Deployment

Task 3

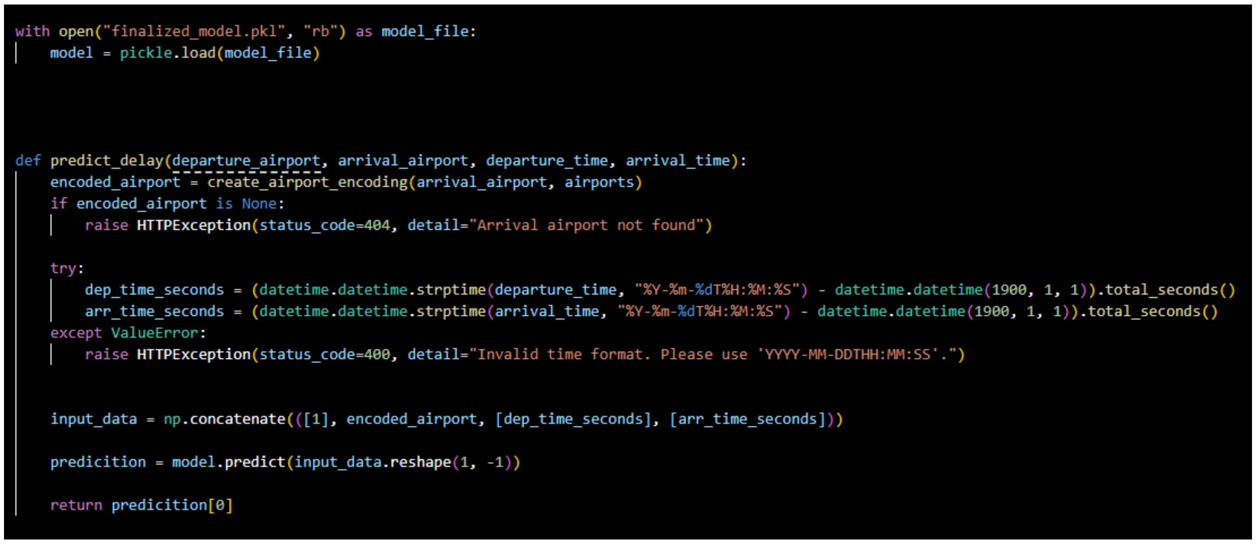
Nathan Hefner

B. Write an API with the code templates provided in either the FastAPI package in Python or the plumber package in R that accepts the following HTTP endpoints.

1. "/" should return a JSON message indicating that the API is functional.

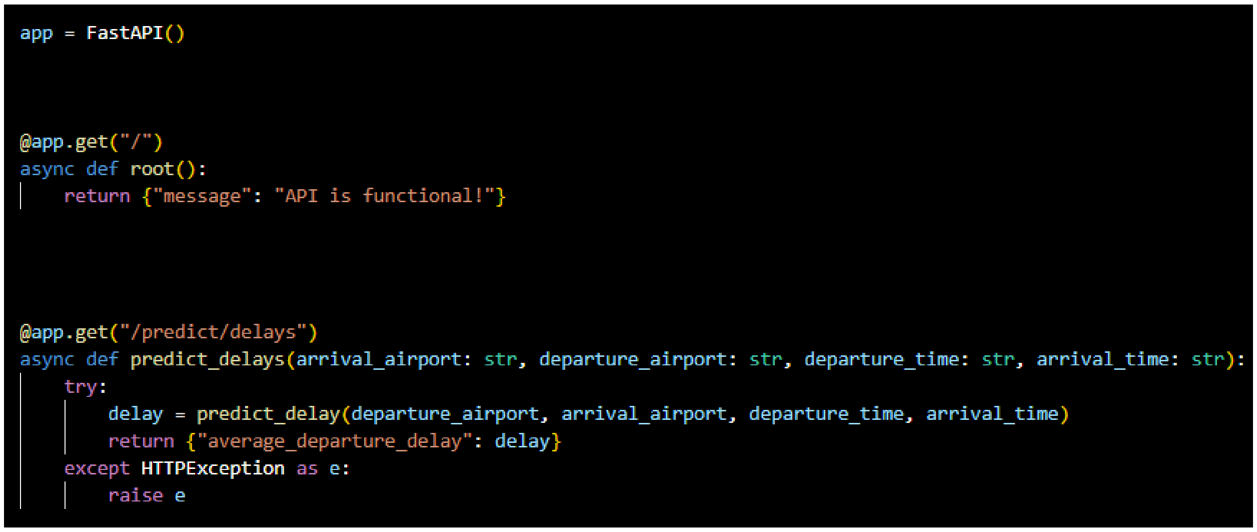
2. "/predict/delays" should accept a GET request specifying the arrival airport, the local departure time, and the local arrival time. It should return a JSON response indicating the average departure delay in minutes.

Writing the api, for the most part, was straightforward. I ran into issues while converting the departure and arrival times. I received NameErrors, but was able to work through them. Below is the code for outputting my prediction.



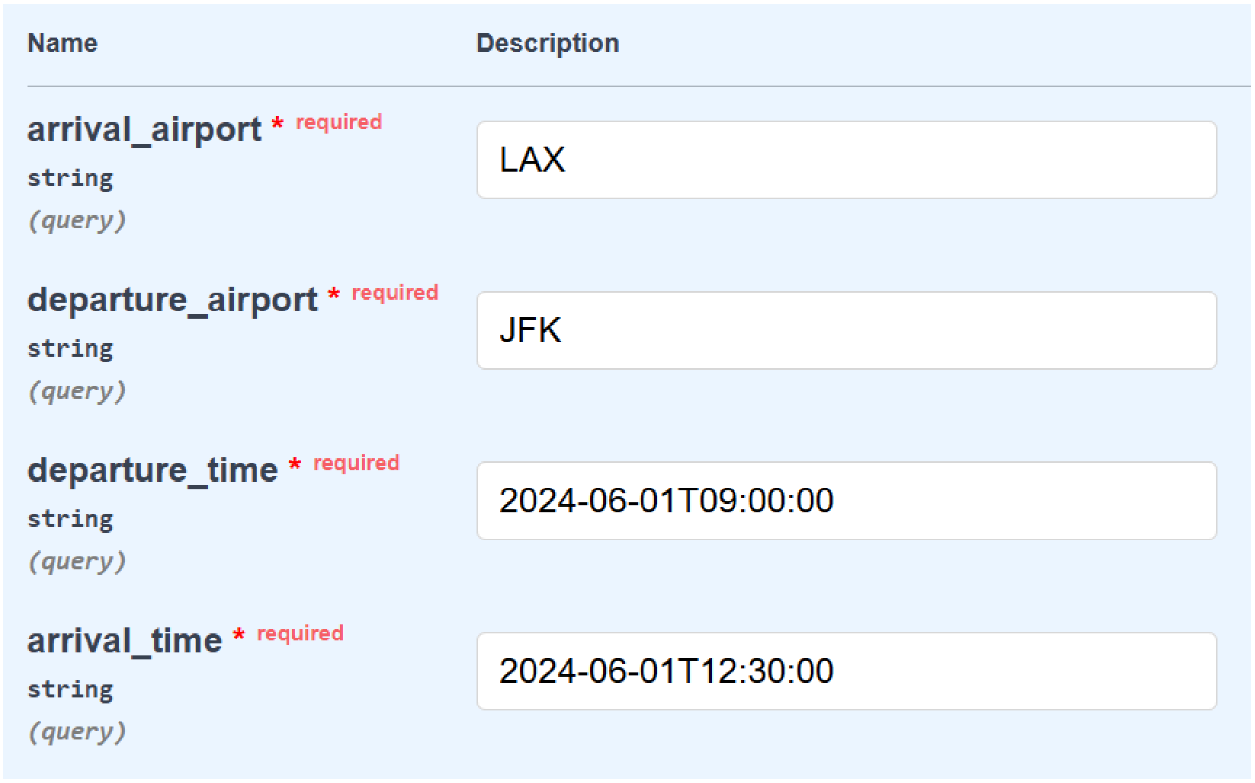
```python
with open("finalized_model.pkl", "rb") as model_file:
    model = pickle.load(model_file)



def predict_delay(departure_airport, arrival_airport, departure_time, arrival_time):
    encoded_airport = create_airport_encoding(arrival_airport, airports)
    if encoded_airport is None:
        raise HTTPException(status_code=404, detail="Arrival airport not found")

    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'.")


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

    predicition = model.predict(input_data.reshape(1, -1))

    return predicition[0]
```

Below is the FastAPI section for the get "/" and get "/predict/delays" commands.

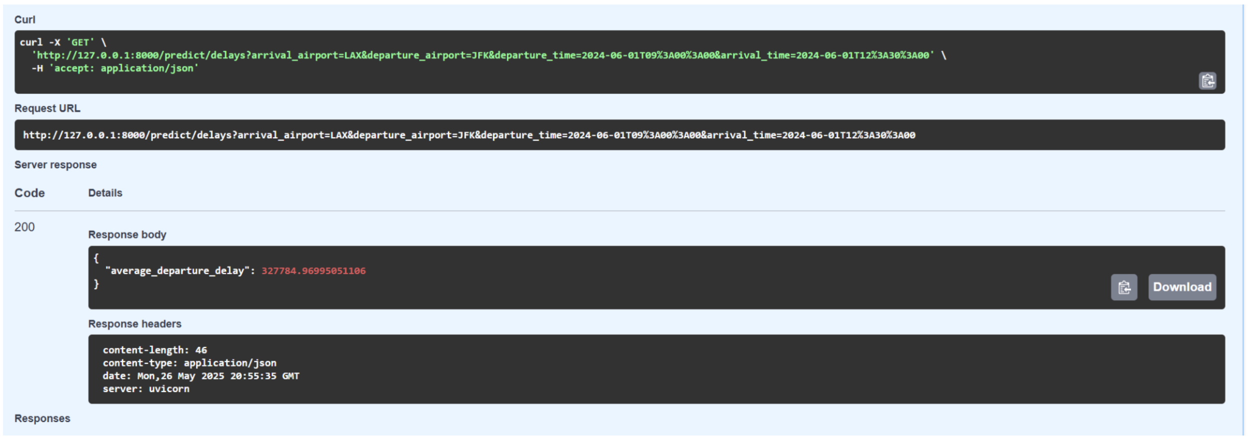


```python
app = FastAPI()



@app.get("/")
async def root():
    return {"message": "API is functional!"}




@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
```

Below are the results after I tested my API. To do so, I had to install many packages, run the api in my terminal, and go to a locally hosted web address.



Name	Description
arrival_airport * required string *(query)*	LAX
departure_airport * required string *(query)*	JFK
departure_time * required string *(query)*	2024-06-01T09:00:00
arrival_time * required string *(query)*	2024-06-01T12:30:00



Curl

```
curl -X 'GET' \
  'http://127.0.0.1:8000/predict/delays?arrival_airport=LAX&departure_airport=JFK&departure_time=2024-06-01T09%3A00%3A00&arrival_time=2024-06-01T12%3A30%3A00' \
  -H 'accept: application/json'
```

Request URL

```
http://127.0.0.1:8000/predict/delays?arrival_airport=LAX&departure_airport=JFK&departure_time=2024-06-01T09%3A00%3A00&arrival_time=2024-06-01T12%3A30%3A00
```

Server response

Code Details

200

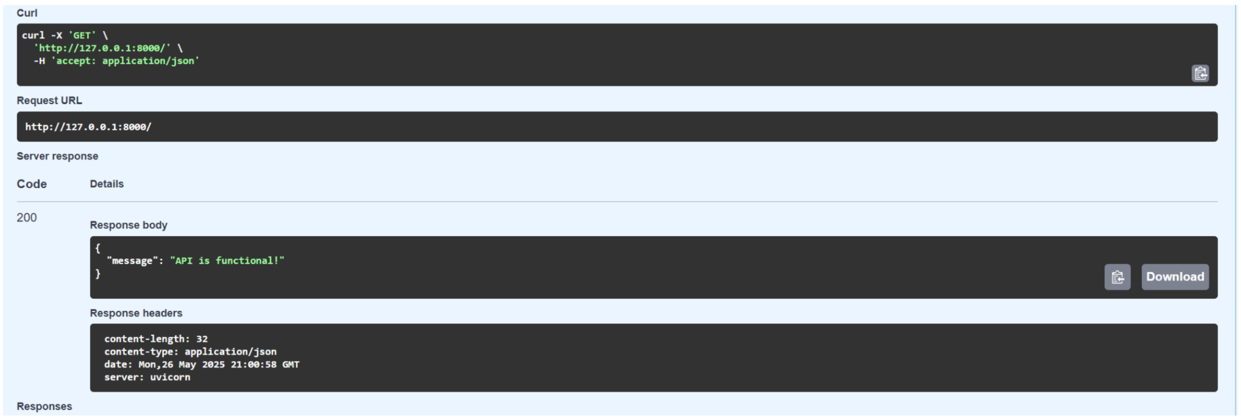
Response body

```
{
  "average_departure_delay": 327784.96995051106
}
```

Response headers

```
 content-length: 46
 content-type: application/json
 date: Mon,26 May 2025 20:55:35 GMT
 server: uvicorn
```

Responses



Curl

```
curl -X 'GET' \
  'http://127.0.0.1:8000/' \
  -H 'accept: application/json'
```

Request URL

```
http://127.0.0.1:8000/
```

Server response

Code Details

200

Response body

```
{
  "message": "API is functional!"
}
```

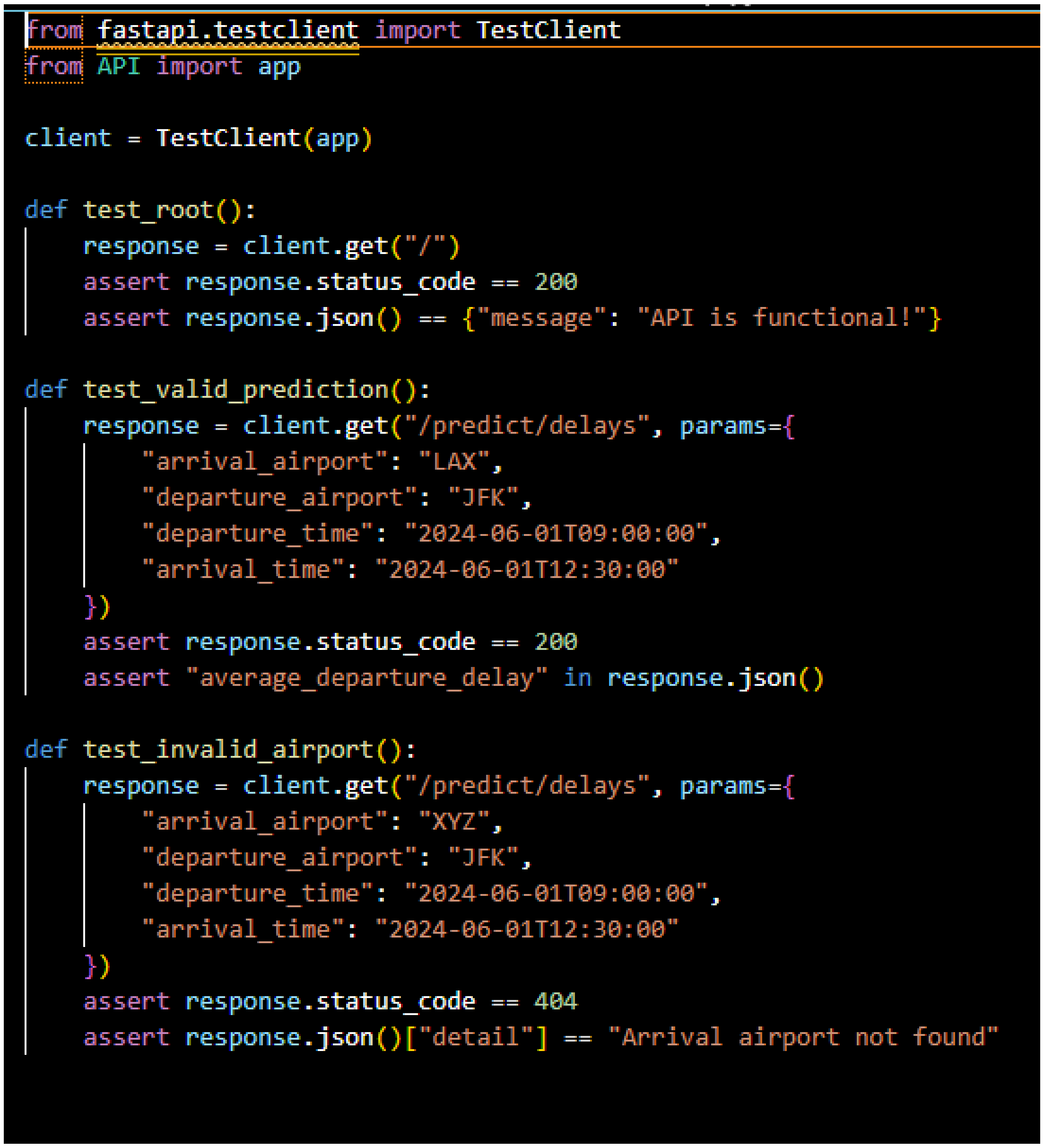
Response headers

```
 content-length: 32
 content-type: application/json
 date: Mon,26 May 2025 21:00:58 GMT
 server: uvicorn
```

Responses

C. Write at least three unit tests for your API code, using the pytest package in Python or the testthat package in R, that test features of endpoints given both correctly formatted and incorrectly formatted requests.

Below is my code that shows the responses given for correctly and incorrectly formatted requests.



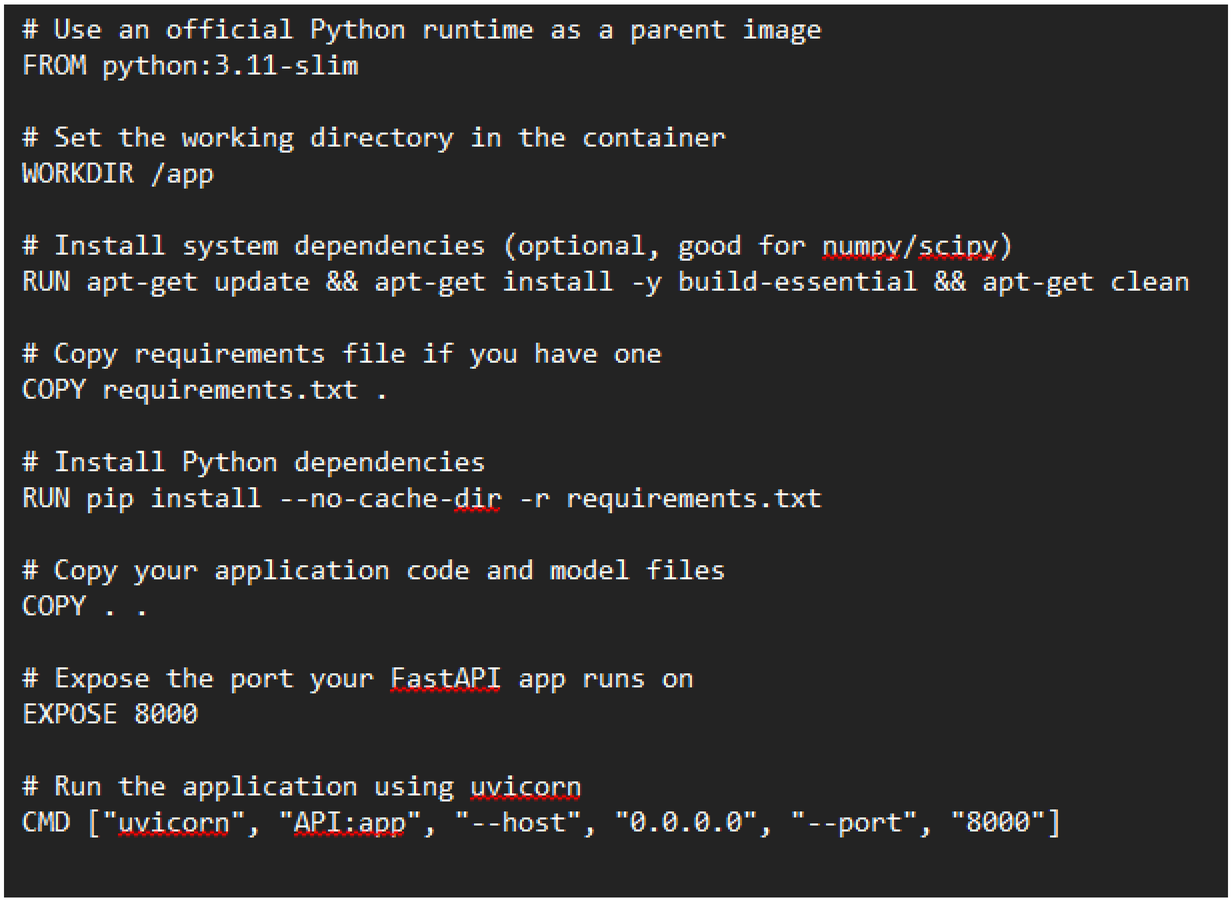
```python
from fastapi.testclient import TestClient
from API import app

client = TestClient(app)

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

def test_valid_prediction():
    response = client.get("/predict/delays", params={
        "arrival_airport": "LAX",
        "departure_airport": "JFK",
        "departure_time": "2024-06-01T09:00:00",
        "arrival_time": "2024-06-01T12:30:00"
    })
    assert response.status_code == 200
    assert "average_departure_delay" in response.json()

def test_invalid_airport():
    response = client.get("/predict/delays", params={
        "arrival_airport": "XYZ",
        "departure_airport": "JFK",
        "departure_time": "2024-06-01T09:00:00",
        "arrival_time": "2024-06-01T12:30:00"
    })
    assert response.status_code == 404
    assert response.json()["detail"] == "Arrival airport not found"
```

It shows the results of valid and invalid requests.

This part of the task wasn't too difficult as the inputs were straightforward and strict, so asserting the wrong inputs would be easy to identify and give a different response.

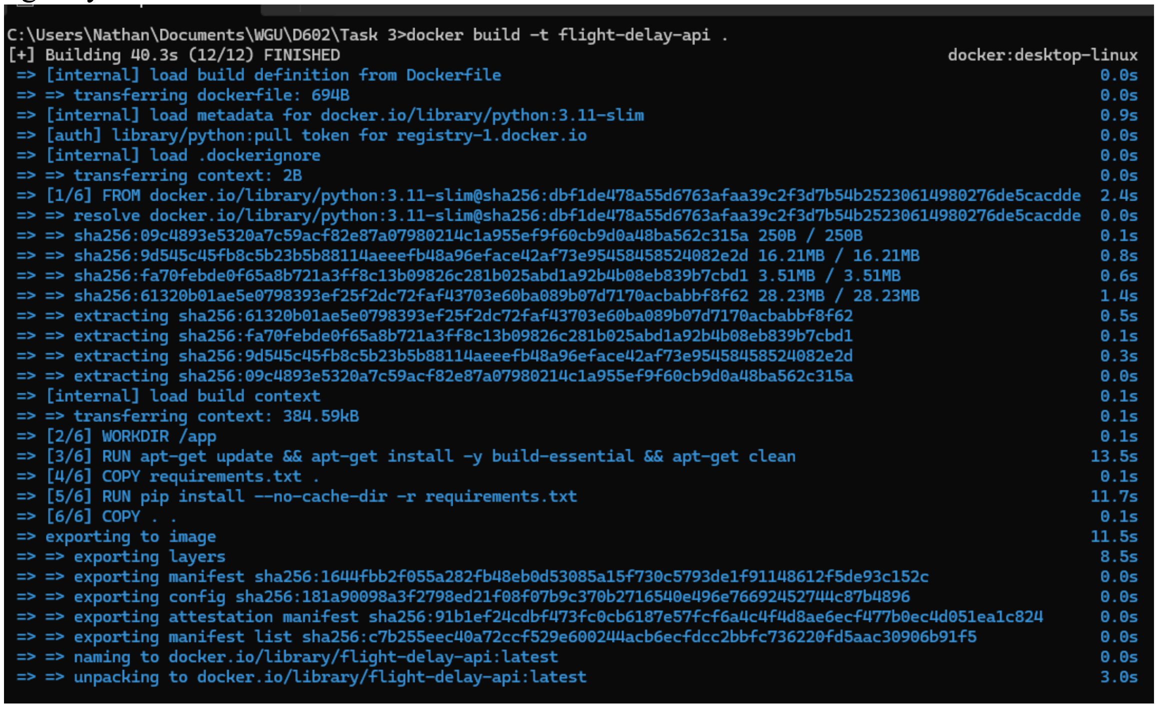
D. Write a Dockerfile referencing the requirements.txt file as appropriate that packages your API code and runs a web server to allow HTTP requests to your API.

Below is the code for my Docker file.



```
# Use an official Python runtime as a parent image
FROM python:3.11-slim

# Set the working directory in the container
WORKDIR /app

# Install system dependencies (optional, good for numpy/scipy)
RUN apt-get update && apt-get install -y build-essential && apt-get clean

# Copy requirements file if you have one
COPY requirements.txt .

# Install Python dependencies
RUN pip install --no-cache-dir -r requirements.txt

# Copy your application code and model files
COPY . .

# Expose the port your FastAPI app runs on
EXPOSE 8000

# Run the application using uvicorn
CMD ["uvicorn", "API:app", "--host", "0.0.0.0", "--port", "8000"]
```

This part was difficult for me, as configuring the Docker container properly proved challenging. I believe that when I first installed Docker, something was wrong, as I kept getting errors no matter what I tried until I uninstalled and reinstalled the software, allowing it to run through my terminal.



```
C:\Users\Nathan\Documents\WGU\D602\Task 3>docker build -t flight-delay-api .
[+] Building 40.3s (12/12) FINISHED                                                              docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                             0.0s
 => => transferring dockerfile: 694B                                                                             0.0s
 => [internal] load metadata for docker.io/library/python:3.11-slim                                              0.9s
 => [auth] library/python:pull token for registry-1.docker.io                                                    0.0s
 => [internal] load .dockerignore                                                                                0.0s
 => => transferring context: 2B                                                                                  0.0s
 => [1/6] FROM docker.io/library/python:3.11-slim@sha256:dbf1de478a55d6763afaa39c2f3d7b54b25230614980276de5cacdde  2.4s
 => => resolve docker.io/library/python:3.11-slim@sha256:dbf1de478a55d6763afaa39c2f3d7b54b25230614980276de5cacdde  0.0s
 => => sha256:09c4893e5320a7c59acf82e87a07980214c1a955ef9f60cb9d0a48ba562c315a 250B / 250B                        0.1s
 => => sha256:9d545c45fb8c5b23b5b88114aeeefb48a96eface42af73e95458458524082e2d 16.21MB / 16.21MB                  0.8s
 => => sha256:fa70febde0f65a8b721a3ff8c13b09826c281b025abd1a92b4b08eb839b7cbd1 3.51MB / 3.51MB                    0.6s
 => => sha256:61320b01ae5e0798393ef25f2dc72faf43703e60ba089b07d7170acbabbf8f62 28.23MB / 28.23MB                  1.4s
 => => extracting sha256:61320b01ae5e0798393ef25f2dc72faf43703e60ba089b07d7170acbabbf8f62                         0.5s
 => => extracting sha256:fa70febde0f65a8b721a3ff8c13b09826c281b025abd1a92b4b08eb839b7cbd1                         0.1s
 => => extracting sha256:9d545c45fb8c5b23b5b88114aeeefb48a96eface42af73e95458458524082e2d                         0.3s
 => => extracting sha256:09c4893e5320a7c59acf82e87a07980214c1a955ef9f60cb9d0a48ba562c315a                         0.0s
 => [internal] load build context                                                                                0.1s
 => => transferring context: 384.59kB                                                                            0.1s
 => [2/6] WORKDIR /app                                                                                           0.1s
 => [3/6] RUN apt-get update && apt-get install -y build-essential && apt-get clean                             13.5s
 => [4/6] COPY requirements.txt .                                                                                0.1s
 => [5/6] RUN pip install --no-cache-dir -r requirements.txt                                                    11.7s
 => [6/6] COPY . .                                                                                               0.1s
 => exporting to image                                                                                          11.5s
 => => exporting layers                                                                                          8.5s
 => => exporting manifest sha256:1644fbb2f055a282fb48eb0d53085a15f730c5793de1f91148612f5de93c152c                 0.0s
 => => exporting config sha256:181a90098a3f2798ed21f08f07b9c370b2716540e496e76692452744c87b4896                   0.0s
 => => exporting attestation manifest sha256:91b1ef24cdbf473fc0cb6187e57fcf6a4c4f4d8ae6ecf477b0ec4d051ea1c824     0.0s
 => => exporting manifest list sha256:c7b255eec40a72ccf529e600244acb6ecfdcc2bbfc736220fd5aac30906b91f5            0.0s
 => => naming to docker.io/library/flight-delay-api:latest                                                       0.0s
 => => unpacking to docker.io/library/flight-delay-api:latest                                                    3.0s
```

The Docker file also requires a requirements.txt file. This, along with my script, the airport encodings JSON file, and the finalized model pkl file, were all the pieces I needed to run the API successfully.

Sources:
No sources were used except WGU Material