CSC3065 Cloud Computing

**Assessment 2: Web Calculator**

Submission

Student Number: 40177912

Name: Peter Scott

**Task A**

**Mathematical Function:** Multiply

**Repository URL:** <http://gitlab.hal.davecutting.uk/pscott15/webcalc-mult>

**Live Service URL:** <http://multiply.40177912.qpc.hal.davecutting.uk?x=4&y=2>

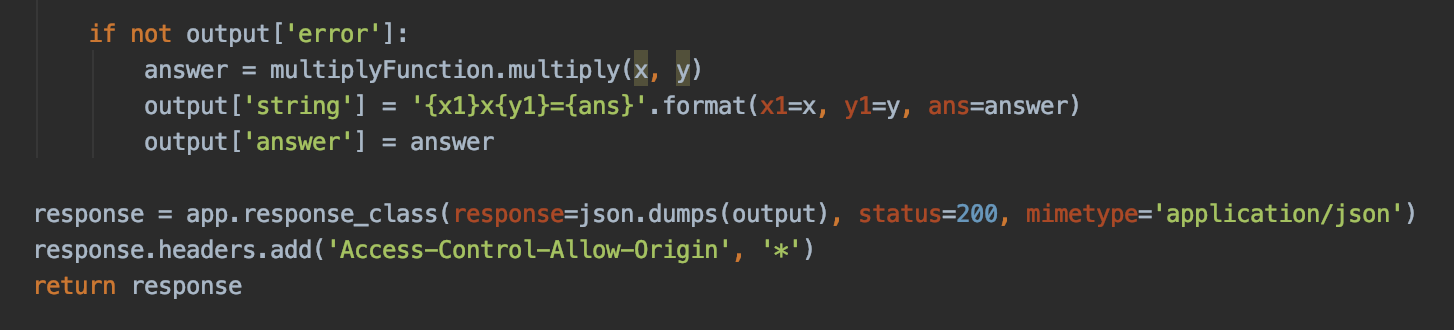
**Description of Implementation**

The multiply function is developed using Python, implementing Flask. The code implements a Python dictionary to store the values of “error”, “string” and “answer” as shown below

A screen shot of a computer

Description automatically generated

The rest of the code for this function is essentially checking the validity of the parameters entered implementing IF statements. To check if values are entered, it makes use of the None value which is equivalent to a null value. If values are entered, it then validates if they are acceptable using the .isDigit() function which determines whether the string consists of inly digits. If no errors have been thrown, we can determine that the parameters are valid and performs the following code:



* Calls the multiply function using x and y as its values
* Sets the string using the x, y and answer variables
* Sets the answer as the result of the multiply function

Finally, we set the response

* json.dumps() is used to format the dictionary into a JSON acceptable version
* Sets the response to 200
* mimeType is used to set the Content-Type which in this case is application/json
* We then add an additional header - Access-Control-Allow-Origin, \* for CORS.
* We then return the response

**Description of Testing:**

A gitlab-ci.yml has been set up to run GitLab CI tests on the function. A screenshot of the tests are below:



These are just some of the tests used for testing the multiply function. We’re testing

* 2 positive values
* 2 negative numbers
* A positive and negative combination
* Multiplication using 0

When any code changes are made, the gitlab-ci.yml file is ran, running the tests in this Python file.

**Mathematical Function:** Divide

**Repository URL:** <http://gitlab.hal.davecutting.uk/pscott15/webcalc-divide>

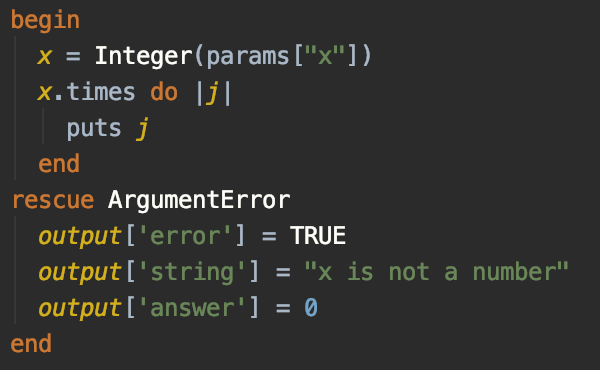
**Live Service URL:** <http://divide.40177912.qpc.hal.davecutting.uk?x=10&y=2>

**Description of Implementation**

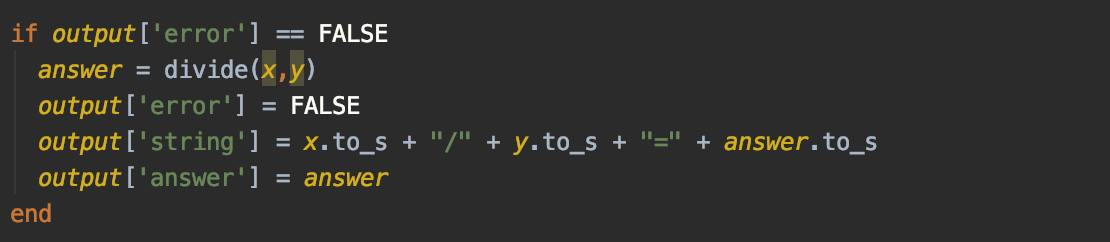
This divide function has been implemented using Ruby Sinatra. After setting the ports to 80 etc, I have set the headers to return the response as a JSON and added CORS. Just like Python, I have implemented a dictionary type method to set the error, string and answer.



After this, I am simply reading in the parameters of x and y and performing checks to determine whether they are valid inputs. I have developed different IF statements to evaluate whether there are any values entered, checking if the user is attempting to divide by 0 and finally if the value entered is not a number. Below is the code for checking this:

****

It starts by parsing the parameter to an integer. If this is valid, no further actions is required. If we try to parse the value “six”, there would be an error and an Argument error would be thrown and execute the above code. This is similar to a try catch method in Java. If there are no errors thrown, the below code is executed:



We calculate the answer of the parameters entered and set the values to error, string and answer. Finally, we return the full response. Ideally, I would have used JSON.generate(response), however, due to issues with Gems regarding JSON, I had to manually write the response in a JSON acceptable format.

**Description of Testing:**

Below is a screenshot of tests for the divide function:

A screenshot of a cell phone

Description automatically generated

Here we are testing different scenarios:

* Testing with 2 positive numbers
* Testing with a negative integer
* Testing a value of x which is less than y (this produces 0 as it can only handle integers)
* Testing a value of x which is bigger than y (this should also produce a remainder however, as before, the calculator only handles integers.

Just like the Python testing, these tests are automatically ran after code changes using the gitlab-ci.yml file.

**Mathematical Function:** Squared

**Repository URL:** <http://gitlab.hal.davecutting.uk/pscott15/webcalc-squared>

**Live Service URL:** <http://squared.40177912.qpc.hal.davecutting.uk?x=4>

**Description of Implementation (language, methods, etc):**

This squared function is developed using Go.

A screenshot of a cell phone

Description automatically generated

The above code is the beginning of my main method for developing the web server. I have set two parameters in this method to access the HTTP writer and HTTP request. I begin by setting the Content Type and CORS headers in the response. After this, I have set a message structure for returning the functions.

The next step to this code is reading the parameter and checking the validity of it. Below is a screenshot of this stage:

A screenshot of a social media post

Description automatically generated

We begin by checking the length of the parameter. If it’s 0 then we can determine no parameter has been entered. We then convert the string to an integer by using strconv.Atoi(). If the string is not a number, then we throw an exception saying the parameter entered is not valid. Finally, we return the response as shown below:

A screenshot of a cell phone

Description automatically generated

The response is parsed into JSON format using json.Marshal() and finally we print to the response writer the response marshalled into the variable correctResJSON.

**Description of Testing:**

Below is a screenshot of the tests used for the squared function. Of course, as this only uses one parameter, there are not as many tests required compared to those functions which use more than one parameter.

**A picture containing screenshot

Description automatically generated**

**Mathematical Function:** Cubed

**Repository URL:** <http://gitlab.hal.davecutting.uk/pscott15/cube>

**Live Service URL:** <http://cubed.40177912.qpc.hal.davecutting.uk?x=2>

**Description of Implementation:**

**A screenshot of a cell phone

Description automatically generated**This function was developed using Java Spring

The above snippet of code shows the main functionality of the cube function. It begins with setting the two headers (Content Type and Access-Control-Allow-Origin). It then begins checking the validity of the parameter passed. If no parameters were passed, it would then set the error to true and the answer to 0. If the value is valid, it will then calculate the answer and set the string value also.

**A screenshot of a cell phone

Description automatically generated**

The above code is called as an Exception Handler. If the parameter is not an integer, this piece of code is called. It sets the headers, error value, string value and answer value. Just like the other set of code, it will then create a new JSON object and send the results in the response body as JSON.

**Description of Testing:**

@SpringBootTest  
class CubeApplicationTests {  
  
 @Autowired  
 private CubeApplication cubeApplication;  
  
 @Test  
 void contextLoads() {  
  
 }  
  
 @Test  
 void testInteger(){  
 int values = cubeApplication.cubedFunction(3);  
 *assertEquals*(27, values);  
  
 }  
  
 @Test  
 void testNegative(){  
 int values = cubeApplication.cubedFunction(-2);  
 *assertEquals*(-8, values);  
  
 }  
  
 @Test  
 void testZero(){  
 int values = cubeApplication.cubedFunction(0);  
 *assertEquals*(0, values);  
  
 }  
  
}

Here we have the tests for the cubed function. Just like the others, these are automatically run by the gitlab-ci.yml after each push.

**Task B**

**Frontend repository URL:** <http://gitlab.hal.davecutting.uk/pscott15/webcalc-frontend>

In this section, I have added a range of different frontend error handling. I will present different error checks

A close up of a device

Description automatically generated

The above shows the calculator awaiting a response. The buttons are greyed out and no input is allowed. This also blocks any requests being made while a request is sent and awaiting a response. This s I have set the timeout as 5 seconds as seen below in the code:

A screenshot of a cell phone

Description automatically generated

Once the HTTP Request is created, if there is no response back within 5000ms (5 seconds), it alerts to the user with the alert, “Calculation Request Timed Out”. This can be seen below:

A screenshot of a cell phone

Description automatically generated

When the error is accepted, it resets the calculator back to its original state.

The next error handling part I will demonstrate will be the response we get when containers are down. In this example I will scale down the squared function in Rancher

A screenshot of a cell phone

Description automatically generated

When I now go to the front end and try to create a request using the squared function, I get the following error:

A screenshot of a cell phone

Description automatically generated

We get an alert in the front end saying the backend service is down and we cannot proceed with the calculation. When the alert is accepted, the front end sets the value to zero and starts waiting for a new request to be made.

The next error handling I have created is error handling with mathematical functions. A prime example of this is dividing by 0. The code for this can be seen below:

A black sign with white text

Description automatically generated

If the second value (y) is zero then it will throw this error. We can see this in action in the front end as seen below:

A screenshot of a cell phone

Description automatically generated

When reset, the calculator returns to its startup state, waiting for a new response.

I have also added error handling to each of the four new functions created. These examples can be seen in section A. However, regarding the add and subtract – the two functions we were given – I have built upon these with sufficient error handling

**Add:**

A screenshot of a cell phone

Description automatically generated

Here we can see that I have added error handling for many different situations:

* If x is not inputted
* If y is not inputted
* If there are no parameters added
* If the parameters are not valid numbers

First image shows no parameters entered and the second shows both parameters are words

|  |  |
| --- | --- |
| A screenshot of a social media post  Description automatically generated |  |

**Subtract**

The below screenshot shows the error handling to the subtract function:

A screenshot of a cell phone screen with text

Description automatically generated

Here we can see that I have added error handling for many different situations:

* If x is not inputted
* If y is not inputted
* If there are no parameters added
* If the parameters are not valid numbers

First image shows no y parameter added and the second image shows no x value added

|  |  |
| --- | --- |
|  |  |

I have shown the error handling of the two existing functions in detail and below are requests being sent from Postman to the four new ones showing a successful result and a failed result:

**Multiply:**

|  |  |
| --- | --- |
| First image is a successful request the second image shows no parameters entered |  |

**Divide:**

|  |  |
| --- | --- |
|  |  |

First image is a successful request the second image shows no x parameter entered

**Squared:**

|  |  |
| --- | --- |
|  |  |

First image is a successful request the second image shows the result of the parameter as a word

**Cubed:**

First image is a successful request the second image shows the result of no parameter entered

|  |  |
| --- | --- |
|  |  |

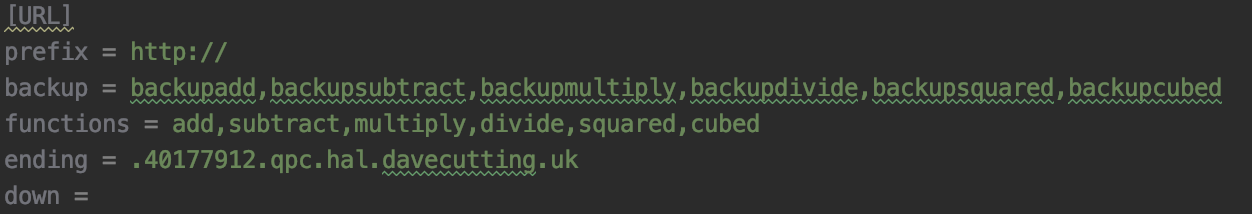
**Task C**

**Challenge Code and Title:** C1. Custom proxy router

**Repository URL:** <http://gitlab.hal.davecutting.uk/pscott15/webcalc-proxy>

**Implementation Details** (how you implemented):

The router has been developed using Python. It has been developed to not make use of any static URLs. When a request is made from the front end, it sends the parameters and the operator to the router. From there, the proxy will search through a configurations file to see if the operator is included. A screenshot of the configurations file is below:



The proxy will search through the functions and match the operator sent in the request to the correct value in the functions list. If this value is not found, it will send this JSON response to the front end

output = {  
 'error': True,  
 'string': "Operator not found",  
 'answer': 0  
}

It will search through a Python Dictionary for the value. The dictionary is defined below:

def function\_lists():  
 parser = SafeConfigParser()  
 parser.read('Configurations.properties')  
  
 response = parser.get('URL', 'functions')  
 myList = response.split(",")  
 listOfFunctions = {}  
  
 prefix = parser.get('URL', 'prefix')  
 ending = parser.get('URL', 'ending')  
  
 i = 0  
 while i < len(myList):  
 listOfFunctions[myList[i]] = prefix + myList[i] + ending  
 i += 1  
  
 return listOfFunctions

It searches through the file using “configparser” and puts the values into a list called “myList”. It then takes each value in myList and uses it as a key value in the dictionary and creates a URL using the prefix (http://), value in myList and the ending (.40177912.qpc.hal.davecutting.uk) from the config file and uses it to create a URL.

**Configurable Options**

I have also made the router very configurable by adding the ability to modify what functions are available in the router. If a new function is added to the cluster and frontend (e.g. modulo), we can easily add it to the router buy using a PUT request. We can also delete some of the functions using a DELETE request and finally we can check the status of each of the end points.

**PUT Request**

Below is the link for adding a new function to the configurations file.

[**http://proxy.40177912.qpc.hal.davecutting.uk/functions/modulo**](http://proxy.40177912.qpc.hal.davecutting.uk/functions/modulo)

The code for this PUT request can be seen below:

@app.route("/functions/<string:method>", methods=['PUT'])  
def add\_to\_configurations(method):  
 parser = SafeConfigParser()  
 parser.read('Configurations.properties')  
  
 response = parser.get('URL', 'functions')  
 backupResponse = parser.get('URL', 'backup')  
 myList = response.split(",")  
  
 method = request.view\_args['method']  
  
 i = 0  
 existing = 0  
 while i < len(myList):  
 if method == myList[i]:  
 existing = 1  
 break  
 else:  
 i += 1  
  
 if existing == 0:  
 parser.set('URL', 'functions', response + "," + method)  
 parser.set('URL', 'backup', backupResponse + ",backup" + method)  
 else:  
 return method + " already added to list of functions"  
  
 with open('Configurations.properties', 'w') as configfile:  
 parser.write(configfile)  
  
 now = datetime.now()  
 current\_time = now.strftime("%H:%M:%S")  
  
 return "Added " + method + " at " + current\_time

The method added in the path parameter in the URL is searched against the methods already in the configurations file. If it already exists, it returns with a response to the user saying the method already exists. If it doesn’t exist, then it will add it to the end of the both the functions list and backup list (explained in C2). It also sends the time the function was executed so it ensures the user knows the response is not an old version.

**EXAMPLE:**

**A screenshot of a cell phone

Description automatically generated**

Above we can see the current content of the configurations file. We can see the 6 functions we expect, the ones already added in the calculator. We can send a put request to the server to add “modulo” and when done we get a message back confirming it has been added with the time to confirm it is not an old message

A screenshot of a cell phone

Description automatically generated

If we go back to the configurations file, we can see the new function modulo has been added to the list of functions.

A screenshot of a cell phone

Description automatically generated

If we try to add an existing function in the functions section – in this instance I will be adding modulo again – we receive the following error message and the PUT request is not completed. This ensures we do not have duplicated endpoints for each function

A screenshot of a cell phone

Description automatically generated

**Delete Request**

As there is a PUT request, we also need a delete function which can be seen below on the next page. The URL for this is:

<http://proxy.40177912.qpc.hal.davecutting.uk/functions/round>

@app.route("/functions/<string:method>", methods=['DELETE'])  
def delete\_from\_configurations(method):  
 parser = SafeConfigParser()  
 parser.read('Configurations.properties')  
  
 response = parser.get('URL', 'functions')  
 myList = response.split(",")  
 backupResponse = parser.get('URL', 'backup')  
 backupList = backupResponse.split(',')  
 method = request.view\_args['method']  
  
 i = 0  
 found = 0  
  
 while i < len(myList):  
 if method == myList[i]:  
 del myList[i]  
 found = 1  
 break  
 else:  
 i += 1  
  
 j = 0  
 backupFound = 0  
 backupMethod = "backup" + method  
 while j < len(backupList):  
 if backupMethod == backupList[j]:  
 del backupList[j]  
 backupFound = 1  
 break  
 else:  
 j += 1  
  
 if found == 1 and backupFound == 1:  
 string = json.dumps(myList)  
 string = string.replace(" ", "")  
 string = string.replace('"', '')  
 string = string.replace("[", "")  
 string = string.replace("]", "")  
 bString = json.dumps(backupList)  
 bString = bString.replace(" ", "")  
 bString = bString.replace('"', '')  
 bString = bString.replace("[", "")  
 bString = bString.replace("]", "")  
 parser.set('URL', 'functions', string)  
 parser.set('URL', 'backup', bString)  
 with open('Configurations.properties', 'w') as configfile:  
 parser.write(configfile)  
  
 now = datetime.now()  
 current\_time = now.strftime("%H:%M:%S")  
  
 return "Removed " + method + " at " + current\_time  
 else:  
 return method + " is not a valid method"

This does a very similar process as the PUT request. This is a delete request, so when activated, it will delete the function in the web URL parameter.

The function begins parsing the functions into a list. It then searches through the function list and if found, deletes it from the list and sets a variable found to 1. There is then an IF statement which checks whether the function was found, if it was, it writes back the new list to the configurations list, and if not, returns a response stating the method was not found.

**A screenshot of a cell phone

Description automatically generated**

We can see that modulo is in the function list, so when the web request is sent, it will remove it from the functions list. Below is a screenshot of the request being sent.

**A screenshot of a cell phone

Description automatically generated**

I have then sent the delete request to the router and received a confirmation message that it has been deleted. Just like the PUT request, we receive a time of deletion to confirm that the request is not a stale one.

The next step is to check the execute shell in Rancher to confirm that modulo has been deleted.

**A screenshot of a cell phone

Description automatically generated**

We can see from the above that the modulo function has been deleted from the list and we cannot use it anymore. If we send the request again using modulo, we receive the following error message as the function is deleted from the list.

**A screenshot of a cell phone

Description automatically generated**

We can see that modulo is not a valid method in the functions list so nothing is deleted here.

**Health Checks**

Health checks are provided to check the status of each of the functions in the Configurations file. The URL for this is:

<http://proxy.40177912.qpc.hal.davecutting.uk/functions/health>

The point of the function is to check the status of each of the functions and return whether they are healthy or not. The code for this can be seen below:

@app.route("/functions/health/", methods=['GET'])  
def getFunctionList():  
 parser = SafeConfigParser()  
 parser.read('Configurations.properties')  
  
 response = parser.get('URL', 'functions')  
 myList = response.split(",")  
  
 responseBackups = parser.get('URL', 'backup')  
 backupList = responseBackups.split(",")  
  
 listOfFunctions = {}  
 listOfURLs = []  
 checkHealth = {}  
  
 prefix = parser.get('URL', 'prefix')  
 ending = parser.get('URL', 'ending')  
  
 i = 0  
 while i < len(myList):  
 listOfFunctions[myList[i]] = prefix + myList[i] + ending  
 i += 1  
  
 k = 0  
 counted = 0  
  
 while k < len(myList) \* 2:  
 listOfURLs.insert(len(listOfURLs), prefix + myList[counted] + ending)  
 listOfURLs.insert(len(listOfURLs), prefix + backupList[counted] + ending)  
 counted += 1  
 k += 2  
  
 j = 0  
  
 while j < len(listOfURLs):  
 r = requests.get(listOfURLs[j])  
 checkHealth[listOfURLs[j]] = "Status Code = " + str(r.status\_code)  
 j += 1  
  
 response = app.response\_class(response=json.dumps(checkHealth), status=200, mimetype='application/json')  
 return response

The results of the health check can be seen below:

A screenshot of a social media post

Description automatically generated

If we bring down the containers for add, we can see the health for it below:

A screenshot of a cell phone

Description automatically generated

We can see that both the add and add backup URLs return with a 503 error as the workload has been scaled down to zero. If we now add modulo for example and run the health check again, we get the following results:

A close up of text on a white background

Description automatically generated

We can see the status codes for both modulo functions are 404 – Not Found, which is expected

For this, we are checking through a list of URLs (both function and backup) and printing it out using its status code.

**Review of Success:**

I would say I have been successful in creating a custom proxy router as it handles requests in the front end and is configurable to be modified by admin users.

**Additional Features**

<http://proxy.40177912.qpc.hal.davecutting.uk/functions/functionlist>

This is a list of URLs for each function. From this we can see the list of functions and their corresponding links The code for this is below:

@app.route("/functions/functionlist/", methods=['GET'])  
def printFunctions():  
 parser = SafeConfigParser()  
 parser.read('Configurations.properties')  
  
 response1 = parser.get('URL', 'functions')  
 myList = response1.split(",")  
 listOfFunctions = {}  
  
 prefix = parser.get('URL', 'prefix')  
 ending = parser.get('URL', 'ending')  
  
 i = 0  
 while i < len(myList):  
 listOfFunctions[myList[i]] = prefix + myList[i] + ending  
 i += 1  
  
 response = app.response\_class(response=json.dumps(listOfFunctions), status=200, mimetype='application/json')  
 return response

When the request is made, we get the following response:

A screenshot of a cell phone

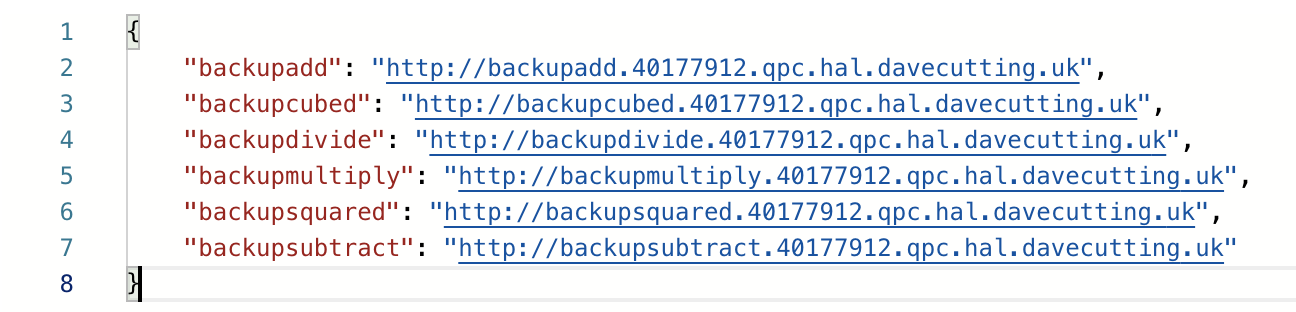
Description automatically generated

<http://proxy.40177912.qpc.hal.davecutting.uk/functions/backupfunctionlist/>

This is a list of URLs for each backup function. It is the same principle as the main function list but pulls the data from the backup section. The code for this can be seen below:

@app.route("/functions/backupfunctionlist/", methods=['GET'])  
def printbackups():  
 parser = SafeConfigParser()  
 parser.read('Configurations.properties')  
  
 response = parser.get('URL', 'backup')  
 myList = response.split(",")  
 backupList = {}  
  
 prefix = parser.get('URL', 'prefix')  
 ending = parser.get('URL', 'ending')  
  
 i = 0  
 while i < len(myList):  
 backupList[myList[i]] = prefix + myList[i] + ending  
 i += 1  
  
 response = app.response\_class(response=json.dumps(backupList), status=200, mimetype='application/json')  
 return response

Likewise, the response for this is:



**Challenge Code and Title:** C2: Frontend service failure handler

**Implementation Details** (how you implemented):

For the implementation of this challenge, I have built upon C1 to develop a proxy router which does the following:

* Takes in the values from the front end

A black and silver text

Description automatically generated

* Creates second request using the backup function

A screenshot of a cell phone

Description automatically generated

* If the first request using the main function is down and the backup endpoint is up, then it sets the final request using the backup endpoint
* If the main function is ok but the backup endpoint is down, it uses the main endpoint
* If they’re both up, we make use of the round robin load balancing

Below we can see an example of the backup function being implemented. We are checking the status of the first request and can determine it does not return a successful response however the backup endpoint does. We then write to the configurations file and add the URL to the down section.

if r.status\_code != 200 and r1.status\_code == 200:  
 finalR = requests.get(execUrl1, params=PARAMS)  
 if len(string) != 0:  
 parser.set('URL', 'down', down + "," + r.url)  
 else:  
 parser.set('URL', 'down', r.url)  
 with open('Configurations.properties', 'w') as configfile:  
 parser.write(configfile)

There are a number of different conditions – another being if both endpoints are down. The code for this can be seen below:

elif r.status\_code != 200 and r1.status\_code != 200:  
 bothDown = 1  
 if len(string) != 0:  
 parser.set('URL', 'down', down + "," + r.url + "," + r1.url)  
 else:  
 parser.set('URL', 'down', r.url + "," + r1.url)  
 with open('Configurations.properties', 'w') as configfile:  
 parser.write(configfile)  
 output = {  
 'error': True,  
 'string': "Backend Down",  
 'answer': 0  
 }

If both are down then we send the response – “Backend down” to the frontend by returning this response:

response = app.response\_class(response=json.dumps(output), status=200, mimetype='application/json')

If both are up and working with a 200 response code, we perform the following code

elif r.status\_code == 200 and r1.status\_code == 200:  
 if backupLastUsed == 0:  
 finalR = requests.get(execUrl1, params=PARAMS)  
 backupLastUsed = 1  
 else:  
 finalR = requests.get(execUrl, params=PARAMS)  
 backupLastUsed = 0

We check the global variable “backupLastUsed” depending on the value we choose which endpoint to hit. If we hit the main endpoint we then set the backupLastUsed to false (0) and then next time, the backup endpoint will be used and we continue using this method to achieve round robin load balancing.

To show this working I will show the URL being used in requests:

First Request:

A screenshot of a cell phone

Description automatically generated

We are using our main endpoint in this instance. We have then set the backupLastUsed to 0 and when we run the same request again, we get the following response:

Second Request:

A screenshot of a cell phone

Description automatically generated

Now I will bring down one of the functions in Rancher and send off a request. Here you can see I have brought down add:

A screenshot of a cell phone

Description automatically generated

To ensure this is down, we can send off a health check to confirm this:

A close up of text on a white background

Description automatically generated

We can confirm that add is down. Now we can confirm in the proxy execute shell that nothing is in the down list:

A screenshot of a cell phone

Description automatically generated

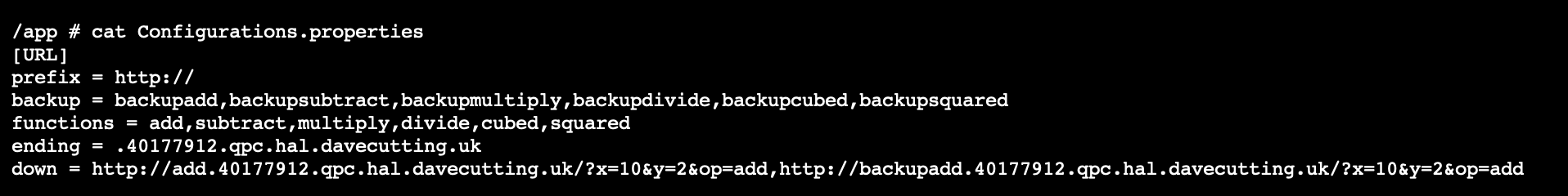
Currently, nothing is added into this down list, however if we send a request using add, we should have two new requests added to this list (both the main URL and the backup URL)

We run this request:

A screenshot of a cell phone

Description automatically generated

We can see that we get an error and if we check the configurations file, we can see the following:



Both main and backup functions are added. Now how do we remove these from this list if we bring up the service again?

In the main method in the server, I have set up a scheduler as shown below that runs a method called “checkDown()” every 5 seconds. The code is as below:

if \_\_name\_\_ == "\_\_main\_\_":  
 scheduler = BackgroundScheduler()  
 scheduler.start()  
 scheduler.add\_job(checkDown, 'interval', seconds=5)  
 app.run(host='0.0.0.0', port=80, debug=True)

I have initialised this Background Scheduler as seen and it checks this method:

def checkDown():  
 parser = SafeConfigParser()  
 parser.read('Configurations.properties')  
 listInDown = downList()  
  
 if len(listInDown) >= 1:  
 i = 0  
 changed = 0  
 while i < len(listInDown):  
 url = listInDown[i]  
 r = requests.get(url)  
 if r.status\_code == 200:  
 del listInDown[i]  
 changed = 1  
 i += 1  
 else:  
 i += 1  
  
 if changed == 1:  
 string = json.dumps(listInDown)  
 string = string.replace(" ", "")  
 string = string.replace('"', '')  
 string = string.replace("[", "")  
 string = string.replace("]", "")  
 parser.set('URL', 'down', string)  
 with open('Configurations.properties', 'w') as configfile:  
 parser.write(configfile)  
  
 return  
 else:  
  
 return

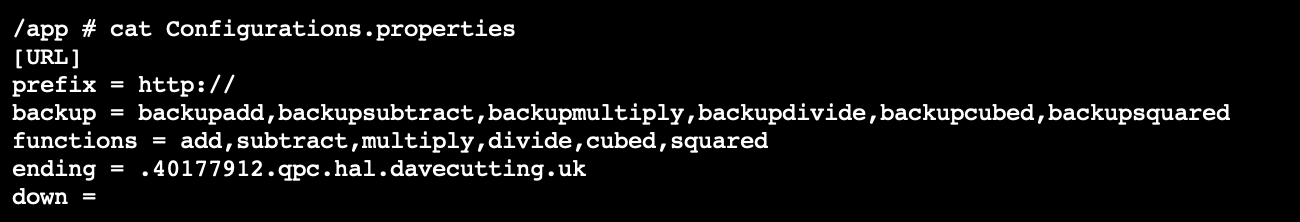
I goes through each of the requests in the down list, checks the status of each response and if it is 200, it removes it from the list and re-writes the list to the configurations file. However, if the status is still not 200, it continues and retries 5 seconds later.

If we now bring up the add function we can see now that the status code is now 200:

A close up of text on a white background

Description automatically generated

Now when we check the configurations file, we can see the 2 requests are now cleared:



This is just one of the examples as to what could happen. If we bring down just one of the endpoints, we will see just that one added to the list in down

I will demonstrate this by bringing down the main instance of cubed. If we check our health we should see only one instance of cube is down:

A screenshot of a cell phone

Description automatically generated

Here we can see the main URL has a status code of 404 yet the backup has a status code of 200. If we send a multiply request, we can see we still return a result:

A screenshot of a cell phone

Description automatically generated

Even though the man endpoint is down, we can still retrieve the data using the backup

If we check our configurations file, we can see the following in the down list:

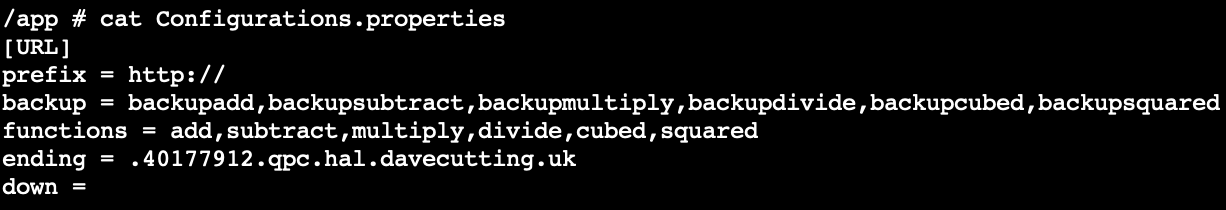


If we bring up the endpoint again and check the health, we see it’s back to 200:

A screenshot of a cell phone

Description automatically generated

And if we check the configurations file, we see it has been removed from the list as it ran the request in the background and retrieved a 200 status code:



This works in the other way to as if we have the backup down, we will still retrieve a response as the main function is still up and the we will send the backup request to the down list to get constant checks.

**Review of Success**

I believe I have achieved a working router which can reroute requests to different endpoints and automatically check these if they are working. I would however change the fact the both ingress controllers for each function point to the same workload. This is due to the 10 pod limit in Rancher. If this was used in production, there would not only be duplicated URLs, but duplicated workloads as if one workload goes down (as we’ve seen in the example), it will not affect both endpoints and potentially still return a successful response.

**List of Useful URLs**

<http://proxy.40177912.qpc.hal.davecutting.uk/functions/functionlist> - Gets list of functions and their respective URLs

<http://proxy.40177912.qpc.hal.davecutting.uk/functions/backupfunctionlist/> - Gets list of backup functions and their respective URLs

<http://proxy.40177912.qpc.hal.davecutting.uk/?x=10&op=cubed> – Sends request to the proxy router

<http://add.40177912.qpc.hal.davecutting.uk?x=1&y=2> – Sends request to add workload

<http://subtract.40177912.qpc.hal.davecutting.uk?x=1&y=2> – Sends request to subtract workload

<http://multiply.40177912.qpc.hal.davecutting.uk?x=4&y=2> – Sends request to multiply workload

<http://divide.40177912.qpc.hal.davecutting.uk?x=10&y=2> – Sends request to divide workload

<http://squared.40177912.qpc.hal.davecutting.uk?x=4> – Sends request to squared workload

<http://cubed.40177912.qpc.hal.davecutting.uk?x=2> – Sends request to cubed workload

<http://proxy.40177912.qpc.hal.davecutting.uk/functions/health> - Checks health on each of the URLs

<http://proxy.40177912.qpc.hal.davecutting.uk/functions/modulo> - PUT request to add modulo

<http://proxy.40177912.qpc.hal.davecutting.uk/functions/modulo> - DELETE request to remove modulo