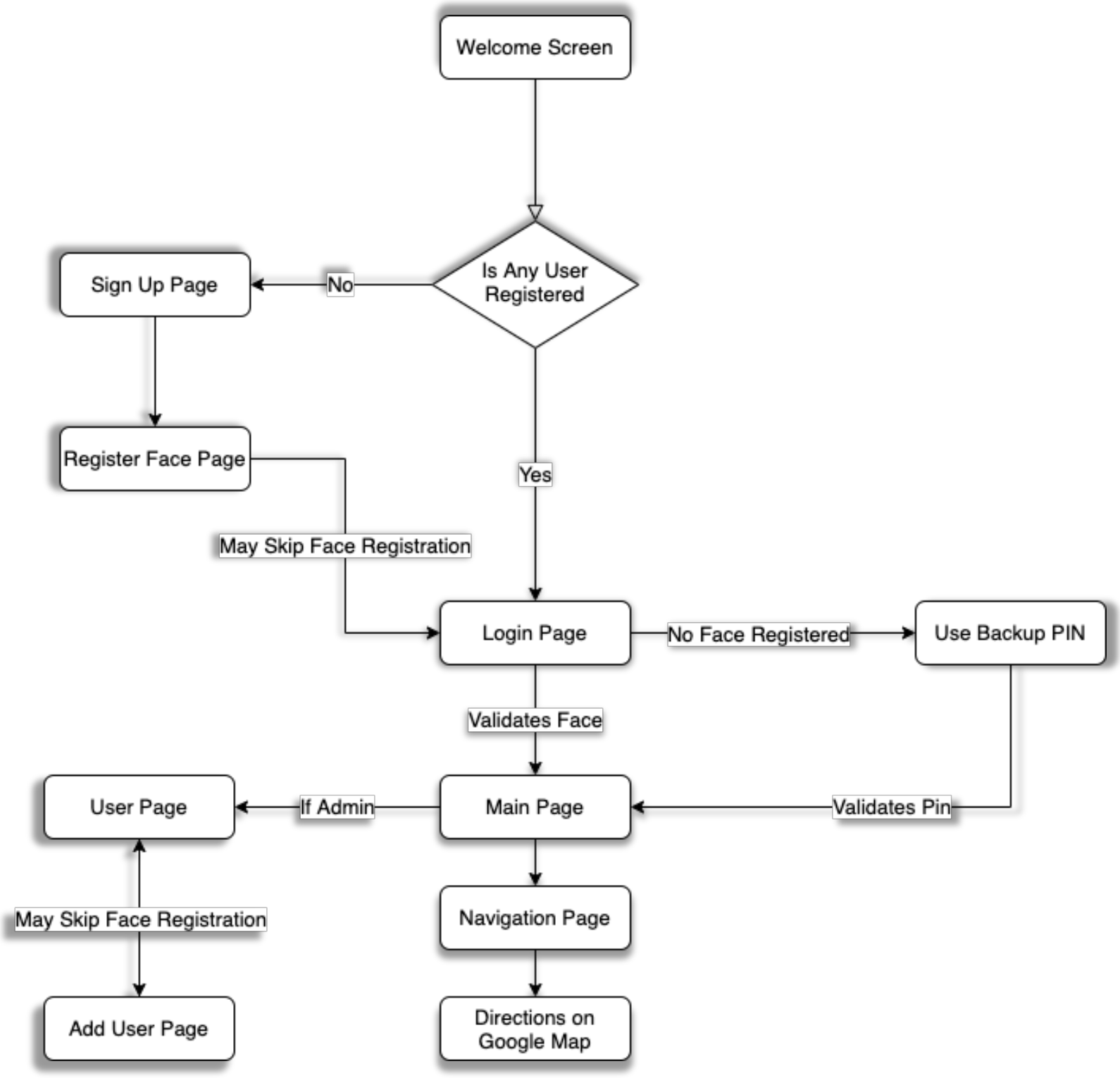
Dashboard Layout



Navigation Screen

Navigation Screen (Favorites)

User Flow Diagram



Facial Recognition

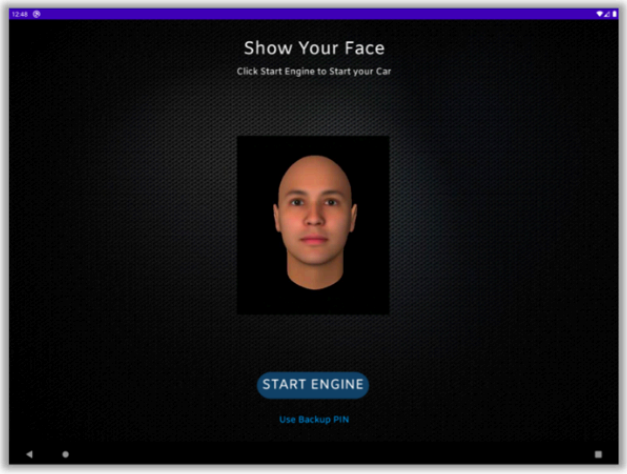
The Facial Recognition used in this app runs entirely using the ***face_recognition*** library in Python. The model used for facial recognition has a *99.38%* accuracy on the Labelled Faces in the Wild dataset. Python is linked to Android Studio using the open source ***Chaquopy plugin***. All facial recognition data is stored on a Firebase database.

During the facial recognition stage, the front camera of the Android device is used to capture the image. Then the image is passed as a raw byte stream to the Python backend to extract the facial features. This is then compared against the feature encoding stored in the Firebase database to validate the face.

```
def check_similarity(base_enc, enc2):
  vals = []
  for f in enc2:
    mat = face_recognition.compare_faces([f], base_enc, tolerance=0.6)
    vals.append(mat[0])
  return any(x for x in vals)

def check_match_with_image(encoding, img):
  image = Image.open(io.BytesIO(img))
  image.save(os.environ["HOME"] + "/temp.jpg")
  fr_image = face_recognition.load_image_file(os.environ["HOME"] + "/temp.jpg")
  face_encodings = face_recognition.face_encodings(fr_image)
  return check_similarity(encoding, face_encodings)

def get_encoding_with_file(path):
  fr_image = face_recognition.load_image_file(path)
  face_encoding = face_recognition.face_encodings(fr_image)
  if len(face_encoding) > 1 or len(face_encoding) == 0:
    raise Exception("Please show one face")
  return face_encoding[0]

def get_encoding_with_image(img):
  image = Image.open(io.BytesIO(img))
  image.save(os.environ["HOME"] + "/temp.jpg")
  return get_encoding_with_file(os.environ["HOME"] + "/temp.jpg")
```

These are the main functions used to process faces in images.

Facial Recognition



Chaquopy plugin

face_recognition library

```
([-1.25474021e-01,  4.30163741e-03,  2.50592083e-02, -6.11674339e-02,
  -3.18518914e-02, -5.66688962e-02,  5.47048636e-02, -7.81638995e-02,
   1.60959512e-01, -1.16157725e-01,  2.28007063e-01, -7.74169341e-02,
  -2.01009691e-01, -1.11756742e-01,  3.03803906e-02,  1.02945186e-01,
  -1.14417970e-01, -1.14191815e-01, -1.20461933e-01, -7.59568363e-02,
   2.08363403e-02, -8.18593577e-02, -1.42102260e-02,  6.66017830e-02,
  -1.43667817e-01, -3.90285760e-01, -1.20794192e-01, -1.35852322e-01,
   3.07489792e-03, -9.18435529e-02, -4.00098078e-02,  3.75721008e-02,
  -1.22249305e-01, -5.98395392e-02, -6.17018417e-02,  5.98258823e-02,
  -2.94239987e-02, -7.18261003e-02,  1.84724033e-01, -6.76385034e-03,
  -1.94659337e-01, -5.52332550e-02, -3.48776579e-04,  2.41776228e-01,
   1.87749296e-01,  1.69469193e-02,  4.13279235e-03, -7.20760971e-03,
   9.44364071e-02, -2.60828912e-01,  4.43785638e-02,  7.82113224e-02,
   8.16664398e-02,  2.60556620e-02,  9.03658867e-02, -8.43490511e-02,
   2.20645592e-02,  9.92065445e-02, -2.66967118e-01,  2.77326927e-02,
  -6.60007820e-03, -1.24366567e-01, -4.61388752e-02, -1.70268975e-02,
   2.33746380e-01,  2.25252047e-01, -7.59788752e-02, -8.69664252e-02,
   2.13570178e-01, -1.98724985e-01,  1.14265606e-02, -1.07026733e-02,
  -8.74682218e-02, -2.19989210e-01, -2.67576039e-01,  8.24404806e-02,
   3.94626886e-01,  2.02386737e-01, -1.48090839e-01,  3.77145372e-02,
  -1.10428751e-01, -2.82877982e-02,  1.54161215e-01,  2.74406187e-02,
  -1.13311373e-01,  3.40634771e-02, -1.01073988e-01,  4.22147363e-02,
   1.58401936e-01, -2.31090933e-03, -4.32495736e-02,  2.13270009e-01,
  -4.24002670e-02,  3.45717072e-02,  8.97298455e-02, -2.39595175e-02,
  -8.55992958e-02,  2.76423246e-02, -1.21746182e-01,  1.85502619e-02,
   1.31400898e-01, -1.03786439e-01,  6.87233433e-02,  7.03739300e-02,
  -1.45974010e-01,  9.09243822e-02, -4.81949095e-03, -2.21767239e-02,
   2.23567225e-02,  8.97686556e-03, -1.24209151e-01, -6.91418350e-02,
   1.47161692e-01, -2.26013437e-01,  1.37712449e-01,  1.61237642e-01,
   2.29968876e-02,  1.64134055e-01,  9.35962647e-02,  2.10202113e-02,
   7.73406029e-03, -5.32441735e-02, -1.75140992e-01,  3.95164229e-02,
   9.94656160e-02,  1.16615593e-02,  8.28401968e-02,  3.96566689e-02])
```

compare encodings

Firebase

CENTER OF EXCELLENCE
ARTIFICIAL INTELLIGENCE & ROBOTICS

APIs Used

- **<u>Open Weather Map API</u>:** This is used to get weather updates on the Dashboard. It calls for update every hour and also if the distance travelled exceeds 50km.

```
def get_weather(lat, lon):
  global geoLoc, geoData, lastTime
  currTime = time.time()
  if lastTime == None or currTime - lastTime > 3600 or abs(geoLoc[0] - lat) > 0.5
        or abs(geoLoc[1] - lon) > 0.5:
    apiKey = "5335f4dd5e54d57760b199b8330173e1"
    url = "https://api.openweathermap.org/data/2.5/onecall?lat=%f&lon=%f&appid=%s&units=metric"
          % (lat, lon, apiKey)
    response = requests.get(url)
    geoData = json.loads(response.text)
    geoLoc = [lat, lon]
    lastTime = currTime
  return geoData["timezone"] + " / " +  str(geoData["current"]["temp"]) + " / "
         + geoData["current"]["weather"][0]["main"] + " / " + str(geoData["current"]["humidity"])
         + " / " + str(geoData["current"]["wind_speed"])
```

- **<u>Google Maps API</u>:** This is used to display the map on the Dashboard. It is also used to calculate the speed of the vehicle.

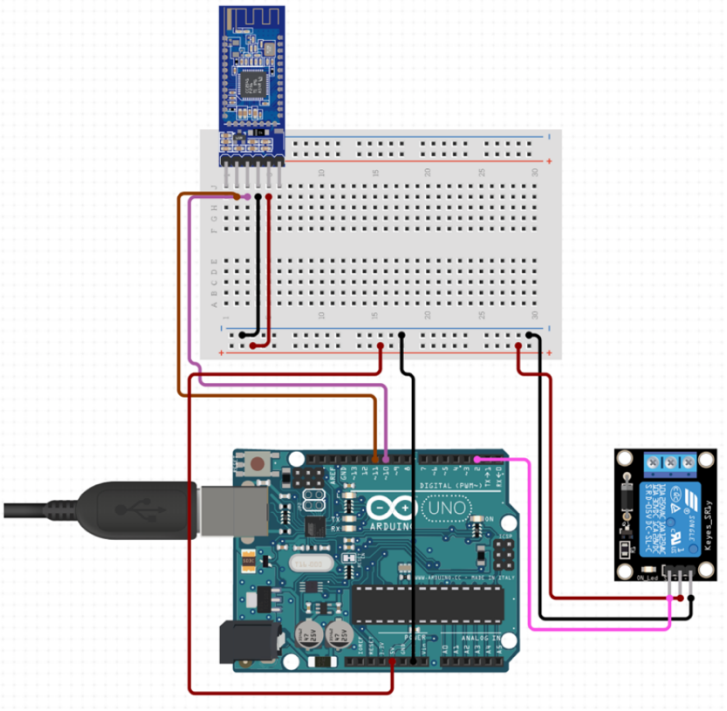
- **<u>YouTube Music API</u>:** This is used to make the Music Player on the Dashboard. It uses the ***ytmusicapi*** and ***pafy*** library in Python.

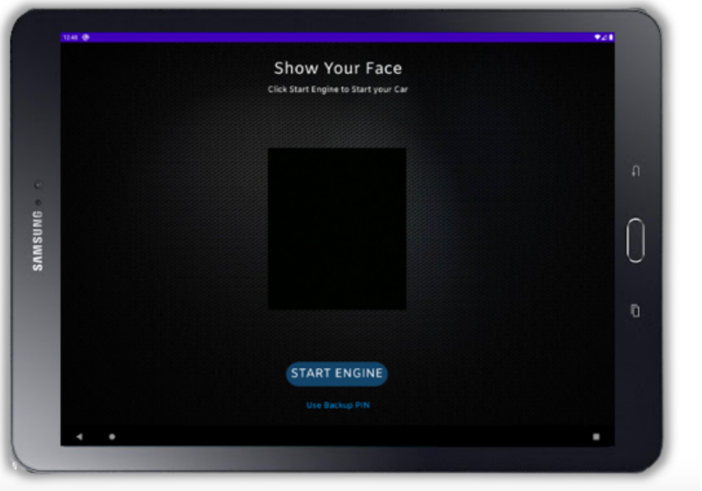
```
def get_songs(name):
  results = ytmusic.search(name)
  songs = []
  for r in results:
    try:
      tmp = [int(x) for x in r["duration"].split(":")]
      songs.append(str({"title": r["title"], "id": r["videoId"], "url": get_url(r["videoId"]),
      "artists": ", ".join(x["name"] for x in r["artists"]),
      "duration": "{:02d}:{:02d}".format(tmp[0], tmp[1]), "thumbnail": r["thumbnails"][0]["url"]}))
    except:
      continue
  return " / ".join(s for s in songs)

def get_url(songID):
  video = pafy.new("https://youtube.com/watch?v="+songID)
  return video.getbest().url

def get_song(songID):
  details = ytmusic.get_song(songID)['videoDetails']
  return {"title": details["title"], "id": songID, "url":get_url(songID),
  "artists": details["author"], "duration": sec_to_time(details["lengthSeconds"]),
  "thumbnail": details["thumbnail"]["thumbnails"][0]["url"]}
```

- **<u>Photon OpenStreetMaps API</u>**: This is used to get the search results for places in navigation.

Future Prospects

The app could be connected to an Arduino on the vehicle and when logged in, the app sends a message to the Arduino via Bluetooth to switch on the relay and start the engine.





THANK

YOU