D602 Task 3
By Eric Williams

The purpose of this file is to meet the requirement E to “Provide an explanation of how you wrote your code, including any challenges you encountered and how you addressed those challenges.”

The beginning section of the code was given and remained largely untouched:

[25]:
```python
#!/usr/bin/env python
# coding: utf-8

# import statements
from fastapi import FastAPI, HTTPException
import json
import numpy as np
import pickle
import datetime

# Import the airport encodings file
f = open('airport_encodings.json')

# returns JSON object as a dictionary
airports = json.load(f)

def create_airport_encoding(airport: str, airports: dict) -> np.array:
    """
    create_airport_encoding is a function that creates an array the length of all arrival airports from the chosen
    departure aiport.  The array consists of all zeros except for the specified arrival airport, which is a 1.

    Parameters
    ----------
    airport : str
        The specified arrival airport code as a string
    airports: dict
        A dictionary containing all of the arrival airport codes served from the chosen departure airport

    Returns
    -------
    np.array
        A NumPy array the length of the number of arrival airports.  All zeros except for a single 1
        denoting the arrival airport.  Returns None if arrival airport is not found in the input list.
        This is a one-hot encoded airport array.

    """
    temp = np.zeros(len(airports))
    if airport in airports:
        temp[airports[airport]] = 1
        temp = temp.T
        return temp
    else:
        return None

# TODO:  write the back-end logic to provide a prediction given the inputs
# requires finalized_model.pkl to be loaded
# the model must be passed a NumPy array consisting of the following:
# (polynomial order, encoded airport array, departure time as seconds since midnight, arrival time as seconds since midnight)
# the polynomial order is 1 unless you changed it during model training in Task 2
# YOUR CODE GOES HERE

# TODO:  write the API endpoints.
```

Here are the commented out requirements above and a breakdown of my code for each piece

#1. Back-end logic to provide a prediction given the inputs

```
def predict_delay(departure_airport, arrival_airport, departure_time, arrival_time):
    # Get the one-hot encoded airport for arrival
    encoded_airport = create_airport_encoding(arrival_airport, airports)
    if encoded_airport is None:
        raise HTTPException(status_code=404, detail="Arrival airport not found")

    #Converting times to seconds since midnight
    try:
        dep_time_seconds = (datetime.datetime.strptime(departure_time, "%Y-%m-%dT%H:%M:%S") - datetime.datetime(1900, 1, 1)).total_seconds()
        arr_time_seconds = (datetime.datetime.strptime(arrival_time, "%Y-%m-%dT%H:%M:%S") - datetime.datetime(1900, 1, 1)).total_seconds()
    except ValueError:
        raise HTTPException(status_code=400, detail="Invalid time format. Please use 'YYYY-MM-DDTHH:MM:SS'.")

    # Preparing the input array with only the expected 4 features (polynomial order 1, length of airports,
    # departure after midnight, arrival after midnight)

    input_data = np.concatenate(([1], encoded_airport, [dep_time_seconds], [arr_time_seconds]))  # 4 features total

    #Prediction
    delay = model.predict(input_data.reshape(1, -1))  # The model expects a 2D array

    return delay[0]
```

I defined a prediction model for the four parameters we needed (departure airport, arrival airport, departure time, arrival time), then encoded it. This converted the variables to a binary input. I also added the exception error for when the arrival airport is not found.

#2. Requires finalized_model.pkl to be loaded

This was the most simple line, especially because pickle was already imported. This line just loads the model:

```
#Loading the trained model
with open('finalized_model.pkl', 'rb') as model_file:
    model = pickle.load(model_file)
```

#3. The model must be passed a NumPy array consisting of the following: (polynomial order, encoded airport array, departure time as seconds since midnight, arrival time as seconds since midnight):

First I had to concatenate the data so the NumPy array was only the four expected features. I ran into an error where 13 variables had been imported, so I decided to create the input data from the ground up, using just the necessary variables:

```
    # Preparing the input array with only the expected 4 features (polynomial order 1, length of airports,
    # departure after midnight, arrival after midnight)

    input_data = np.concatenate(([1], encoded_airport, [dep_time_seconds], [arr_time_seconds]))

    #Prediction
    delay = model.predict(input_data.reshape(1, -1))  #This is needed because the model expects a 2D array

    return delay[0]
```

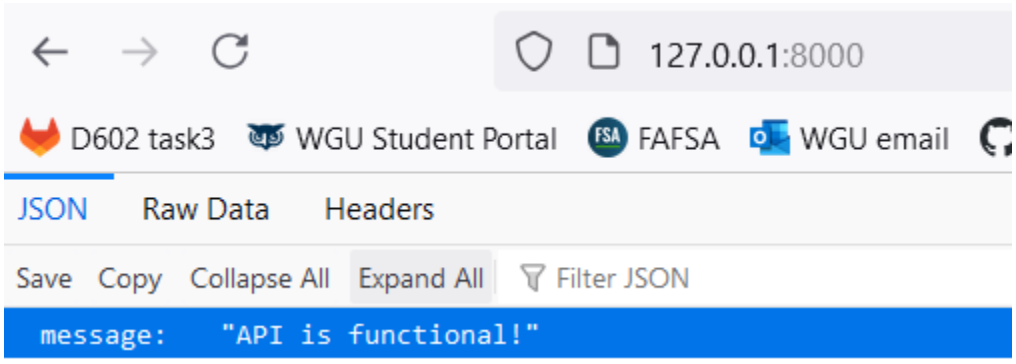
This also includes [1] for the polynomial order, and I reshaped the array to the proper size.
I then initialized the API test. I included a check I will show in my video of the API showing “API is functional!” Here I met the requirements for what would show “/” should return a JSON message indicating that the API is functional and “/predict/delays” should accept a GET request specifying the arrival airport, the local departure time, and the local arrival time.

```python
#Initializing FastAPI app
app = FastAPI()

#Root endpoint to check if the API is functional
@app.get("/")
async def root():
    return {"message": "API is functional!"}

#Prediction endpoint to get the average departure delay
@app.get("/predict/delays")
async def predict_delays(arrival_airport: str, departure_airport: str, departure_time: str, arrival_time: str):
    try:
        delay = predict_delay(departure_airport, arrival_airport, departure_time, arrival_time)
        return {"average_departure_delay": delay}
    except HTTPException as e:
        raise e
```

Here you can see the message returned that the API is functional, returned in a JSON file:



C:API TEST

Here are my three tests:

```python
from fastapi.testclient import TestClient
from API import app

client = TestClient(app)

def test_root():
    """Testing the root endpoint"""
    response = client.get("/")
    assert response.status_code == 200
    assert response.json() == {"message": "API is functional!"}

def test_invalid_airport():
    """Testing the /predict/delays endpoint (with an invalid airport)"""
    response = client.get("/predict/delays", params={
        "arrival_airport": "XYZ",  # Invalid airport code
        "departure_airport": "LAX",
        "departure_time": "2024-10-31T14:30:00",
        "arrival_time": "2024-10-31T22:15:00"
    })
    assert response.status_code == 404
    assert response.json() == {"detail": "Arrival airport not found"}

def test_invalid_time_format():
    """Testing the /predict/delays endpoint (with an invalid time format)"""
    response = client.get("/predict/delays", params={
        "arrival_airport": "JFK",
        "departure_airport": "LAX",
        "departure_time": "14:30",  # Invalid format
        "arrival_time": "2024-10-31T22:15:00"
    })
    assert response.status_code == 400
    assert response.json() == {"detail": "Invalid time format. Please use 'YYYY-MM-DDTHH:MM:SS'."}
```

As required, they contain test features of endpoints with both correctly formatted and incorrectly formatted requests. In the first test, I tested the root endpoint. In the second endpoint, I included an invalid departure code while everything else was valid. In the last test, I included an invalid time format while everything else was valid. I chose to use the well known airport codes rather than the airport ID numbers since this was not required by the rubric. Note that each test includes an accurate and appropriate return code, both for when the input is correct and incorrect.

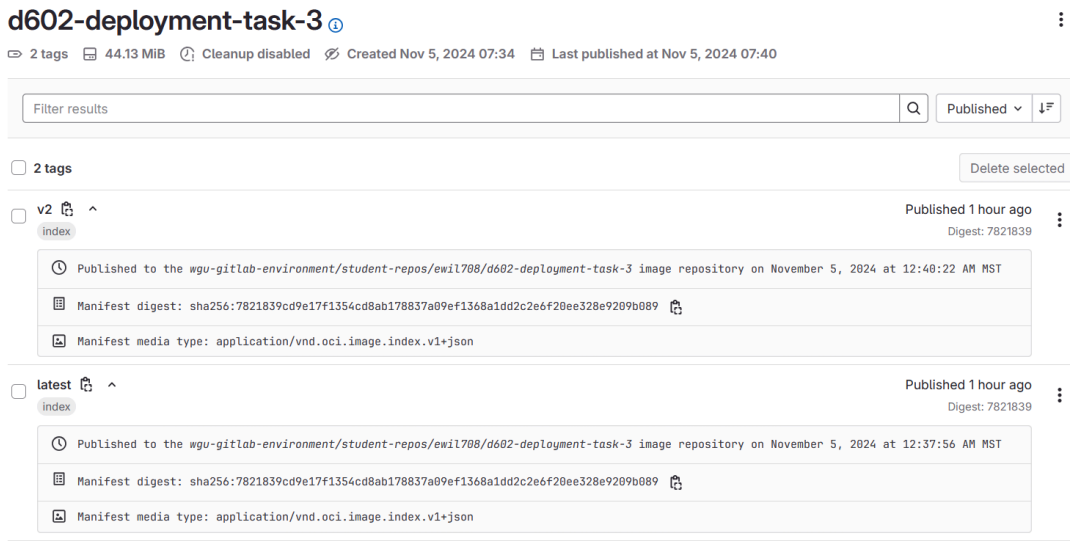
Code was not required for the dockerfile, but I did run into some major issues. This first screenshot shows when I tried to build the file, but I had too many large files in the reference file and had to clean it out:

```
(base) C:\Users\18014>docker build -t registry.gitlab.com/wgu-gitlab-environment/student-repos/ewil708/d602-deployment-task-3 .
[+] Building 45.7s (5/9)                                                                    docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                        0.0s
 => => transferring dockerfile: 1.27kB                                                                      0.0s
 => WARN: FromAsCasing: 'as' and 'FROM' keywords' casing do not match (line 7)                              0.0s
 => [internal] load metadata for docker.io/library/python:3.9-slim                                          0.5s
 => [internal] load .dockerignore                                                                           0.0s
 => => transferring context: 2B                                                                             0.0s
 => ERROR [internal] load build context                                                                    45.0s
 => => transferring context: 237.26MB                                                                      44.9s
 => CACHED [1/5] FROM docker.io/library/python:3.9-slim@sha256:7a9cd42706c174cdcf578880ab9ae3b6551323a7ddbc2a89ad  0.1s
 => => resolve docker.io/library/python:3.9-slim@sha256:7a9cd42706c174cdcf578880ab9ae3b6551323a7ddbc2a89ad6e5b20a  0.1s
------
 > [internal] load build context:
------
```

Then I had an issue with access because I missed one of the boxes when creating my access token:

```
C:\Users\18014>docker push registry.gitlab.com/wgu-gitlab-environment/student-repos/ewil708/d602-deployment-task-3
Using default tag: latest
The push refers to repository [registry.gitlab.com/wgu-gitlab-environment/student-repos/ewil708/d602-deployment-task-3]
70631fcd0b78: Waiting
6999734a3aaa: Waiting
aa77a90b11d9: Waiting
bf916c16d664: Waiting
a480a496ba95: Waiting
f281ad68d612: Waiting
0dd753b657c9: Waiting
d7c3dc9735c0: Waiting
push access denied, repository does not exist or may require authorization: server message: insufficient_scope: authorization failed
```

After creating a new access token with the appropriate permissions, I was able to push the container registry in two versions:



Sources

No materials were used except for official WGU course materials