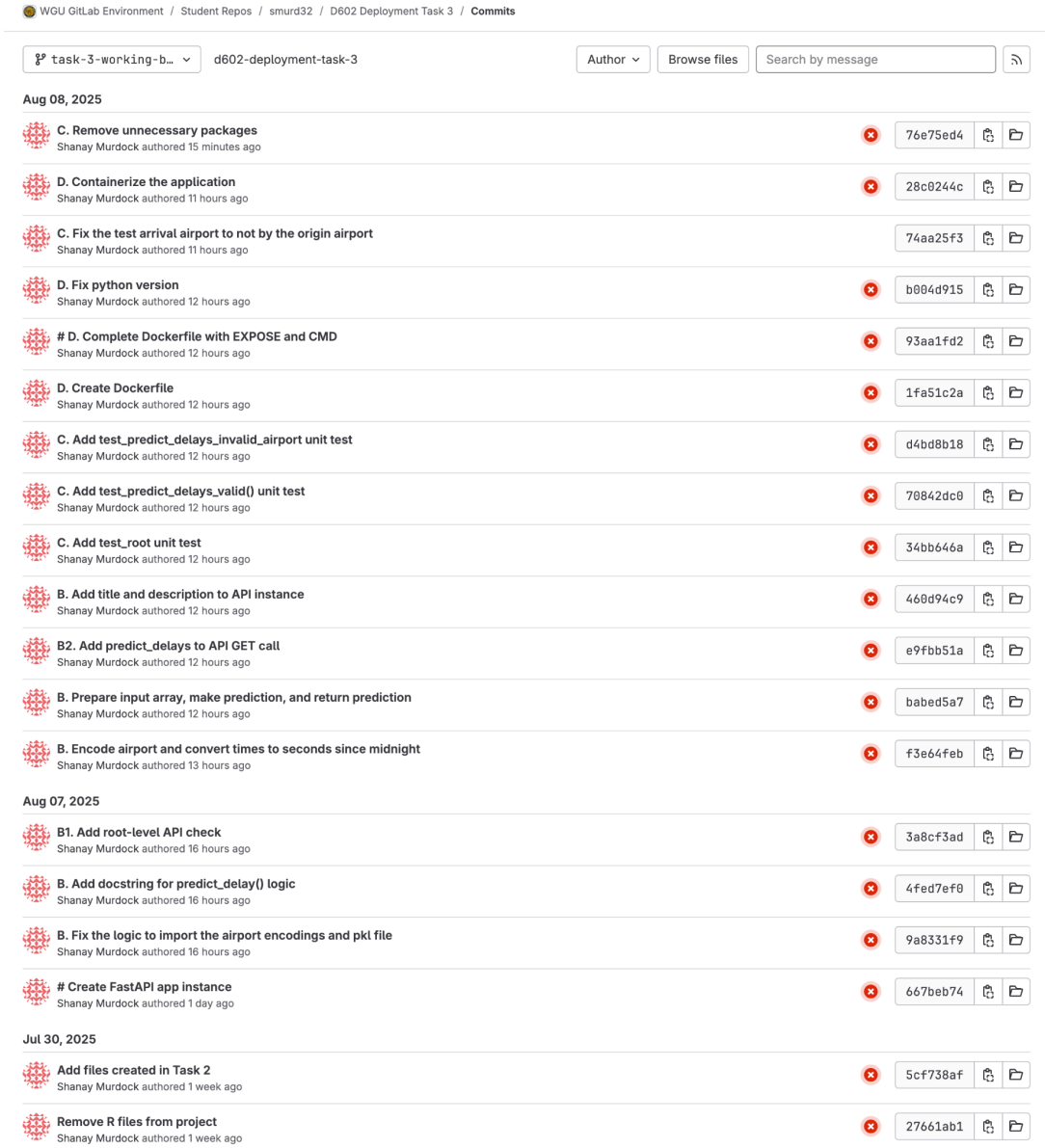
D602 - Deployment
QBN1 Task 3: Program Deployment
Prepared by Shanay Murdock

A. GitLab Repository

https://gitlab.com/wgu-gitlab-environment/student-repos/smurd32/d602-deployment-task-3/-/tree/task-3-working-branch?ref_type=heads

Screenshot of repository branch history:



B. API Endpoints

B1. API Endpoint “/”

Below is a screenshot of the logic and API endpoint (in `app.py`) that demonstrates the API is functional:

WGU GitLab Environment / Student Repos / smurd32 / D602 Deployment Task 3 / **Repository**

```python

# 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
def predict_delay(arrival_airport: str, departure_time: str, arrival_time: str):
    """
    Predict the flight delay based on the arrival airport, departure time, and arrival time.

    Parameters
    ----------
    arrival_airport : str
        The airport code for the arrival airport.
    departure_time : str
        The departure time in HH:MM format.
    arrival_time : str
        The arrival time in HH:MM format.

    Returns
    -------
    float
        The predicted flight delay in minutes.
    """
    # Get one-hot encoding for the arrival airport
    encoded_airport = create_airport_encoding(arrival_airport, airports)
    if encoded_airport is None:
        raise HTTPException(status_code=400, detail="Invalid arrival airport code")

    # Convert departure and arrival times to seconds since midnight
    try:
        dep_time = datetime.datetime.strptime(departure_time, "%H:%M")
        arr_time = datetime.datetime.strptime(arrival_time, "%H:%M")

        dep_seconds = dep_time.hour * 3600 + dep_time.minute * 60
        arr_seconds = arr_time.hour * 3600 + arr_time.minute * 60
    except ValueError:
        raise HTTPException(status_code=400, detail="Invalid time format. Use HH:MM format.")

    # Prepare the input array for the model
    input_data = np.concatenate(([1], encoded_airport, [dep_seconds], [arr_seconds]))

    # Make prediction using the model
    prediction = model.predict(input_data.reshape(1, -1))

    return float(prediction[0])  # Return the predicted delay in minutes

# TODO:  write the API endpoints.
# YOUR CODE GOES HERE
@app.get("/")
async def read_root():
    return {"message": "API is functional."}
```

B2. API Endping “/Predict/Delays”

Below is a screenshot of the API endpoint (in `app.py`) that specifies the arrival airport, the local departure time, and the local arrival time, resulting in a JSON response indicating the average departure delay in minutes:

```python

@app.get("/predict/delays")
async def predict_delays(arrival_airport: str, departure_time: str, arrival_time: str):
    """
    API endpoint to predict flight delays.

    Parameters
    ----------
    arrival_airport : str
        The airport code for the arrival airport.
    departure_time : str
        The departure time in HH:MM format.
    arrival_time : str
        The arrival time in HH:MM format.

    Returns
    -------
    dict
        JSON response containing the predicted delay in minutes.
    """
    try:
        delay = predict_delay(arrival_airport, departure_time, arrival_time)
        return {"predicted_delay": delay}
    except HTTPException as e:
        # Re-raise the HTTPException to return the error response
        raise e
    except Exception as e:
        # Handle any other exceptions and return a generic error message
        raise HTTPException(status_code=500, detail="An error occurred while processing the request.")
```

C. API Test

Below is the code (in `test_app.py`) that performs the unit tests (`test_root()`, `test_predict_delays_valid()`, and `test_predict_delays_invalid_airport()`):

test_app.py 1.16 KiB

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

from fastapi.testclient import TestClient
from app import app

client = TestClient(app)

def test_root():
    """Test root endpoint - correctly formatted request."""
    response = client.get("/")
    assert response.status_code == 200
    assert response.json() == {"message": "API is functional."}


def test_predict_delays_valid():
    """Test /predict/delays endpoint with valid input - correctly formatted request."""
    response = client.get("/predict/delays", params={
        "arrival_airport": "JFK",
        "departure_time": "14:30",
        "arrival_time": "18:30"
    })
    assert response.status_code == 200
    assert "predicted_delay" in response.json()
    assert isinstance(response.json()["predicted_delay"], float)


def test_predict_delays_invalid_airport():
    """Test /predict/delays with invalid airport code - incorrectly formatted request."""
    response = client.get("/predict/delays", params={
        "arrival_airport": "INVALID",
        "departure_time": "14:30",
        "arrival_time": "18:30"
    })
    assert response.status_code == 400
    assert response.json() == {"detail": "Invalid arrival airport code"}
```

The `test_predict_delays_valid()` test checks for correctly formatted requests, and the `test_predict_delays_invalid_airport()` test checks for invalid airport codes and provides a response to advise the user that a valid airport code was not received in the request.

Below is a screenshot of the passing unit tests:

```
● (deployment-env) (base) shanaymurdock@Mac d602-deployment-task-3-2 % pytest test_app.py
========================================================================= test session starts =========================================================================
platform darwin -- Python 3.13.5, pytest-8.4.1, pluggy-1.6.0
rootdir: /Users/shanaymurdock/WGU Repos/d602-deployment-task-3-2
plugins: anyio-4.10.0
collected 3 items

test_app.py ...                                                                                                                                                      [100%]

============================================================================= warnings summary ==============================================================================
deployment-env/lib/python3.13/site-packages/sklearn/base.py:442
  /Users/shanaymurdock/WGU Repos/d602-deployment-task-3-2/deployment-env/lib/python3.13/site-packages/sklearn/base.py:442: InconsistentVersionWarning: Trying to
 unpickle estimator Ridge from version 1.4.1.post1 when using version 1.7.1. This might lead to breaking code or invalid results. Use at your own risk. For more
 info please refer to:
  https://scikit-learn.org/stable/model_persistence.html#security-maintainability-limitations
    warnings.warn(

test_app.py::test_predict_delays_valid
  /Users/shanaymurdock/WGU Repos/d602-deployment-task-3-2/app.py:99: DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is deprecated, and wil
l error in future. Ensure you extract a single element from your array before performing this operation. (Deprecated NumPy 1.25.)
    return float(prediction[0])  # Return the predicted delay in minutes

-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html
========================================================================= 3 passed, 2 warnings in 0.65s =========================================================================
```

D. Dockerfile

Below is a screenshot of the Dockerfile used to reference the `requirements.txt` dependencies and allows the API code to run a web server to allow HTTP requests to the API:

Dockerfile 377 B

```
FROM python:3.12.1-slim

# Set the working directory in the container
WORKDIR /app

# Copy the requirements file into the container at /app
COPY . /app

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

# Expose the port the app runs on
EXPOSE 8000

# Command to run the application
CMD ["uvicorn", "app:app", "--host", "0.0.0.0", "--port", "8000"]
```

Below is an example of a successful API call to LAX with valid times:

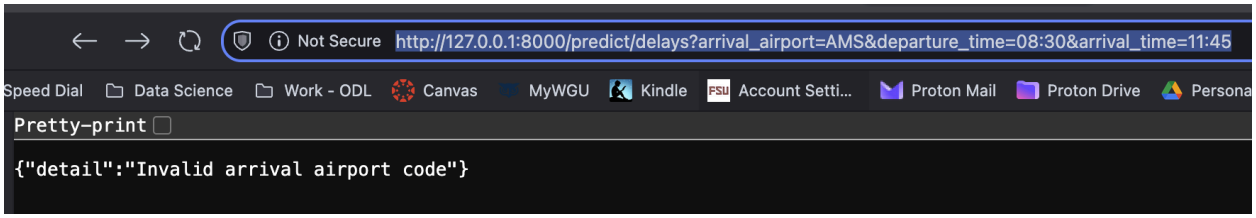
Not Secure http://127.0.0.1:8000/predict/delays?arrival_airport=LAX&departure_time=08:30&arrival_time=11:45

Speed Dial | Data Science | Work - ODL | Canvas | MyWGU | Kindle | Account Setti... | Proton Mail | Proton Drive | Personal

Pretty-print

{"predicted_delay":4.410206138209759}

Below is an example of a failed (but handled) API call to AMS (valid airport, invalid for this list):



E. Explanation

I have listed in Parts B-D the process of writing the code and testing its functionality.

Below I will address some of the challenges I faced and the solutions I found.

Challenge: Loading and using model artifacts -- when running `test_app.py`, I received errors that Scikit-Learn could not be found.

Solution: I updated the virtual environment using `pip install sckit-learn` as it no longer installs using `sklearn` like it was set up in `requirements.txt`. I ensured there was proper error handling for file loading and model initialization.

Challenge: Understanding the requirements for writing the logic in `app.py`. Nowhere in the project requirements or rubric does this get mentioned.

Solution: Use the provided `create_airport_encoding` function and instructions in the #TODO comments for one-hot encoding, convert time strings to seconds since midnight, and piece together the feature vector in the correct order.

Challenge: Receiving "Failed pipeline for <branch>" emails for every GitLab commit.

Solution: I found in the course FAQs (which the supplemental course materials themselves were hard to find and in an unintuitive spot) that this is a normal part of the project based on the `yml` file attached to the pipeline.

Challenge: Test design strategy for both valid and invalid scenarios.

Solution: I wrote three tests covering valid predictions with known airport codes, invalid airport codes (or valid codes but unknown to this airport), all needing to recognize proper time formats.

Challenge: Part F: Provide a video demonstrating the live API running from a deployed Docker container. There was no link provided for use of Panopto in the Task or course.

Solution: Opened D603 to access Panopto to allow for recording.

F. Panopto Demonstration

https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=6eaebde3-f41c-4311-9c15-b33300fd842d

G. Sources

WGU Course Materials