Development Process

The API was implemented using **FastAPI**, a Python framework that allows for easy creation of web APIs. The goal was to provide endpoints that accept airport codes and flight times, then return the predicted average departure delay.

Key Steps in Development

1. **Defining the API Structure**
    - The root endpoint (`"/"`) simply returns a message confirming the API is functional.
    - The `/predict/delays` endpoint accepts **GET** requests with parameters for the arrival airport, departure airport, departure time, and arrival time.
    - The API uses **Pydantic** for request validation to ensure correct input format.
2. **Loading Essential Data**
    - The API loads a JSON file (`airport_encodings.json`) to map airport codes to one-hot encoded vectors.
    - A pre-trained model (`finalized_model.pkl`) is loaded using `pickle` to generate predictions.
3. **Data Processing & Validation**
    - Airport codes are validated to ensure they are three-letter IATA codes.
    - Time inputs are converted to **seconds since midnight** for numerical consistency.
    - A helper function `create_airport_encoding()` converts the airport code into a **one-hot encoded NumPy array**, ensuring proper feature representation for the model.
4. **Prediction Logic**
    - The input data is formatted into a NumPy array consisting of:
        - A constant polynomial order value (1).
        - The encoded airport vector.
        - Departure and arrival times as seconds since midnight.
    - This array is passed into the machine learning model to generate a prediction.

d602-deployment-task-3 > Task3-C-Part1.py > ...

```python
import pytest
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_predict_delays_correct_format():
    response = client.get("/predict/delays", params={
        "arrival_airport": "JFK",
        "departure_airport": "LAX",
        "departure_time": "2025-03-03T08:00:00",
        "arrival_time": "2025-03-03T16:00:00"
    })
    assert response.status_code == 200
    assert "average_departure_delay" in response.json()

def test_predict_delays_incorrect_format():
    response = client.get("/predict/delays", params={
        "arrival_airport": "JFK",
        "departure_airport": "LAX",
        "departure_time": "invalid-time-format",
        "arrival_time": "2025-03-03T16:00:00"
    })
    assert response.status_code == 400
    assert response.json() == {"detail": "Invalid time format. Please use 'YYYY-MM-DDTHH:MM:SS'."}

def test_predict_delays_missing_airport():
    response = client.get("/predict/delays", params={
        "arrival_airport": "INVALID",
        "departure_airport": "LAX",
        "departure_time": "2025-03-03T08:00:00",
        "arrival_time": "2025-03-03T16:00:00"
    })
    assert response.status_code == 404
    assert response.json() == {"detail": "Arrival airport not found"}
```

Challenges & Solutions

Challenge	Solution
Ensuring valid airport codes	Implemented validation to check if the provided airport exists in the airport encoding dictionary. If not found, the API returns an HTTP 404 error.
Handling incorrect time formats	Used `datetime.strptime()` with error handling to validate time input format. If incorrect, an HTTP 400 error is raised.
Ensuring compatibility with the model	The model requires a **2D NumPy array** as input. To prevent shape mismatches, `.reshape(1, -1)` was used before passing the array to the model.
Ensuring API robustness	Added exception handling using FastAPI's `HTTPException` to provide clear error messages for invalid inputs.

d602-deployment-task-3 > Task3-D-Part2 > FROM

```dockerfile
FROM python:3.12.8-slim

# Set working directory
WORKDIR C:\Users\kyleh\OneDrive\Desktop\D602\Task 3\d602-deployment-task-3

# Copy requirements and install dependencies
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Copy application code
COPY . .

# Expose port
EXPOSE 8000

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

Unit Testing

The API was tested using **pytest** and **FastAPI's TestClient**. The test cases included:

1. **Basic API functionality** – Ensuring `"/"` returns a valid response.
2. **Valid predictions** – Checking that the model returns a delay value when given correct inputs.
3. **Incorrectly formatted requests** – Testing:
    - Invalid time formats
    - Missing or incorrect airport codes
4. **Additional edge cases** – Predicting delays for **past and future dates** and when **departure and arrival airports are the same**.

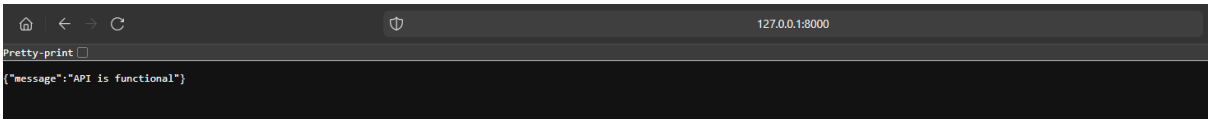
Dockerization

To containerize the API:

1. A **Dockerfile** was created that:
    - Installs dependencies from `requirements.txt`
    - Exposes the necessary ports
    - Runs FastAPI using **uvicorn**
2. The API was tested within a **Docker container** to ensure it functioned consistently across environments.

d602-deployment-task-3 > Dockerfile > FROM

```dockerfile
FROM python:3.12.8-slim

# Set working directory
WORKDIR /app

# Copy requirements and install dependencies
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Copy application code
COPY . .

# Expose port
EXPOSE 8000

# Run the app
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000", "--reload"]
```



Progression of Work

- **Initial version**: Implemented the API structure with basic validation.
- **Second version**: Improved input validation, added exception handling, and ensured the model input format was correct.
- **Final version**: Completed unit tests and added further robustness checks.