Ryerson University

Department of Electrical, Computer, & Biomedical Engineering
Faculty of Engineering & Architectural Science

Course Title:	Distributed and Cloud Computing
Course Number:	COE892
Semester/Year (e.g.F2016)	W2023

Instructor:	Dr. Muhammad Jaseemuddin

Assignment/Lab Number:	4-5
Assignment/Lab Title:	FastAPI and Containers

Submission Date:	Wednesday April 12th 2023
Due Date:	Wednesday April 12th 2023

Student LAST Name	Student FIRST Name	Student Number	Section	Signature*
Faisal	Mahir	500896206	01	MahirF.

*By signing above you attest that you have contributed to this written lab report and confirm that all work you have contributed to this lab report is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a “0” on the work, an “F” in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct, which can be found online at: http://www.ryerson.ca/senate/current/pol60.pdf

Introduction

The primary objective of this lab assignment is to use Fast API services to develop a web application that handles the dispatch of land mine detection robots that must traverse a given land space. This procedure consists of the three parts. The first part involves programming the Fast API backend service for the application. The second part involves developing the operator program whose purpose is to directly contact the server in order to fetch and display requests on to the terminal window. The final part involves deploying the server backend component of the application to the Microsoft Azure cloud service.

Part 1 – Server Backend Program

1. the first step that I took in the back-end's development process was that I created two base model classes that are going to be used to create rover objects and mine bomb objects. I then proceeded to define two global list variables. One that stores rover class objects and another that stores mine class objects.

2. Next I started to create all the methods for each of the endpoint routes that are listed in the lab manual starting with the map.

3. For the map's endpoints, I reused the map.txt file from lab 1 and reused the map list creation method code that I originally had for lab 1, for lab 4. If the map.txt file is not found in the project directory, the fast API method will return the appropriate error codes for this case (i.e., 404 not found). If the file does exist, the request response will be a default 10 by 10 matrix of the land space data given that the map has not been updated by the user yet, or the updated map, if the user has requested to update the map prior to viewing it.

4. When calling the update request, the user must specify the new height and width of the map and then upon doing so, the update request service, copies the old map array data into a new array with the newly specified map dimensions and either shrinks or expands the original array based off of the dimensions provided as input. If the user tries to input dimensions that are less than 0, then the request method will appropriately throw an error saying that it is unable to update the map properly. This is to be expected as height and width obviously cannot be negative.

5. For the mines endpoints, for the get method, it returns the list of mine class objects. In this response, the API service sends back a list of items which consist of the xy coordinate and the serial number for each respective mine object. For the put method, we are using this request call to update a mine object in the list. First, we specify the id number, then we specify the parameter that we wish to update. Then if the parameter specified is a valid parameter or the new value entered is a valid value that the object's parameter can be updated to, then then the request will return all the details regarding the newly updated mine object. Otherwise, it will respond with a 400 bad request error if the parameter name or value is invalid or a 404 not found error if the request is trying to call a mine object that does not exist. I also coded another get method that returns a specific mine object along with its details. If the mine object does not exist, it returns a list of error codes. Next, I created another endpoint method which deletes a mine object based on its id number and if the id number does not exist, then it returns an error code. If deletion is successful, a success code is returned. Lastly, I created one final endpoint method for the mines which consists of creating and adding a new mine object to the global mine list. If the mine object already exists or if the user specifies invalid xy coordinates for the mine during the creation

process, then it appropriately returns an error. Otherwise, if mine creation is successful, it returns the id number of the mine object.

6. Next, we have the rover endpoint methods. Starting with the rover's get method, it simply returns the list of all of the rovers in the global rover list. The post method is used to create a rover and the user has to specify the initial list of commands that the user wants the rover to have as a string. If the rover already exists, it appropriately returns an error. The put method adds commands to the pre-existing list of commands for the rover. Lastly, the delete command is used to delete a rover from the rover object list. If the rover does not exist, it returns a 404 error otherwise it returns a success code. The final rover endpoint that I added for the assignment was a post request whose purpose is to dispatch rover based off of its id number. If the id number exists then the rover is dispatched, if not it returns an error. If the rover is dispatched, it returns the rover id, the current position (outputted in xy concatenated form [03 for example means x coordinate 0 and y coordinate 3]), the rover's status, and the list of executed commands. This endpoint method also reuses the map traversal, direction change and mine disarming functions that I developed in lab 1 to carefully traverse the land map and sequentially execute all of the rover's specified commands Realistically, I also added an additional check where if the rover is still moving, or it has been destroyed by a mine bomb, it cannot be dispatched as the rover either hasn't finished executing all of its commands yet and is thus still moving on the land space, or if that particular rover has been destroyed by a bomb, obviously it can't be dispatched as it is not operational. The main reason why I did this stems from the fact that if the rover has been eliminated or is still moving, you are unable to add more commands to it as well. Otherwise, if the rover does not exist it returns an error.

Part 2 – Operator Client

1. First I created two global list variables. One is used to store the default map.txt and the other is used to store the updated map data if the user chooses to update the map.

2. Then I created a global boolean variable to verify if the map has been updated by the server back-end or not.

3. Next I created a fancy command-menu-like user-friendly design on the terminal which lists all the possible options that the user can pick from. If user enters a number that is less than 1 or greater than/equal to 14, the application exits because either the input is invalid, or the user is entering the operator's application exit command.

4. Next a while loop is created which houses input possibilities for every input listed in the command menu. For example, if the user inputs 1 they can view the default map data, if the user inputs 2 they can update the map data based off of the newly specified dimensional parameters and so on (view lab4Operator.py file for more details on command options). Each input number that is entered, if it is a valid command menu option, contacts the Fast API back end and uses it to retrieve or send the necessary information that is specified by the user. If the communication process is successful or ends in a failure, the api will respond by sending a list of content that contains the expected output results for the request including an HTTP 200 success code in certain cases or a list of error codes indicating that the communication process has failed.

5. For object creation commands, the operator returns the id of the newly created objects. Create commands are assigned to either mines or rovers.

6. For object updating commands, the operator returns a list of the newly updated details for that object. Update commands are reserved for the land map and mine objects.

7. for get commands, there are two types. You either get a specific object's details specified by its id, or you get the entire list of objects for that specific object type (i.e. map, mines, or rovers). Maps are gotten by the entire list while mines and rovers can be retrieved either individually per ID number or by the entire list.

8. There is also a command in the menu that calls the rover dispatch request and for this request the operator displays a map matrix that indicates all the spots on the map that have been visited (indicated by *) and all of the spots where mines have been disarmed on the map, indicated by a # symbol. It also returns the xy position, id number and the status of the rover as well.

Part 3 – Deployment Procedure

1. First, I created a Dockerfile following the format listed in the following Fast API docs page: https://fastapi.tiangolo.com/deployment/docker/#build-a-docker-image-for-fastapi

2. Next I created my Microsoft Azure cloud service account and created a container registry in order for the docker image to be pushed on to there.

3. I used the docker build -t <registry name> command to build my docker image

4. I then used the docker login command to login into my Microsoft azure container service.

5. I used the docker push <registry name> command to push my newly built docker image to my container registry as a repository.

6. I created a new Web app for containers service that uses the docker image that has been stored in my container registry.

7. I took the website link that was provided to me after the creation of my web app and used to ping my APIs with the postman API manager service. The website link that can be used to access the web application is the following: coe892lab42023gmahirfaisal.azurewebsites.net

8. See Appendix section below for proof that the web application link works in processing all required requests

NOTE: In order to thoroughly check all Fast API requests via the web application link, it is highly recommended that user has postman installed on to their computer or has access of postman through a web browser.

Conclusion

In the aftermath of this lab assignment, I have learned how to develop and deploy a web application onto a cloud service via a docker container registry. I initially also had issues with the deployment of the docker image but upon further research on how to various docker commands, I was able to properly execute the correct docker commands with little to no issues whatsoever.

Appendix

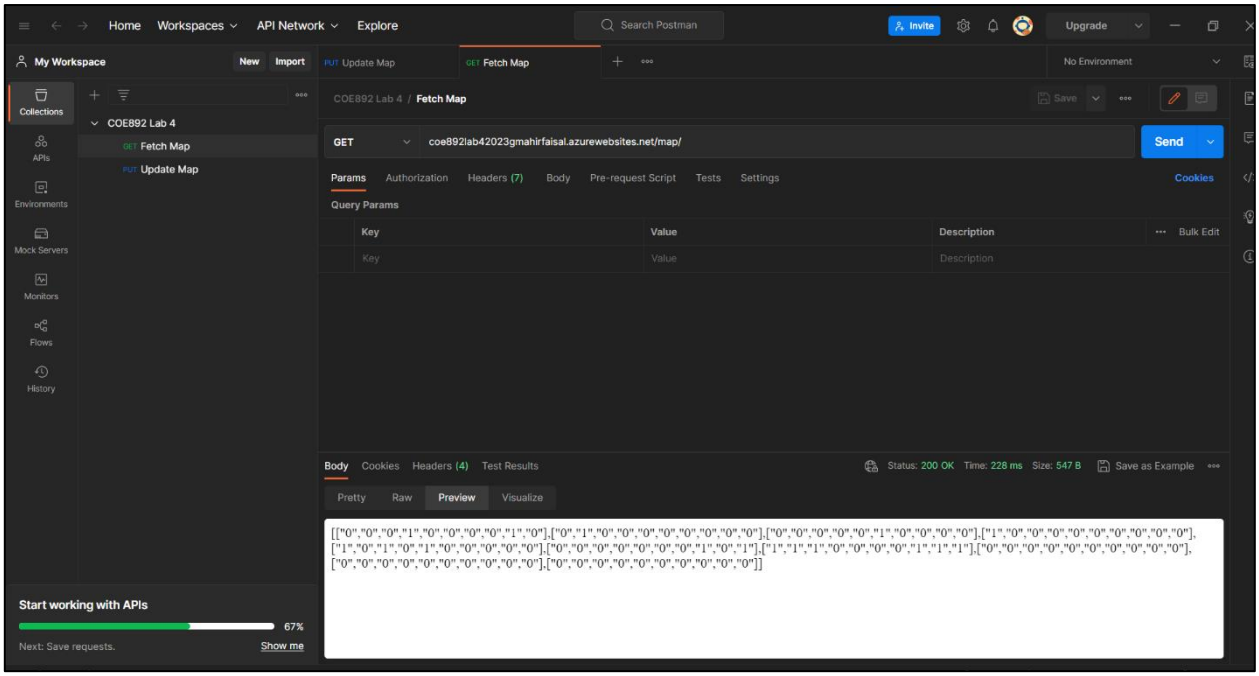

Figure 1: Postman API Manager Screenshot showcasing functionality of GET request for map data.