DEPARTMENT: COMPUTER SCIENCE

COURSE: CONCURRENT PROGRAMMING (CSC 302/310)

GROUP: 3

DATE: 15/11/2022

MATRIC NO.	NAME	SIGN
190805503	OLORUNFEMI-OJO DANIEL TOMIWA	
180805047	MUMUNI ABDULLAH	
180805050	SANUSI ABDULAZEEZ ADEYEMI	

SEARCH ENGINE RESULT PAGES

PROBLEM DEFINITION

Search Engine Results Pages (SERP) are the pages displayed by search engines in response to a query by a user. Each result displayed normally includes a title, a link that points to the actual page on the Web, and a short description showing where the keywords have matched content within the page for organic results. The SERP are ranked based on relevance for organic results. Considering the semantic search of users' query gives more accurate SERP. More importantly, summarizing the relevant content of SERP for users instead of the displayed titles and links will be more useful to users.

Design and implement a multithreaded program for returning summarized relevant content of SERP and ranked each summary in order of relevance. Your implementation should also include visualization of your results.

ANALYSIS

Implementing a multithreaded program returning relevant content implies the process of fetching content contained in each link should happen on its own thread. The user enters a query and the query is passed to a search engine. The search engines response contains links to relevant content. These are the organic results. The number of threads to be created depends on the number of links in the organic results. Meaning, if there are nine organic results, nine threads will be created, each one fetching the content contained in link n. The main event loop (i.e. the main thread on which the GUI is running on) and other threads should coexist simultaneously. The pitfall to be cautious of is "Dirty Writing". All content fetched is written to the same data structure. Therefore, a thread safety measure should be put in place to avoid this pitfall. A lock is put in place to allow only one thread to write to the data structure. Once all the threads are done fetching the pages content, the main event loop should be signaled to further process the content and rank them based on relevancy. A Countdown Latch is used to achieve that. A latch is created and its count is set to the number of organic results. Once a thread finishes fetching and saving the content, it decreases the latch. Once the latch count reaches zero, it means all the threads are done fetching the content and further processing can begin. The logic used to calculate relevancy is, Term Frequency. The number of times the query appears within the document divided by the total number of words within the document.

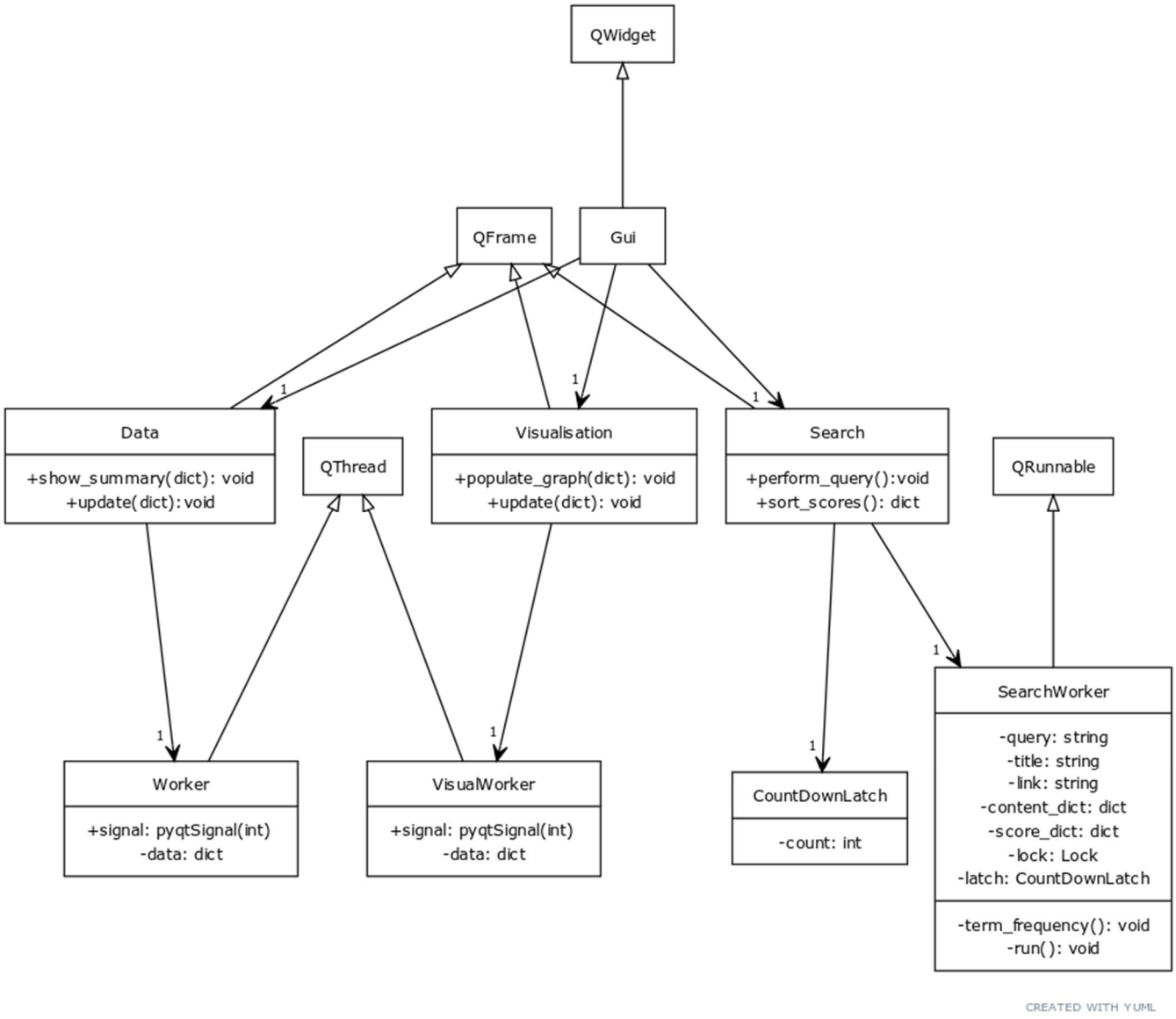
$$tf = \frac{n(q)}{n(w)}$$

q = query, w = words in the document

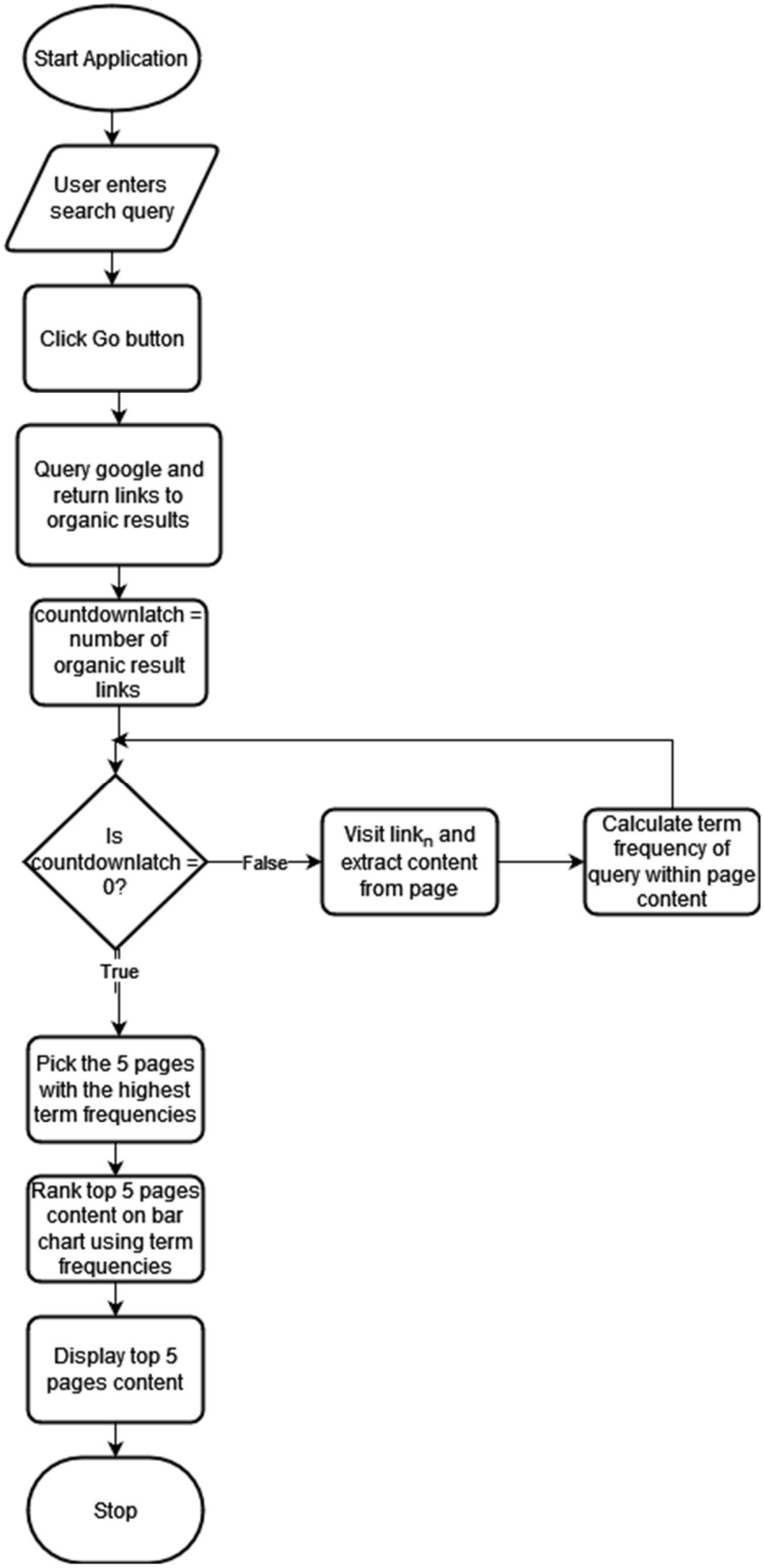
The contents with the top five scores are selected as the most relevant, displayed and the scores visualised on a bar chart.

DESIGN

UML Class Diagram



Flowchart


Start Application
User enters search query
Click Go button
Query google and return links to organic results
countdownlatch = number of organic result links
Is countdownlatch = 0?
False
Visit link$_n$ and extract content from page
Calculate term frequency of query within page content
True
Pick the 5 pages with the highest term frequencies
Rank top 5 pages content on bar chart using term frequencies
Display top 5 pages content
Stop

IMPLEMENTATION

main.py

```
#!/usr/bin/env python3

from PyQt5.QtWidgets import QApplication
from view import Gui


if __name__ == '__main__':
    app = QApplication([])
    gui = Gui()
    app.exec()
```

__init__.py

```
from PyQt5.QtCore import Qt
from PyQt5.QtWidgets import QWidget, QSplitter, QHBoxLayout

from view.data import Data
from view.visualisation import Visualisation
from view.search import Search


class Gui(QWidget):

    def __init__(self):
        super().__init__()

        self.resize(1000, 650)
        self.setMinimumSize(1000, 650)
        # self.setMaximumSize(1000, 1000)
        self.setWindowTitle('Search Engine Results Pages')

        self.data = Data()
        self.visual = Visualisation()
        self.search = Search(self.data, self.visual)

        frame2_split = QSplitter(Qt.Horizontal)
        frame2_split.addWidget(self.data)
        frame2_split.addWidget(self.visual)
        frame2_split.setSizes([480, 480])

        split = QSplitter(Qt.Vertical)
        split.addWidget(self.search)
```

```
        split.addWidget(frame2_split)
        split.setSizes([120, 680])

        hbox = QHBoxLayout()
        hbox.addWidget(split)

        self.setLayout(hbox)
        self.show()
```

countdownlatch.py

```
from threading import Condition


class CountDownLatch:

    def __init__(self, count = 5) -> None:
        self.count = count
        self.condition = Condition()

    def set_count(self, count):
        self.count = count

    def countdown(self):
        with self.condition:
            if self.count == 0:
                return
            self.count -= 1
            if self.count == 0:
                self.condition.notify_all()

    def wait(self):
        with self.condition:
            if self.count == 0:
                return
            self.condition.wait()
```

data.py

```
from PyQt5.QtCore import Qt, QThread, pyqtSignal
from PyQt5.QtWidgets import QFrame, QHBoxLayout, QTabWidget, QTextEdit, QLabel

class Data(QFrame):

    def __init__(self) -> None:
```

```
        super().__init__()
        self.setFrameShape(QFrame.StyledPanel)

        self.tab = [QTextEdit(), QTextEdit(), QTextEdit(), QTextEdit(),
QTextEdit()]
        self.tabs = QTabWidget()
        for i, tab in enumerate(self.tab):
            self.tabs.addTab(tab, str(i + 1))

        self.text_view = QLabel('No summary available')
        # self.text_view.setReadOnly(True)
        self.text_view.setAlignment(Qt.AlignCenter)
        self.hbox = QHBoxLayout()
        self.hbox.addWidget(self.tabs)

        self.setLayout(self.hbox)

    def show_summary(self, data):
        for i, key in enumerate(data):
            self.tabs.setTabText(i, key)
            self.tab[i].setDocumentTitle(key)
            self.tab[i].setText(data[key])
            self.tab[i].setReadOnly(True)

    def update(self, data):
        self.worker = Worker(data)
        self.worker.signal.connect(self.show_summary)
        self.worker.start()
        self.worker.wait()

class Worker(QThread):

    signal = pyqtSignal(dict)

    def __init__(self, data, parent=None) -> None:
        QThread.__init__(self, parent)
        self.data = data

    def run(self):
        self.signal.emit(self.data)
```

search.py

```
import operator
import requests
```

```
import justext
from threading import Lock
from serpapi import GoogleSearch
from PyQt5.QtCore import QThreadPool, QRunnable, pyqtSlot
from PyQt5.QtWidgets import QFrame, QHBoxLayout, QPushButton, QLineEdit

from view.countdownlatch import CountDownLatch


class Search(QFrame):

    def __init__(self, data, visual) -> None:
        super().__init__()

        self.search_links = []
        self.search_content = {}
        self.search_scores = {}
        self.latch = CountDownLatch()
        self.lock = Lock()

        self.threadpool = QThreadPool()
        self.data = data
        self.visual = visual

        self.search_button = QPushButton('Go')
        self.search_button.setFixedHeight(35)
        self.search_button.clicked.connect(self.perform_query)

        self.text_field = QLineEdit()
        self.text_field.setPlaceholderText('Search...')
        self.text_field.setFixedHeight(35)

        hbox = QHBoxLayout()
        hbox.addWidget(self.text_field)
        hbox.addWidget(self.search_button)

        self.setLayout(hbox)

    def perform_query(self):
        params = {
            "engine": "google",
            "q": self.text_field.text(),
            "api_key":
"218b1b0472f6b7a5c1b59f6a995da2b6dd16bbbcb7c65f3cb837bfcfca3e4ac9"
        }
```

```
        search = GoogleSearch(params)
        response = search.get_dict()
        self.latch.set_count(len(response['organic_results']))
        for res in response['organic_results']:
            self.search_links.append(res['link'])
            worker = Worker(self.text_field.text(), res['title'], res['link'],
self.search_content, self.search_scores, self.lock, self.latch)
            self.threadpool.start(worker)
        self.latch.wait()
        scores_sum = 0
        for key in self.search_scores:
            scores_sum += self.search_scores[key]
        for key in self.search_scores:
            self.search_scores[key] = (self.search_scores[key] / scores_sum) *
100
        sorted_results = self.sort_scores()
        print(sorted_results)
        self.visual.update(sorted_results)
        result = {}
        for key in self.sort_scores():
            result[key] = self.search_content[key]
        self.data.update(result)
        self.search_links = []
        self.search_content = {}
        self.search_scores = {}

    def sort_scores(self):
        not_sorted = dict(sorted(self.search_scores.items(),
key=operator.itemgetter(1), reverse=True))
        val = {}
        for i, key in enumerate(not_sorted):
            val[key] = not_sorted[key]
            if i == 4:
                break
        return val



class SearchWorker(QRunnable):

    def __init__(self, query, title, link, content_dict, score_dict, lock, latch)
-> None:
        super().__init__()
        self.query = query
        self.title = title
```

```
        self.link = link
        self.content_dict = content_dict
        self.score_dict = score_dict
        self.latch = latch
        self.lock = lock
        self.content = ''

    @pyqtSlot()
    def run(self):
        try:
            req = requests.get(self.link)
            paragraphs = justext.justext(req.content,
justext.get_stoplist('English'))
            for paragraph in paragraphs:
                if not paragraph.is_boilerplate:
                    self.content += str(paragraph.text)
            self.lock.acquire()
            self.content_dict[self.title] = self.content.replace('\n', '')
            self.score_dict[self.title] = self.term_frequency()
            self.lock.release()
        except:
            pass
        finally:
            self.latch.countdown()

    def term_frequency(self):
        words = self.content.split(' ')
        term_length = 0
        for word in words:
            if word.lower() in self.query.lower().split(' '):
                term_length += 1
        return (term_length / len(words))
```

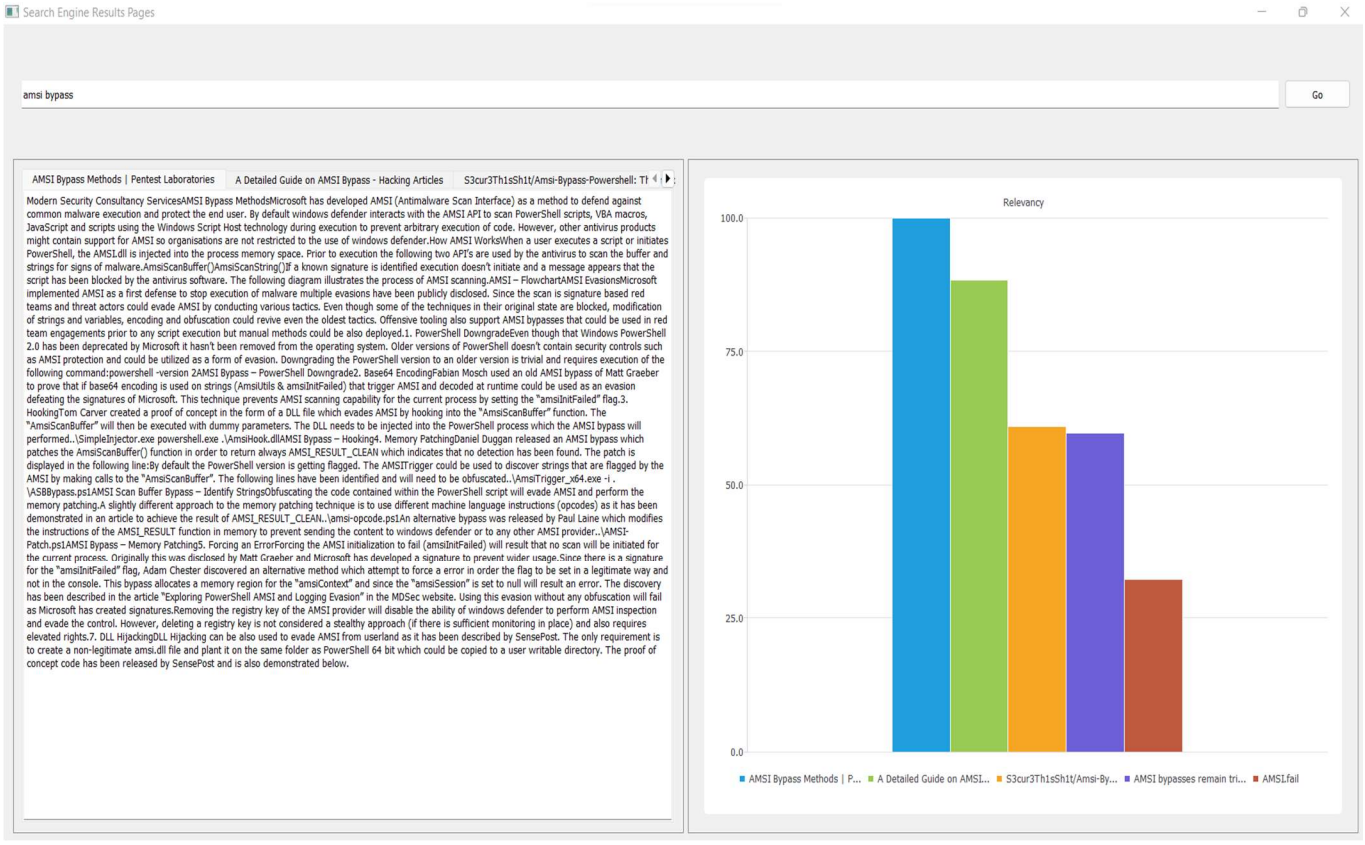
visualisation.py

```
from PyQt5.QtCore import Qt, QThread, pyqtSignal
from PyQt5.QtWidgets import QFrame, QHBoxLayout
from PyQt5.QtChart import QChart, QBarSet, QChartView, QValueAxis, QBarSeries

class Visualisation(QFrame):

    def __init__(self) -> None:
        super().__init__()
        self.setFrameShape(QFrame.StyledPanel)
```

```
        self.sets = None

        self.series = QBarSeries()

        self.chart = QChart()
        self.chart.setTitle('Relevancy')
        self.chart.setAnimationOptions(QChart.SeriesAnimations)

        y_axis = QValueAxis()
        y_axis.setRange(0, 100)

        self.chart.addAxis(y_axis, Qt.AlignLeft)
        self.chart.legend().setVisible(True)
        self.chart.legend().setAlignment(Qt.AlignBottom)

        self.chart_view = QChartView(self.chart)

        hbox = QHBoxLayout()
        hbox.addWidget(self.chart_view)

        self.setLayout(hbox)


    def populate_graph(self, data):
        self.chart.removeSeries(self.series)
        self.series = QBarSeries()
        self.sets = [QBarSet(''), QBarSet(''), QBarSet(''), QBarSet(''),
QBarSet('')]
        for i, key in enumerate(data):
            self.sets[i].setLabel(key)
            self.sets[i].append(data[key])
            self.series.append(self.sets[i])
        self.chart.addSeries(self.series)


    def update(self, data):
        self.worker = Worker(data)
        self.worker.signal.connect(self.populate_graph)
        self.worker.start()
        # self.worker.wait()


class VisualWorker(QThread):

    signal = pyqtSignal(dict)
```

```
def __init__(self, data, parent=None) -> None:
    QThread.__init__(self, parent)
    self.data = data

def run(self):
    print(self.data)
    self.signal.emit(self.data)
```

OUTPUT



TRACKING FLEET VEHICLES

PROBLEM DEFINITION

Implement the ***Tracking Fleet Vehicles*** (a vehicle tracker for dispatching fleet vehicles such as taxicabs, police cars or delivery trucks) example in section 4.2.2. of *Java Concurrency In Practice* by Brian Göetz et. al.

ANALYSIS

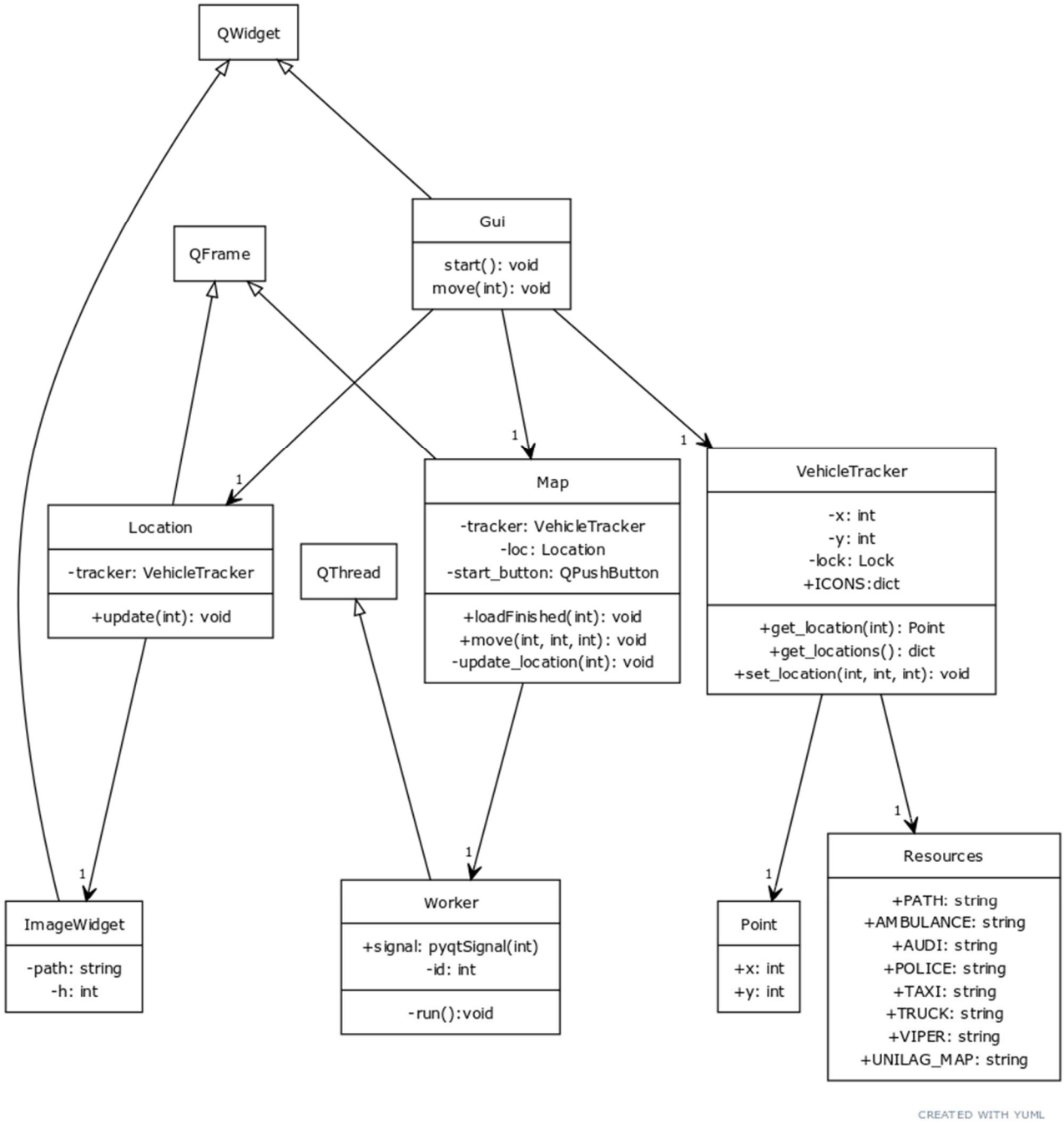
Tracking fleet vehicles entails real time tracking of their locations. The geographic coordinate system is the method used for referencing the location of the fleet vehicles. This project aims at keeping track vehicles heading to a destination simultaneously. The movement of the vehicles is independent of each other. The speed of the vehicles is irrelevant for this simulation and hence, all cars are moving at the same speed.

There is a model in form of a class called VehicleTracker, which contains all the instances of the vehicles and their locations, and another class called Point for holding the latitude and longitude of a vehicle (as seen in example in section 4.2.2. of *Java Concurrency In Practice*). The location of the vehicles (instances of Point class) are stored within the same data structure. Therefore, a thread safety measure has to be put in place when reading and updating the locations because multiple updater threads will access the locations structure. In this simulation, locks are used to restrict access to the structure containing locations. Access is granted to update or read the structure only if no other updater thread currently has access.

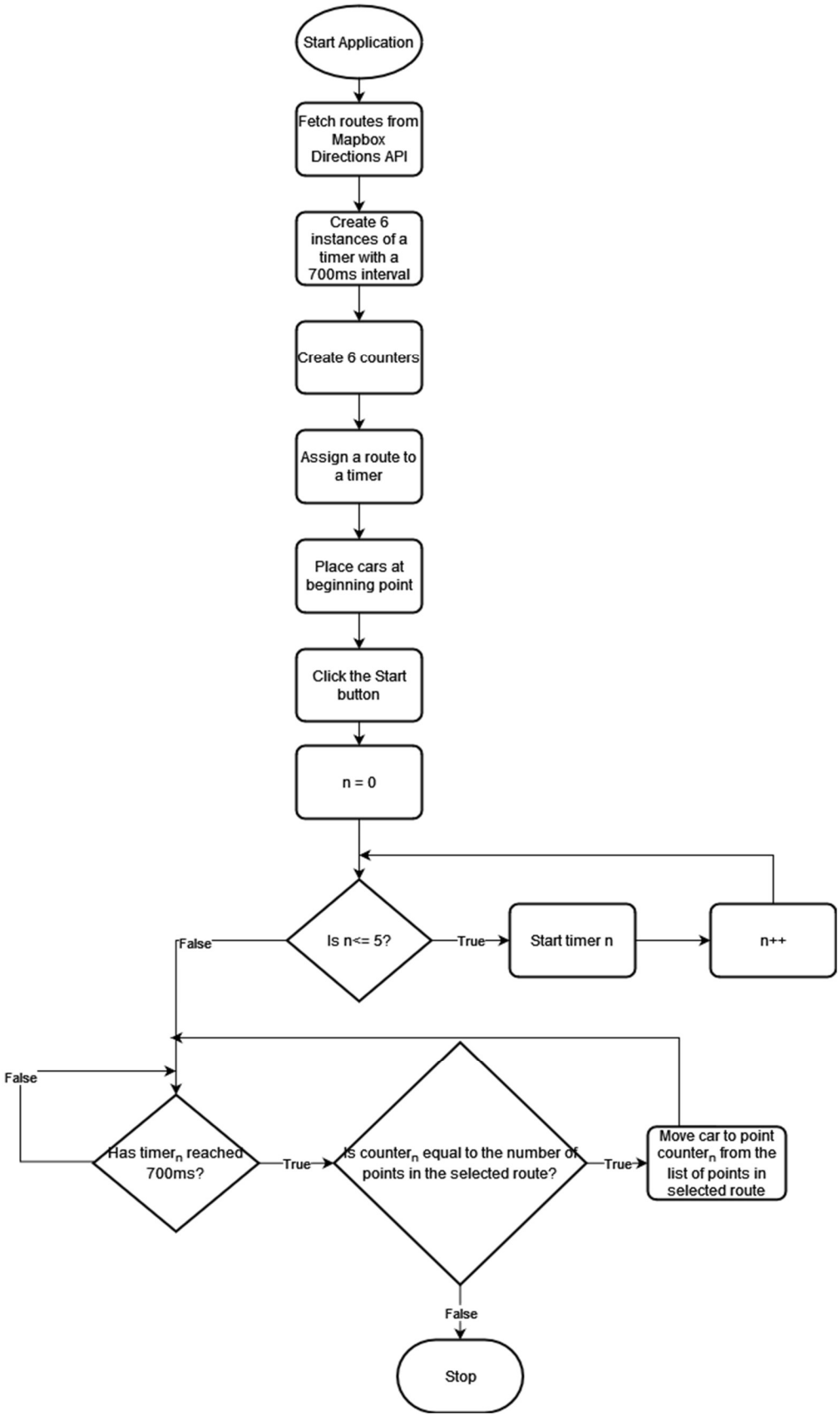
The map of the University of Lagos is being used for this simulation. The start point is the main entrance into the university, the stop point is the Faculty of Science, and they are marked on the map. The map is shown on the GUI. The GUI is running on the view thread (i.e. the main thread or event loop). Using a third party program, two routes have been identified from the main gate to the Faculty of Science. Each vehicle is assigned a route. The movement of a vehicle happens on its own individual thread and another thread is created to update their locations on the VehicleTracker class when the vehicle moves to another coordinate. Upon getting to the destination, movement is brought to a halt. All threads are stopped except the main event loop (i.e. the updater threads are stopped and the view thread keeps on running).

DESIGN

UML Class Diagram



Flowchart


Start Application
Fetch routes from Mapbox Directions API
Create 6 instances of a timer with a 700ms interval
Create 6 counters
Assign a route to a timer
Place cars at beginning point
Click the Start button
n = 0
Is n<= 5?
False
True
Start timer n
n++
False
Has timer$_n$ reached 700ms?
True
Is counter$_n$ equal to the number of points in the selected route?
True
Move car to point counter$_n$ from the list of points in selected route
False
Stop

IMPLEMENTATION

main.py

```
#!/usr/bin/env python3

from PyQt5.QtWidgets import QApplication
from view import Gui


if __name__ == '__main__':
    app = QApplication([])
    gui = Gui()
    app.exec()
```

__init__.py

```
import requests
from threading import Lock
from PyQt5 import QtGui, QtTest
from PyQt5.QtCore import Qt, QTimer
from PyQt5.QtWidgets import QWidget, QSplitter, QHBoxLayout, QPushButton
from model.vehicle import VehicleTracker
from view.location import Location
from view.map import Map
from view.resources import Resources



class Gui(QWidget):

    def __init__(self):
        super().__init__()
        res =
requests.get('https://api.mapbox.com/directions/v5/mapbox/driving/3.384685%2C6.51
7622%3B3.397718%2C6.515656?alternatives=true&geometries=geojson&language=en&overv
iew=full&steps=true&access_token=pk.eyJ1IjoidG9taXdhLW90IiwiYSI6ImNsOXIzN3N5YTBmd
W4zcGw3ZDY0cnlzcWoifQ.92XuIcVS4lNarudJU42f8w')
        self.route1 = res.json()['routes'][0]['geometry']['coordinates']
        self.route2 = res.json()['routes'][1]['geometry']['coordinates']
        self.routes = [self.route1, self.route2]
        self.lock = Lock()
        self.tracker = VehicleTracker(self.route1[0][0], self.route1[0][1],
self.lock)
```

```
        self.timers = [QTimer(), QTimer(), QTimer(), QTimer(), QTimer(),
QTimer()]
        for i, timer in enumerate(self.timers):
            timer.setInterval(700)
            timer.timeout.connect(lambda i=i: self.move(i))
        self.index = [1] * 6

        self.resize(1500, 600)
        self.setMinimumSize(1450, 650)
        self.setMaximumSize(1450, 650)
        self.setWindowTitle('Tracking Vehicle Fleet')
        self.setWindowIcon(QtGui.QIcon(Resources.audi))

        self.start_button = QPushButton('Start', self)
        self.start_button.clicked.connect(self.start)
        self.start_button.setEnabled(False)

        horizontal_split = QSplitter(Qt.Horizontal)
        self.loc = Location(self.tracker)
        horizontal_split.addWidget(self.loc)
        self.map = Map(self.tracker, self.loc, self.start_button)
        horizontal_split.addWidget(self.map)
        horizontal_split.setSizes([350, 1100])

        vertical_split = QSplitter(Qt.Vertical)
        vertical_split.addWidget(horizontal_split)
        vertical_split.addWidget(self.start_button)
        vertical_split.setSizes([650, 50])

        hbox = QHBoxLayout()
        hbox.addWidget(vertical_split)

        self.setLayout(hbox)
        self.show()

    def start(self):
        for timer in self.timers:
            timer.start()
            QtTest.QTest.qWait(4000)

    def move(self, id):
        if self.index[id] == len(self.routes[id % 2]):
            self.timers[id].stop()
            self.index[id] = 0
            return
```

```
        self.map.move(id, str(self.routes[id % 2][self.index[id]][0]),
str(self.routes[id % 2][self.index[id]][1]))
        self.index[id] += 1
```

location.py

```
from PyQt5.QtCore import Qt
from PyQt5.QtWidgets import QWidget
from PyQt5.QtGui import QFont, QPixmap, QPainter
from PyQt5.QtWidgets import QAbstractItemView
from PyQt5.QtWidgets import QFrame, QTableWidget, QTableWidgetItem, QHBoxLayout

from model.vehicle import VehicleTracker

class Location(QFrame):

    def __init__(self, tracker) -> None:
        super().__init__()
        self.tracker = tracker
        self.setFrameShape(QFrame.StyledPanel)

        font = QFont()
        font.setBold(True)
        font.setPixelSize(20)

        self.table = QTableWidget()
        self.table.setRowCount(7)
        self.table.setColumnCount(3)
        self.table.setRowHeight
        self.table.setEditTriggers(QAbstractItemView.NoEditTriggers)
        for i in range(1, 7):
            self.table.setRowHeight(i, 70)

        v = QTableWidgetItem('Vehicle')
        v.setFont(font)
        v.setTextAlignment(Qt.AlignHCenter)
        self.table.setItem(0, 0, v)

        x = QTableWidgetItem('x')
        x.setFont(font)
        x.setTextAlignment(Qt.AlignHCenter)
        self.table.setItem(0, 1, x)

        y = QTableWidgetItem('y')
        y.setFont(font)
```

```
        y.setTextAlignment(Qt.AlignHCenter)
        self.table.setItem(0, 2, y)

        for i in range(1, 7):
            self.table.setCellWidget(i, 0, ImageWidget(VehicleTracker.icons[i -
1], self.table.rowHeight(1)))

        for i in range(3):
            self.table.setColumnWidth(i, 90)

        for i in range(1, 7):
            self.update(i)

        hbox = QHBoxLayout()
        hbox.addWidget(self.table)

        self.setLayout(hbox)

    def update(self, i) -> None:
        x = QTableWidgetItem(f'{self.tracker.locations[i - 1].x}')
        x.setTextAlignment(Qt.AlignCenter)
        y = QTableWidgetItem(f'{self.tracker.locations[i - 1].y}')
        y.setTextAlignment(Qt.AlignCenter)
        self.table.setItem(i, 1, x)
        self.table.setItem(i, 2, y)


class ImageWidget(QWidget):

    def __init__(self, path, h) -> None:
        super().__init__()
        self.pic = QPixmap(path)
        self.h = h

    def paintEvent(self, event) -> None:
        painter = QPainter(self)
        painter.drawPixmap(0, 0, self.h, self.h, self.pic)
```

map.py

```
from PyQt5.QtWidgets import QFrame, QHBoxLayout
from PyQt5.QtWebEngineWidgets import QWebEngineView
from PyQt5.QtCore import QThread, pyqtSignal
from view.resources import Resources
```

```
class Map(QFrame):

    def __init__(self, tracker, loc, start_button) -> None:
        super().__init__()
        self.tracker = tracker
        self.loc = loc
        self.start_button = start_button
        self.setFrameShape(QFrame.StyledPanel)

        self.web_view = QWebEngineView()
        with open(Resources.unilag_map, 'r') as fh:
            self.web_view.setHtml(fh.read())
        self.web_view.loadFinished.connect(self.loadFinished)

        hbox = QHBoxLayout()
        hbox.addWidget(self.web_view)

        self.setLayout(hbox)

    def loadFinished(self, ok):
        if ok:
            self.start_button.setEnabled(True)

    def move(self, id, x, y):
        self.tracker.set_location(id, x, y)
        self.worker = Worker(id)
        self.worker.signal.connect(self.loc.update)
        self.worker.start()
        self.worker.wait()
        self.web_view.page().runJavaScript(f"move({id}, \"{x}\", \"{y}\")",
self.update_location)

    def update_location(self, val):
        pass


class Worker(QThread):

    signal  = pyqtSignal(int)

    def __init__(self, id, parent=None) -> None:
        QThread.__init__(self, parent)
        self.id = id

    def run(self):
```

```
        self.signal.emit(self.id + 1)
```

resources.py

```
import os

class Resources:

    path = os.path.dirname(__file__)

    ambulance = os.path.join(path, 'assets', 'Ambulance.png')
    audi = os.path.join(path, 'assets', 'Audi.png')
    police = os.path.join(path, 'assets', 'Police.png')
    taxi = os.path.join(path, 'assets', 'Taxi.png')
    truck = os.path.join(path, 'assets', 'Mini_truck.png')
    viper = os.path.join(path, 'assets', 'Black_viper.png')
    unilag_map = os.path.join(path, 'assets', 'unilag.html')
```

unilag.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>Demo: Add custom markers in Mapbox GL JS</title>
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <link
      href="https://fonts.googleapis.com/css?family=Open+Sans"
      rel="stylesheet"
    />
    <script src="https://api.tiles.mapbox.com/mapbox-gl-js/v2.9.2/mapbox-
gl.js"></script>
    <link
      href="https://api.tiles.mapbox.com/mapbox-gl-js/v2.9.2/mapbox-gl.css"
      rel="stylesheet"
    />
    <style>
      body {
        margin: 0;
        padding: 0;
      }
      #map {
        position: absolute;
        top: 0;
        bottom: 0;
```

```
        width: 100%;
      }
      .marker {
        background-image: url('https://raw.githubusercontent.com/Tomiwa-Ot/cs-
assignments/master/concurrent%20programming/vehicle%20fleet%20tracker/view/assets
/Marker.png');
        background-size: cover;
        width: 50px;
        height: 50px;
        border-radius: 50%;
        cursor: pointer;
      }
      .ambulance {
        background-image: url('https://raw.githubusercontent.com/Tomiwa-Ot/cs-
assignments/master/concurrent%20programming/vehicle%20fleet%20tracker/view/assets
/Ambulance.png');
        background-size: cover;
        width: 50px;
        height: 50px;
        border-radius: 50%;
        cursor: pointer;
        -webkit-transition: all 0.9s ease-in-out;
        -moz-transition: all 0.9s ease-in-out;
        -ms-transition: all 0.9s ease-in-out;
        -o-transition: all 0.9s ease-in-out;
        transition: all 0.9s ease-in-out;
      }
      .audi {
        background-image: url('https://raw.githubusercontent.com/Tomiwa-Ot/cs-
assignments/master/concurrent%20programming/vehicle%20fleet%20tracker/view/assets
/Audi.png');
        background-size: cover;
        width: 50px;
        height: 50px;
        border-radius: 50%;
        cursor: pointer;
        -webkit-transition: all 0.9s ease-in-out;
        -moz-transition: all 0.9s ease-in-out;
        -ms-transition: all 0.9s ease-in-out;
        -o-transition: all 0.9s ease-in-out;
        transition: all 0.9s ease-in-out;
      }
      .police {
```

```
        background-image: url('https://raw.githubusercontent.com/Tomiwa-Ot/cs-assignments/master/concurrent%20programming/vehicle%20fleet%20tracker/view/assets/Police.png');
        background-size: cover;
        width: 50px;
        height: 50px;
        border-radius: 50%;
        cursor: pointer;
        -webkit-transition: all 0.9s ease-in-out;
        -moz-transition: all 0.9s ease-in-out;
        -ms-transition: all 0.9s ease-in-out;
        -o-transition: all 0.9s ease-in-out;
        transition: all 0.9s ease-in-out;
      }
      .truck {
        background-image: url('https://raw.githubusercontent.com/Tomiwa-Ot/cs-assignments/master/concurrent%20programming/vehicle%20fleet%20tracker/view/assets/Mini_truck.png');
        background-size: cover;
        width: 50px;
        height: 50px;
        border-radius: 50%;
        cursor: pointer;
        -webkit-transition: all 0.9s ease-in-out;
        -moz-transition: all 0.9s ease-in-out;
        -ms-transition: all 0.9s ease-in-out;
        -o-transition: all 0.9s ease-in-out;
        transition: all 0.9s ease-in-out;
      }
      .taxi {
        background-image: url('https://raw.githubusercontent.com/Tomiwa-Ot/cs-assignments/master/concurrent%20programming/vehicle%20fleet%20tracker/view/assets/Taxi.png');
        background-size: cover;
        width: 50px;
        height: 50px;
        border-radius: 50%;
        cursor: pointer;
        -webkit-transition: all 0.9s ease-in-out;
        -moz-transition: all 0.9s ease-in-out;
        -ms-transition: all 0.9s ease-in-out;
        -o-transition: all 0.9s ease-in-out;
        transition: all 0.9s ease-in-out;
      }
      .viper {
```

```
        background-image: url('https://raw.githubusercontent.com/Tomiwa-Ot/cs-assignments/master/concurrent%20programming/vehicle%20fleet%20tracker/view/assets/Black_viper.png');
        background-size: cover;
        width: 50px;
        height: 50px;
        border-radius: 50%;
        cursor: pointer;
        -webkit-transition: all 0.9s ease-in-out;
        -moz-transition: all 0.9s ease-in-out;
        -ms-transition: all 0.9s ease-in-out;
        -o-transition: all 0.9s ease-in-out;
        transition: all 0.9s ease-in-out;
      }
      .mapboxgl-popup {
        max-width: 200px;
      }
      .mapboxgl-popup-content {
        text-align: center;
        font-family: 'Open Sans', sans-serif;
      }
    </style>
  </head>
  <body>
    <div id="map"></div>

    <script>
        let routes = undefined
        var vehicles = {
            0 : 'ambulance',
            1 : 'audi',
            2 : 'police',
            3 : 'taxi',
            4 : 'truck',
            5 : 'viper'
        }
        let vehicleMarkers = {
            'ambulance' : undefined,
            'audi' : undefined,
            'police' : undefined,
            'taxi' : undefined,
            'truck' : undefined,
            'viper' : undefined
        }
```

```
        mapboxgl.accessToken =
'pk.eyJ1IjoidG9taXdhLW90IiwiYSI6ImNsOXIzN3N5YTBmdW4zcGw3ZDY0cnlzcWoifQ.92XuIcVS4l
NarudJU42f8w';

        const map = new mapboxgl.Map({
            container: 'map',
            style: 'mapbox://styles/mapbox/streets-v10',
            center: [3.391004, 6.515844],
            zoom: 15.7,
        });
        map.addControl(new mapboxgl.NavigationControl());

        function move(id, x, y) {
            vehicleMarkers[vehicles[id]].setLngLat([x, y])
        }

        async function getStartPoint(url) {
            const response = await fetch(url)
            if(!response.ok)
                throw Error(response.statusText)
            const json = await response.json()
            return json
        }

        (async function(){
            let url =
'https://api.mapbox.com/directions/v5/mapbox/driving/3.384685%2C6.517622%3B3.3977
18%2C6.515656?alternatives=true&geometries=geojson&language=en&overview=full&step
s=true&access_token=pk.eyJ1IjoidG9taXdhLW90IiwiYSI6ImNsOXIzN3N5YTBmdW4zcGw3ZDY0cn
lzcWoifQ.92XuIcVS4lNarudJU42f8w'

            routes = await getStartPoint(url)

            const geojson = {
                'type': 'FeatureCollection',
                'features': [
                {
                    'type': 'Feature',
                    'geometry': {
                    'type': 'Point',
                    'coordinates':
routes['routes'][0]['geometry']['coordinates'][0]
                    },
                    'properties': {
                    'title': 'Start',
```

```
                    'description': 'Start'
                    }
                },
                {
                    'type': 'Feature',
                    'geometry': {
                    'type': 'Point',
                    'coordinates':
routes['routes'][0]['geometry']['coordinates'][routes['routes'][0]['geometry']['c
oordinates'].length - 1]
                    },
                    'properties': {
                    'title': 'Stop',
                    'description': 'Stop'
                    }
                }
                ]
            };

            // add markers to map
            for (const feature of geojson.features) {
                // create a HTML element for each feature
                const el = document.createElement('div');
                el.className = 'marker';

                // make a marker for each feature and add it to the map
                new mapboxgl.Marker(el)
                .setLngLat(feature.geometry.coordinates)
                .setPopup(
                    new mapboxgl.Popup({ offset: 25 }) // add popups
                    .setHTML(
                        `<h3>${feature.properties.title}</h3><p>${feature.propert
ies.description}</p>`
                    )
                )
                .addTo(map);
            }
            for (i = 0; i <= 5; i++) {
                const el = document.createElement('div');
                el.className = vehicles[i];

                // make a marker for each feature and add it to the map
                vehicleMarkers[vehicles[i]] = new mapboxgl.Marker(el)
                .setLngLat(routes['routes'][0]['geometry']['coordinates'][0])
                .setPopup(
```

```
                    new mapboxgl.Popup({ offset: 25 }) // add popups
                )
                .addTo(map);
            }
        })()

    </script>
  </body>
</html>
```

OUTPUT