Traffic Monitoring using IoT Technologies: FINAL REPORT.

Or counting vehicles with a Camera and a Cloud.
Group №18

Vishal Dabba (25591602), Samuel Orman-Chan (25659005), Harry Fitchett (25717330), Haadiya Ahmed (25995974), Dmytro Steblyna (26817765) & Ahmed Ahmed (26018267)

UNIVERSITY OF
LINCOLN

School of Computer Science
University of Lincoln
United Kingdom
May 11, 2023

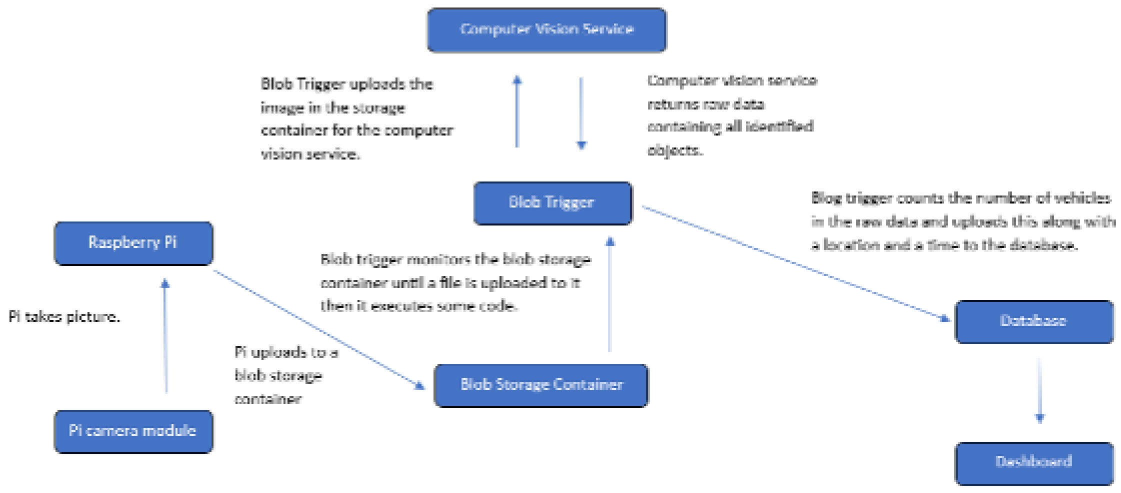
—

Supervisor:
Derek Foster.

1 Software Engineering:

For the implementation of this prototype, we collectively chose to go with the Waterfall strategy. The waterfall strategy uses a logical progression of steps for a project, it has distinct endpoints for each phase of development. Once an endpoint has been completed it cannot be revisited. We felt the nature of the strategy was approachable and feasible with the amount of time we had, its simplicity and linear nature made easy to implement when developing the prototype. We were able to split the development of the prototype into 6 different phases, these consisted of, requirements and documentation, system design, implementation, testing, deployment, and maintenance. During phase 1, requirements and documentation, all the potential requirements, deadlines, and guidelines for the project were placed into a document, this was useful as we were able to refer to this documentation at any point during the development of the project and it always kept us on track ensuring we kept up to date and inline with deadlines and guidelines. Phase 2 consisting of system design, a document outlining the technical design of the project was made, this included hardware needed, software languages being used, and services being used to help the development of the project. Phase 3 of the waterfall strategy is implementation the source code is developed in line with the specifications and guidelines in the prior phases of the project. This part was split into different parts and each member of the group was designated a specific role developing a specific component of the project. Once all parts had been completed we combined all the components together. There are no version controls present due the fact this is a waterfall strategy meaning the code must be implemented all together at once hence only being one version. We had used GitHub to prepare code until we are ready to deploy our code. Phase 4 consists of testing we ensure that integrity of the software, errors did occur, and we were forced to repeat the coding stage for debugging purposes. However, once it did pass all the tests required the waterfall continued its journey. Phase 5 consists of deployment, after the testing the application is deemed to be functional and can be deployed into the environment. The final phase is maintenance, which in our project is not really met due to the fact that we will not be maintaining this prototype for further development.

A diagram showing how the prototype was constructed:


Computer Vision Service
Blob Trigger uploads the image in the storage container for the computer vision service.
Computer vision service returns raw data containing all identified objects.
Blob Trigger
Blog trigger counts the number of vehicles in the raw data and uploads this along with a location and a time to the database.
Raspberry Pi
Blob trigger monitors the blob storage container until a file is uploaded to it then it executes some code.
Pi takes picture.
Database
Pi uploads to a blob storage container
Blob Storage Container
Pi camera module
Dashboard

When choosing a strategy there were many options to choose from these included, Kanban, Scrum, and Agile. We chose to use waterfall because it has many advantages that we thought would benefit the prototype in our environment. We need to ensure everyone was able to follow a structure and the waterfall strategy provided that. Splitting the different parts of the project into phases made planning and designing the prototype more digestible. It made documentation and planning much easier. Using the waterfall method, we can determine the end goal at an early stage of development, unlike the other methods suggest such as Kanban or Scrum. Once committing to this strategy, you are committing to an end goal. For smaller scale projects like ours the waterfall method is good for making the team aware of the goals from the very beginning, as there is less potential for confusion as the project moves forward. Furthermore, the waterfall method is highly methodical, so when transferring from phase to phase there is clean transfer of information. Waterfall prioritises accessible information such as the documentation made for our project meaning if anyone does become confused or unsure they are always able to refer back to the documentation created at the earlier stages unlike other strategies where changes can become confusing . However, as many advantages as there are for waterfall there are some disadvantages. As we were developing our prototype we realised sometimes we had innovating ideas during later phases such as coding implementation, however, with waterfall once a phase is completed you cannot go back, meaning there was never any room for unexpected changes or revisions. If faced with a roadblock it can hinder the entire project thus far. Another limitation with waterfall strategy is that it delays testing till after completion. Doing testing after completion can be very risky as there could be significant amounts of tests the projects fail to meet meaning there could large revisions to the plan or cause significant delays. These delays in a commercial setting can cause major issues not just for a company but for clients too as time may be a constraint and there may be no room for error. Another disadvantage is that there is a lack of focus on the end user, once a plan is set out with the waterfall strategy there is very little involvement with the client, meaning there is no chance for the client to give opinions as the project moves forward.

If we were given a chance to develop a second prototype another strategy that we would possibly consider is Agile. This type of strategy involves more of the client and end user and allows for changes amid project as there is an increased in flexibility.

2 Implementation:

2.1 Video:

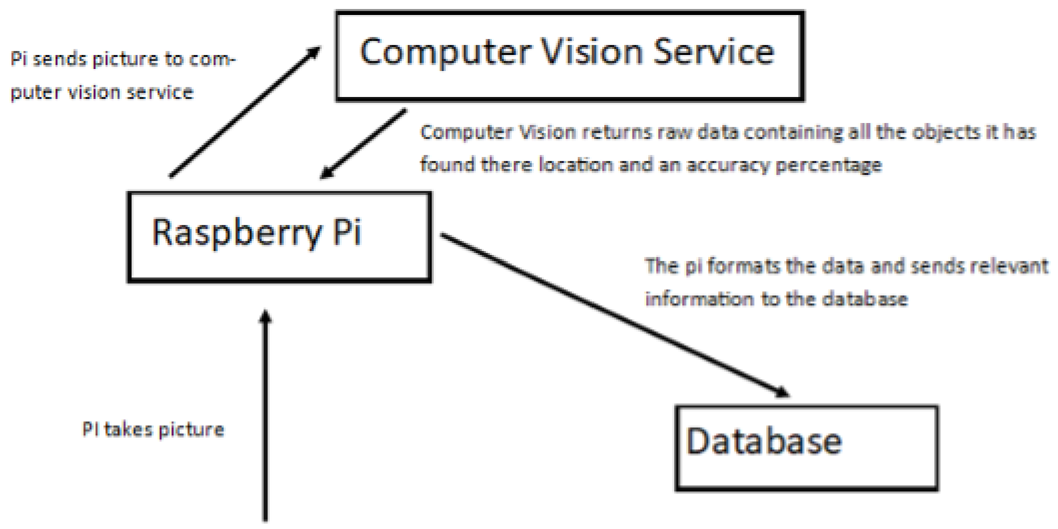
A Video showing the Artefact in use is available at the following url:
`https://www.youtube.com/watch?v=Ut4ODaQWDq0`

2.2 The Computer Vision and Trigger Component:

We began building our cloud infrastructure by deploying a recourse group TSE_Project_IOT. After familiarizing ourselves with the azure portal we decided to deploy a Computer Vision resource called TSE_IOT_PROJECTY-59960. We elected to choose Computer Vision services compared to other image processing services like image analysis or optical character recognition because it capable of “image understanding” – unknown (n.d.b) making it able to recognise concepts and images such as cars, had good documentation and was therefore easy to learn for new developers and was scalable so dependant on subscription could service one Pi to a traffic network worth of Pis.

After some trial and error, we were able to build code that could upload an image URL and count the number of objects categorised as “car”, “van”, “Land Vehicle”, “bicycle” with a confidence value of 0.5 or higher. This code is referred to as test_run_2.py on our GitHub repository and can decode the raw data returned by azure computer vision services. See Fig.1

After this success we began considering the architecture of our artefact here is a representation of one of the first suggestions:


Computer Vision Service
Pi sends picture to computer vision service
Computer Vision returns raw data containing all the objects it has found there location and an accuracy percentage
Raspberry Pi
The pi formats the data and sends relevant information to the database
PI takes picture
Database

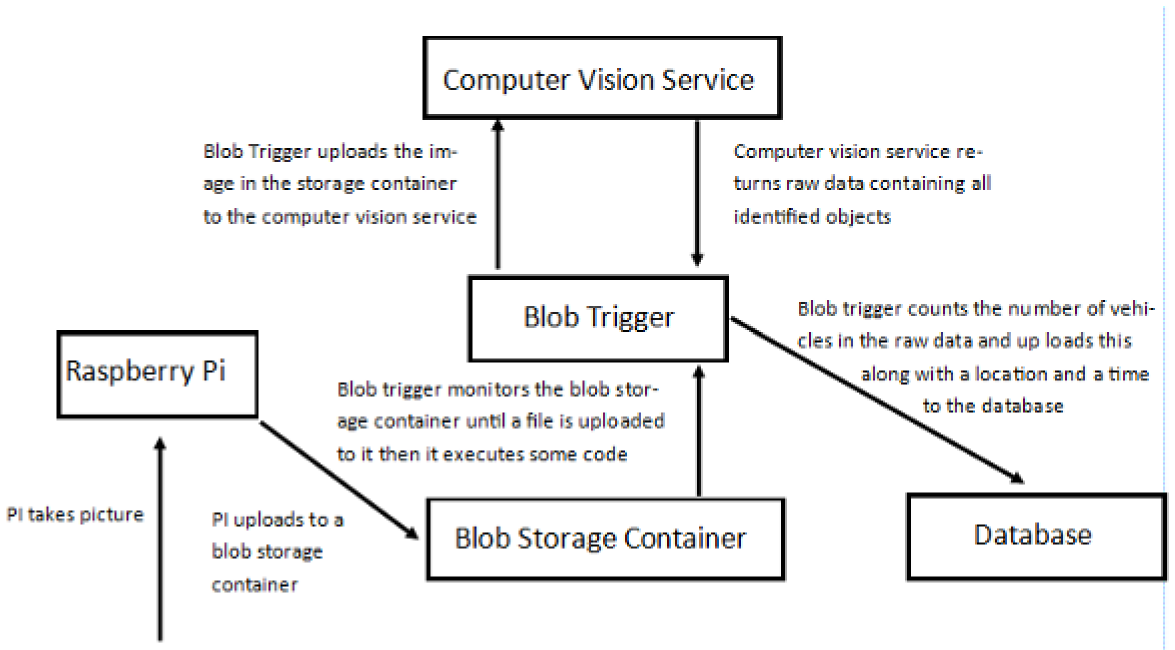
This suggestion had some flaws the main one being the Pi would do all the data processing and would have to upload to various cloud services twice as well as receive data from the cloud. We decided to improve upon this design as when the Pi is deployed it will likely have a poor internet connection and increasing the computational demands of a Pi would reduce its lifespan and increase replacement costs.

After further research the group decided the pi should upload to an initial storage service and a virtually hosted function to feed images from the storage service to the computer vision service and data to the database. We decided to use a Blob storage container we named ‘imagessquidrabbitdogrex’ located in the ‘tsestorageaccount59940’, at this point we thought azure services had to have unique names which is why these names are not streamlined. We elected to use a Blob storage container as it was cheap to host costing our customers nothing with the downsides being its “unorganised data structure” – unknown (n.d.a) meaning it is harder to query, it has a limited storage capacity, and it Is not a viable long term storage option with “azure blob containers storing blobs for a maximum of nine days” – (azure.microsoft.com, n.d). These downsides do not affect our artefact as the blob storage container is supposed to be temporary storage.

Eventually code was produced to upload a text file containing “hello world” to our blob storage container called Python_Bucket_Uploader.py on our GitHub: Fig.2

We now had a blob container working and after a bit of research and trial we decided to deploy a blob trigger a blob trigger has three advantages in our artefact first its low cost to maintain compared to alternatives like hosting a virtual machine to run code, second it automatically monitors a blob storage container and will trigger upon a new blob being uploaded so we do not have to produce additional code to monitor the blob storage container, Finally despite its limited processing power it will be perfectly capable of executing the previously made vision code.

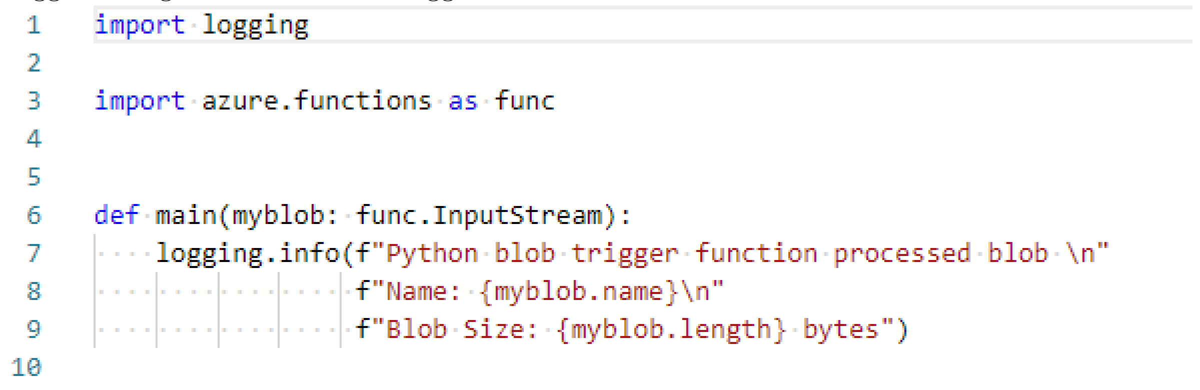
Our theoretical model now looked like this:


Computer Vision Service
Blob Trigger uploads the image in the storage container to the computer vision service
Computer vision service returns raw data containing all identified objects
Blob Trigger
Blob trigger counts the number of vehicles in the raw data and up loads this along with a location and a time to the database
Raspberry Pi
Blob trigger monitors the blob storage container until a file is uploaded to it then it executes some code
PI takes picture
PI uploads to a blob storage container
Blob Storage Container
Database

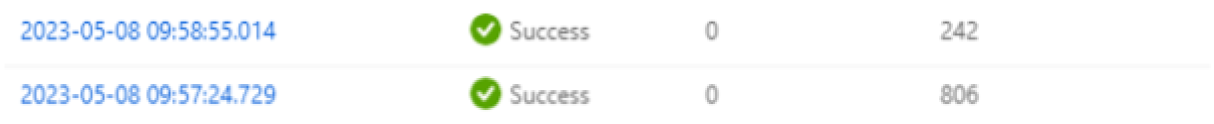
First, we had built a function app however hosting a function app using python is currently cannot use a windows-based operating system and had to be hosted via Linux the other alternative was we use .Net framework and program in C# which can use windows. Eventually we decided to use python because we already had built the existing computer vision code in python, and we had grown more confident in our Linux skills from deploying the Pi.

We created *FunctionApp3LinuxPY* which contained *BlobTrigger1*.

After successfully deploying these resources, we were able to monitor our first successful triggers using the default blob trigger code.



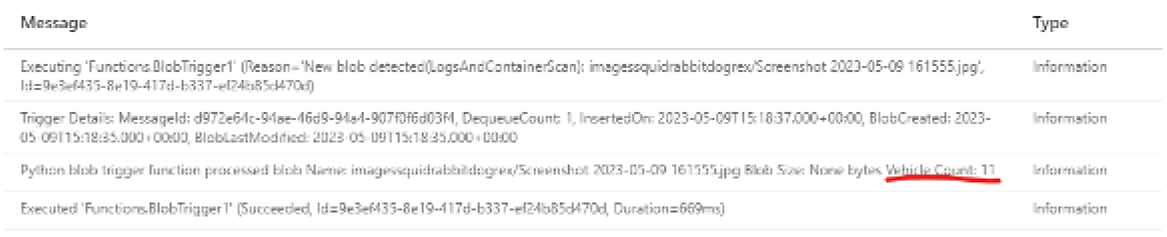
```
import logging

import azure.functions as func


def main(myblob: func.InputStream):
    logging.info(f"Python blob trigger function processed blob \n"
                 f"Name: {myblob.name}\n"
                 f"Blob Size: {myblob.length} bytes")

```



```
2023-05-08 09:58:55.014    ✓ Success    0    242
2023-05-08 09:57:24.729    ✓ Success    0    806
```

The final step was to add our computer vision code to the trigger and after we figured out how to install a requirements file from our GitHub containing the relevant libraries our code looks like this in Fig.3

And the code counts vehicles:



Message	Type
Executing 'Functions.BlobTrigger1' (Reason='New blob detected(LogsAndContainerScan): imagessquidrabbitdogrex/Screenshot 2023-05-09 161555.jpg', Id=9e3ef435-8e19-417d-b337-ef24b85d470d)	Information
Trigger Details: MessageId: d972e64c-94ae-46d9-94a4-907f0f6d03f4, DequeueCount: 1, InsertedOn: 2023-05-09T15:18:37.000+00:00, BlobCreated: 2023-05-09T15:18:35.000+00:00, BlobLastModified: 2023-05-09T15:18:35.000+00:00	Information
Python blob trigger function processed blob Name: imagessquidrabbitdogrex/Screenshot 2023-05-09 161555.jpg Blob Size: None bytes Vehicle Count: 11	Information
Executed 'Functions.BlobTrigger1' (Succeeded, Id=9e3ef435-8e19-417d-b337-ef24b85d470d, Duration=669ms)	Information

When deploying this code, It is recommended to build an individual blob storage container per raspberry pi camera and to edit the variables confidenceVar and vehicleTypes to represent how accurate your system must be as well as what vehicles you are looking for.

2.3 The Dashboard & Database Components:

Our exploration will be split into three main sections: Raspberry Pi implementation, cloud services, and database and dashboard creation. In the first section, we delve into the reasoning behind our choice of Raspberry Pi, the technical challenges we faced during implementation, and how we overcame them.

The second part discusses our cloud infrastructure, where we used Microsoft Azure's Computer Vision services and Blob Storage. We explain our architectural decisions, how we optimized data processing, and how we set up blob triggers for automated image processing.

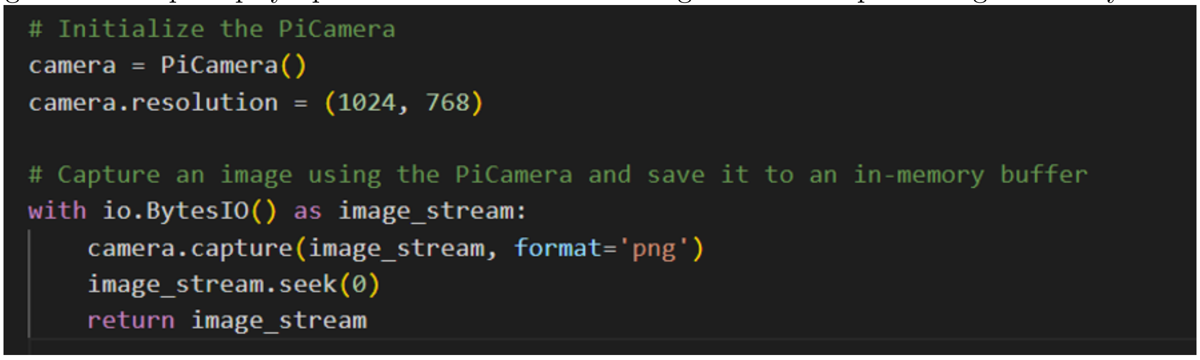
The final section focuses on our use of a cloud-based Postgres database solution, Supabase, and the development of a dashboard for data visualization using Next.js and Tailwind. We also shed light on the issues faced during the dashboard creation process and how we adapted to them.

2.3.1 Raspberry Pi

The traffic monitoring system prototype we have built utilizes cloud computing and the Internet of Things (IoT), specifically the Raspberry Pi 3 and its camera module, to form a robust, real-time traffic monitoring solution.

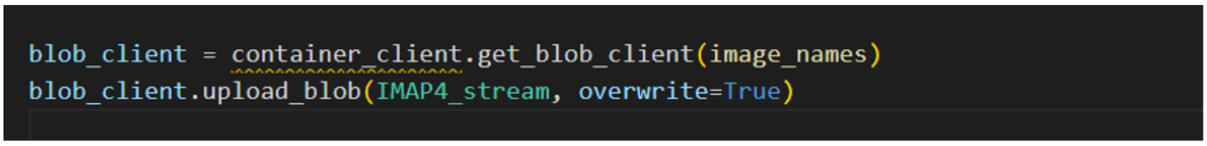
The Raspberry Pi 3, equipped with a camera module, was selected for its compact size, cost-effectiveness, and the availability of extensive community support. We used Python, known for its simplicity and the variety of libraries it offers, which greatly facilitated our development process.

Our original design involved streaming video from the Raspberry Pi to the cloud for processing. However, due to technical constraints and the need for a more efficient solution, we adapted our design to instead capture images when a certain trigger condition is met. These images are then promptly uploaded to Azure Blob Storage for further processing and analysis.



```python
# Initialize the PiCamera
camera = PiCamera()
camera.resolution = (1024, 768)

# Capture an image using the PiCamera and save it to an in-memory buffer
with io.BytesIO() as image_stream:
    camera.capture(image_stream, format='png')
    image_stream.seek(0)
    return image_stream
```

Azure Blob Storage was chosen for its scalability, high security, and redundancy. Its ability to store large amounts of data in a cost-effective manner, coupled with the flexibility of data access from anywhere, made it a suitable choice for our cloud-based traffic monitoring system.



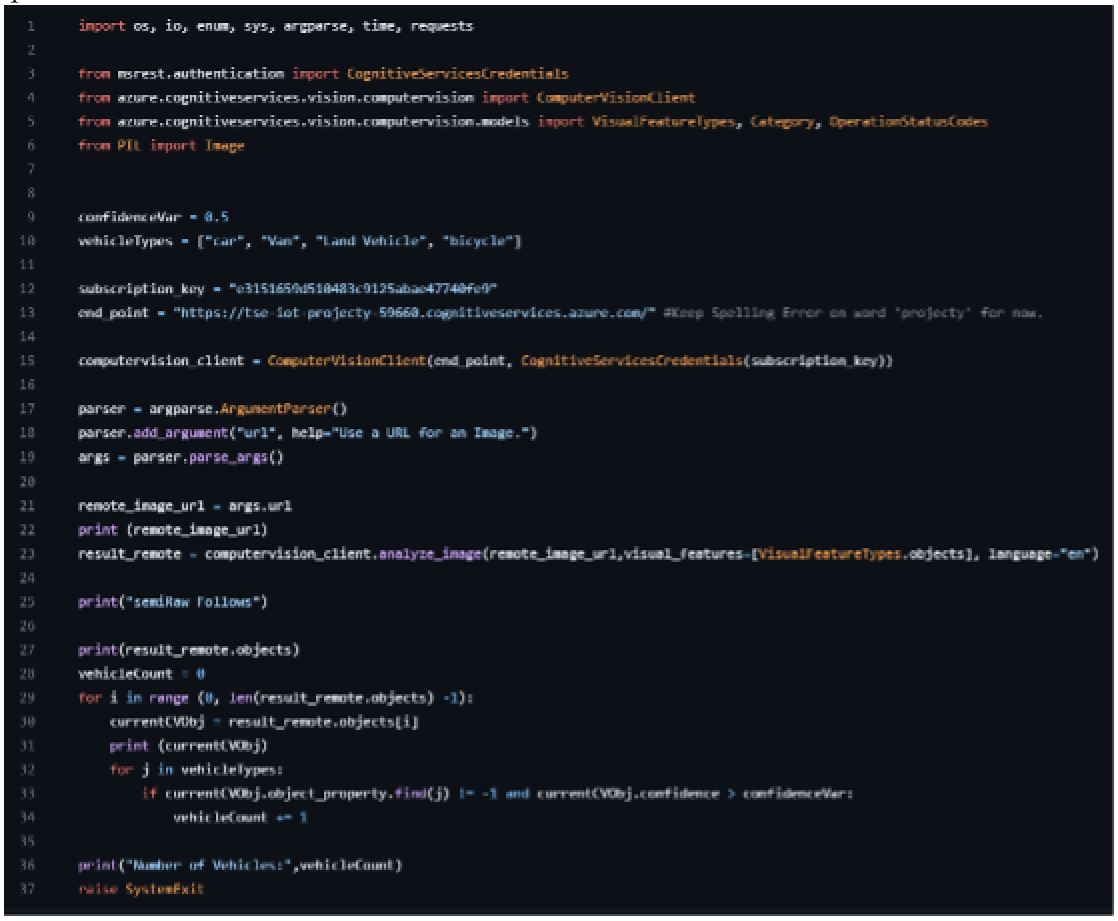
```
blob_client = container_client.get_blob_client(image_names)
blob_client.upload_blob(IMAP4_stream, overwrite=True)
```

One of the challenges faced was the latency in image capture and upload due to network constraints. To mitigate this, we implemented a queue system that temporarily stored the images on the Raspberry Pi, allowing them to be uploaded in the background, thereby not impeding the image capture process.

Despite the challenges encountered, this project provided us with a valuable learning opportunity, especially in terms of problem-solving and adapting to unforeseen circumstances.

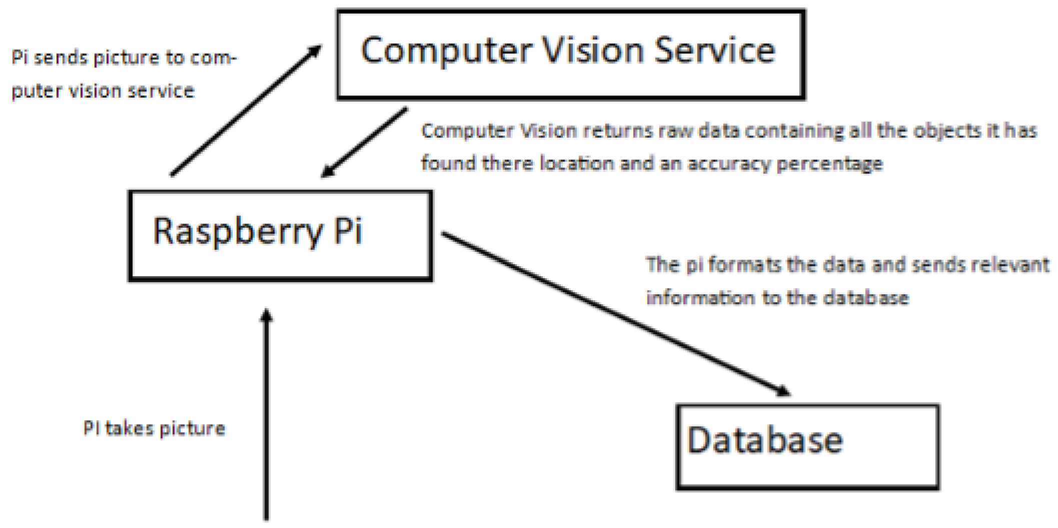
2.3.2 Blob Trigger / Computer Vision

We began building our cloud infrastructure by deploying a recourse group TSE_Project_IOT. After familiarizing ourselves with the azure portal we decided to deploy a Computer Vision resource called TSE_IOT_PROJECTY-59960. We elected to choose Computer Vision services compared to other image processing services like image analysis or optical character recognition because it capable of "image understanding" – (azure.microsoft.com, n.d) making it able to recognise concepts and images such as cars, had good documentation and was therefore easy to learn for new developers and was scalable so dependant on subscription could service one Pi to a traffic network worth of Pis. After some trial and error, we were able to build code that could upload an image URL and count the number of objects categorised as "car", "van", "Land Vehicle", "bicycle" with a confidence value of 0.5 or higher. This code is referred to as test_run_2.py on our GitHub repository and can decode the raw data returned by azure computer vision services.



```
import os, io, enum, sys, argparse, time, requests

from msrest.authentication import CognitiveServicesCredentials
from azure.cognitiveservices.vision.computervision import ComputerVisionClient
from azure.cognitiveservices.vision.computervision.models import VisualFeatureTypes, Category, OperationStatusCodes
from PIL import Image


confidenceVar = 0.5
vehicleTypes = ["car", "Van", "Land Vehicle", "bicycle"]

subscription_key = "e3151659d510483c9125abae47740fe9"
end_point = "https://tse-iot-projecty-59660.cognitiveservices.azure.com/" #Keep Spelling Error on word 'projecty' for now.

computervision_client = ComputerVisionClient(end_point, CognitiveServicesCredentials(subscription_key))

parser = argparse.ArgumentParser()
parser.add_argument("url", help="Use a URL for an Image.")
args = parser.parse_args()

remote_image_url = args.url
print (remote_image_url)
result_remote = computervision_client.analyze_image(remote_image_url,visual_features=[VisualFeatureTypes.objects], language="en")

print("semiRaw Follows")

print(result_remote.objects)
vehicleCount = 0
for i in range (0, len(result_remote.objects) -1):
    currentCVObj = result_remote.objects[i]
    print (currentCVObj)
    for j in vehicleTypes:
        if currentCVObj.object_property.find(j) != -1 and currentCVObj.confidence > confidenceVar:
            vehicleCount += 1

print("Number of Vehicles:",vehicleCount)
raise SystemExit
```

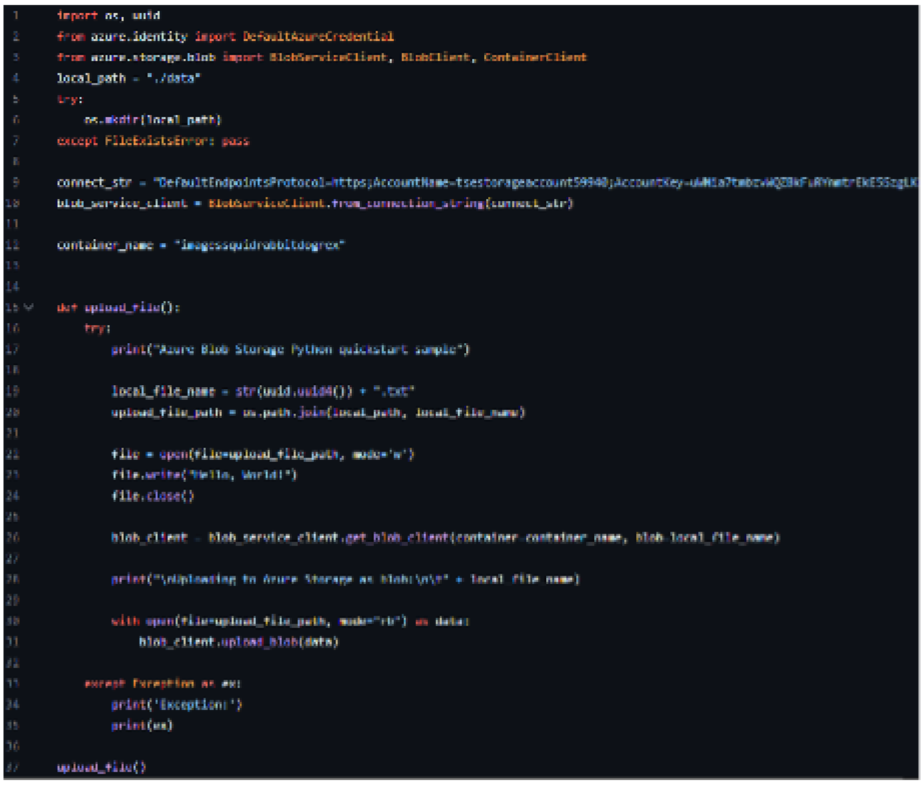
After this success we began considering the architecture of our artefact here is a representation of one of the first suggestions:


Computer Vision Service
Pi sends picture to computer vision service
Computer Vision returns raw data containing all the objects it has found there location and an accuracy percentage
Raspberry Pi
The pi formats the data and sends relevant information to the database
PI takes picture
Database

This suggestion had some flaws the main one being the Pi would do all the data processing and would have to upload to various cloud services twice as well as receive data from the cloud. We decided to improve upon this design as when the Pi is deployed it will likely have a poor internet connection and increasing the computational demands of a Pi would reduce its lifespan and increase replacement costs.

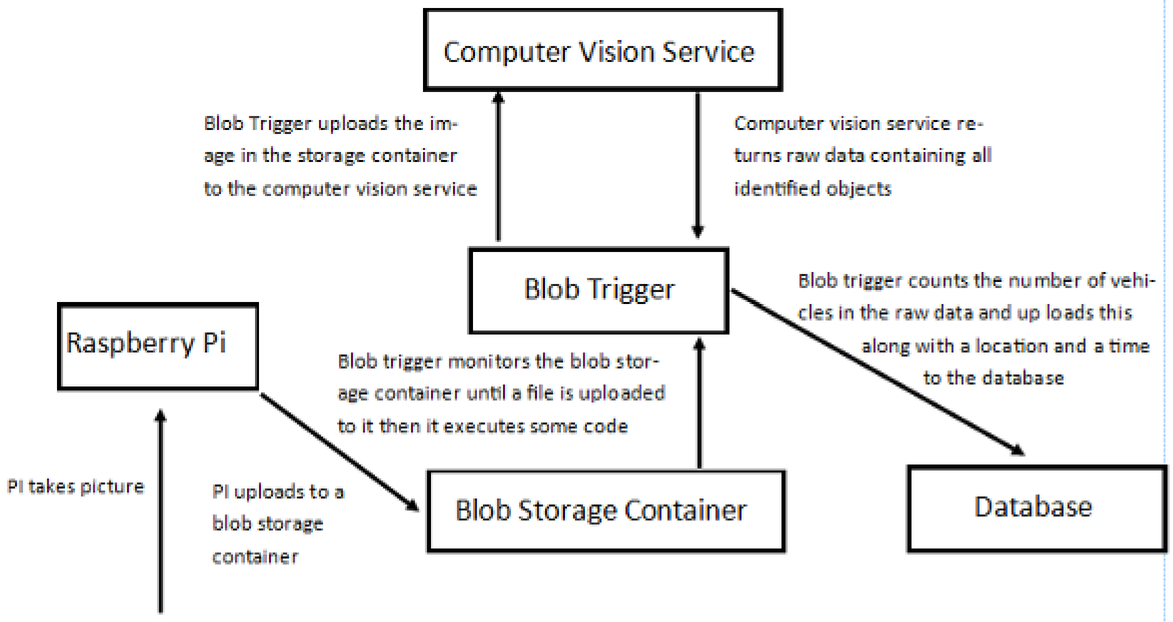
After further research the group decided the pi should upload to an initial storage service and a virtually hosted function to feed images from the storage service to the computer vision service and data to the database. We decided to use a Blob storage container we named imagessquidrabbitdogrex located in the tsestorageaccount59940, at this point we thought azure services had to have unique names which is why these names are not streamlined. We elected to use a Blob storage container as it was cheap to host costing our customers nothing with the downsides being its "unorganised data structure" – (azure.microsoft.com, n.d) meaning it is harder to query, it has a limited storage capacity, and it Is not a viable long term storage option with "azure blob containers storing blobs for a maximum of nine days" – (azure.microsoft.com, n.d). These downsides do not affect our artefact as the blob storage container is supposed to be temporary storage.

Eventually code was produced to upload a text file containing "hello world" to our blob storage container called *Python_Bucket_Uploader.py* on our GitHub



```python
import os, uuid
from azure.identity import DefaultAzureCredential
from azure.storage.blob import BlobServiceClient, BlobClient, ContainerClient
local_path = "./data"
try:
    os.mkdir(local_path)
except FileExistsError: pass

connect_str = "DefaultEndpointsProtocol=https;AccountName=tsestorageaccount59940;AccountKey=uWN1a7tmbzWQEBkFuRYmtrEkESSzgLK
blob_service_client = BlobServiceClient.from_connection_string(connect_str)

container_name = "imagessquidrabbitdogrex"


def upload_file():
    try:
        print("Azure Blob Storage Python quickstart sample")

        local_file_name = str(uuid.uuid4()) + ".txt"
        upload_file_path = os.path.join(local_path, local_file_name)

        file = open(file=upload_file_path, mode='w')
        file.write("Hello, World!")
        file.close()

        blob_client = blob_service_client.get_blob_client(container=container_name, blob=local_file_name)

        print("\nUploading to Azure Storage as blob:\n\t" + local_file_name)

        with open(file=upload_file_path, mode="rb") as data:
            blob_client.upload_blob(data)

    except Exception as ex:
        print('Exception:')
        print(ex)

upload_file()
```

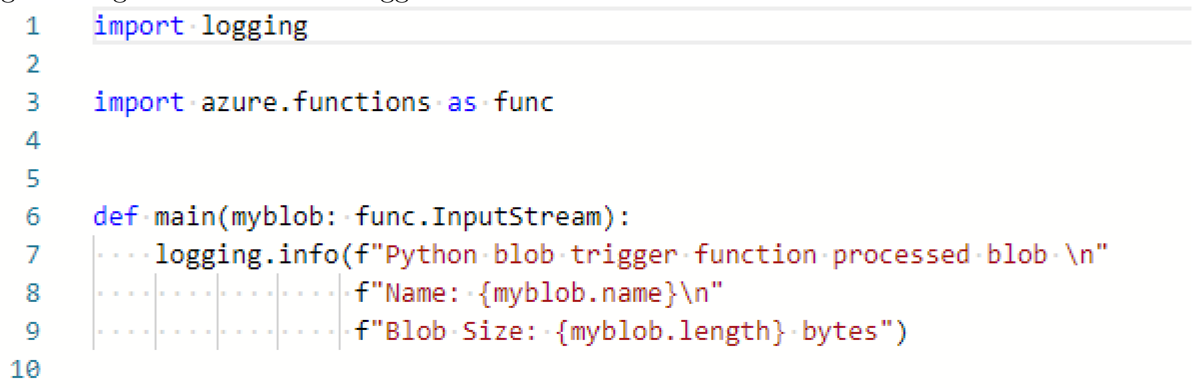
We now had a blob container working and after a bit of research and trial we decided to deploy a blob trigger a blob trigger has three advantages in our artefact first its low cost to maintain compared to alternatives like hosting a virtual machine to run code, second it automatically monitors a blob storage container and will trigger upon a new blob being uploaded so we do not have to produce additional code to monitor the blob storage container, Finally despite its limited processing power it will be perfectly capable of executing the previously made vision code. Our theoretical model now looked like this:



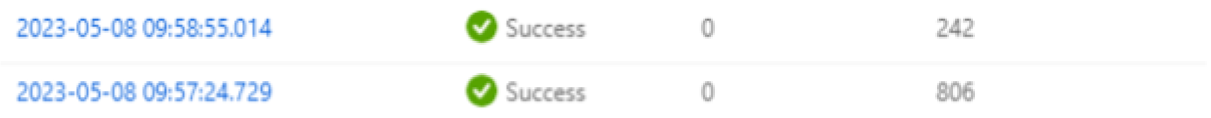
Computer Vision Service
Blob Trigger uploads the image in the storage container to the computer vision service
Computer vision service returns raw data containing all identified objects
Blob Trigger
Blob trigger counts the number of vehicles in the raw data and up loads this along with a location and a time to the database
Raspberry Pi
Blob trigger monitors the blob storage container until a file is uploaded to it then it executes some code
PI takes picture
PI uploads to a blob storage container
Blob Storage Container
Database

First, we had built a function app however hosting a function app using python is currently cannot use a windows-based operating system and had to be hosted via Linux the other alternative was we use .Net framework and program in C# which can use windows. Eventually we decided to use python because we already had built the existing computer vision code in python, and we had grown more confident in our Linux skills from deploying the Pi. We created FunctionApp3LinuxPY which contained BlobTrigger1.

After successfully deploying these resources, we were able to monitor our first successful triggers using the default blob trigger code.

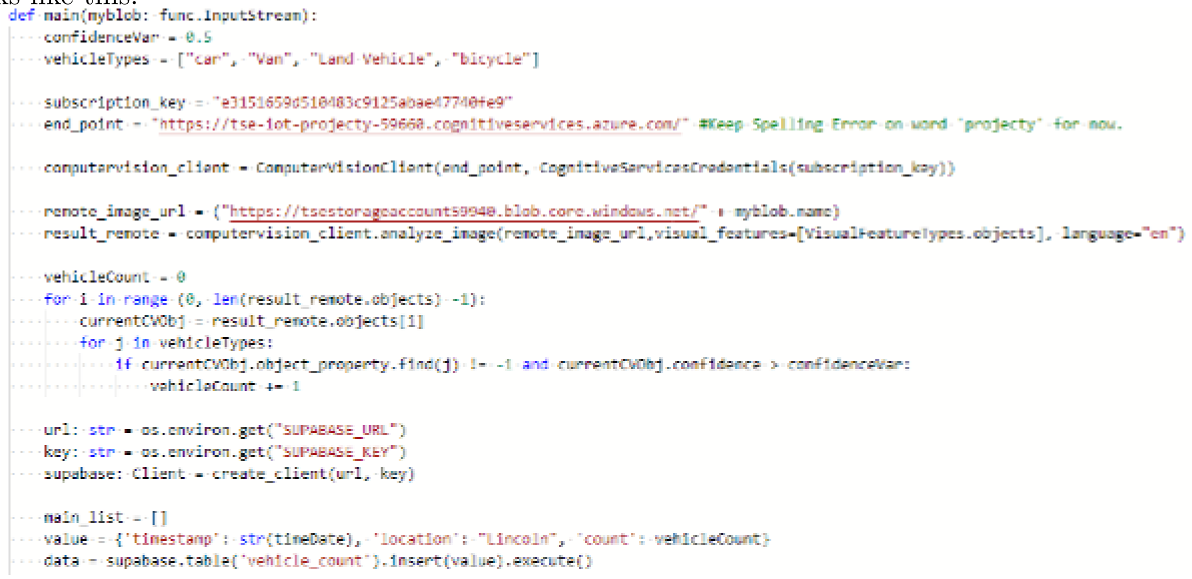


```python
import logging

import azure.functions as func


def main(myblob: func.InputStream):
    logging.info(f"Python blob trigger function processed blob \n"
                 f"Name: {myblob.name}\n"
                 f"Blob Size: {myblob.length} bytes")

```



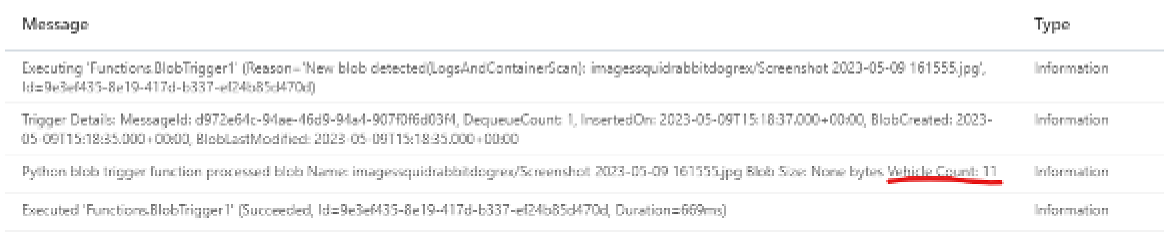
2023-05-08 09:58:55.014	✓ Success	0	242
2023-05-08 09:57:24.729	✓ Success	0	806

The final step was to add our computer vision code to the trigger and after we figured out how to install a requirements file from our GitHub containing the relevant libraries our code looks like this:



```python
def main(myblob: func.InputStream):
    confidenceVar = 0.5
    vehicleTypes = ["car", "Van", "Land Vehicle", "bicycle"]

    subscription_key = "e3151659d510483c9125abae47740fe9"
    end_point = "https://tse-iot-projecty-59660.cognitiveservices.azure.com/" #Keep Spelling Error on word "projecty" for now.

    computervision_client = ComputerVisionClient(end_point, CognitiveServicesCredentials(subscription_key))

    remote_image_url = ("https://tsestorageaccount59940.blob.core.windows.net/" + myblob.name)
    result_remote = computervision_client.analyze_image(remote_image_url,visual_features=[VisualFeatureTypes.objects], language="en")

    vehicleCount = 0
    for i in range (0, len(result_remote.objects) -1):
        currentCVObj = result_remote.objects[1]
        for j in vehicleTypes:
            if currentCVObj.object_property.find(j) != -1 and currentCVObj.confidence > confidenceVar:
                vehicleCount += 1

    url: str = os.environ.get("SUPABASE_URL")
    key: str = os.environ.get("SUPABASE_KEY")
    supabase: Client = create_client(url, key)

    main_list = []
    value = {'timestamp': str(timeDate), 'location': "Lincoln", 'count': vehicleCount}
    data = supabase.table('vehicle_count').insert(value).execute()
```

And the code counts vehicles:

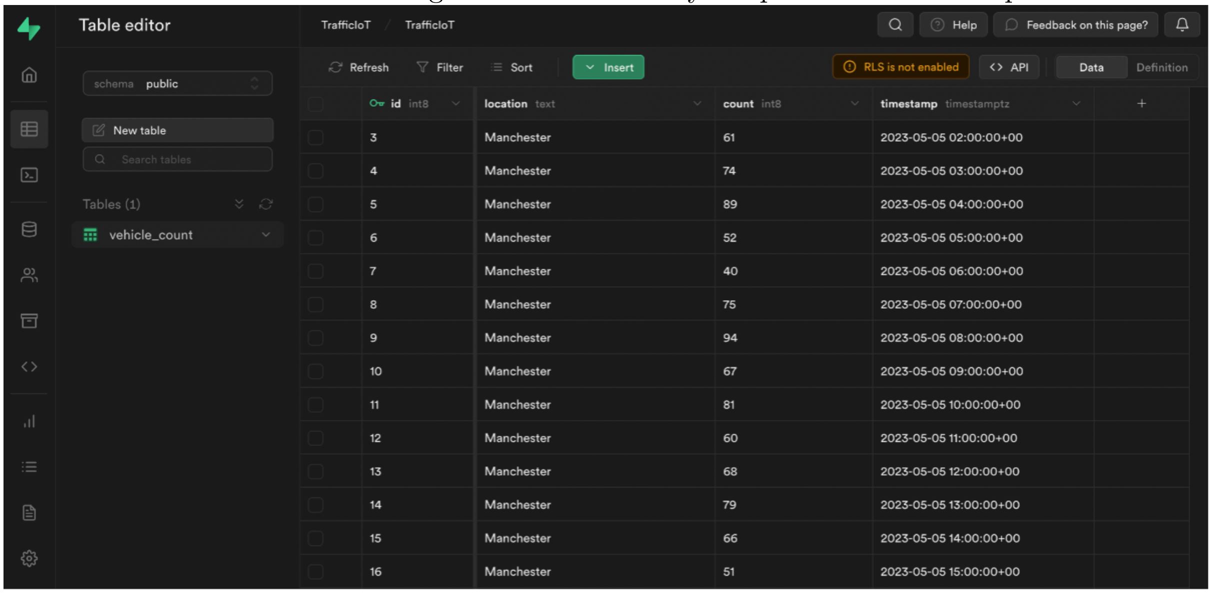


Message	Type
Executing 'Functions.BlobTrigger1' (Reason='New blob detected(LogsAndContainerScan): imagessquidrabbitdogrex/Screenshot 2023-05-09 161555.jpg', Id=9e3ef435-8e19-417d-b337-ef24b85d470d)	Information
Trigger Details: MessageId: d972e64c-94ae-46d9-94a4-907f0f6d03f4, DequeueCount: 1, InsertedOn: 2023-05-09T15:18:37.000+00:00, BlobCreated: 2023-05-09T15:18:35.000+00:00, BlobLastModified: 2023-05-09T15:18:35.000+00:00	Information
Python blob trigger function processed blob Name: imagessquidrabbitdogrex/Screenshot 2023-05-09 161555.jpg Blob Size: None bytes Vehicle Count: 11	Information
Executed 'Functions.BlobTrigger1' (Succeeded, Id=9e3ef435-8e19-417d-b337-ef24b85d470d, Duration=669ms)	Information

When deploying this code, It is recommended to build an individual blob storage container per raspberry pi camera and to edit the variables confidenceVar and vehicleTypes to represent how accurate your system must be as well as what vehicles you are looking for.

2.4 Database

We have considered two database options: Postgres and MySQL. We chose Postgres as it provides role-level security out of the box as well as being able to handle more requests using the same resources. Supabase was chosen as a cloud Postgres database solution. It provides a managed database instance running on AWS, and easy to use management UI, allowing users to administrate the database from any device. It also provides API and libraries for multiple languages (Python, JavaScript, and others) to access the database easily through the API. Administrators can create users and give them access only to specific tables and operations.



id int8	location text	count int8	timestamp timestamptz
3	Manchester	61	2023-05-05 02:00:00+00
4	Manchester	74	2023-05-05 03:00:00+00
5	Manchester	89	2023-05-05 04:00:00+00
6	Manchester	52	2023-05-05 05:00:00+00
7	Manchester	40	2023-05-05 06:00:00+00
8	Manchester	75	2023-05-05 07:00:00+00
9	Manchester	94	2023-05-05 08:00:00+00
10	Manchester	67	2023-05-05 09:00:00+00
11	Manchester	81	2023-05-05 10:00:00+00
12	Manchester	60	2023-05-05 11:00:00+00
13	Manchester	68	2023-05-05 12:00:00+00
14	Manchester	79	2023-05-05 13:00:00+00
15	Manchester	66	2023-05-05 14:00:00+00
16	Manchester	51	2023-05-05 15:00:00+00

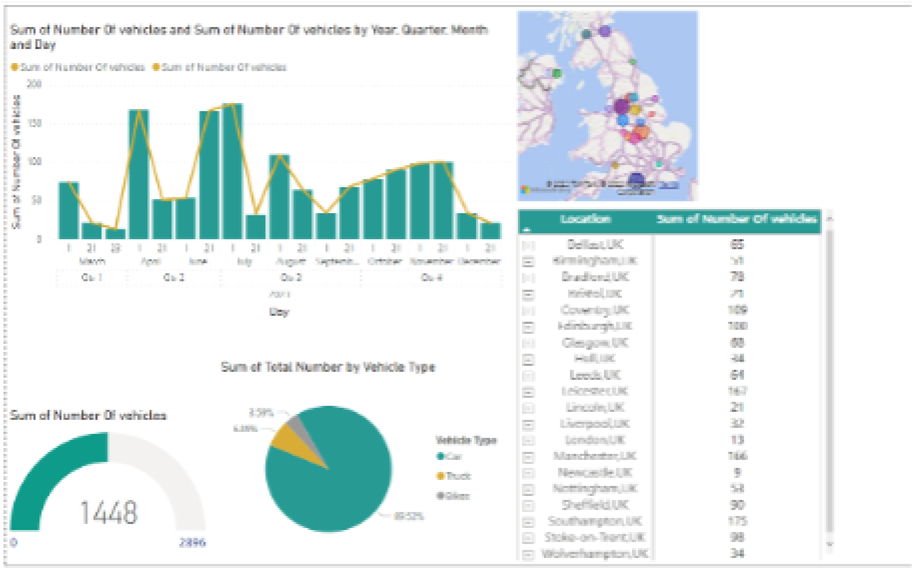
Supabase database management UI

In the current prototype, the database contains one table storing records about the number of cars recognised on each image at a given time. After analysing data by the AI model, a script uploading the result to the database is triggered. On the other end, the dashboard is connected to the database through the Supabase library. Then these records are visualised on the dashboard.

2.4.1 Dashboard

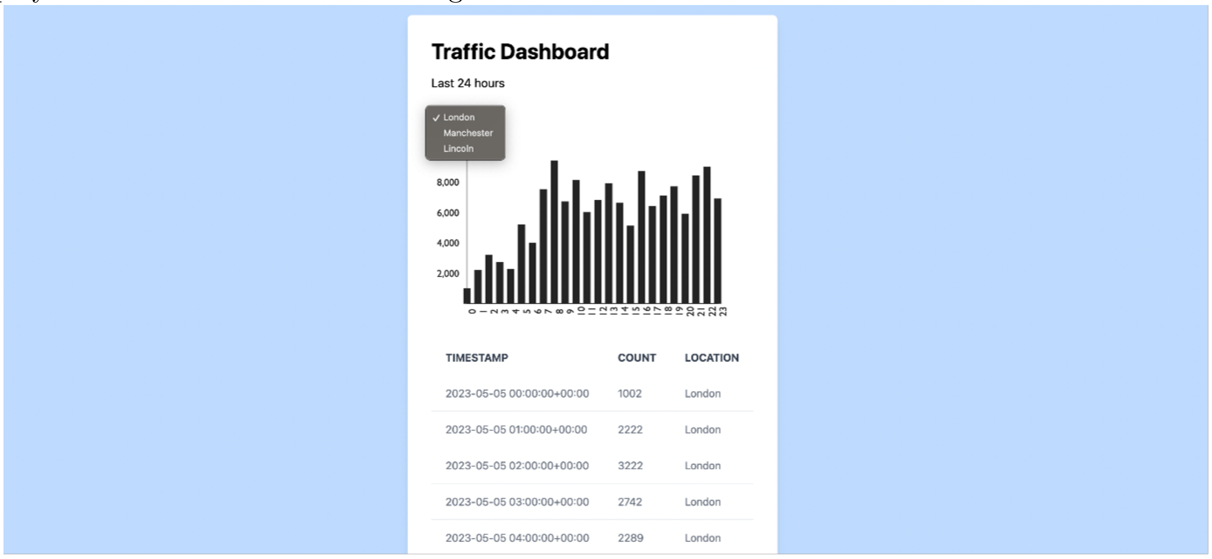
The initial dashboard prototype was developed using the Power BI platform. To create this prototype, a test dataset was set up in an Excel table, which contained ordered data on the location, type, and time of each vehicle. The table was then connected to the Power BI dashboard. The main aim of the dashboard is to present data in a visual format, allowing for easier analysis and report creation. The Power BI platform grouped the data by type and presented it in graphical form, making it easier to visualize. In addition, a bubble chart on a map of the UK was included to help visualize the relative density of vehicles in different regions.

First prototype of the dashboard using the Power BI platform


Sum of Number Of vehicles and Sum of Number Of vehicles by Year, Quarter, Month and Day
Sum of Number Of vehicles
Sum of Number Of vehicles
Sum of Number Of vehicles
200
150
100
50
0
March
April
June
July
August
September
October
November
December
Qtr 1
Qtr 2
Qtr 3
Qtr 4
2023
Day
Sum of Total Number by Vehicle Type
3.59%
6.89%
89.52%
Vehicle Type
● Car
● Truck
● Other
Sum of Number Of vehicles
1448
0
2896

Location	Sum of Number Of vehicles
Belfast,UK	65
Birmingham,UK	51
Bradford,UK	78
Bristol,UK	71
Coventry,UK	109
Edinburgh,UK	100
Glasgow,UK	68
Hull,UK	44
Leeds,UK	64
Leicester,UK	167
Lincoln,UK	21
Liverpool,UK	32
London,UK	13
Manchester,UK	166
Newcastle,UK	9
Nottingham,UK	58
Sheffield,UK	90
Southampton,UK	175
Stoke-on-Trent,UK	98
Wolverhampton,UK	34

However, we encountered problems with compatibility between the database and the dashboard, which required additional installation and licensing. Due to time constraints, a new dashboard web-application was developed with fewer features, although it still allowed for the display of vehicle locations and filtering.



TIMESTAMP	COUNT	LOCATION
2023-05-05 00:00:00+00:00	1002	London
2023-05-05 01:00:00+00:00	2222	London
2023-05-05 02:00:00+00:00	3222	London
2023-05-05 03:00:00+00:00	2742	London
2023-05-05 04:00:00+00:00	2289	London

The final dashboard using Next.js under the hood

The dashboard web-application could be done in simple HTML and a bit of JavaScript, but after additional research, Next.js was chosen for the dashboard engine as it is way more powerful and provides a solid foundation for future development. Tailwind was chosen instead of raw CSS to develop the UI and make it responsive quickly. React hooks allowed us to improve user experience by implementing real-time updates from the database without the need to refresh the page. Vercel is used to host the dashboard. It leverages CDN (Content Delivery Network) to deliver the page fast in any part of the world. Also, Vercel provides CI for automatic building and deploying the dashboard which we successfully used in the project. This project provided valuable insights into the importance of establishing compatibility before proceeding with specific design and functionality.

2.4.2 Conclusion

In conclusion, the implementation of our traffic monitoring system demonstrates the transformative power of the Internet of Things (IoT) combined with cloud computing. Despite the challenges faced, such as hardware availability, technical issues, and latency concerns, we were able to design and build a system that effectively captures and processes real-time traffic data.

Overall, this project not only taught us valuable problem-solving skills but also demonstrated the significant potential of IoT and cloud computing in real-world applications. It provided us with a scalable and adaptable framework that can be extended for other similar IoT applications, showcasing the potential of such systems in improving urban life and infrastructure planning.

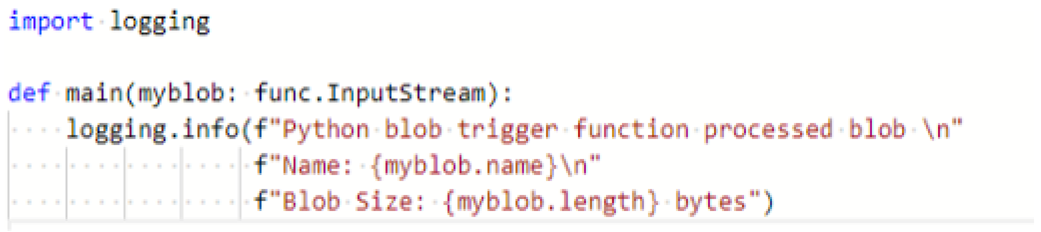
3 Testing:

Throughout the implementation of the prototype, we ensured testing had been carried out, this is so all requirements of the prototype are met, and no unexpected results would be produced. One testing method that we implemented in our project was called Black Box testing. Black box testing assesses the functionality of an application with needing the knowledge of the internal structure of the project itself. Black box main focuses on the input and output of the software application.

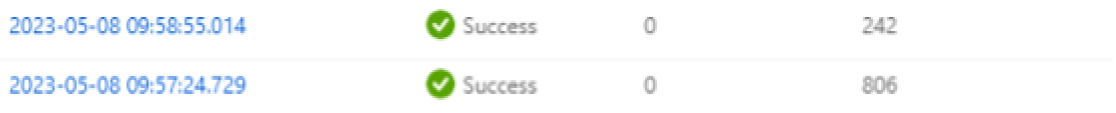
We can split Black Box testing into two subcategories functional and non-functional testing. Functional testing relates to the functional requirements of the software, whereas, non-functional relates to the performance, scalability, and usability. Functional testing includes smoke testing, integration testing, end to end testing, regression testing and user acceptance testing, whereas, non-functional testing consists of Usability testing, Load testing, Stress testing and Scalability testing.

Smoke testing verifies whether the core features of the prototype are as expected. Through this testing we were able to check to see if the system can be installed and configured without any issues and then validate that that the system can perform basic operations without errors. This test indicates whether our prototype is ready for further testing or if the design is flawed. It allowed us to evaluate the prototype and allowed to make changes if any needed to be made. Integration testing focuses on checking the data communication amongst the modules. Within integration testing there are many testing methods that we were able to choose from, however, the method we decided to advance with was Incremental testing. Incremental testing allows us to check the integrity of the module as soon as they have been developed. This is so if one module has not yet been created we are still able to check other modules that have been created. This informs us step by step which part of the module is working or causing problems allowing us to go back take the necessary steps to avoid these issues.

First test just logging if the things connected (testing whether blob trigger is connected to the blob):

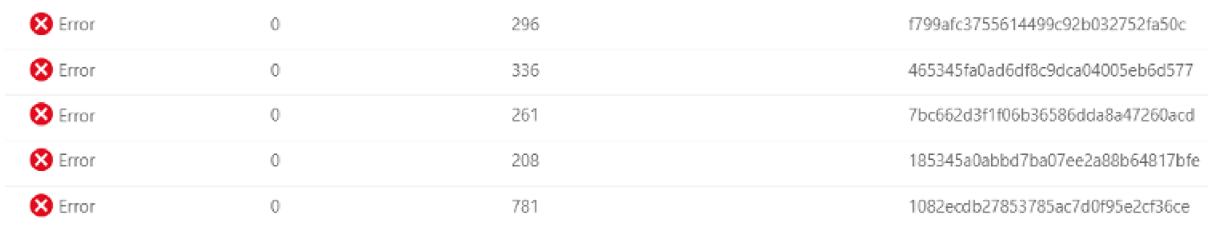


```python
import logging

def main(myblob: func.InputStream):
    logging.info(f"Python blob trigger function processed blob \n"
                 f"Name: {myblob.name}\n"
                 f"Blob Size: {myblob.length} bytes")
```



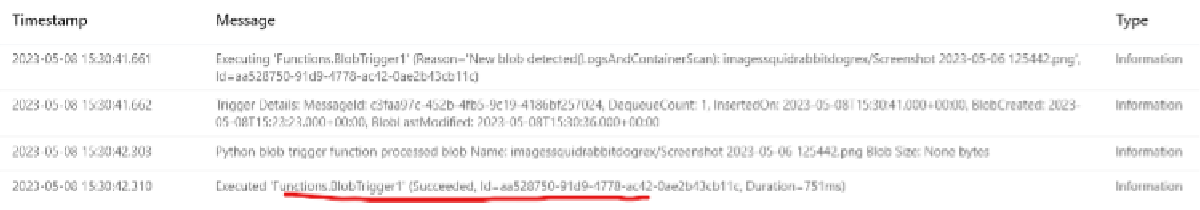
```
2023-05-08 09:58:55.014    Success    0    242
2023-05-08 09:57:24.729    Success    0    806
```

Second test using the cognitive vision code (import the python code, errors occurred from uninstalled libraries): See Fig3



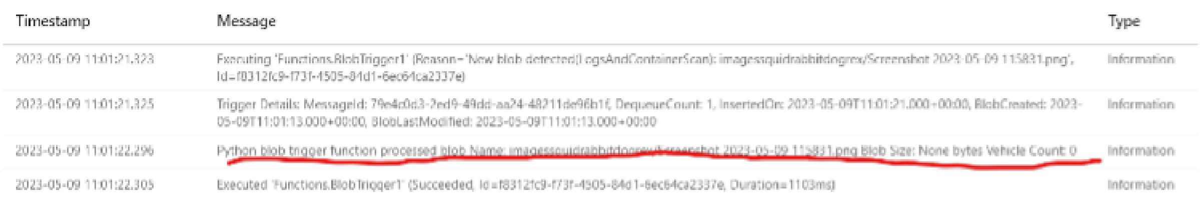
✖ Error	0	296	f799afc3755614499c92b032752fa50c
✖ Error	0	336	465345fa0ad6df8c9dca04005eb6d577
✖ Error	0	261	7bc662d3f1f06b36586dda8a47260acd
✖ Error	0	208	185345a0abbd7ba07ee2a88b64817bfe
✖ Error	0	781	1082ecdb27853785ac7d0f95e2cf36ce

Error from unimported libraries. After importing Libraries:



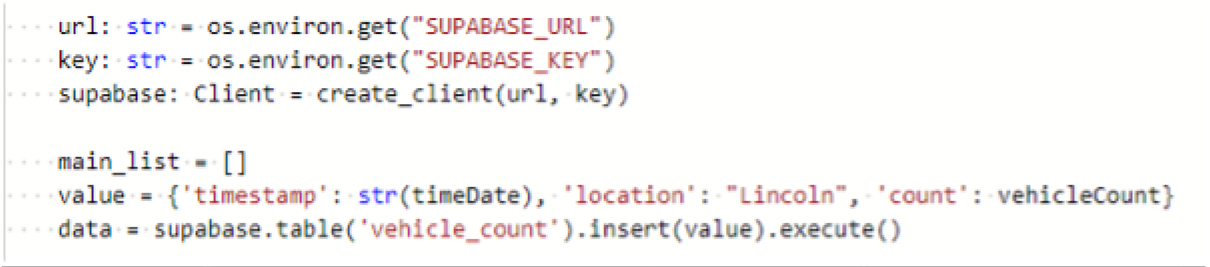
Timestamp	Message	Type
2023-05-08 15:30:41.661	Executing 'Functions.BlobTrigger1' (Reason='New blob detected(LogsAndContainerScan): imagessquidrabbitdogrex/Screenshot 2023-05-06 125442.png', Id=aa528750-91d9-4778-ac42-0ae2b43cb11c)	Information
2023-05-08 15:30:41.662	Trigger Details: MessageId: c3faa97c-452b-4fb5-9c19-4186bf257024, DequeueCount: 1, InsertedOn: 2023-05-08T15:30:41.000+00:00, BlobCreated: 2023-05-08T15:23:23.000+00:00, BlobLastModified: 2023-05-08T15:30:36.000+00:00	Information
2023-05-08 15:30:42.303	Python blob trigger function processed blob Name: imagessquidrabbitdogrex/Screenshot 2023-05-06 125442.png Blob Size: None bytes	Information
2023-05-08 15:30:42.310	Executed 'Functions.BlobTrigger1' (Succeeded, Id=aa528750-91d9-4778-ac42-0ae2b43cb11c, Duration=751ms)	Information

Third test superbase code integrated and improved logging (successfully downloaded libraries so the python program can run, the code that uploads the superbase code now is functional):

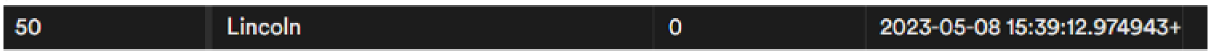


Timestamp	Message	Type
2023-05-09 11:01:21.323	Executing 'Functions.BlobTrigger1' (Reason='New blob detected(LogsAndContainerScan): imagessquidrabbitdogrex/Screenshot 2023-05-09 115831.png', Id=f8312fc9-f73f-4505-84d1-6ec64ca2337e)	Information
2023-05-09 11:01:21.325	Trigger Details: MessageId: 79e4d0d3-2ed9-49dd-aa24-48211de96b1f, DequeueCount: 1, InsertedOn: 2023-05-09T11:01:21.000+00:00, BlobCreated: 2023-05-09T11:01:13.000+00:00, BlobLastModified: 2023-05-09T11:01:13.000+00:00	Information
2023-05-09 11:01:22.296	Python blob trigger function processed blob Name: imagessquidrabbitdogrex/Screenshot 2023-05-09 115831.png Blob Size: None bytes Vehicle Count: 0	Information
2023-05-09 11:01:22.305	Executed 'Functions.BlobTrigger1' (Succeeded, Id=f83121c9-f73f-4505-84d1-6ec64ca2337e, Duration=1103ms)	Information

In superbase:



```
    url: str = os.environ.get("SUPABASE_URL")
    key: str = os.environ.get("SUPABASE_KEY")
    supabase: Client = create_client(url, key)

    main_list = []
    value = {'timestamp': str(timeDate), 'location': "Lincoln", 'count': vehicleCount}
    data = supabase.table('vehicle_count').insert(value).execute()
```



50	Lincoln	0	2023-05-08 15:39:12.974943+

The person who generated the error would initially take the blame and try to correct their error, however if the person did not have certain expertise in an error they were then allowed to confer with other members of the team for example in the images above one member didn't know how to import pip libraries to a virtual azure function. However, after asking for aide from other team members they were successful able to install the requirements.txt file from GitHub.

Computer vision testing data log: Table. 2

End to end testing allowed to check the validity of the entire prototype. By using end to end we can also validate the processing of the data from the start to the end. To ensure all aspects of the project were met we created a list of features that should be present within the prototype checking them off as the testing was carried out. By carrying out this test we are able to validate or not whether the has the capability of handling real-world scenarios.

Regression testing is a type of software testing that confirms that a recent program or code change has not adversely affected any of the other features present. This is to ensure that new code does not have any adverse side affects on the existing functionality. Regression testing is needed to determine how new code affects the pre-existing code and other parts of the software application. To ensure that we used regression testing appropriately we performed this test each time a functional defect was fixed, ensuring the code implemented to fix the issue wasn't going to affect any other parts of the software application.

User Acceptance testing is a type of testing that is performed by the end client to verify the integrity of the software system before moving the software application towards a commercial setting. The main purpose of carrying out this testing is to verify if the software application is ready for commercial use. Now in our case it would be to see if the prototype we have made is commercially feasible or not. It's not like the other testing strategies that I have explain thus far, this kind of testing focuses on where end users are heavily involved. This is to ensure that developers had included features on their own understanding and that meets the requirements of the client who is wanting to use the software.

Now we move onto the non-functional testing part of black box testing. To start with we used Usability testing. This testing focuses heavily on user friendliness. To be able to fully take advantage of this testing we introduced two individuals who were familiar with software themselves to see if they were able to understand and navigate their way through it without facing any trouble. Here are the comments they made after having tested the software prototype.

The design presented was very easy to interpret, I enjoyed the visual graphical data showing the course of vehicles across a period of time.
- Individual 1

The GUI presented was very user friendly, it was easy to read the data presented to me on the dash, and I was easily able to monitor the different kind of vehicles across a period of time
- Individual 2

Load testing is testing process in which the performance of the software application is tested under a specific yet realistic load. It helps us to determine how the application behaves if multiple requests are being called upon or if the software is being use by multiple users. The end goal of this test is to ensure stability and smooth functioning of the software application before being deployed. The test helps us to identify what the software's limits are and whether the current infrastructure we have is sufficient enough to run the application.

Stress testing is a software testing that verifies the integrity and stability of a software application. The goal of stress testing is to measure how well error handling capabilities are under heavy conditions ensuring the software does not crash. We implemented this type of testing to see if under extreme conditions of many updates occurring at once can all parts of the prototype keep up with each other.

The final test within the non-functional testing is called scalability testing. This method measures the performance of a system when the number of user requests are scaled up or down. The purpose of this test is to ensure that system can handle projected increase in user traffic and data volume. Performing this test allowed to evaluate how the prototype handles increased traffic and how well it can handle scaling up, it also allowed to observe the robustness and degradation of the application itself. Through doing this we are able to how the project handles response time, screen transition and throughput.

By implementing these tests, we were able to create a functional prototype that meets all the needed requirements.

4 Release

For the release, the decision was made to use GitHub as the primary point of release for the code, with 'README.MD' files being included to instruct the implementor on installation. This decision was primarily based on two factors, one being that we were all familiar with GitHub & two being that as much of the implementation was based on code that would need to be directly input into the correct locations, it made more sense to provide the code directly.

Additionally, the usage of GitHub allows for issues to be raised in the form of 'Discussions' or as 'Pull Requests', with the latter also offering the ability for end-users to offer fixes or feature suggestions or complete 'forks' of the code.

Furthermore, the project is current licensed under the 'GPL v3' license, meaning that as of present, it is considered Open-Source. However, owing to the project being considered pre-release and not completely polished, that license is liable to be changed in the future. However, as the repository is set to private, owing to a few authentication strings being exposed, raw, in the source-code as well as the usage of Azure and Supabase (AWS) Services, which unfortunately due to the complexity of the licensing mean that we are unsure as to which license the code may be properly released under.

Also, via the use of GitHub for the release of the solution, we are able to allow users to exam the source-code, thus allowing any glaring security issues to be identified. Additionally, by using 'README' files, we are able to provide instructions that can be parsed both in Markdown and plain-text text editors.

5 Evaluation:

While this project successfully achieved its primary objective of monitoring traffic flow, there are opportunities for improvement in certain areas, and the adoption of alternative methods can enable us to operate more efficiently, producing more accurate results with a broader range of features. In this regard, if we were to undertake future projects or had additional time, we would consider implementing the following enhancements to the project:

Data Categorisation: The Azure Computer Vision Cognitive Service recognises each vehicle as a separate brand, which impedes our code's ability to recognise it as a vehicle, therefore, the data is not added. To address this issue, we propose creating a file containing a list of brands that should be classified under specific categories such as cars, trucks, bikes, and other vehicle types. This would enable us to group all the output into categories, making the data more manageable for our dashboard and database. Although the Azure Computer Vision Cognitive Service is a powerful tool, it requires careful customisation to ensure effective deployment for our project.

Image Streaming: The initial goal of the project was to capture an image when the traffic light changed colours. Consequently, the camera only took a picture after receiving input from the keyboard. However, we propose to capture a continuous stream of images throughout the day, without any gaps, to monitor traffic flow patterns accurately. Achieving this would necessitate establishing a timestamp for the intervals between each picture to capture every vehicle that passes through the road. With a continuous stream of images, the system can provide more precise data, enhancing its analysis and reporting capabilities.

Custom hardware: Custom hardware can be developed and manufactured instead of the Raspberry pi to fit the project's precise needs resulting in a highly optimised system having more performance and being more cost-efficient.

Self-hosted AI: Azure Computer Vision Cognitive Service is a good solution for prototyping. In production, it would be better to self-host the AI model. This approach will require more effort but will result in having more control over the model, is more secure, and is cheaper. We are planning to train the AI model in the cloud and then self-host the model on each device. This will save vast amounts of resources drastically decreasing in the bandwidth and storage used and making it more secure as we don't have to send the images over the network – we analyse it on the device and send only the text data. Furthermore, we don't need to use such services as Azure Computer Vision Cognitive Service which will save the budget.

Database: Supabase is good for prototyping and admin panel with easy-to-use UI. But this service is the "man in the middle" as it uses AWS Postgres database instance under the hood, which rises some security concerns, and results in increased latency as well. Supabase is an open-source solution. This means that we can use our own AWS, GCP, or a self-hosted database, and deploy Supabase opensource admin panel. In this way, we will have full control over the database ensuring that all the data is safe.

Dockerization: All the components can be dockerized to integrate and streamline the entire software development lifecycle (development, testing, and deployment processes), providing a cohesive and efficient approach for the development of a large-scale system in the future.

Testing: 80% of the code of each component can be covered with the unit and automated functional tests. The tests can be triggered by the CI to automatically check if the code works as expected and immediately return the report to the developer.

Kubernetes cluster: A helm chart can be developed to define the deployment which can be running on a Kubernetes cluster, for example. Kubernetes will control the pods to be healthy and will efficiently scale them depending on the load. Furthermore, it will provide easy rollout updates, so that no downtime will be for users when the product is updating.

Development automation: Having implemented the previously discussed improvements, we will be able to automate the whole software development lifecycle. The flow is the following: Developer pushes code to his branch, CI is triggered by that commit, and if tests are passing, CI gives the developer the green light. After that, the feature can be merged into the main branch. After the merge, another flow is activated: tests – docker image build – tagging the build with version - updating the Kubernetes cluster with a new image. This flow can be integrated into each of our components.

GPS Integration: Adding a GPS will enable automatic detection of the camera's location, making it more portable and easier to move as needed. Currently, the camera's location must be manually inputted into the system, making it more time-consuming if another location needs to be monitored.

Extension modules: A solar panel and GSM modules together with a built-in GPS module make the device fully autonomous, easy to install, and easy to use. The solar-panel module with a battery ensures a continuous and renewable energy supply when a GSM module provides a network connection without any wires.

Image cache: We intend to implement a backup feature for recent images taken by the device. If the connection is unstable or lost, the photos taken will be stored on the device. After the connection is renewed, all the images are sent to the centralised storage. Furthermore, we plan to introduce an alert feature that will notify the user in case of a loss of connection or any other system issues, enabling prompt action to be taken.

Compatibility: The dashboard was able to display the vehicles in different locations and allowed filtering, but it did not provide the same number of features as the original prototype. For future projects, we intend to establish compatibility before continuing with the specific design and functionality to ensure that the dashboard and database are fully compatible with each other.

Security: Authorisation can be added to the dashboard and endpoints to make the connection private and secure. Introducing encryption with TLS certificates is another crucial step in ensuring the project's security. This will safeguard the project's integrity and protect sensitive data.

In conclusion, by implementing these improvements across the entire project infrastructure, we can significantly enhance not only the project's functionality, accuracy, and security but also its overall performance and robustness making it more secure, scalable, accessible, and user-friendly while providing more comprehensive data for analysis and reporting enabling clients to make informed decisions, identify trends, and derive actionable conclusions.

6 Group Work Conclusion

The group worked well together. Despite varying degrees of knowledge in this area team members have worked to their strengths while complementing each other's short comings.

6.1 Successes:

- The group has maintained regular in-person meetings since November these have been concise and have allowed team members to communicate their progress, ask for additional support from other members and allowed us to collectively plan the next steps to the project as we became more familiar with its components
- In addition, the group has maintained an active GitHub repository which has allowed us to communicate meeting notes, code snippets and documents as well as log our progress according to the Gantt Chart made last year.
- The group has been able to make decisions at these meetings via consensus. This allowed team members with existing expertise or with research knowledge to advise our decision making.
- We decided to modularise our code to allow for easy work allocation and for convenient function testing. Allowing us to verify the integrity of code prior to deployment. If we were to create another prototype the group would maintain this practice.
- The group produced documentation in the form of readMe files to assist each other and potential third parties when using, integrating, or deploying code. In future prototypes we would continue with this practice

6.2 Areas of Improvement:

- Our lack of experience with azure meant the group was not able to follow the previously laid out Gantt Chart. Instead, we had to adapt our work based on our knowledge at the time. In future if the group were to continue with a second iteration of prototypes, we would be able to deploy a more accurate Gantt Chart however we believe a Pert chart might work better as it would allow us to commit team members to separate branches that converge later, reducing the amount of time spent having to explain code snippets to each other.

- Likewise, when we wrote our initial report, we had a basic understanding of azure and its services. This led to the architecture of our artefact being unrealistic, for example in our report we suggested training our own AI model however after familiarising ourselves with this software we developed a different model demonstrated in the implementation section. In future prototypes our theoretical models should closer resemble the output.

6.3 Individual Involvement

Ahmed — Ahmed built and managed the Raspberry Pi. Despite receiving it from the technicians late Ahmed displayed excellent time management and resourcefulness deploying code to the Pi.

Dmytro — Dmytro is the most experienced member of the team and was responsible for deploying the database & Dashboard. Dmytro routinely provided less experienced members with help and insight when stuck

Haadiya — Haadiya deployed the dashboard and despite us converting between multiple different formats and platforms. Haadiya has produced a dashboard that we consider to be to an industrial quality.

Harry — Harry managed and organised the team though out the project. He also deployed most of the azure services and built the blob trigger.

Sam — Sam pioneered the computer vision services and associated code. He routinely provided helpful insights with his knowledge of Linux to assist in the setup of the pi and the virtual function app.

Vishal — Vishal wrote a large amount of the documentation and worked together with other team members to select and format relevant information.

In Conclusion the group found this project challenging but after we figured out how to work together and organise ourselves very rewarding and after the work and the stress, we found the project fun.

7 Artefact:

The Code of the Artefact is available at:
`https://github.com/SJO-C/TSEG18-TrafficIoT`
The Video, as previously mentioned is available at:
`https://www.youtube.com/watch?v=Ut4ODaQWDq0`

Name	Relative Contribution
Ahmed Ahmed	$\frac{1}{6}$
Dmytro Steblyna	$\frac{1}{6}$
Haadiya Ahmed	$\frac{1}{6}$
Harry Fitchett	$\frac{1}{6}$
Samuel Orman-Chan	$\frac{1}{6}$
Vishal Dabba	$\frac{1}{6}$

Table 1: Relative Contributions of each Group Member.

A Figures

```
import os, io, enum, sys, argparse, time, requests

from msrest.authentication import CognitiveServicesCredentials
from azure.cognitiveservices.vision.computervision \
      import ComputerVisionClient
from azure.cognitiveservices.vision.computervision.models\
      import VisualFeatureTypes,\
      Category, OperationStatusCodes
from PIL import Image


confidenceVar = 0.5
vehicleTypes = ["car", "Van", "Land Vehicle", "bicycle"]

subscription_key = "REDACTED"
end_point = "REDACTED"

computervision_client = ComputerVisionClient(end_point,\
            CognitiveServicesCredentials(subscription_key))

parser = argparse.ArgumentParser()
parser.add_argument("url", help="Use a URL for an Image.")
args = parser.parse_args()

remote_image_url = args.url
print (remote_image_url)
result_remote = computervision_client.analyze_image(remote_image_url,\
            visual_features=[VisualFeatureTypes.objects], language="en")


vehicleCount = 0
for i in range (0, len(result_remote.objects) -1):
    currentCVObj = result_remote.objects[i]
    print (currentCVObj)
    for j in vehicleTypes:
        if currentCVObj.object_property.find(j) != -1 and\
           currentCVObj.confidence > confidenceVar:
            vehicleCount += 1

print("Number of Vehicles:",vehicleCount)
raise SystemExit
```

Figure 1: The Code of the File: “test_run_2.py”

```
import os, uuid
from azure.identity import DefaultAzureCredential
from azure.storage.blob import BlobServiceClient,\
    BlobClient, ContainerClient
local_path = "./data"
try:
    os.mkdir(local_path)
except FileExistsError: pass

connect_str = "REDACTED"
blob_service_client = \
    BlobServiceClient.from_connection_string(connect_str)

container_name = "REDACTED"


def upload_file():
    try:
        print("Azure Blob Storage Python quickstart sample")

        local_file_name = str(uuid.uuid4()) + ".txt"
        upload_file_path = os.path.join(local_path, local_file_name)

        file = open(file=upload_file_path, mode='w')
        file.write("Hello, World!")
        file.close()

        blob_client = blob_service_client.\
            get_blob_client(container=container_name, blob=local_file_name)

        print("\nUploading to Azure Storage as blob:\n\t" + local_file_name)

        with open(file=upload_file_path, mode="rb") as data:
            blob_client.upload_blob(data)

    except Exception as ex:
        print('Exception:')
        print(ex)

upload_file()
```

Figure 2: The Code of the File: "Python_Bucket_Uploader.py"

```
# logging
import logging
import azure.functions as func

# computer vision
from msrest.authentication import CognitiveServicesCredentials
from azure.cognitiveservices.vision.computervision \
     import ComputerVisionClient
from azure.cognitiveservices.vision.computervision.models \
     import VisualFeatureTypes, Category, OperationStatusCodes
from PIL import Image

# supabase
import os
import datetime
from supabase import create_client, Client
timeDate = datetime.datetime.now()

def main(myblob: func.InputStream):
    confidenceVar = 0.5
    vehicleTypes = ["car", "Van", "Land Vehicle", "bicycle"]

    subscription_key = "REDACTED"
    end_point = "REDACTED"

    computervision_client = ComputerVisionClient\
            (end_point, CognitiveServicesCredentials(subscription_key))

    remote_image_url = ("REDACTED" + myblob.name)
    result_remote = computervision_client.analyze_image\
            (remote_image_url,visual_features=[VisualFeatureTypes.objects],\
             language="en")

    vehicleCount = 0
    for i in range (0, len(result_remote.objects) -1):
        currentCVObj = result_remote.objects[i]
        for j in vehicleTypes:
            if currentCVObj.object_property.find(j) != -1 \
               and currentCVObj.confidence > confidenceVar:
                vehicleCount += 1

    url: str = os.environ.get("SUPABASE_URL")
    key: str = os.environ.get("SUPABASE_KEY")
    supabase: Client = create_client(url, key)

    main_list = []
    value = {'timestamp': str(timeDate), 'location':\
            "Lincoln", 'count': vehicleCount}
    data = supabase.table('vehicle_count').insert(value).execute()



    logging.info(f"Python blob trigger function processed blob \n"
                 f"Name: {myblob.name}\n"
                 f"Blob Size: {myblob.length} bytes\n"
                 f"Vehicle Count: {vehicleCount}")
```

Figure 3: The Code of the File: “azure_blob_trigger.py”

Test Nature	Expected Result	Output	Pass/Fail
AI: Computer Vision: Upload Image of Car	Vehicle Count <0	1	Pass
AI: Computer Vision: Upload Image of Not a Car (Image of Person CF)	Vehicle Count = 0	0	Pass
AI: Computer Vision: Upload Image of several Cars	Vehicle Count <0	11	Pass (But did not count all cars.)
AI: Upload image of person SO w/o any cars	Vehicle Count = 0	0	Pass
AI: Computer Vision: Upload Image of different Car	Vehicle Count = 1	0	Fail. (May have recognised Car as Branded Car that was not in detection list, or not having sufficient confidence to positively ID car)
AI: Computer Vision: Upload Image of several Cars (II)	Vehicle Count = 5	3	Fail. (May have recognised some Cars as Branded Car that was not in detection list, or not having sufficient confidence to positively ID some cars)

Table 2: Computer Vision Functional Testing Table

Bibliography

9 Key Benefits of Using the Agile Methodology [Online], n.d. https://kissflow.com/project/agile/benefits-of-agile/. [Accessed 2023-05-10].

Centre for the Protection of National Infrastructure, 2020. Storage and Retention of Recorded CCTV Images [Online]. [Accessed 2022-11-14].

chez-charlie, 2022. Create event-based triggers - Azure Data Factory & Azure Synapse [Online]. https://learn.microsoft.com/en-us/azure/data-factory/how-to-create-event-trigger. [Accessed 2023-05-10].

Database — Supabase Docs [Online], 2023. https://supabase.com/docs/guides/database/overview. [Accessed 2023-05-10].

Hamilton, T., 2020a. Integration Testing: What is, Types with Example [Online]. https://www.guru99.com/integration-testing.html. [Accessed 2023-05-10].

Hamilton, T., 2020b. Load Testing Tutorial: What is? How to? (with Examples) [Online]. https://www.guru99.com/load-testing-tutorial.html. [Accessed 2023-05-10].

Hamilton, T., 2020c. Test Cases Example for Web Application (Checklist) [Online]. https://www.guru99.com/complete-web-application-testing-checklist.html. [Accessed 2023-05-10].

Hamilton, T., 2020d. What is BLACK Box Testing? Techniques, Types & Example [Online]. https://www.guru99.com/black-box-testing.html. [Accessed 2023-05-10].

Hamilton, T., 2020e. What is END-To-END Testing? E2E Example [Online]. https://www.guru99.com/end-to-end-testing.html. [Accessed 2023-05-10].

Hamilton, T., 2020f. What is Regression Testing? Test Cases (Example) [Online]. https://www.guru99.com/regression-testing.html. [Accessed 2023-05-10].

Hamilton, T., 2020g. What is Scalability Testing? Learn with Example [Online]. https://www.guru99.com/scalability-testing.html. [Accessed 2023-05-10].

Hamilton, T., 2020h. What is Smoke Testing? [Online]. https://www.guru99.com/smoke-testing.html. [Accessed 2023-05-10].

Hamilton, T., 2020i. What is STRESS Testing in Software Testing? [Online]. https://www.guru99.com/stress-testing-tutorial.html. [Accessed 2023-05-10].

Hamilton, T., 2023. What is User Acceptance Testing (UAT)? Examples [Online]. https://www.guru99.com/user-acceptance-testing.html. [Accessed 2023-05-10].

HM Home Office, 2021. Surveillance Camera Code of Practice [Online]. [Accessed 2022-11-14].

Orman-Chan, S., n.d. Requirements.txt.

pauljewellmsft, 2023. Quickstart: Azure Blob Storage client library for Python - Azure Storage [Online]. https://learn.microsoft.com/en-us/azure/storage/blobs/storage-quickstart-blobs-python. [Accessed 2023-05-10].

Picamera — Picamera 1.13 Documentation [Online], n.d. https://picamera.readthedocs.io/en/release-1.13/. [Accessed 2023-05-10].

Pillow, n.d. https://pillow.readthedocs.io/en/stable/index.html. [Accessed 2023-05-10].

The Pros and Cons of Waterfall Methodology, 2017. https://www.lucidchart.com/blog/pros-and-cons-of-waterfall-methodology. [Accessed 2023-05-10].

unknown, n.d.a. Azure Blob Storage — Microsoft Azure [Online]. https://azure.microsoft.com/en-gb/products/storage/blobs. [Accessed 2023-05-10].

unknown, n.d.b. Azure Cognitive Service for Vision with OCR and AI — Microsoft Azure [Online]. https://azure.microsoft.com/en-us/products/cognitive-services/vision-services. [Accessed 2023-05-10].

unknown, n.d.c. Azure Cognitive Service for Vision with OCR and AI — Microsoft Azure [Online]. https://azure.microsoft.com/en-us/products/cognitive-services/vision-services. [Accessed 2023-05-10].

unknown, n.d.d. Computer Vision — Microsoft Azure [Online]. https://azure.microsoft.com/en-gb/products/cognitive-services/computer-vision/. [Accessed 2022-11-10].

unknown, n.d.e. Computer Vision — Microsoft Azure [Online]. https://azure.microsoft.com/en-gb/products/cognitive-services/computer-vision/. [Accessed 2023-05-10].

unknown, 2022a. About this guidance [Online]. https://ico.org.uk/for-organisations/guide-to-data-protection/key-dp-themes/guidance-on-ai-and-data-protection/about-this-guidance/. [Accessed 2022-11-10].

unknown, 2022b. How do we ensure individual rights in our AI systems? [Online]. https://ico.org.uk/for-organisations/guide-to-data-protection/key-dp-themes/guidance-on-ai-and-data-protection/how-do-we-ensure-individual-rights-in-our-ai-systems/. [Accessed 2022-11-10].

unknown, 2022c. How should we assess security and data minimisation in AI? [Online]. https://ico.org.uk/for-organisations/guide-to-data-protection/key-dp-themes/guidance-on-ai-and-data-protection/how-should-we-assess-security-and-data-minimisation-in-ai/. [Accessed 2022-11-10].

unknown, 2022d. Traffic Management Market Size To Surpass USD 61.9 Billion. [Accessed 2022-11-22].

VSC-Service-Account, 2022. Image processing in Python - Code Samples [Online]. https://learn.microsoft.com/en-us/samples/azure-samples/azure-search-python-samples/python-sample-image-processing/. [Accessed 2023-05-10].

What is the Waterfall Model? - Definition and Guide, n.d. https://www.techtarget.com/searchsoftwarequality/definition/waterfall-model. [Accessed 2023-05-10].