**Task 3: Program Deployment**

Joseph Emmanuel Cayetano

Western Governors University

D602: Deployment Task 3

December 14, 2024

1. **Create your subgroup and project in GitLab**

**GitLab repository URL:** <https://gitlab.com/wgu-gitlab-environment/student-repos/jcayet5/d602-deployment-task-3/-/tree/Task3Branch?ref_type=heads>

**Repository branch history screenshot:**

A screenshot of a computer

Description automatically generated

1. **Write an API with the code templates provided in the FastAPI package in Python that accepts the following HTTP endpoints**

**B1. “/” should return a JSON message indicating that the API is functional**

A screenshot of a computer

Description automatically generated

This code is from app.py (modified API\_Python\_1.0.0.py)

**B2. “/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**

A screen shot of a computer code

Description automatically generated

This code is from app.py (modified API\_Python\_1.0.0.py)

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

A screenshot of a computer code

Description automatically generated

This code is from test\_app.py. The three unit tests are test\_root, test\_valid\_prediction, and test\_invalid\_arrival\_airport. The test\_valid\_prediction contains the correctly formatted request, while the test\_invalid\_arrival\_airport contains the incorrectly formatted request.

A screen shot of a computer

Description automatically generated

This image confirms that all three unit tests from test\_app.py have passed.

1. **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**

A screenshot of a computer program

Description automatically generated

This is the Dockerfile

1. **Provide an explanation of how you wrote your code, including any challenges you encountered and how you addressed those challenges**

In app.py (API\_Python\_1.0.0.py), I started by loading the finalized\_model.pkl file using the load() function from the pickle library. The pkl file is the trained machine learning model from Task 2, and it predicts flight departure delays. Next, I created two API endpoints. The root endpoint (“/”) outputs a JSON response that says the API is functional. By receiving this response, it confirms the API is up and running. The prediction endpoint (“predict/delays”) predicts the average departure delay in minutes. It accepts three parameters, which are the arrival airport code, local departure time, and local arrival time. In this endpoint, I began by transforming the arrival airport code into a one-hot encoded format through the create\_airport\_encoding() function. I added a validation check to ensure the arrival airport code exists in the encoding map. If it doesn’t, the function returns None and raises the 400 Bad Request error with a message that says, “Invalid arrival airport.” Next, both departure and arrival times are converted from the HH:MM format into seconds since midnight. After the conversion, I used the NumPy array() function to create an input array consisting of the polynomial order, the encoded arrival airport array, the departure time in seconds, and the arrival time in seconds. The array then gets used by the model to predict the average departure delay in minutes, and outputs the prediction in JSON format. If there is any unexpected issue within the program, the 500 Internal Server error is raised.

The test\_app.py contains three unit tests for the FastAPI application that was created in app.py. I started by importing the library TestClient from FastAPI and the application from app.py. Then, I initialized a TestClient instance to send HTTP requests to the application. The first unit test is called “test\_root,” and it sends a GET request to the root endpoint (“/”). This test ensures that the root endpoint is working and returns a 200 OK status code with a JSON response that says, “API is functional.” The second unit test is called “test\_valid\_prediction,” and it sends a GET request to the /predict/delays endpoint with valid input parameters (arrival\_airport, departure\_time, arrival\_time). This test ensures that the /predict/delays endpoint is working when given valid inputs and returns a 200 OK status code with a JSON response indicating the average departure delay in minutes. The third unit test is called “test\_invalid\_arrival\_airport,” and it sends a GET request to the /predict/delays endpoint with an invalid arrival airport code. This test ensures that the prediction endpoint correctly handles invalid arrival airport inputs by returning a 400 Bad Request error with a JSON response that says, “Invalid arrival airport.”

I created a Dockerfile that creates a docker image for the FastAPI application from app.py. I used the lightweight version of Python 3.11 as the base image for the Docker container. Next, I set the working directory inside the container to /app and copied the application code (app.py) and configuration files from the current host directory into the /app directory in the container. Then, I installed the Python dependencies listed in requirements.txt by adding the line “RUN pip install --no-cache-dir -r requirements.txt” inside the Dockerfile. I’ve also included the line “EXPOSE 8000” to inform docker that the container listens on port 8000 at runtime. Finally, I added the line “CMD ["uvicorn", "app:app", "--host", "0.0.0.0", "--port", "8000"]” to set the default command to run when the container starts. Basically, the command uses Uvicorn to run the FastAPI application from app.py and runs the application on port 8000.

There are two challenges I’ve encountered while completing this task. The first challenge was running the FastAPI application from app.py using Uvicorn. To define Uvicorn, it is a “lightning-fast ASGI server implementation, perfect for running FastAPI applications” (Lu, 2024, par. 1). For some reason, I couldn’t run the application inside a Conda environment, as doing so would give me the “your app cannot run on your PC” error message. I even created a new Conda environment, and it would still show the same error. Through research, I discovered that this could be due to a conflict with the Uvicorn executable or the Python installation inside the Conda environment was corrupted or improperly set up. To solve this issue, I created a virtual environment instead and ran the FastAPI application with Uvicorn from there. The application ran without any errors.

The second challenge was handling FastAPI’s HTTPException within app.py. To define HTTPException, it is a class that “can be used to raise HTTP errors with custom status codes and detail messages” (Nikenoueba, 2024, par. 9). I had an issue where the HTTPException with a 400 Error for an invalid airport code was raised, but it was mistakenly caught by the generic except block. This block then re-raised it as a new HTTPException with a 500 Internal Server Error, hiding the original 400 error. To fix this issue, I added the line “except HTTPException as e:” before the generic except block to explicitly re-raise the HTTPException for FastAPI to handle. This prevents the broad except block from re-mapping HTTPException. As a result, the program now works as expected.

1. **Provide a video demonstrating the live API running from a deployed Docker container. You must issue at least 1 well-formatted request and 1 ill-formatted request and demonstrate that the API responds appropriately**

Panopto video: https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=5184cccf-5390-491c-8066-b2470128b31b

**References**

Lu, H. (2024, March 20). *FASTAPI with Uvicorn: A comprehensive tutorial*. Orchestra. <https://www.getorchestra.io/guides/fastapi-with-uvicorn-a-comprehensive-tutorial>

Nikenoueba, J.-S. (2024, September 5). *Error and exception handling in fastapi*. Medium. https://blog.stackademic.com/error-and-exception-handling-in-fastapi-c0949bb42e1b