Configuration Manual

MSc Research Project
Cloud Computing

Akash Parapurath Govindarajan
Student ID:x20122101

School of Computing
National College of Ireland

Supervisor: Divya Elango

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Akash Parapurath Govindarajan
Student ID:	x20122101
Programme:	Cloud Computing
Year:	2021
Module:	MSc Research Project
Supervisor:	Divya Elango
Submission Due Date:	16/12/2021
Project Title:	Configuration Manual
Word Count:	1400
Page Count:	8

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Akash Parapurath Govindarajan
Date:	31st January 2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Akash Parapurath Govindarajan
x20122101

1 Introduction

The configuration manual provides us with the complement implementation of our integrated Masked Face Identification model web application. This project is done with Deep learning and MLOps. In this documentation, we will have a clear view of how the application files are handled in our project.

2 System Configuration

2.1 Hardware Requirements For Local Machine

The implementation is done in Local for primarily training the algorithm results. For evaluating, we need a huge computational GPU system to run these Deep Learning algorithms. The system consists of the following configuration:

Operating System: Windows 10 x64

Processor: Intel Core i5-9300H @ 2.4 GHz

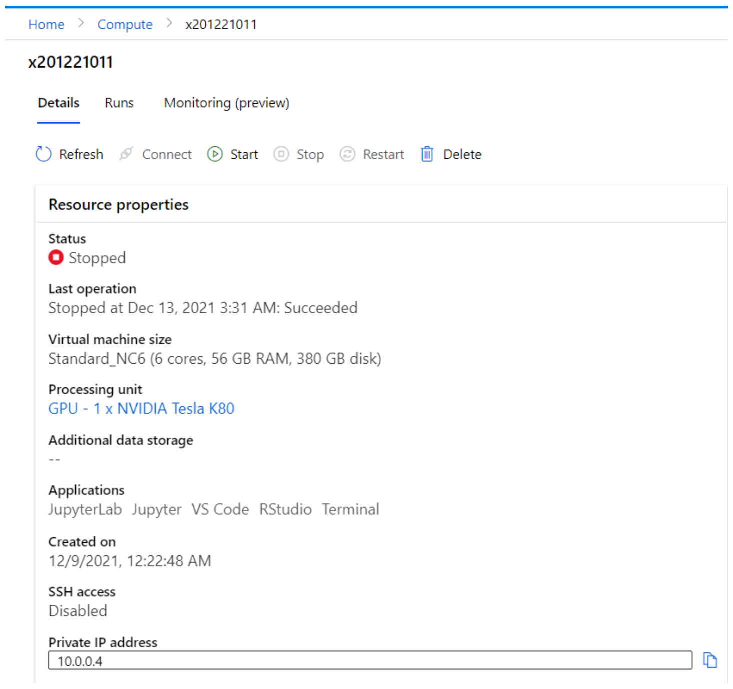
RAM: 16 GB

Hard drive: 215 GB SSD

GPU: GEFORCE GTX 1650 Ti

2.2 Hardware Requiremnets For Cloud Instance

This heading explains the requirements for our application to run in the Azure Cloud. Here this compute helps to run the deep learning model in the cloud and can be accessed by REST API generated by the cloud. We use the free standard GPU computation provided by Azure. Here we have a computation of 6 cores, 56GB RAM, 380 GB Disk, and GPU of NVIDIA Tesla K80. Figure 1 shows the computational setup we have in our cloud.



x201221011

Resource properties

Status
Stopped

Last operation
Stopped at Dec 13, 2021 3:31 AM: Succeeded

Virtual machine size
Standard_NC6 (6 cores, 56 GB RAM, 380 GB disk)

Processing unit
GPU - 1 x NVIDIA Tesla K80

Additional data storage
--

Applications
JupyterLab Jupyter VS Code RStudio Terminal

Created on
12/9/2021, 12:22:48 AM

SSH access
Disabled

Private IP address
10.0.0.4

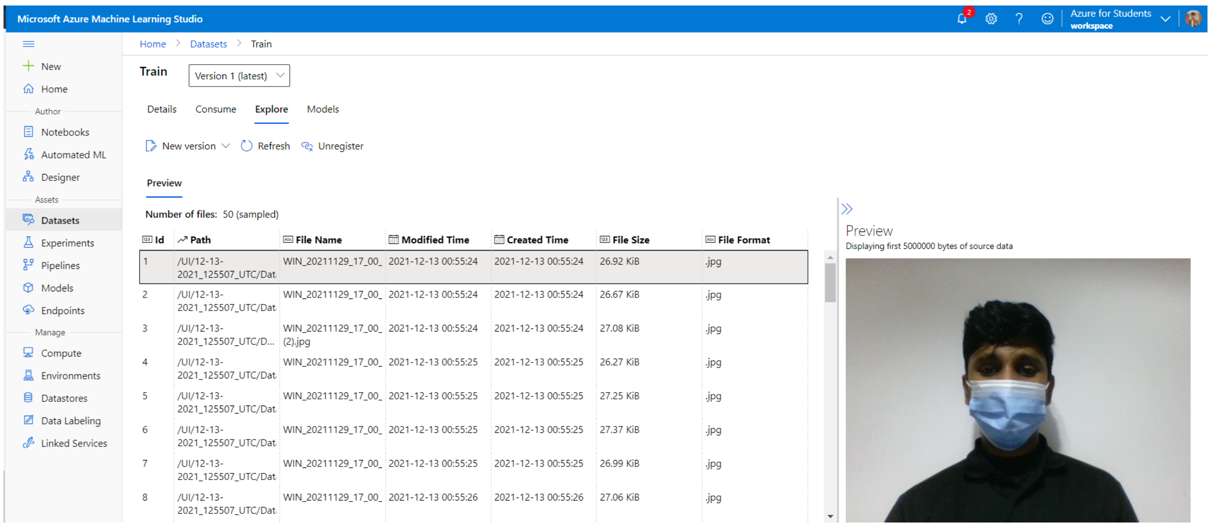
Figure 1: Computational Requirements in Azure Cloud

2.3 Software Requirements

This complete project is implemented in Python 3.6 for building the deep learning model and it is completely used in the back-end. The Masked Face Identification model is implemented using Custom CNN and Inception V3 deep learning models. These algorithms are adapted from previous research. We also use HTML, CSS, JavaScript for the front-end to develop the UI for our application. We use Azure Cloud Service to implement the MLOps concept in our application. This also helps to deploy our application live. The following are the software and packages used to build our project.
1. Python 3.6
2. HTML
3. CSS
4. JavaScript
5. Azureml
6. Jupyter Notebook
7. VS Code IDE
8. TensorFlow
9. Keras
10. pandas
11. matplotlib
12. os
13. urllib
14. json
15. tqdm
16. opencv

3 Dataset Description

The data set for our project is manually generated by OpenCV. This data set is stored in the Azure Blob storage and Google drive that can be accessed to perform the model building process. The figure shows the data sets stored in the Azure Cloud that are used to do the model building with Azure Machine Learning.



Number of files: 50 (sampled)

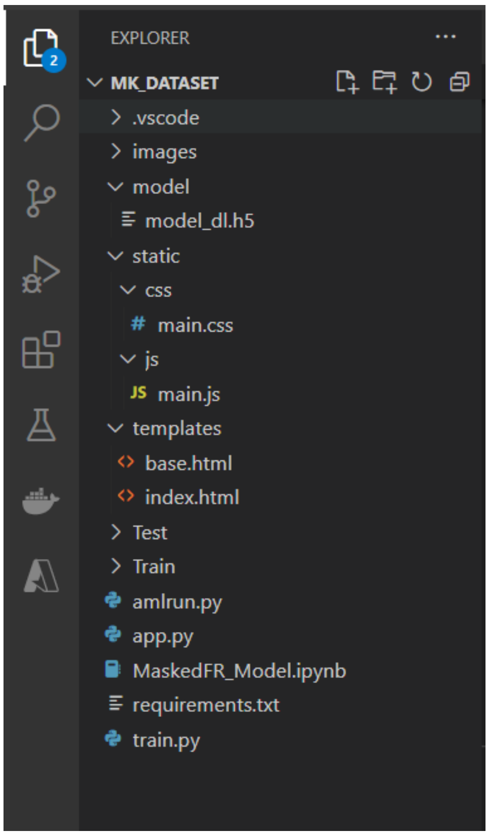
Id	Path	File Name	Modified Time	Created Time	File Size	File Format
1	/UI/12-13-2021_125507_UTC/Dat	WIN_20211129_17_00_	2021-12-13 00:55:24	2021-12-13 00:55:24	26.92 KiB	.jpg
2	/UI/12-13-2021_125507_UTC/Dat	WIN_20211129_17_00_	2021-12-13 00:55:24	2021-12-13 00:55:24	26.67 KiB	.jpg
3	/UI/12-13-2021_125507_UTC/D...	WIN_20211129_17_00_ (2).jpg	2021-12-13 00:55:24	2021-12-13 00:55:24	27.08 KiB	.jpg
4	/UI/12-13-2021_125507_UTC/Dat	WIN_20211129_17_00_	2021-12-13 00:55:25	2021-12-13 00:55:25	26.27 KiB	.jpg
5	/UI/12-13-2021_125507_UTC/Dat	WIN_20211129_17_00_	2021-12-13 00:55:25	2021-12-13 00:55:25	27.25 KiB	.jpg
6	/UI/12-13-2021_125507_UTC/Dat	WIN_20211129_17_00_	2021-12-13 00:55:25	2021-12-13 00:55:25	27.37 KiB	.jpg
7	/UI/12-13-2021_125507_UTC/Dat	WIN_20211129_17_00_	2021-12-13 00:55:25	2021-12-13 00:55:25	26.99 KiB	.jpg
8	/UI/12-13-2021_125507_UTC/Dat	WIN_20211129_17_00_	2021-12-13 00:55:26	2021-12-13 00:55:26	27.06 KiB	.jpg

Preview
Displaying first 5000000 bytes of source data

Figure 2: Azure Blob Storage

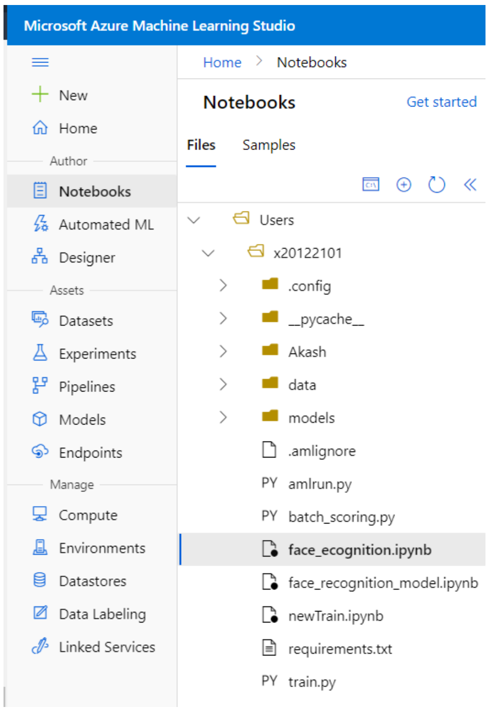
4 Environment Setup

The environment setup is done in Cloud as well as Local. The cloud performs the model building with Deep Learning Neural Network for the masked face recognition. And we perform web application development in our Local Machine with Flask, HTML, CSS, JavaScript. Firstly, we use the Jupyter Notebook to Train, Test, and Build our model. Figure 4 shows the environment setup made in Azure Machine Learning. Azure Machine Learning, it provides us to create a workspace with GPU computation which is discussed in the previous heading. Azure ML also provides an MLOps(Machine Learning Operation) feature for our deep Learning Model with the Azure Machine Learning Pipeline option on the left. Secondly, the web development environment for our application is set up in our local machine with the help of Virtual Studio Code(VS Code). By running *"pip install -r requirements.txt"* all the required packages that need for our model building will be installed. This VS Code also supports the Azure Cloud Service with an extension that can be connected with this IDE. Figure 3 shows the folder structure that is created for our Integrated Model web application. The web application is built with HTML, CSS, and JavaScript that is listed under the templates and static folder. The app.py is the main file that handles all the integration of the model with the application and also this file is used to start the Service. Let's see the complete running of our code in the upcoming heading.



```
EXPLORER
MK_DATASET
  .vscode
  images
  model
    model_dl.h5
  static
    css
      main.css
    js
      main.js
  templates
    base.html
    index.html
  Test
  Train
  amlrun.py
  app.py
  MaskedFR_Model.ipynb
  requirements.txt
  train.py
```

Figure 3: Folder Structure Created For the Web Application in VS Code IDE



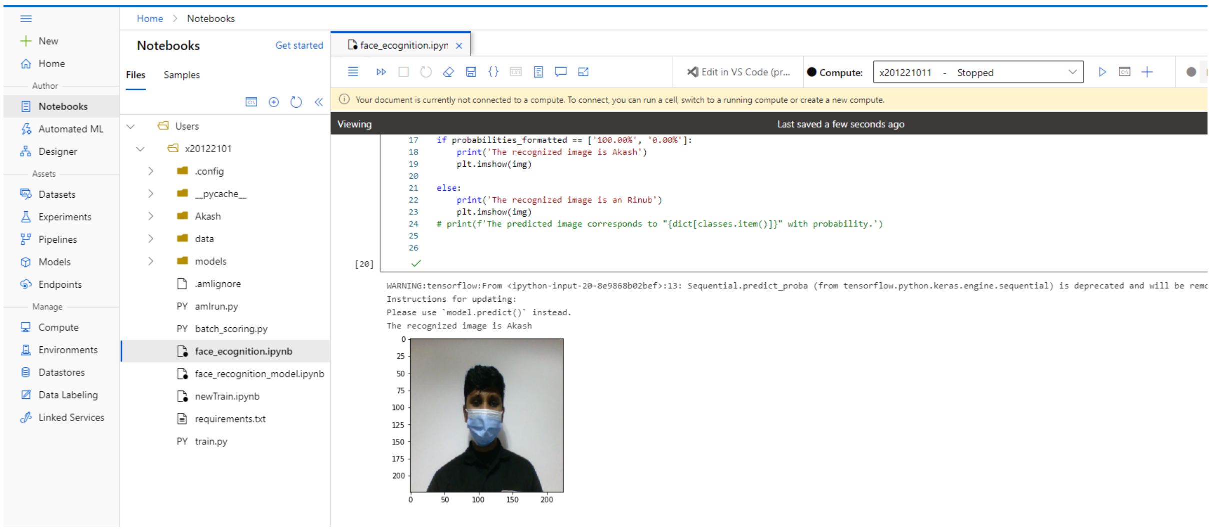
```
Microsoft Azure Machine Learning Studio

New                 Home > Notebooks
Home                Notebooks                Get started
Author              Files   Samples
Notebooks           Users
Automated ML          x20122101
Designer                .config
Assets                  __pycache__
Datasets                Akash
Experiments             data
Pipelines               models
Models                  .amlignore
Endpoints               amlrun.py
Manage                  batch_scoring.py
Compute                 face_ecognition.ipynb
Environments            face_recognition_model.ipynb
Datastores              newTrain.ipynb
Data Labeling           requirements.txt
Linked Services         train.py
```

Figure 4: Folder Structure Jupyter Notebook in Azure Machine Learning Workspace

5 Training and Implementation of Model with Cloud

This training and Implementation heading gives an overall flow of how to handle all the files that are used to implement our Masked Face Recognition model with MLOps. Here Firstly, we will see the cloud setup which we have made for our Model building and MLOps. Figure 5 shows the model running and identification of a person in the Azure Machine Learning workspace. This is a workspace we can build the model by creating Jupyter Notebooks. Here we train this ***"Face_Recognition.ipynb"*** file that is used to train the model for our Masked Face Recognition. If we run This program the model will start train based on the code. The figures 6,7,8 shows the pipeline creation for this MLOps process. The Azure Machine Learning has the inbuilt feature of MLOps (Machine Learning Operation) that can be accessed by the options Azure ML Pipeline. The Pipeline can be accessed by the trained model with the library *"azure ml"* package in our code. The running pipeline is shown clearly in the second and third figures that are clicked to view the experiment outputs and also if the code is changed in the workspace and if you run all the model building pipeline will trigger to create the new endpoint that can be accessed by the web application to implement the model in web application.



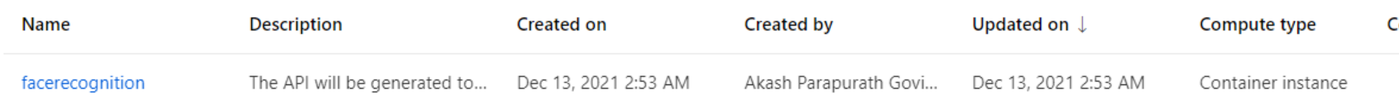
Your document is currently not connected to a compute. To connect, you can run a cell, switch to a running compute or create a new compute.

```
    if probabilities_formatted == ['100.00%', '0.00%']:
        print('The recognized image is Akash')
        plt.imshow(img)

    else:
        print('The recognized image is an Rinub')
        plt.imshow(img)
    # print(f'The predicted image corresponds to "{dict[classes.item()]}" with probability.')
```

```
[20]
WARNING:tensorflow:From <ipython-input-20-8e9868b02bef>:13: Sequential.predict_proba (from tensorflow.python.keras.engine.sequential) is deprecated and will be remc
Instructions for updating:
Please use `model.predict()` instead.
The recognized image is Akash
```

Figure 5: Model Running in Azure Machine Learning Workspace

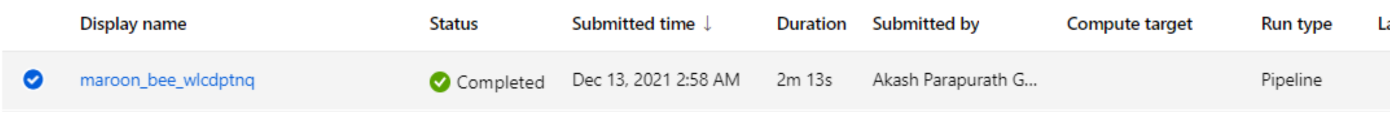


Name	Description	Created on	Created by	Updated on ↓	Compute type	C
facerecognition	The API will be generated to...	Dec 13, 2021 2:53 AM	Akash Parapurath Govi...	Dec 13, 2021 2:53 AM	Container instance	

Figure 6: Build model is running in Azure Container

maroon_bee_wlcdptnq	Masked-Face-Recognition	✓ Completed	Dec 13, 2021 2:58 AM

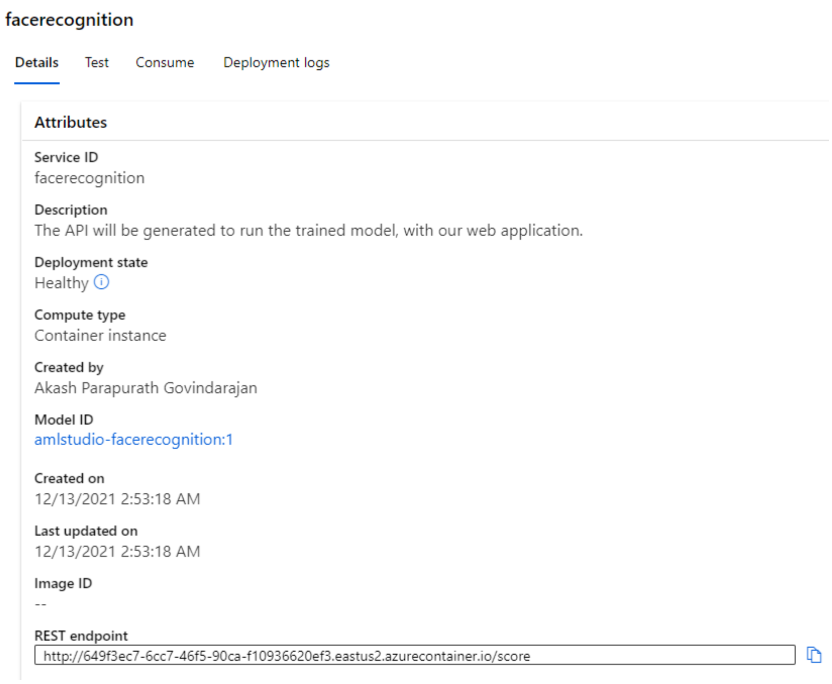
Figure 7: ML Pipeline For MLOps



	Display name	Status	Submitted time ↓	Duration	Submitted by	Compute target	Run type	La
✓	maroon_bee_wlcdptnq	✓ Completed	Dec 13, 2021 2:58 AM	2m 13s	Akash Parapurath G...		Pipeline	

Figure 8: ML Pipeline For MLOps

The Completely trained model will generate a REST API, Through this REST API, you can easily access the model to test and validate. This model-trained end-point API can be accessed with the flask in the back end to easily Validate the authentication process of a Masked Face person. Figure 9 shows the end-point that is generated for our model and is integrated with the web application.



facerecognition

Details Test Consume Deployment logs

Attributes

Service ID
facerecognition

Description
The API will be generated to run the trained model, with our web application.

Deployment state
Healthy

Compute type
Container instance

Created by
Akash Parapurath Govindarajan

Model ID
amlstudio-facerecognition:1

Created on
12/13/2021 2:53:18 AM

Last updated on
12/13/2021 2:53:18 AM

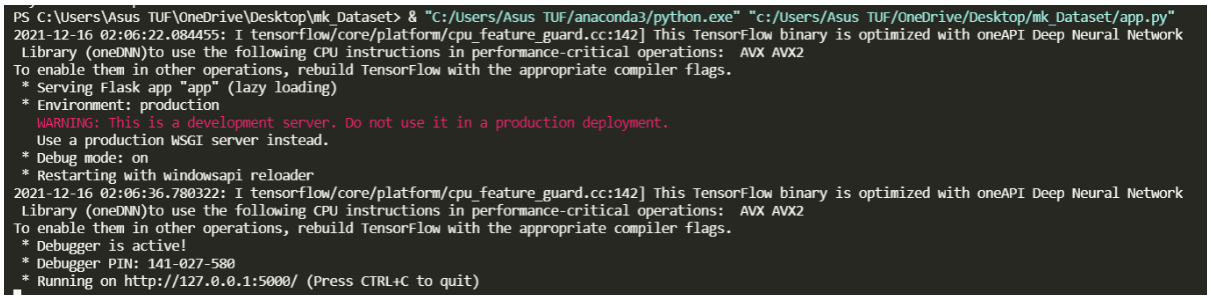
Image ID
--

REST endpoint
http://649f3ec7-6cc7-46f5-90ca-f10936620ef3.eastus2.azurecontainer.io/score

Figure 9: The Rest API end-point for accessing the model output

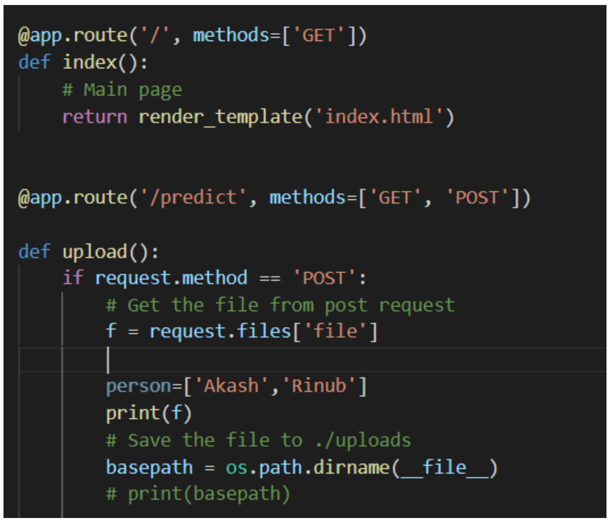
Finally, we will discuss the model integration part with the web application. Previous figures show the implementation and model building with Azure Cloud Service. The model was built and integrated with the MLOps part with the help of the Azure Machine Learning service. Now we see how to run the developed web application model with flask. From the figure 3, we see the environment setup we made in our local machine to run the web application. From the image 10, we see the *"app.py"* which is the heart of the project. This is the file that uses the flask framework and triggers all UI to communicate with the back-end developed model. After running the main file app.py the server started that server is communicated with the model that is deployed in the Azure Container. This interaction of recognizing the person is identified by REST API which is generated as an end-point that we discussed in the previous paragraph.

The next figure 11 shows the process of Recognizing the masked face with the help of a flask. Generally, the flask is a web framework, that performs web actions like GET, POST, PUT, DELET methods. Here in that figure, we create a post method function



```
PS C:\Users\Asus TUF\OneDrive\Desktop\mk_Dataset> & "C:/Users/Asus TUF/anaconda3/python.exe" "c:/Users/Asus TUF/OneDrive/Desktop/mk_Dataset/app.py"
2021-12-16 02:06:22.084455: I tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary is optimized with oneAPI Deep Neural Network
 Library (oneDNN)to use the following CPU instructions in performance-critical operations:  AVX AVX2
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with windowsapi reloader
2021-12-16 02:06:36.780322: I tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary is optimized with oneAPI Deep Neural Network
 Library (oneDNN)to use the following CPU instructions in performance-critical operations:  AVX AVX2
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
 * Debugger is active!
 * Debugger PIN: 141-027-580
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Figure 10: Running Server with Flask

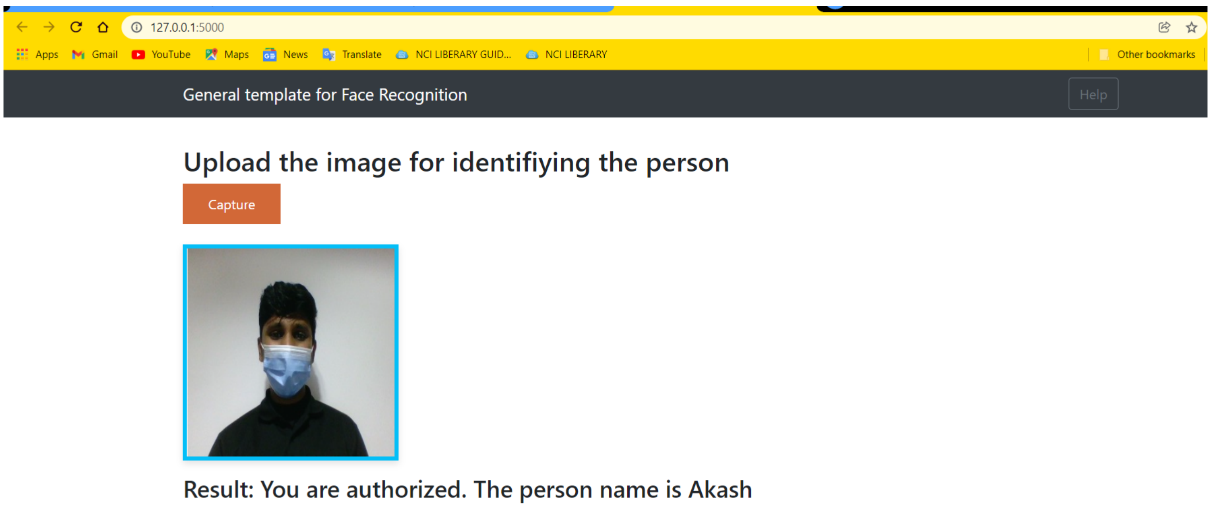


```python
@app.route('/', methods=['GET'])
def index():
    # Main page
    return render_template('index.html')


@app.route('/predict', methods=['GET', 'POST'])

def upload():
    if request.method == 'POST':
        # Get the file from post request
        f = request.files['file']
        
        person=['Akash','Rinub']
        print(f)
        # Save the file to ./uploads
        basepath = os.path.dirname(__file__)
        # print(basepath)
```

Figure 11: Flask which Handles the Captured image from the Front-End for authentication



General template for Face Recognition

Upload the image for identifiying the person

Result: You are authorized. The person name is Akash

Figure 12: Running Implemented Web Application

that handles the image which is captured by the camera in the front-end and the image is passed to the model for authentication. Finally, this figure 12 shows the implementation of the demo. The authentication is performed in this web application. If we run this program all previously mention scenarios will perform in sequential order. By this configuration manual, we will get a proper way to handle this End-To-End Masked Face Recognition application using Deep Learning and MLOps.