

Module 10: Building an Automated CI/CD Pipeline

JHU EP 605.206 - Introduction to Programming Using Python

Introduction

In Module 7 we used the WMATA's publicly available API's to create maps to help disabled patrons navigate the metro system more effectively. However, the WMATA Incidents API had a small limitation. Namely, we could only get a single list of incidents and not request them by unit type (e.g. only escalator incidents or only elevator incidents). In this Assignment we will expand upon the Module 7 Assignment where you were asked to design, but not implement, an API to fix this issue. You will create a wrapper API around the WMATA Incidents API to allow consumers of your microservice to be able to filter their results based on the unit type. Next you'll create a small series of unit tests to verify the functionality of your code. Optionally, you'll have the opportunity to create a CI/CD pipeline using CircleCI to run this series of tests automatically upon a commit to your Git repository.

Skills: REST API's using Flask, version control using Git, unit tests using inheritance, CI/CD pipelines

Programming Assignment Part 1 – Publishing a REST API Using FLASK

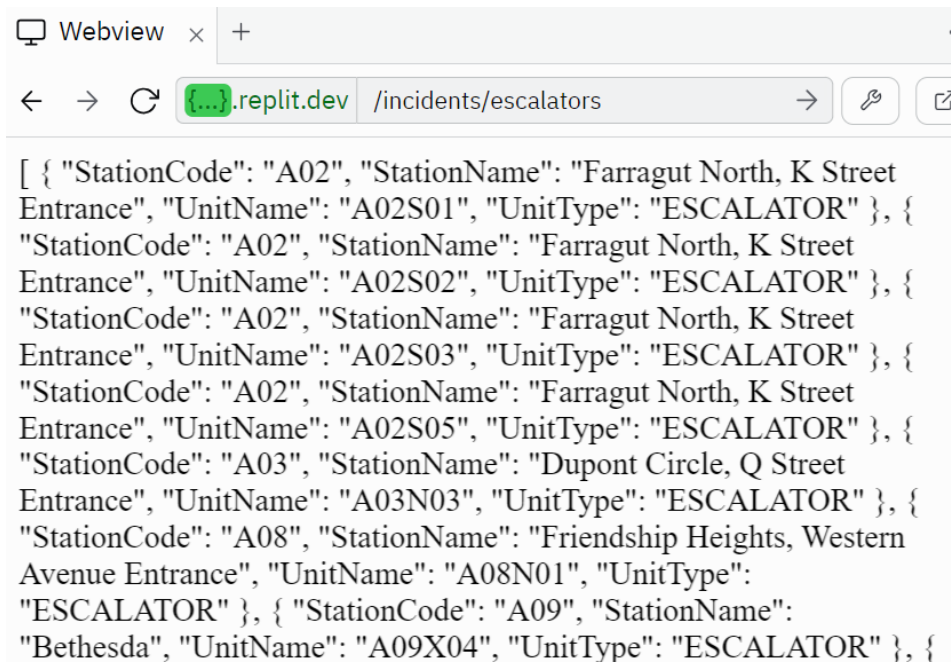
In Module 7 we designed an API endpoint using the OpenAPI specification to retrieve incidents based on their type: either only escalators or only elevators. Recall that this endpoint accepted a path parameter called 'machine_type' and returned an array (list) of 'Machine' objects, with each Machine object (dictionary) containing 4 field: StationCode, StationName, UnitName, and UnitType. In this part of the Assignment we're going to implement that endpoint using Flask.

Begin by downloading **wmata_api.py** from Canvas. Familiarize yourself with this code before writing your solution to get a feel for how it works. We've done all of the Flask setup for you, but pay special attention to lines 16-17 as this is what makes your endpoint work! The key thing to note is that the endpoint accepts a path parameter named 'unit_type', which will be either "escalators" or "elevators", and this is passed into the 'get_incidents' function as a parameter.

Requirements:

1. Install/import any necessary libraries: **json, requests, flask**
2. Configure your WMATA API key from Module 7
3. Complete the code within the 'get_incidents' function:
 - a. This function should:
 - i. Accept "**unit_type**" as a parameter (will be either "elevators" or "escalators")
 - ii. Call the WMATA Incidents API and convert the response to JSON
 - iii. Create a list of dictionaries (objects) of the specified unit_type
 1. Each dictionary should conform to the API schema from Module 7
 - a. StationCode, StationName, UnitName, UnitType
 - iv. Return the list of dictionaries to the endpoint caller as a JSON string
4. Run (expose) your API endpoint by executing **wmata_api.py**:
 - a. If you're using REPL you have two options to interact with your endpoint:
 - i. Access the API URL directly in the Webview tab
 - ii. Copy the dev URL from the Websview address bar and paste it into a browser
 - iii. Whichever option you chose you will then:
 1. Add the API endpoint path: /incidents
 - a. Add the "elevators" path parameter: /elevators
 - i. Capture a screenshot of the returned response
 - b. Add the "escalators" path parameter: /escalators
 - i. Capture a screenshot of the returned response
 - b. If you're using VS Code/PyCharm you can access your endpoint by:
 - i. Using the base URL which should be your loopback IP address: 127.0.0.1
 - ii. Using a browser or Postman:
 1. Add the API endpoint path: /incidents
 - a. Add the "elevators" path parameter
 - i. Capture a screenshot of the returned response
 - b. Add the "escalators" path parameter
 - i. Capture a screenshot of the returned response

Part 1 Sample Outputs



A screenshot of a web browser window with a single tab titled 'Webview'. The address bar shows the URL '...replit.dev /incidents/escalators'. The main content area displays a JSON array of objects, each representing an escalator unit. The objects contain fields for 'StationCode', 'StationName', 'UnitName', and 'UnitType'.

```
[ { "StationCode": "A02", "StationName": "Farragut North, K Street Entrance", "UnitName": "A02S01", "UnitType": "ESCALATOR" }, { "StationCode": "A02", "StationName": "Farragut North, K Street Entrance", "UnitName": "A02S02", "UnitType": "ESCALATOR" }, { "StationCode": "A02", "StationName": "Farragut North, K Street Entrance", "UnitName": "A02S03", "UnitType": "ESCALATOR" }, { "StationCode": "A02", "StationName": "Farragut North, K Street Entrance", "UnitName": "A02S05", "UnitType": "ESCALATOR" }, { "StationCode": "A03", "StationName": "Dupont Circle, Q Street Entrance", "UnitName": "A03N03", "UnitType": "ESCALATOR" }, { "StationCode": "A08", "StationName": "Friendship Heights, Western Avenue Entrance", "UnitName": "A08N01", "UnitType": "ESCALATOR" }, { "StationCode": "A09", "StationName": "Bethesda", "UnitName": "A09X04", "UnitType": "ESCALATOR" }, { }
```



A screenshot of a web browser window with a single tab titled 'Webview'. The address bar shows the URL '...replit.dev /incidents/elevators'. The main content area displays a JSON array of objects, each representing an elevator unit. The objects contain fields for 'StationCode', 'StationName', 'UnitName', and 'UnitType'.

```
[ { "StationCode": "B01", "StationName": "Gallery Pl-Chinatown, 7th and G Street/Arena Entrance", "UnitName": "B01E02", "UnitType": "ELEVATOR" }, { "StationCode": "C05", "StationName": "Rosslyn", "UnitName": "C05E03", "UnitType": "ELEVATOR" }, { "StationCode": "D09", "StationName": "Minnesota Ave", "UnitName": "D09X03", "UnitType": "ELEVATOR" }, { "StationCode": "D09", "StationName": "Minnesota Ave", "UnitName": "D09X04", "UnitType": "ELEVATOR" }, { "StationCode": "E08", "StationName": "Prince George's Plaza", "UnitName": "E08X04", "UnitType": "ELEVATOR" }, { "StationCode": "E09", "StationName": "College Park-U of MD", "UnitName": "E09X04", "UnitType": "ELEVATOR" }, { "StationCode": "E09", "StationName": "College Park-U of MD", "UnitName": "E09X05", "UnitType": "ELEVATOR" } ]
```

Programming Assignment Part 2 – Creating Unit Tests for Manual Execution

For Part 2 of this Assignment you'll create a series of unit tests to verify the functionality of your endpoint.

Download **test_wmata_api.py** from Canvas. It contains a class called **WMATATest** that is derived from the `unittest.TestCase` class and contains skeleton code for 4 unit test cases:

You will fill-in the test code for the 4 unit tests below:

1. **test_http_success**: this unit test uses the Flask `test_client()` to make an HTTP get request to the `/incidents/escalators` and `/incidents/elevators` endpoints and verify that they both return a 200 response code.
2. **test_required_fields**: this unit test will verify that the 4 required fields are contained in each incident returned by your endpoint. The 4 required fields are: `StationCode`, `StationName`, `UnitName`, and `UnitType`.
3. **test_escalators**: this unit test will ensure that all incidents returned from your `/incidents/escalators` endpoint have a `UnitType` equal to "ESCALATOR".
4. **test_elevators**: this unit test will ensure that all incidents returned from your `/incidents/escalators` endpoint have a `UnitType` equal to "ELEVATOR".

Once you've completed the 4 unit tests you will execute them manually by:

1. Running `python3 -m unittest` in your terminal
 - a. `unittest` finds test cases based on the "test_" prefix in the module name
2. Executing `test_wmata_api.py` directly
 - a. `unittest.main()` executes the test cases for you directly

Capture a screenshot of the successfully passing tests for submission later. Examples of passing tests in both REPL and PyCharm can be found in the Part 2 Sample Outputs section on the next page.

Part 2 Sample Output

REPL

```
~/WMATATestAPI$ python3 -m unittest
....
-----
Ran 4 tests in 1.106s

OK
```

PyCharm

```
✓ Tests passed: 4 of 4 tests – 3 sec 339 ms
C:\Users\jkovba\AppData\Local\Programs\Python\Python39\python.exe
Testing started at 4:04 PM ...
Launching unittests with arguments python -m unittest C:\Users\jko

Process finished with exit code 0

Ran 4 tests in 3.346s
```

Part 3 – Automating Unit Testing with a CI/CD Pipeline (Optional)

Important Note: Since this is the first time running this Assignment we're looking for your help! **Any student who makes an honest attempt to complete this part of the Assignment will receive 50% credit back on their lowest Assignment.** So, for example, if you received a 40% on an Assignment, you will receive half of the 60% credit lost back increasing the 40% to a 70%. This will be applied to your lowest grade from Module 1-8 and cannot result in a grade above 100%. To be eligible for credit you must make an attempt to complete Part 3 and answer the following questions:

1. Were you able to complete Part 3 successfully?
 - a. If so, how much total time did it take?
2. Did you find the provided instructions clear enough to complete Part 3?
 - a. If not, which areas can be improved and how specifically?

For Part 3, the final (optional) part of the Assignment, you will create an automated CI/CD pipeline to deploy and test your code automatically upon every code commit to your GitHub repository.

1. Create a free [GitHub](#) and [CircleCI](#) account
2. Install [Git](#) on your local machine
 - a. You can skip this step if you're using [REPL's GitHub integration](#)
3. [Install](#) and [configure](#) GitHub Desktop on your local machine (Recommended)
 - a. You can technically skip this step if you're using [REPL GitHub Integration](#) or the PyCharm/VS Code terminal but this will help avoid a lot of hassle with generating keys
4. [Create a new empty repository](#) inside of your GitHub account named "module10"
5. [Clone](#) your empty repo from Step 4 using GitHub Desktop
6. Add the following files/directories to the cloned repository:
 - a. Files
 - i. `wmata_api.py`
 - ii. `test_wmata_api.py`
 - iii. `requirements.txt` (downloaded from Canvas)
 - b. A directory named `.circleci` containing `config.yml` (downloaded from Canvas)
7. Add, commit, and push the files/directory from your local to your GitHub repository
 - a. Take a screenshot of your GitHub repository after the files/directory have been pushed
 - i. See Part 3 Sample Output(s) section below for an example
8. [Connect your GitHub repository to CircleCI](#) by using the instructions under "Configuring CircleCI"
 - a. Name your CircleCI project "WMATA API CICD Pipeline"
9. Commit a change to your "module10" GitHub repository to initiate a CircleCI build
 - a. CircleCI will build a Docker container, install Python and the required libraries from `requirements.txt`, check your code out from GitHub and deploy it into the container, and execute the unit tests and report their status as shown below. Please capture a screenshot of your successful CI/CD pipeline to be submitted later.
 - i. See Part 3 Sample Output(s) section below for an example

Part 3 Sample Output(s)

GitHub Repository

main ▾

1 Branch

0 Tags

Add file ▾

Code ▾

jkovba Updates

c814e19 · now

66 Commits

.circleci	Update config.yml	17 minutes ago
requirements.txt	Update requirements.txt	last month
test_wmata_api.py	Update test_wmata_api.py	1 hour ago
wmata_api.py	Update wmata_api.py	47 minutes ago

CircleCI CI/CD Pipeline

WMATA API
CICD Pipeline 1

Success

workflow

GitHub: [main](#)
6b37c15 Update
config.yml
Triggered by: jkovba

28m ago

22s

Deliverables

readme.txt

So-called “read me” files are a common way for developers to leave high-level notes about their applications. Here’s an example of a [README file](#) for the Apache Spark project. They usually contain details about required software versions, installation instructions, contact information, etc. For our purposes, your readme.txt file will be a way for you to describe the approach you took to complete the assignment so that, in the event you may not quite get your solution working correctly, we can still award credit based on what you were trying to do. Think of it as the verbalization of what your code does (or is supposed to do). Your readme.txt file should contain the following:

1. **Name:** Your name and JHED ID
2. **Module Info:** The Module name/number along with the title of the assignment and its due date
3. **Approach:** a detailed description of the approach you implemented to solving the assignment. Be as specific as possible. If you are sorting a list of 2D points in a plane, describe the class you used to represent a point, the data structures you used to store them, and the algorithm you used to sort them, for example. The more descriptive you are, the more credit we can award in the event your solution doesn’t fully work.
4. **Known Bugs:** describe the areas, if any, where your code has any known bugs. If you’re asked to write a function to do a computation but you know your function returns an incorrect result, this should be noted here. Please also state how you would go about fixing the bug. If your code produces results correctly you do not have to include this section.

Please submit your ***wmata_api.py*** and ***test_wmata_api.py*** source code files along with a Word/PDF file named JHEDID_mod10.pdf (ex: jkovba1_mod10.pdf) containing the screenshots of your outputs. Please do not ZIP your files together.

Recap:

1. *wmata_api.py*
2. *test_wmata_api.py*
3. A Word or PDF file named ‘JHEDID_mod10’ (ex: jkovba1_mod10.pdf/.doc) containing:
 - a. Two screenshots from *wmata_api.py*:
 - i. A screenshot of the output from */incidents/escalators*
 - ii. A screenshot of the output from */incidents/elevators*
 - b. One screenshot of your test results from *test_wmata_api.py*
 - c. **(Optional)** Two from the GitHub + CircleCI integration:
 - i. A screenshot of your populated GitHub repository
 - ii. A screenshot of your executed CircleCI CI/CD pipeline

Please let us know if you have any questions via Teams or email!

Grading

We wanted to provide a few notes on grading to help you be as successful as possible on the assignments:

1. Following instructions closely is an important part of good software development practices. Even small details like a file in the wrong format can negatively impact software interoperability. For this reason, **we ask that you please follow the instructions as closely as possible. This includes matching the output format of the Sample Solutions exactly, submitting the required files in the correct formats, etc.**
2. The Readings + Video Lectures provide 90%+ of the skills necessary to complete this Assignment. While we encourage experimentation, we ask that you focus on the use of what we learned in this Module, and previous Modules, to develop your solution. **If you'd like to try other techniques, we ask that you please practice them separately and do not submit them as your assignment solution.**
3. Because the Readings + Video Lectures constitute a large majority of skills required to complete this assignment, we ask that you use external references as judiciously as possible. That is, **you may refer to external resources as needed, but such resources should be (informally) cited in your README and code borrowed from these resources must not be copy/pasted.**
4. **Unless otherwise specified, you do not need to worry about any error-checking or input sanitization.** We'll learn techniques for detecting and correcting errors later in the course and explicitly state when it's required.
5. **The output of your solution should match that of the Sample Outputs exactly.** It is important not just to have a functional solution but also one that adheres to the exact formatting specifications. For example, if another piece of code was expecting your results in the specified format but you added an extra word, or even an extra space, it could potentially break that code. Producing the proper output is also a good measure of your understanding in how to manipulate and display information that your code produces.

Note: the format of the output is different than how the formatted output is displayed on the screen. Depending on your screen size, the width of your IDE window, etc., the positioning/wrapping of your output text may appear slightly differently, though still in the same format, as ours.