CLOUD APPLICATION DEVELOPMENT

PROJECT TITLE:

IMAGE RECOGNITION WITH IBM CLOUD VISUAL RECOGNITION COMPUTING

STEPS TO DEPLOY ON IBM CLOUD FOUNDARY:

1.Login to IBM cloud.

2.Target a cloud foundary organization and space .

3.Push your application.

4.Bind IBM cloud visual recognition service .

5.Restage your application .

6.Access your application.

7.Monitoring and scaling.

SAMPLE CODE:

```
from flask import Flask,request,jsonify

from ibm_watson import VistualRecognitionV3

from ibm_cloud _sdk_core.authenticators import IAMAuthenticator

app=Flask(_ _ name_ _)

API_KEY=   ‘YOUR_API_KEY’

MODEL-ID=  ‘YOUR-MODEL-ID’

authenticator =    IAMAuthennticator(API_KEY)

visual_recognition = VisualRecognitionV3(version=’2018-03-19’,authenticator=authenticator)

visual_recognition.set_service-url(‘https://api.us-south.visual-recognition.watson.cloud.ibm.com’)

@app.route(‘/predict’,methods=[‘POST’])

def  predict():

 try:

file=request.files[‘file’]
```

```
classes=visual_recognition.classify(file=file , threshold=’0.6’,classifier_ids=[MODEL_ID]).get_result()
predictions=[]
for class_result in classess [‘images’][0][‘classifiers’[0][‘classes’]:
predictions.append({‘class’ : class_result[‘class’],’score’:class_result[‘score’]})
return jsonify(predictions)
except Exception as e:
return jsonify({‘error’:str(e)})
if_ _nmae_ _ ==’_ _ main _ _’:
app.run(debug=True)
```

PERFROM DIFFERENT FUNCTIONS FOR PROJECT REQUIREMENTS:

1.UPLOAD AND PROCESS IMAGE:

```
From flask import request
@app.route(‘/upload’,methods=[‘POST’])
def upload_image():
try:
file = request.files[‘file’]
predictions = recognize_image(file)
return jsonify(predictions)
except Exception as e:
return jsonify({‘error’:str(e)})
```

2.IMAGE RECOGNITION FUNCTION:

```
def  recognize_image(image_file):
classes=visual_recognition.classify(file=image_file,threshold=’0.6’,,classifier_ids=[MODEL_ID]).
get_result()
predictions=[]
for class_result in classes[‘images’][0][‘classifiers’][0][‘classes’]:
```

```
predictions.append({‘class’: class_result[‘class’],’score’:class_result[‘score’]})

return predictions
```

3.LIST AVAILABLE CLASSIFIERS:

```
@app.route(‘/classifiers’,methods=[‘GET’])

def list_classifiers():

classifiers=visual_recognition.list_classifiers().get_result()

return jsonify(classifiers)
```

4.CREATE   NEW   CLASSIFIER:

```
@app.route(‘/classifiers ‘,methods=[‘POST’])

def create_classifier():

try:

with open(‘positive_examples.zip’,’rb’) as positive_examples,\

open(‘negative_examples.zip’,’rb’) as negative_examples:

classifiers=visual_recognition.create_classifier(

positive_examples = positive_examples,

negative_examples = negative_examples,

name=’NewClassifier’

).get_result()

return jsonify(classifier)

except Exception as e:

return jsonify({‘error’:str(e)})
```

5.DELETE CLASSIFIER:

```
@app.route(‘/classifiers/<classifier_id>’,methods=[‘DELETE’])

def delete_classifier(classifier_id):

try:

response=visual_recognition.delete_classifier(classifier_id).get_result()

return jsonify(response)
```

```
except Exception as e:

return jsonify({‘error’:str(e)})
```

6.ERROR HANDLING:

```
@app.errorhandler(Exception)

def handle_error(error):

response=jsonify({‘error’:str(error)})

response.status_code+500 return response
```