PUBLIC TRANSPORT OPTIMIZATION USING IOT

PHASE -05

TEAM MEMBERS

1.Mopuri Lokesh Dhananjayan - 112821104047
2.Karthikeyan S - 112821104041
3.Mohammed Asraf Ali S - 112821104043
4.mukesh Varma P - 112821104048
5.varaprasath P - 112821104069

INTRODUCTION

Public transport optimization using the Internet of Things (IoT) is the use of IoT sensors and data to improve the efficiency and effectiveness of public transport systems. IoT sensors can be used to collect data on a variety of factors, such as:

- Vehicle location and occupancy
- Passenger demand
- Traffic conditions
- Weather conditions

This data can then be used to optimize public transport routes and schedules, reduce travel times, and improve passenger satisfaction.

There are a number of different ways in which IoT can be used to optimize public transport. Here are a few examples:

Real-time route optimization:

IoT sensors can be used to track the location and occupancy of public transport vehicles in real time. This data can then be used to optimize public transport routes and schedules to avoid congestion and meet passenger demand.

Predictive maintenance:

IoT sensors can be used to monitor the condition of public transport vehicles and identify potential problems early on. This data can then be used to schedule preventive maintenance, which can help to reduce the number of vehicle failures and disruption to services.

Passenger information systems:

IoT sensors can be used to provide passengers with real-time information on the arrival and departure times of public transport vehicles, as well as information on traffic conditions and other disruptions. This can help passengers to plan their journeys more effectively and reduce their waiting times.

IoT-based public transport optimization is still a relatively new field, but it has the potential to revolutionize the way that public transport systems are operated and managed.

Here are some of the benefits of using IoT to optimize public transport:

Reduced travel times for passengers:

IoT can be used to optimize public transport routes and schedules to avoid congestion, which can lead to reduced travel times for passengers.

Increased efficiency of public transport operations:

IoT can be used to improve the efficiency of public transport operations by optimizing vehicle scheduling and maintenance.

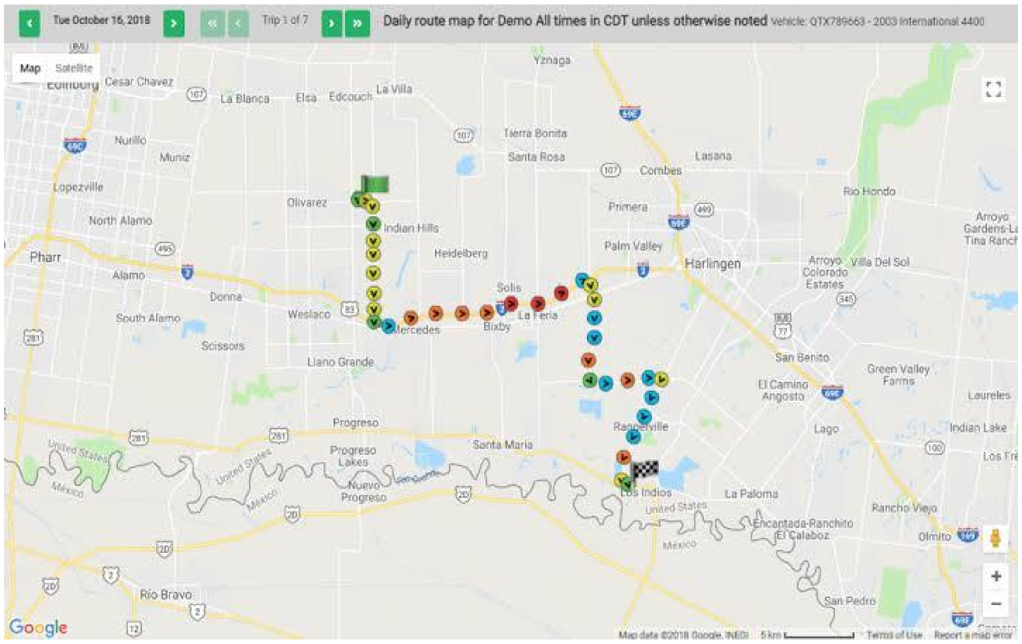
Reduced emissions from public transport vehicles:

By optimizing public transport routes and schedules, IoT can help to reduce the amount of time that public transport vehicles spend on the road, which can lead to reduced emissions.

Improved passenger satisfaction:

By providing passengers with real-time information and reducing travel times, IoT can help to improve passenger satisfaction with public transport services.

Overall, IoT-based public transport optimization has the potential to provide a number of benefits to both public transport operators and passengers.



SENSORS USED IN PUBLIC TRANSPORT

A wide variety of sensors can be used in public transportation to improve efficiency, safety, and passenger experience.

GPS sensors:

GPS sensors are used to track the location of vehicles in real time. This information can be used to provide passengers with accurate arrival times and to help dispatchers reroute vehicles around traffic congestion or other disruptions.

Passenger counting sensors:

Passenger counting sensors are used to track the number of passengers on vehicles and at stations. This data can be used to forecast demand for services and to optimize vehicle scheduling and routing.

Environmental sensors:

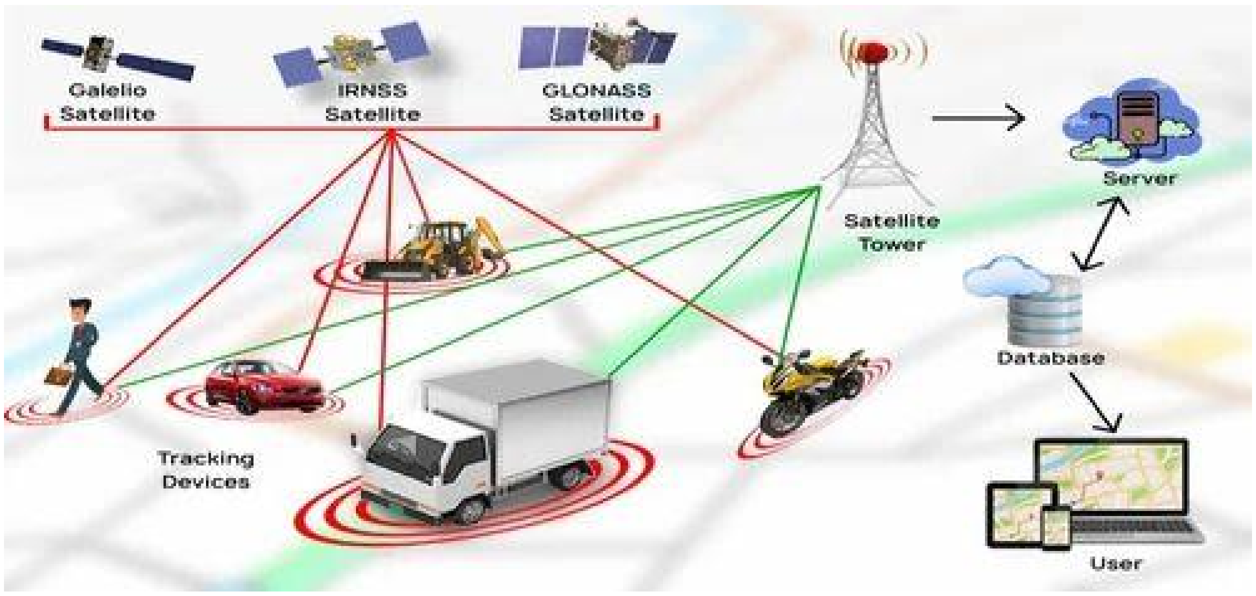
Environmental sensors are used to monitor temperature, humidity, air quality, and other environmental conditions on vehicles and at stations. This data can be used to improve passenger comfort and safety.

Door sensors:

Door sensors are used to detect whether doors are open or closed. This information can be used to prevent accidents and to ensure that vehicles are secure.

Safety sensors:

Safety sensors are used to detect objects and people in the path of vehicles. This information can be used to prevent collisions and to protect passengers and pedestrians.



GPS SENSOR

GPS sensors play a vital role in public transport optimization. They can be used to track the location of vehicles in real time, which can be used to:

- Provide passengers with accurate arrival times and route information.

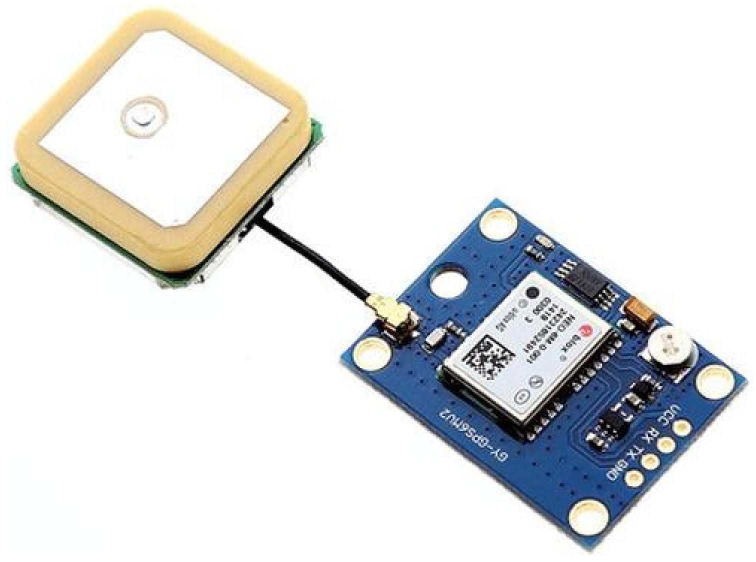
- Help dispatchers reroute vehicles around traffic congestion or other disruptions

- Optimize vehicle scheduling and routing to improve efficiency and reduce costs.

- Improve passenger safety by identifying and responding to potential hazards.

GPS sensors are also being used to develop new and innovative ways to improve public transport. For example, some cities are using GPS sensors to develop real-time bus tracking apps that allow passengers to track the location of their bus and see its estimated arrival time. Other cities are using GPS sensors to develop systems that can predict demand for public transportation services and adjust bus and train schedules accordingly.

As GPS technology continues to develop, we can expect to see even more innovative and effective ways to use GPS sensors to optimize public transport.



GPS SENSOR

Algorithms used in GPS

GPS sensors in IoT use a variety of algorithms to determine a device's location. These algorithms typically work by using the signals from multiple GPS satellites to calculate the device's position in three-dimensional space (latitude, longitude, and altitude).

Trilateration algorithm:

Trilateration works by using the distances between the device and at least three GPS satellites to calculate its location. To do this, the device measures the time it takes to receive a signal from each satellite. This information is then used to calculate the distance between the device and each satellite. Once the distances are known, the device can use trilateration to calculate its location.

Multipathing algorithm:

Multipathing occurs when a GPS signal is reflected off of objects in the environment, such as buildings or trees. This can cause the device to receive multiple signals from the same satellite, which can make it difficult to accurately calculate the device's location. The multipathing algorithm helps to address this problem by identifying and filtering out reflected signals.

In addition to trilateration and multipathing, GPS sensors also use a variety of other algorithms to improve the accuracy and reliability of their location estimates.

The specific algorithms that are used in a GPS sensor will vary depending on the type of sensor and its intended application. However, all GPS sensors use some form of algorithm to calculate the device's location based on the signals from multiple GPS satellites.

Python script to find arrival time of public vehicles

```
import time
import requests

def get_bus_location(bus_id):
Url=
"https://api.example.org/get_bus_location?bus_id={}".format(b
us_id)
    response = requests.get(url)
    return response.json()

def calculate_arrival_time(bus_location, destination):
    def main():
        bus_location = get_bus_location(bus_id=1234)
        arrival_time=calculate_arrival_time(bus_location=
                    bus_location, destination="Aravil")

  print("The bus will arrive at Aravil at {}.".format(arrival_time))

    if __name__ == "__main__":
        main()
```

approaches and examples for implementing public transport optimization in Python:

1. Route Optimization:

○ One way to optimize routes is by identifying the most effective connections based on traffic and population. This involves finding the shortest or most efficient paths between stops.

○ You can use graph algorithms (e.g., Dijkstra's algorithm) to find optimal routes.

○Check out the Public-Transport-System-Optimization repository on GitHub.It provides a solution for public transportation systems, including route optimization, dynamic timetable generation, and real-time information for passengers1.

2. Mathematical Programming:

○ Mathematical programming models can help minimize costs in multimodal transportation systems.

○ The multimodal-transportation-optimization project uses mathematical programming (DOcplex and CVXPY) to model and solve overall cost minimization problems2.

3. Visualization and Insights:

○ Visualizing road transportation network performance can provide valuable insights.

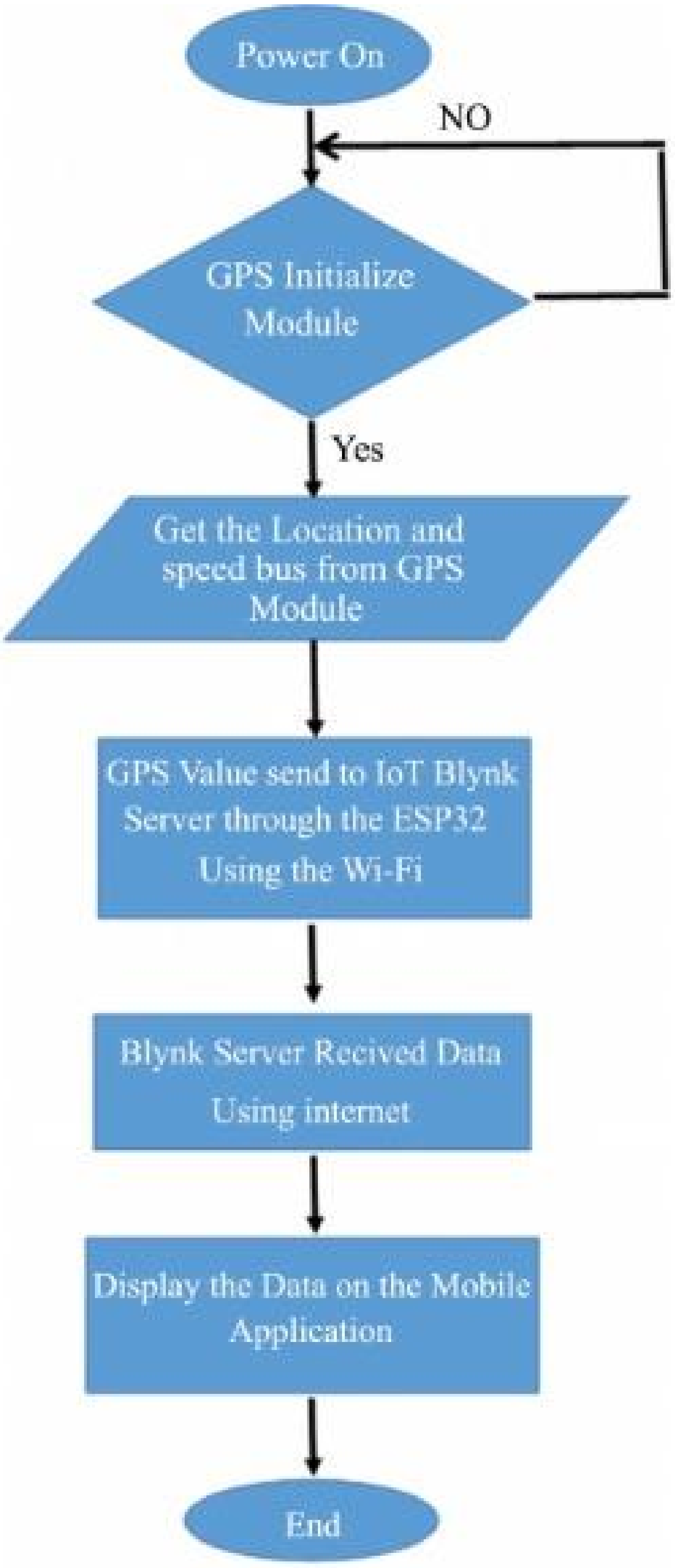
○ Consider building visualizations using Python libraries (e.g., Matplotlib,Plotly) to analyze transportation data3.

4. AI-Driven Solutions:

○ Implementing AI algorithms can enhance public transport optimization.

○ For example, SAP AI Core has worked on AI-driven urban public transport optimization for the London bus service4.

Remember that each transportation system has unique requirements, and the choice of approach depends on factors like network size, available data, and computational resources. Feel free to explore these resources and adapt them to your specific use case.


Power On
NO
GPS Initialize Module
Yes
Get the Location and speed bus from GPS Module
GPS Value send to IoT Blynk Server through the ESP32 Using the Wi-Fi
Blynk Server Recived Data Using internet
Display the Data on the Mobile Application
End

CODE IMPLEMENTATION FOR PUBLIC TRANSPORT OPTIMIZATION:-

```
import numpy as np, json, random, solver, operator,
pandas as pd
from flask import *
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import csv
app = Flask(__name__)

city_name_data =
pd.read_csv('namelist.csv',header=None)
city_dist_data = pd.read_csv('distlist.csv',header=None)
city_weight_data =
pd.read_csv('poplist.csv',header=None)
city_coord_data =
pd.read_csv('Latlong.csv',header=None)
print(city_name_data)
print(city_dist_data)
print(city_coord_data)

class City:
    def __init__(self,name,population,coord):
        self.name=name
        self.population=population
        self.coord=coord

    def distance(self, city):
        distance=city_dist_data.iloc[self.name,city.name]
        return distance

    def CityName(self):
        return str(city_name_data.iloc[self.name,0])

    def CityCoord(self):
        return self.coord

    def __repr__(self):
        return "\""+str(city_name_data.iloc[self.name,0]) + "
"+str(self.coord[0])+", "+str(self.coord[1])+"" +"\""

class Fitness:
    def __init__(self, route):
        self.chromosome = route
        self.distance = 0
        self.fitness= 0.0
        self.total_population= 0

    def routeDistance(self):
        pathDistance = 0
        for i in range(0, len(self.chromosome)):
          for j in range(0,len(self.chromosome[0])):
            fromCity = self.chromosome[i][j]
            toCity = None
            if j + 1  < len(self.chromosome[0]):
                toCity = self.chromosome[i][j +1] #doubtfull
            else:
                break
            if(type(toCity) == list): break
            pathDistance += fromCity.distance(toCity)
        self.distance = pathDistance
        return self.distance

    def routePopulation(self):
        path_population = 0
        for i in range(0, len(self.chromosome)):
          for j in range(0,len(self.chromosome[0])):
            City = self.chromosome[i][j]
            if type(City) != list:
              path_population += int(City.population.replace(',',
''))
            else:
              path_population = 0
        self.total_population = path_population
        return self.total_population

    def routeFitness(self):
        if self.fitness == 0:
          if(self.routeDistance()==0):
            return self.routePopulation()
          self.fitness = self.routePopulation() /
float(self.routeDistance())
        return self.fitness

def Diff(l1, l2):
    li_dif = [i for i in l1  if i not in l2]
    return li_dif

def GenerateTiming():
    res = []
    res.append( str(random.randint(6,9)) +
":"+str(random.randint(0,11)*5) )
    res.append( str(random.randint(9,12)) +":"+
str(random.randint(0,12)*5) )
    res.append( str(random.randint(12,15)) +
":"+str(random.randint(0,12)*5) )
    res.append( str(random.randint(15,18)) +
":"+str(random.randint(0,12)*5) )
    res.append( str(random.randint(18,21)) +
":"+str(random.randint(0,12)*5) )
    res.append( str(random.randint(21,23)) +
":"+str(random.randint(0,12)*5) )
    return res
cityRoute = solver.solve()
```

```
def createRoute(cityList):
    tempcityList = cityList.copy()
    chromosome = []
    for i in range(5):
      route = random.sample(tempcityList, 7)
      chromosome.append(route)
      tempcityList = Diff(tempcityList,route)
    return chromosome

def initialPopulation(popSize, cityList):
    population = []
    for i in range(0, popSize):
        population.append(createRoute(cityList))
    return population

def rankRoutes(population):
    fitnessResults = {}
    for i in range(0,len(population)):
        fitnessResults[i] =
Fitness(population[i]).routeFitness()
    return sorted(fitnessResults.items(), key =
operator.itemgetter(1), reverse = True)

def selection(popRanked, eliteSize):
    selectionResults = []
    df = pd.DataFrame(np.array(popRanked),
columns=["Index","Fitness"])
```

```
    df['cum_sum'] = df.Fitness.cumsum()
    df['cum_perc'] = 100*df.cum_sum/df.Fitness.sum()

    for i in range(0, eliteSize):
        selectionResults.append(popRanked[i][0])
    for i in range(0, len(popRanked) - eliteSize):
        pick = 100*random.random()
        for i in range(0, len(popRanked)):
            if pick <= df.iat[i,3]:
                selectionResults.append(popRanked[i][0])
                break
    return selectionResults

def matingPool(population, selectionResults):
    matingpool = []
    for i in range(0, len(selectionResults)):
        index = selectionResults[i]
        matingpool.append(population[index])
    return matingpool

def breed(parent1, parent2):
    copyParent1 = parent1.copy()
    copyParent1 = [ j for i in parent1 for j in i  ]
    copyParent2 = parent2.copy()
    copyParent2 = [ j for i in parent2 for j in i  ]
    child = []
    childP1 = []
    childP2 = []

    geneA = int(random.random() * len(copyParent1))
    geneB = int(random.random() * len(copyParent1))

    startGene = min(geneA, geneB)
    endGene = max(geneA, geneB)

    for i in range(startGene, endGene):
        childP1.append(copyParent1[i])
    for item in copyParent2:
      if item not in childP1 and len(childP1)<35:
        childP1.append(item)
    child = childP1
    offspring = []
    temp=[]
    for i in range(len(child)):
      temp.append(child[i])
      if(len(temp)==7):
        offspring.append(temp)
        temp=[]
    return offspring

def breedPopulation(matingpool, eliteSize):
    children = []
    length = len(matingpool) - eliteSize
    pool = random.sample(matingpool, len(matingpool))
    for i in range(0,eliteSize):
        children.append(matingpool[i])

    for i in range(0, length):
        child = breed(pool[i], pool[len(matingpool)-i-1])
        children.append(child)
    return children

def mutate(individual, mutationRate): # this can be
improved
    for swapped in range(len(individual)):
        if(random.random() < mutationRate):
            swapWith = int(random.random() * len(individual))

            city1 = individual[swapped]
            city2 = individual[swapWith]

            individual[swapped] = city2
            individual[swapWith] = city1
    return individual

def mutatePopulation(population, mutationRate):
    mutatedPop = []

    for ind in range(0, len(population)):
        mutatedInd = mutate(population[ind], mutationRate)
        mutatedPop.append(mutatedInd)
    return mutatedPop

def nextGeneration(currentGen, eliteSize, mutationRate):
    popRanked = rankRoutes(currentGen)
    selectionResults = selection(popRanked, eliteSize)
    matingpool = matingPool(currentGen, selectionResults)
    children = breedPopulation(matingpool, eliteSize)
    nextGeneration = mutatePopulation(children,
mutationRate)
    return nextGeneration

def geneticAlgorithm(population, popSize, eliteSize,
mutationRate, generations):
    pop = initialPopulation(popSize, population)
    print("Initial distance: " +
str(Fitness(pop[rankRoutes(pop)[0][0]]).routeDistance()))
    print("Initial population: " +
str(Fitness(pop[rankRoutes(pop)[0][0]]).routePopulation()))
    for i in range(0, generations):
        pop = nextGeneration(pop, eliteSize, mutationRate)

    total_distance=0
    total_population = 0
    for i in range (0,1):
        bestRouteIndex = rankRoutes(pop)[i][0]
        bestRoute = pop[bestRouteIndex]
        fitness = Fitness(bestRoute)
        fitness.routeFitness()
        for route in bestRoute:
            print(route,"\n")
        total_distance += fitness.distance
        total_population += fitness.total_population

    print("Total distance= "+ str(total_distance))
    FirstbestRouteIndex = rankRoutes(pop)[0][0]
    FirstbestRoute = pop[FirstbestRouteIndex]

    return FirstbestRoute

def toCity(city):
    ind = 0
    for i in range(len(city_name_data)):
        if city == city_name_data[0][i]:
            ind = i
            break
    return City(
name=ind,population=city_weight_data.iloc[ind,0],coord =
LatLongDict[ city_name_data.iloc[ind,0] ])


LatLongDict = {}
for i in range(len(city_coord_data)):
    LatLongDict[city_coord_data.iloc[i,0]] = [
city_coord_data.iloc[i,1] , city_coord_data.iloc[i,2]]

cityList = []
cities=[]
for i in range(0,len(city_name_data)):

cityList.append(City(name=i,population=city_weight_data.i
loc[i,0], coord = LatLongDict[ city_name_data.iloc[i,0] ] ))
    cities.append(city_name_data.iloc[i,0])

X=city_coord_data
X.columns =["Name","latitude","longitude","demand"]
kmeans = KMeans(n_clusters = 5, init ='k-means++')
kmeans.fit(np.array(X.iloc[:,1:3]))
X['cluster_label'] =
kmeans.fit_predict(np.array(X.iloc[:,1:3]))
centers = kmeans.cluster_centers_
labels = kmeans.predict(np.array(X.iloc[:,1:3]))
X.plot.scatter(x = 'latitude', y = 'longitude', c=labels, s=50,
cmap='viridis')
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200,
alpha=0.5)
# plt.show() # uncomment for graph
```

```
FirstbestRoute = geneticAlgorithm(population=cityList,
popSize=60, eliteSize=20, mutationRate=0.15,
generations=10)



for i in cityRoute:
    for j in range(len(i)):
        i[j] = toCity(str(i[j]))
AllRoutes = FirstbestRoute + cityRoute
index = 0
MapRouteToCity={}
for route in AllRoutes:
    if route[0].CityName() not in MapRouteToCity:
        MapRouteToCity[ route[0].CityName() ] = []
    MapRouteToCity[ route[0].CityName() ].append(route)

TotalRoute = []
for FromCity in MapRouteToCity.keys():
    timing = GenerateTiming()
    itr = 0
    for route in MapRouteToCity[ FromCity ]:
        TotalRoute.append( [ timing[itr] , route] )
        itr+=1
        if(itr==5) : itr =0
```

```
@app.route('/getAllCities')
def CityList():
    d = {}
    for i in range(len(cityList)):
        d[i] = [cityList[i].CityName() , cityList[i].CityCoord()]
    return json.dumps(d)

@app.route('/getBusRouteByID')
def getBusRouteByID():
    ID = request.args.get('ID', default = 0, type = int)
    if(ID<0 or ID>int(len(TotalRoute))):
        return "Invalid ID"
    else:
        res = {}
        for i in range(len(TotalRoute[ID][1])):
            res[i] = [ TotalRoute[ID][1][i].CityName() ,
TotalRoute[ID][1][i].CityCoord() ]
        res[-1] = TotalRoute[ID][0]
        return json.dumps(res)


@app.route('/getBusesBySrcDest')
def getBusesBySrcDest():
    src = request.args.get('src', default = 0, type =
str).lower()
```

```
    dest = request.args.get('dest', default = 0, type =
str).lower()
    res = {}
    for routeInd in range(len(TotalRoute)):
        l = [i.CityName().lower() for i in
TotalRoute[routeInd][1]]
        if src in l and dest in l:
            if(l.index(src) < l.index(dest)):
                res[routeInd] = [TotalRoute[routeInd][0]]+[
[i.CityName() , i.CityCoord()]  for i in
TotalRoute[routeInd][1] ]

    if res == {}:
        return "Invalid"
    return json.dumps(res)



if __name__ == '__main__':
   app.run(host="0.0.0.0",port=4000)




import numpy as nimport numpy as np, json, random,
solver, operator, pandas as pd
from flask import *
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import csv
app = Flask(__name__)

city_name_data =
pd.read_csv('namelist.csv',header=None)
city_dist_data = pd.read_csv('distlist.csv',header=None)
city_weight_data =
pd.read_csv('poplist.csv',header=None)
city_coord_data =
pd.read_csv('Latlong.csv',header=None)
print(city_name_data)
print(city_dist_data)
print(city_coord_data)

class City:
    def __init__(self,name,population,coord):
        self.name=name
        self.population=population
        self.coord=coord

    def distance(self, city):
        distance=city_dist_data.iloc[self.name,city.name]
        return distance

    def CityName(self):
        return str(city_name_data.iloc[self.name,0])

    def CityCoord(self):
        return self.coord

    def __repr__(self):
        return "\""+str(city_name_data.iloc[self.name,0]) + "
"+str(self.coord[0])+", "+str(self.coord[1])+"" +"\""

class Fitness:
    def __init__(self, route):
        self.chromosome = route
        self.distance = 0
        self.fitness= 0.0
        self.total_population= 0

    def routeDistance(self):
        pathDistance = 0
        for i in range(0, len(self.chromosome)):
          for j in range(0,len(self.chromosome[0])):
            fromCity = self.chromosome[i][j]
            toCity = None
            if j + 1  < len(self.chromosome[0]):
                toCity = self.chromosome[i][j +1] #doubtfull
            else:
                break
            if(type(toCity) == list): break
            pathDistance += fromCity.distance(toCity)
        self.distance = pathDistance
        return self.distance

    def routePopulation(self):
        path_population = 0
        for i in range(0, len(self.chromosome)):
          for j in range(0,len(self.chromosome[0])):
            City = self.chromosome[i][j]
            if type(City) != list:
              path_population += int(City.population.replace(',',
''))
            else:
              path_population = 0
        self.total_population = path_population
        return self.total_population

    def routeFitness(self):
        if self.fitness == 0:
          if(self.routeDistance()==0):
            return self.routePopulation()
          self.fitness = self.routePopulation() /
float(self.routeDistance())
        return self.fitness

def Diff(l1, l2):
    li_dif = [i for i in l1  if i not in l2]
    return li_dif

def GenerateTiming():
    res = []
    res.append( str(random.randint(6,9)) +
":"+str(random.randint(0,11)*5) )
    res.append( str(random.randint(9,12)) +":"+
str(random.randint(0,12)*5) )
    res.append( str(random.randint(12,15)) +
":"+str(random.randint(0,12)*5) )
    res.append( str(random.randint(15,18)) +
":"+str(random.randint(0,12)*5) )
    res.append( str(random.randint(18,21)) +
":"+str(random.randint(0,12)*5) )
    res.append( str(random.randint(21,23)) +
":"+str(random.randint(0,12)*5) )
    return res
cityRoute = solver.solve()
def createRoute(cityList):
    tempcityList = cityList.copy()
    chromosome = []
    for i in range(5):
      route = random.sample(tempcityList, 7)
      chromosome.append(route)
      tempcityList = Diff(tempcityList,route)
    return chromosome
```

```
def initialPopulation(popSize, cityList):
    population = []
    for i in range(0, popSize):
        population.append(createRoute(cityList))
    return population

def rankRoutes(population):
    fitnessResults = {}
    for i in range(0,len(population)):
        fitnessResults[i] =
Fitness(population[i]).routeFitness()
    return sorted(fitnessResults.items(), key =
operator.itemgetter(1), reverse = True)

def selection(popRanked, eliteSize):
    selectionResults = []
    df = pd.DataFrame(np.array(popRanked),
columns=["Index","Fitness"])
    df['cum_sum'] = df.Fitness.cumsum()
    df['cum_perc'] = 100*df.cum_sum/df.Fitness.sum()

    for i in range(0, eliteSize):
        selectionResults.append(popRanked[i][0])
    for i in range(0, len(popRanked) - eliteSize):
        pick = 100*random.random()
        for i in range(0, len(popRanked)):
            if pick <= df.iat[i,3]:
```

```
            selectionResults.append(popRanked[i][0])
            break
    return selectionResults

def matingPool(population, selectionResults):
    matingpool = []
    for i in range(0, len(selectionResults)):
        index = selectionResults[i]
        matingpool.append(population[index])
    return matingpool

def breed(parent1, parent2):
    copyParent1 = parent1.copy()
    copyParent1 = [ j for i in parent1 for j in i  ]
    copyParent2 = parent2.copy()
    copyParent2 = [ j for i in parent2 for j in i  ]
    child = []
    childP1 = []
    childP2 = []

    geneA = int(random.random() * len(copyParent1))
    geneB = int(random.random() * len(copyParent1))

    startGene = min(geneA, geneB)
    endGene = max(geneA, geneB)

    for i in range(startGene, endGene):
        childP1.append(copyParent1[i])
    for item in copyParent2:
      if item not in childP1 and len(childP1)<35:
        childP1.append(item)
    child = childP1
    offspring = []
    temp=[]
    for i in range(len(child)):
      temp.append(child[i])
      if(len(temp)==7):
        offspring.append(temp)
        temp=[]
    return offspring

def breedPopulation(matingpool, eliteSize):
    children = []
    length = len(matingpool) - eliteSize
    pool = random.sample(matingpool, len(matingpool))

    for i in range(0,eliteSize):
        children.append(matingpool[i])

    for i in range(0, length):
        child = breed(pool[i], pool[len(matingpool)-i-1])
        children.append(child)
    return children
```

```
def mutate(individual, mutationRate): # this can be
improved
    for swapped in range(len(individual)):
        if(random.random() < mutationRate):
            swapWith = int(random.random() * len(individual))

            city1 = individual[swapped]
            city2 = individual[swapWith]

            individual[swapped] = city2
            individual[swapWith] = city1
    return individual

def mutatePopulation(population, mutationRate):
    mutatedPop = []

    for ind in range(0, len(population)):
        mutatedInd = mutate(population[ind], mutationRate)
        mutatedPop.append(mutatedInd)
    return mutatedPop

def nextGeneration(currentGen, eliteSize, mutationRate):
    popRanked = rankRoutes(currentGen)
    selectionResults = selection(popRanked, eliteSize)
    matingpool = matingPool(currentGen, selectionResults)
    children = breedPopulation(matingpool, eliteSize)
    nextGeneration = mutatePopulation(children,
mutationRate)
    return nextGeneration

def geneticAlgorithm(population, popSize, eliteSize,
mutationRate, generations):
    pop = initialPopulation(popSize, population)
    print("Initial distance: " +
str(Fitness(pop[rankRoutes(pop)[0][0]]).routeDistance()))
    print("Initial population: " +
str(Fitness(pop[rankRoutes(pop)[0][0]]).routePopulation()))
    for i in range(0, generations):
        pop = nextGeneration(pop, eliteSize, mutationRate)

    total_distance=0
    total_population = 0
    for i in range (0,1):
        bestRouteIndex = rankRoutes(pop)[i][0]
        bestRoute = pop[bestRouteIndex]
        fitness = Fitness(bestRoute)
        fitness.routeFitness()
        for route in bestRoute:
            print(route,"\n")
        total_distance += fitness.distance
        total_population += fitness.total_population

    print("Total distance= "+ str(total_distance))
    FirstbestRouteIndex = rankRoutes(pop)[0][0]
    FirstbestRoute = pop[FirstbestRouteIndex]

    return FirstbestRoute

def toCity(city):
    ind = 0
    for i in range(len(city_name_data)):
        if city == city_name_data[0][i]:
            ind = i
            break
    return City(
name=ind,population=city_weight_data.iloc[ind,0],coord =
LatLongDict[ city_name_data.iloc[ind,0] ])


LatLongDict = {}
for i in range(len(city_coord_data)):
    LatLongDict[city_coord_data.iloc[i,0]] = [
city_coord_data.iloc[i,1] , city_coord_data.iloc[i,2]]

cityList = []
cities=[]
for i in range(0,len(city_name_data)):

cityList.append(City(name=i,population=city_weight_data.i
loc[i,0], coord = LatLongDict[ city_name_data.iloc[i,0] ] ))
```

```
    cities.append(city_name_data.iloc[i,0])

X=city_coord_data
X.columns =["Name","latitude","longitude","demand"]
kmeans = KMeans(n_clusters = 5, init ='k-means++')
kmeans.fit(np.array(X.iloc[:,1:3]))
X['cluster_label'] =
kmeans.fit_predict(np.array(X.iloc[:,1:3]))
centers = kmeans.cluster_centers_
labels = kmeans.predict(np.array(X.iloc[:,1:3]))
X.plot.scatter(x = 'latitude', y = 'longitude', c=labels, s=50,
cmap='viridis')
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200,
alpha=0.5)
# plt.show() # uncomment for graph


FirstbestRoute = geneticAlgorithm(population=cityList,
popSize=60, eliteSize=20, mutationRate=0.15,
generations=10)



for i in cityRoute:
    for j in range(len(i)):
        i[j] = toCity(str(i[j]))
AllRoutes = FirstbestRoute + cityRoute
```

```
index = 0
MapRouteToCity={}
for route in AllRoutes:
    if route[0].CityName() not in MapRouteToCity:
        MapRouteToCity[ route[0].CityName() ] = []
    MapRouteToCity[ route[0].CityName() ].append(route)

TotalRoute = []
for FromCity in MapRouteToCity.keys():
    timing = GenerateTiming()
    itr = 0
    for route in MapRouteToCity[ FromCity ]:
        TotalRoute.append( [ timing[itr] , route] )
        itr+=1
        if(itr==5) : itr =0




@app.route('/getAllCities')
def CityList():
    d = {}
    for i in range(len(cityList)):
        d[i] = [cityList[i].CityName() , cityList[i].CityCoord()]
    return json.dumps(d)

@app.route('/getBusRouteByID')
```

```
def getBusRouteByID():
   ID = request.args.get('ID', default = 0, type = int)
   if(ID<0 or ID>int(len(TotalRoute))):
      return "Invalid ID"
   else:
      res = {}
      for i in range(len(TotalRoute[ID][1])):
         res[i] = [ TotalRoute[ID][1][i].CityName() ,
TotalRoute[ID][1][i].CityCoord() ]
      res[-1] = TotalRoute[ID][0]
      return json.dumps(res)


@app.route('/getBusesBySrcDest')
def getBusesBySrcDest():
   src = request.args.get('src', default = 0, type =
str).lower()
   dest = request.args.get('dest', default = 0, type =
str).lower()
   res = {}
   for routeInd in range(len(TotalRoute)):
      l = [i.CityName().lower() for i in
TotalRoute[routeInd][1]]
      if src in l and dest in l:
         if(l.index(src) < l.index(dest)):
                res[routeInd] = [TotalRoute[routeInd][0]]+[
[i.CityName() , i.CityCoord()]  for i in
TotalRoute[routeInd][1] ]

    if res == {}:
        return "Invalid"
    return json.dumps(res)



if __name__ == '__main__':
   app.run(host="0.0.0.0",port=4000)
p, json, random, solver, operator, pandas as pd
from flask import *
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import csv
app = Flask(__name__)

city_name_data =
pd.read_csv('namelist.csv',header=None)
city_dist_data = pd.read_csv('distlist.csv',header=None)
city_weight_data =
pd.read_csv('poplist.csv',header=None)
city_coord_data =
pd.read_csv('Latlong.csv',header=None)
```

```
print(city_name_data)
print(city_dist_data)
print(city_coord_data)

class City:
    def __init__(self,name,population,coord):
        self.name=name
        self.population=population
        self.coord=coord

    def distance(self, city):
        distance=city_dist_data.iloc[self.name,city.name]
        return distance

    def CityName(self):
        return str(city_name_data.iloc[self.name,0])

    def CityCoord(self):
        return self.coord

    def __repr__(self):
        return "\""+str(city_name_data.iloc[self.name,0]) + "
"+str(self.coord[0])+", "+str(self.coord[1])+"" +"\""

class Fitness:
    def __init__(self, route):
        self.chromosome = route
        self.distance = 0
        self.fitness= 0.0
        self.total_population= 0

    def routeDistance(self):
        pathDistance = 0
        for i in range(0, len(self.chromosome)):
          for j in range(0,len(self.chromosome[0])):
            fromCity = self.chromosome[i][j]
            toCity = None
            if j + 1  < len(self.chromosome[0]):
                toCity = self.chromosome[i][j +1] #doubtfull
            else:
                break
            if(type(toCity) == list): break
            pathDistance += fromCity.distance(toCity)
        self.distance = pathDistance
        return self.distance

    def routePopulation(self):
        path_population = 0
        for i in range(0, len(self.chromosome)):
          for j in range(0,len(self.chromosome[0])):
            City = self.chromosome[i][j]
            if type(City) != list:
              path_population += int(City.population.replace(',',
''))
```

```
            else:
              path_population = 0
        self.total_population = path_population
        return self.total_population

    def routeFitness(self):
        if self.fitness == 0:
          if(self.routeDistance()==0):
            return self.routePopulation()
          self.fitness = self.routePopulation() /
float(self.routeDistance())
        return self.fitness

def Diff(l1, l2):
    li_dif = [i for i in l1  if i not in l2]
    return li_dif

def GenerateTiming():
    res = []
    res.append( str(random.randint(6,9)) +
":"+str(random.randint(0,11)*5) )
    res.append( str(random.randint(9,12)) +":"+
str(random.randint(0,12)*5) )
    res.append( str(random.randint(12,15)) +
":"+str(random.randint(0,12)*5) )
    res.append( str(random.randint(15,18)) +
":"+str(random.randint(0,12)*5) )
```

```
    res.append( str(random.randint(18,21)) +
":"+str(random.randint(0,12)*5) )
    res.append( str(random.randint(21,23)) +
":"+str(random.randint(0,12)*5) )
    return res
cityRoute = solver.solve()
def createRoute(cityList):
    tempcityList = cityList.copy()
    chromosome = []
    for i in range(5):
      route = random.sample(tempcityList, 7)
      chromosome.append(route)
      tempcityList = Diff(tempcityList,route)
    return chromosome

def initialPopulation(popSize, cityList):
    population = []
    for i in range(0, popSize):
        population.append(createRoute(cityList))
    return population

def rankRoutes(population):
    fitnessResults = {}
    for i in range(0,len(population)):
        fitnessResults[i] =
Fitness(population[i]).routeFitness()
```

```
    return sorted(fitnessResults.items(), key =
operator.itemgetter(1), reverse = True)

def selection(popRanked, eliteSize):
    selectionResults = []
    df = pd.DataFrame(np.array(popRanked),
columns=["Index","Fitness"])
    df['cum_sum'] = df.Fitness.cumsum()
    df['cum_perc'] = 100*df.cum_sum/df.Fitness.sum()

    for i in range(0, eliteSize):
        selectionResults.append(popRanked[i][0])
    for i in range(0, len(popRanked) - eliteSize):
        pick = 100*random.random()
        for i in range(0, len(popRanked)):
            if pick <= df.iat[i,3]:
                selectionResults.append(popRanked[i][0])
                break
    return selectionResults

def matingPool(population, selectionResults):
    matingpool = []
    for i in range(0, len(selectionResults)):
        index = selectionResults[i]
        matingpool.append(population[index])
    return matingpool
```

```
def breed(parent1, parent2):
    copyParent1 = parent1.copy()
    copyParent1 = [ j for i in parent1 for j in i  ]
    copyParent2 = parent2.copy()
    copyParent2 = [ j for i in parent2 for j in i  ]
    child = []
    childP1 = []
    childP2 = []

    geneA = int(random.random() * len(copyParent1))
    geneB = int(random.random() * len(copyParent1))

    startGene = min(geneA, geneB)
    endGene = max(geneA, geneB)

    for i in range(startGene, endGene):
        childP1.append(copyParent1[i])
    for item in copyParent2:
      if item not in childP1 and len(childP1)<35:
        childP1.append(item)
    child = childP1
    offspring = []
    temp=[]
    for i in range(len(child)):
      temp.append(child[i])
      if(len(temp)==7):
        offspring.append(temp)
        temp=[]
    return offspring

def breedPopulation(matingpool, eliteSize):
    children = []
    length = len(matingpool) - eliteSize
    pool = random.sample(matingpool, len(matingpool))

    for i in range(0,eliteSize):
        children.append(matingpool[i])

    for i in range(0, length):
        child = breed(pool[i], pool[len(matingpool)-i-1])
        children.append(child)
    return children

def mutate(individual, mutationRate): # this can be
improved
    for swapped in range(len(individual)):
        if(random.random() < mutationRate):
            swapWith = int(random.random() * len(individual))

            city1 = individual[swapped]
            city2 = individual[swapWith]

            individual[swapped] = city2
            individual[swapWith] = city1
    return individual

def mutatePopulation(population, mutationRate):
    mutatedPop = []

    for ind in range(0, len(population)):
        mutatedInd = mutate(population[ind], mutationRate)
        mutatedPop.append(mutatedInd)
    return mutatedPop

def nextGeneration(currentGen, eliteSize, mutationRate):
    popRanked = rankRoutes(currentGen)
    selectionResults = selection(popRanked, eliteSize)
    matingpool = matingPool(currentGen, selectionResults)
    children = breedPopulation(matingpool, eliteSize)
    nextGeneration = mutatePopulation(children,
mutationRate)
    return nextGeneration

def geneticAlgorithm(population, popSize, eliteSize,
mutationRate, generations):
    pop = initialPopulation(popSize, population)
    print("Initial distance: " +
str(Fitness(pop[rankRoutes(pop)[0][0]]).routeDistance()))
    print("Initial population: " +
str(Fitness(pop[rankRoutes(pop)[0][0]]).routePopulation()))
    for i in range(0, generations):
        pop = nextGeneration(pop, eliteSize, mutationRate)

    total_distance=0
    total_population = 0
    for i in range (0,1):
        bestRouteIndex = rankRoutes(pop)[i][0]
        bestRoute = pop[bestRouteIndex]
        fitness = Fitness(bestRoute)
        fitness.routeFitness()
        for route in bestRoute:
            print(route,"\n")
        total_distance += fitness.distance
        total_population += fitness.total_population

    print("Total distance= "+ str(total_distance))
    FirstbestRouteIndex = rankRoutes(pop)[0][0]
    FirstbestRoute = pop[FirstbestRouteIndex]

    return FirstbestRoute

def toCity(city):
    ind = 0
    for i in range(len(city_name_data)):
        if city == city_name_data[0][i]:
            ind = i
            break
    return City(
name=ind,population=city_weight_data.iloc[ind,0],coord =
LatLongDict[ city_name_data.iloc[ind,0] ])


LatLongDict = {}
for i in range(len(city_coord_data)):
    LatLongDict[city_coord_data.iloc[i,0]] = [
city_coord_data.iloc[i,1] , city_coord_data.iloc[i,2]]

cityList = []
cities=[]
for i in range(0,len(city_name_data)):

cityList.append(City(name=i,population=city_weight_data.i
loc[i,0], coord = LatLongDict[ city_name_data.iloc[i,0] ] ))
    cities.append(city_name_data.iloc[i,0])

X=city_coord_data
X.columns =["Name","latitude","longitude","demand"]
kmeans = KMeans(n_clusters = 5, init ='k-means++')
kmeans.fit(np.array(X.iloc[:,1:3]))
X['cluster_label'] =
kmeans.fit_predict(np.array(X.iloc[:,1:3]))
centers = kmeans.cluster_centers_
labels = kmeans.predict(np.array(X.iloc[:,1:3]))
```

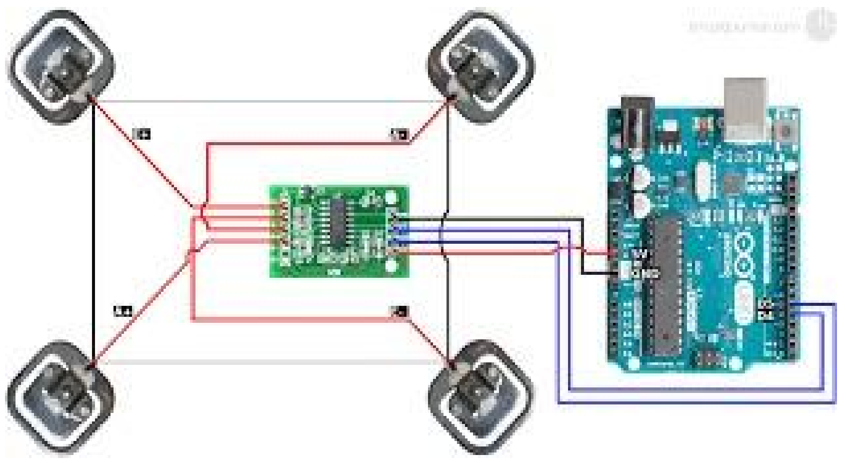
```
X.plot.scatter(x = 'latitude', y = 'longitude', c=labels, s=50,
cmap='viridis')
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200,
alpha=0.5)
# plt.show() # uncomment for graph


FirstbestRoute = geneticAlgorithm(population=cityList,
popSize=60, eliteSize=20, mutationRate=0.15,
generations=10)



for i in cityRoute:
    for j in range(len(i)):
        i[j] = toCity(str(i[j]))
AllRoutes = FirstbestRoute + cityRoute
index = 0
MapRouteToCity={}
for route in AllRoutes:
    if route[0].CityName() not in MapRouteToCity:
        MapRouteToCity[ route[0].CityName() ] = []
    MapRouteToCity[ route[0].CityName() ].append(route)

TotalRoute = []
for FromCity in MapRouteToCity.keys():
    timing = GenerateTiming()
    itr = 0
    for route in MapRouteToCity[ FromCity ]:
        TotalRoute.append( [ timing[itr] , route] )
        itr+=1
        if(itr==5) : itr =0




@app.route('/getAllCities')
def CityList():
    d = {}
    for i in range(len(cityList)):
        d[i] = [cityList[i].CityName() , cityList[i].CityCoord()]
    return json.dumps(d)

@app.route('/getBusRouteByID')
def getBusRouteByID():
    ID = request.args.get('ID', default = 0, type = int)
    if(ID<0 or ID>int(len(TotalRoute))):
        return "Invalid ID"
    else:
        res = {}
        for i in range(len(TotalRoute[ID][1])):
            res[i] = [ TotalRoute[ID][1][i].CityName() ,
TotalRoute[ID][1][i].CityCoord() ]
        res[-1] = TotalRoute[ID][0]
        return json.dumps(res)


@app.route('/getBusesBySrcDest')
def getBusesBySrcDest():
    src = request.args.get('src', default = 0, type =
str).lower()
    dest = request.args.get('dest', default = 0, type =
str).lower()
    res = {}
    for routeInd in range(len(TotalRoute)):
        l = [i.CityName().lower() for i in
TotalRoute[routeInd][1]]
        if src in l and dest in l:
            if(l.index(src) < l.index(dest)):
                res[routeInd] = [TotalRoute[routeInd][0]]+[
[i.CityName() , i.CityCoord()]  for i in
TotalRoute[routeInd][1] ]

    if res == {}:
        return "Invalid"
    return json.dumps(res)



if __name__ == '__main__':
   app.run(host="0.0.0.0",port=4000)
```

PASSENGER COUNTING SENSORS

Sensors can be used to count the number of people in public transport in IoT in a variety of ways. Some of the most common methods include:

Weight sensors:

Weight sensors can be placed on the floor of public transport vehicles to measure the total weight of the passengers on board. This information can then be used to estimate the number of passengers on the vehicle, based on the average weight of a person.

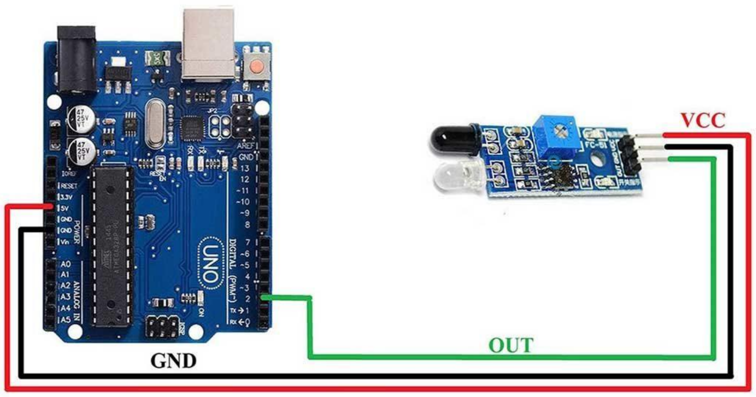


Passenger counting gates:

Passenger counting gates can be installed at the entrance and exit of public transport vehicles to count the number of passengers entering and exiting the vehicle. This information can then be used to estimate the total number of passengers on the vehicle.

Infrared sensors:

Infrared sensors can be used to detect the presence of people in a given area. By placing infrared sensors at the entrance and exit of public transport vehicles, it is possible to count the number of passengers entering and exiting the vehicle.



Video cameras:

Video cameras can be used to count the number of people in a given area by using image processing techniques to identify and track individuals. Video cameras are often used in conjunction with other sensors, such as infrared sensors or weight sensors, to improve the accuracy of the count.

The specific method used to count the number of people in public transport using IoT will depend on a variety of factors, such as the type of public transport vehicle, the budget available, and the desired accuracy of the count.

Python script to counting passengers in public vehicles

Script 1 (weight sensor)

```
import time
import requests

def get_weight_reading(sensor_id):
    url =
"https://api.example.org/get_weight_reading?sensor_id={}".format(sen
sor_id)
  response = requests.get(url)
  return response.json()

def count_passengers(weight_reading, average_weight_per_person):
    return weight_reading / average_weight_per_person

def main():
    weight_reading = get_weight_reading(sensor_id=1234)
    average_weight_per_person = 70
number_of_passengers=count_passengers(weight_reading=weight_r
eading, average_weight_per_person=average_weight_per_person)

print("The number of passengers is:",number_of_passengers)

if __name__ == "__main__":
  main()
```

Script 2 (infrared sensor)

```
import time
import requests

def get_infrared_sensor_reading(sensor_id):
    url =
"https://api.example.org/get_infrared_sensor_reading?sensor_id={}".fo
rmat(sensor_id)
  response = requests.get(url)
  return response.json()

def count_people(infrared_sensor_reading, threshold):
    if infrared_sensor_reading > threshold:
    return 1
  else:
    return 0

def main():
    infrared_sensor_reading =
get_infrared_sensor_reading(sensor_id=1234)

    threshold = 100
    number_of_people =
count_people(infrared_sensor_reading=infrared_sensor_reading,
threshold=threshold)

    print("The number of people is:", number_of_people)

if __name__ == "__main__":
  main()
```

Researchers have been exploring the problem of reducing waiting time as well as reducing bunching in buses.

- applied data mining on data collected using Automatic Passenger Counters. The paper uses clustering to divide data points into different headways based on a decision tree. The method informs whether the existing headway would work for the buses.

- used a sequential clustering method to ensure a fixed order for buses so that time division could be properly applied. They calculated the travel time by incorporating bus dwell time at stops which depends on the number of riders and also the road and intersection time depending on the traffic on the road at that time.

- proposed an approach to minimize waiting time using two models which are based on the assumption of how people arrive at the bus stop.

- proposed a method to optimize the timetable for the subway system by proposing an integer programming model to maximize the overlap time with the headway time.

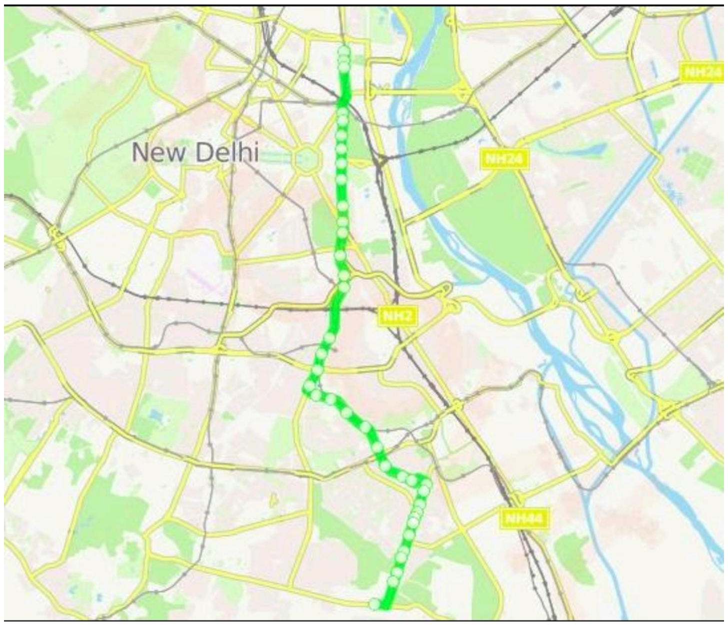
- also proposed an integer programming model which uses a modified generic algorithm for timetabling of buses.

- a mixed integer nonlinear programming model was proposed for timetabling of buses where the objective was to minimize the total waiting time of passengers.

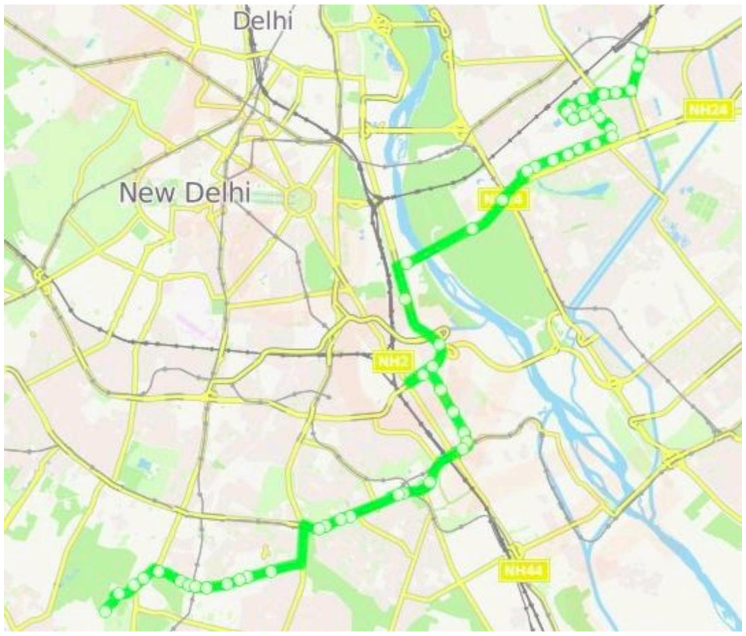
- Studied Open traffic platform which analysis traffic data linked to open street map. They used it to analyze the traffic conditions in Kuala Lumpur. They listed different roads and junctions with heavy traffic flow at different times.

- proposed a dynamic GPS based time-tabling algorithm for public transport that can predict the waiting time considering the real-time location of the user and the bus, and thus predicting estimated time of arrival.

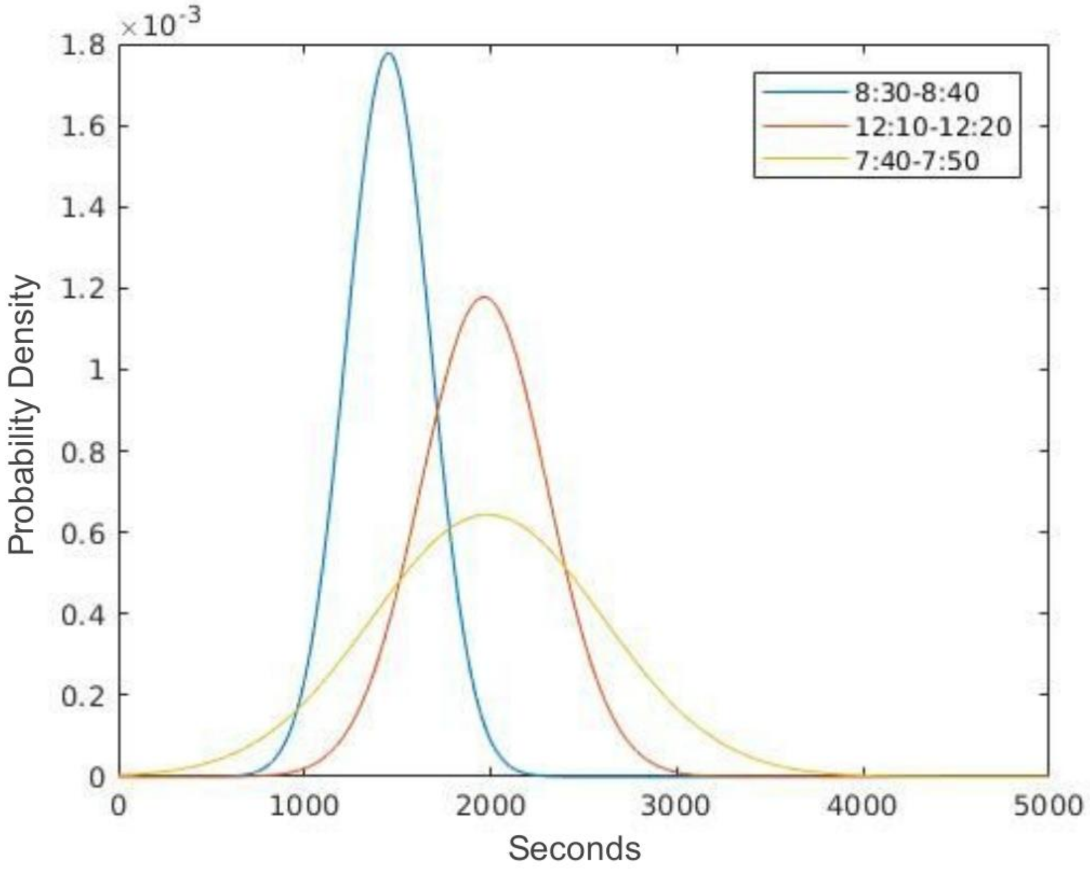
The major issue with current research in the domain of timetable optimization for buses is that there is not a standard dataset which is being used by researchers. Various algorithms are applied on various datasets, which does not provide conclusive evidence of improvement in the state of the art The proposed algorithm used for benchmarking the proposed dataset goes beyond the ones in literature in two ways.Firstly, literature does not take the simplicity of the algorithm into consideration. This is vital primarily as the IT Department of most transportation corporations are not educated enough to work with complex algorithms. The proposed algorithm closely revolves around the traffic behavior to provide the most optimal timetable. This algorithm is being deployed to update the timetable of over a hundred buses in New Delhi, India which would affect the lives of millions of daily bus users.



Route for bus no. 534



Route for bus no. 425



Probability distribution of the arrival time at stop 17 for route no.425 at different times of the day.

The requirements for such a website can vary depending on the specific needs of the transportation system. However, some common requirements include:-

Real-time reporting:

Real-time reporting is essential for improving efficiency and service levels. It helps passengers to plan their trips better and avoid delays.

Flow
Road sensor
Traffic camera
Commercial Vehicle GPS Probe
Consumer Vehicle GPS Probe
GPS-Enabled Smartphone Probe
Cellular Network Probe
Incidents
Driver Generated Report
Accident
Construction
Event

Transportation planning:

Transportation planning is a very complicated problem because planners must take into account multiple competing criteria (e.g., service requirements, asset utilization, cost minimization, workload fairness,etc.), components of the transportation system, and their interconnection.Moreover, transportation planners should consider many factors to create efficient plans, including but not limited to vehicle and driver availability, vehicle size and capacity, traffic details, travel time windows, and passengers’ locations.

Optimization:

Optimization is the process of finding the best solution to a problem. In the context of public transport optimization websites, optimization can be used to improve the efficiency of public transportation systems by reducing travel time, minimizing costs, and maximizing passenger satisfaction.

Integration:

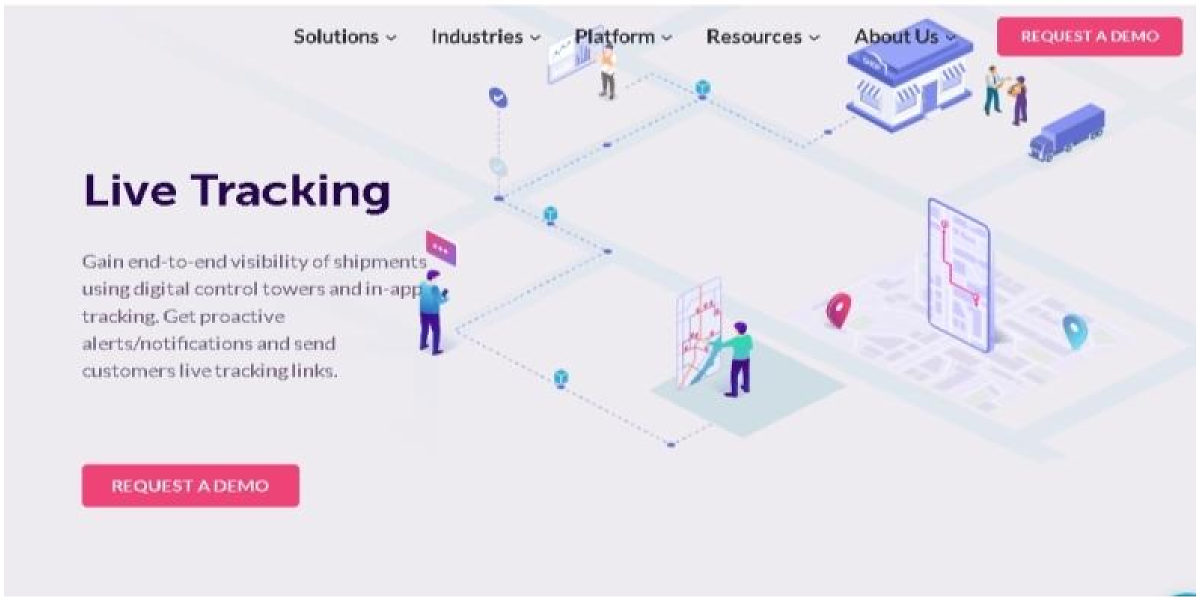
Public transport systems must be planned and operated as a seamless, integrated system. This is particularly important in urban environments where public transport must increasingly compete with private vehicles which offer door-to-door, “one seat” travel irrespective of time of day or day of the week. Successful integration will provide a more customer-friendly experience and make public transport more efficient and cost-effective.

Service planning:

Public transportation planning mainly includes short-term (operational) planning, medium-term (tactical) planning, and long-term (strategic) planning. Strategic planning consists of decisions that have an impact lasting many years. Decisions about the locations of terminals and stops are an example of strategic planning. Operational planning consists of problems like designing bus routes, determining frequencies, and scheduling vehicles and drivers.I hope this helps!

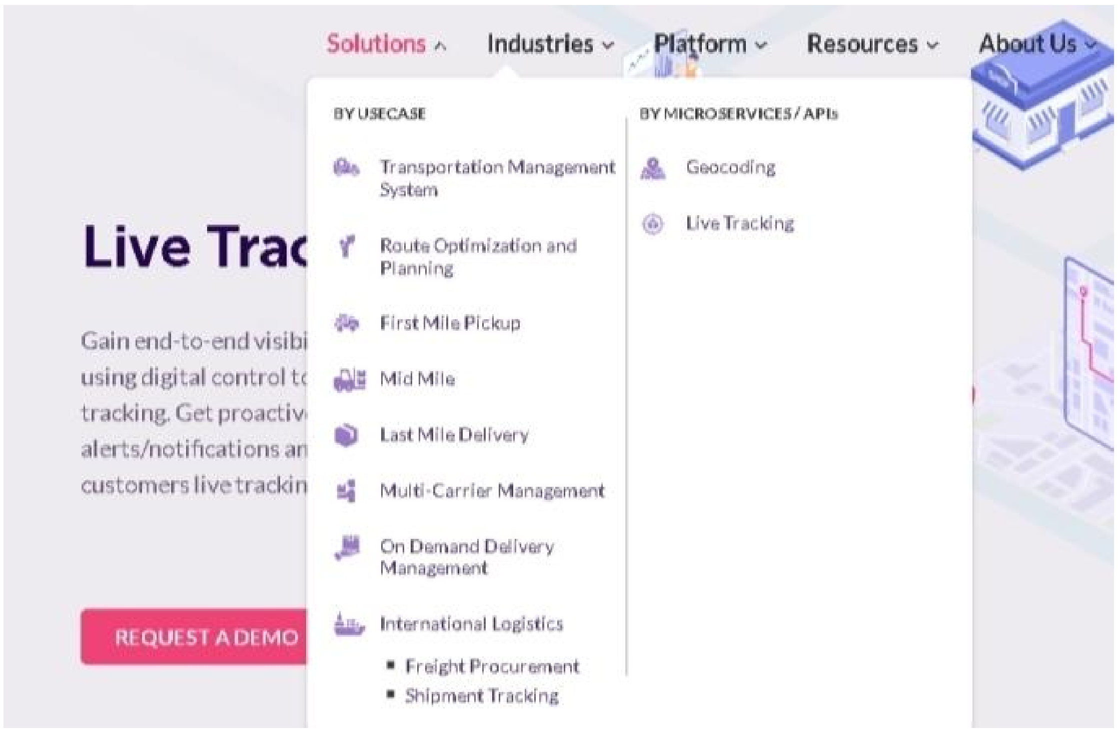
THE MODEL OF OUR HOSTED WEBSITE:-

SITES TOP LOOK:



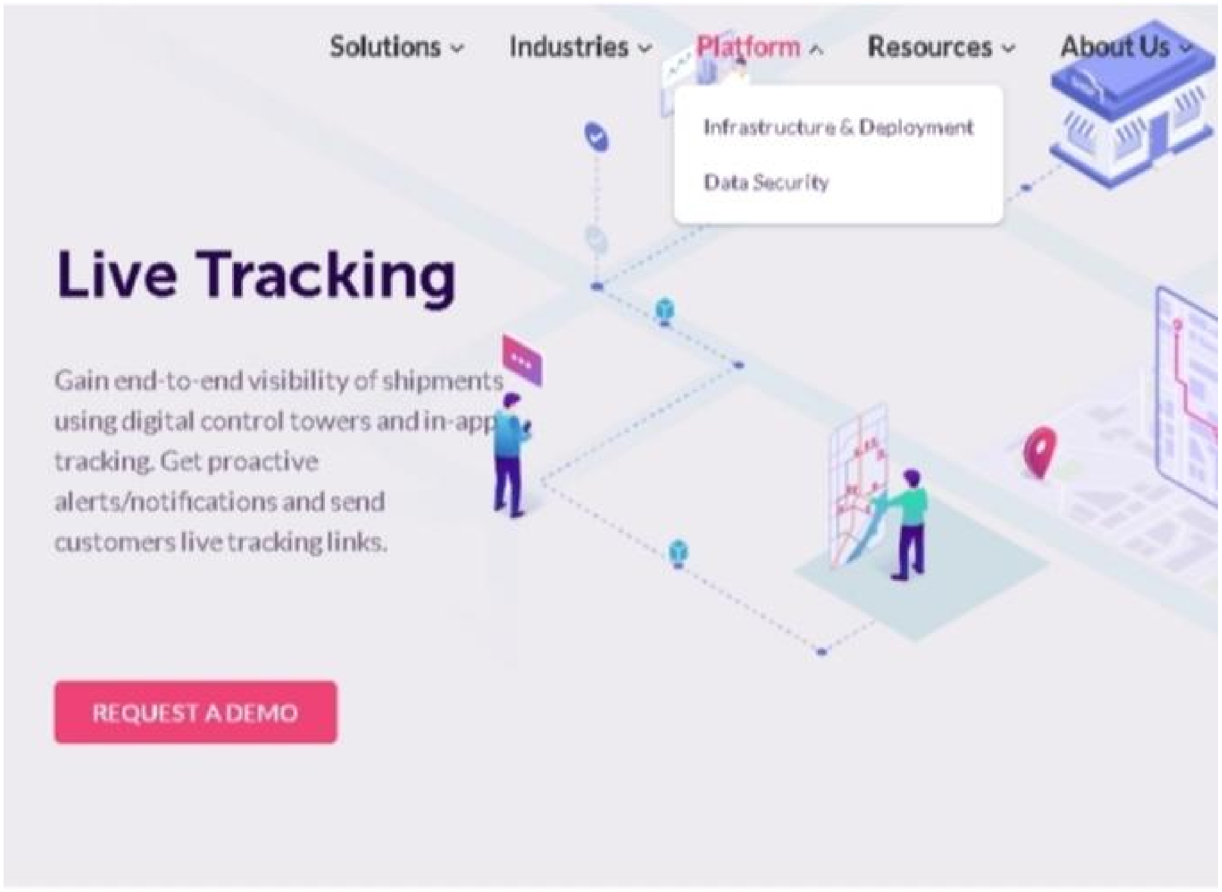
Live Tracking

Gain end-to-end visibility of shipments using digital control towers and in-app tracking. Get proactive alerts/notifications and send customers live tracking links.



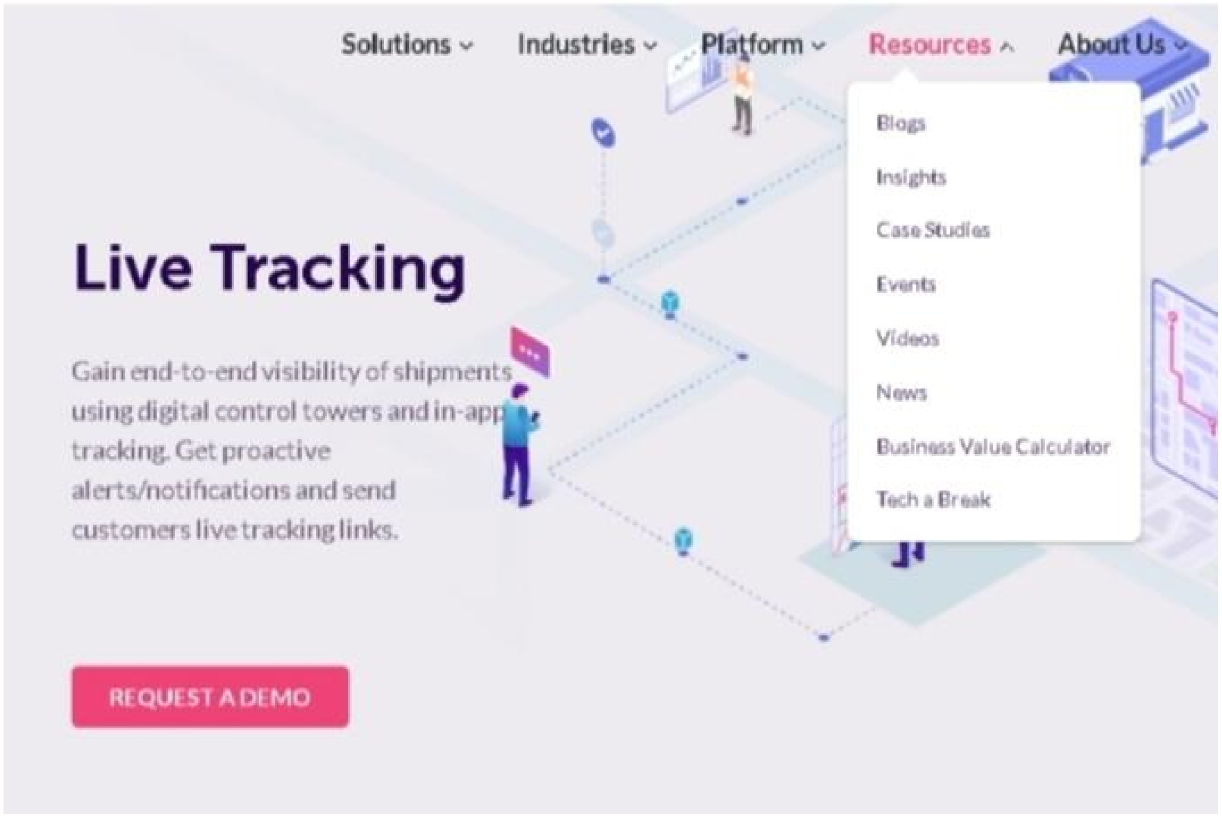
Live Trac

Gain end-to-end visibi using digital control tc tracking. Get proactiv alerts/notifications an customers live trackin



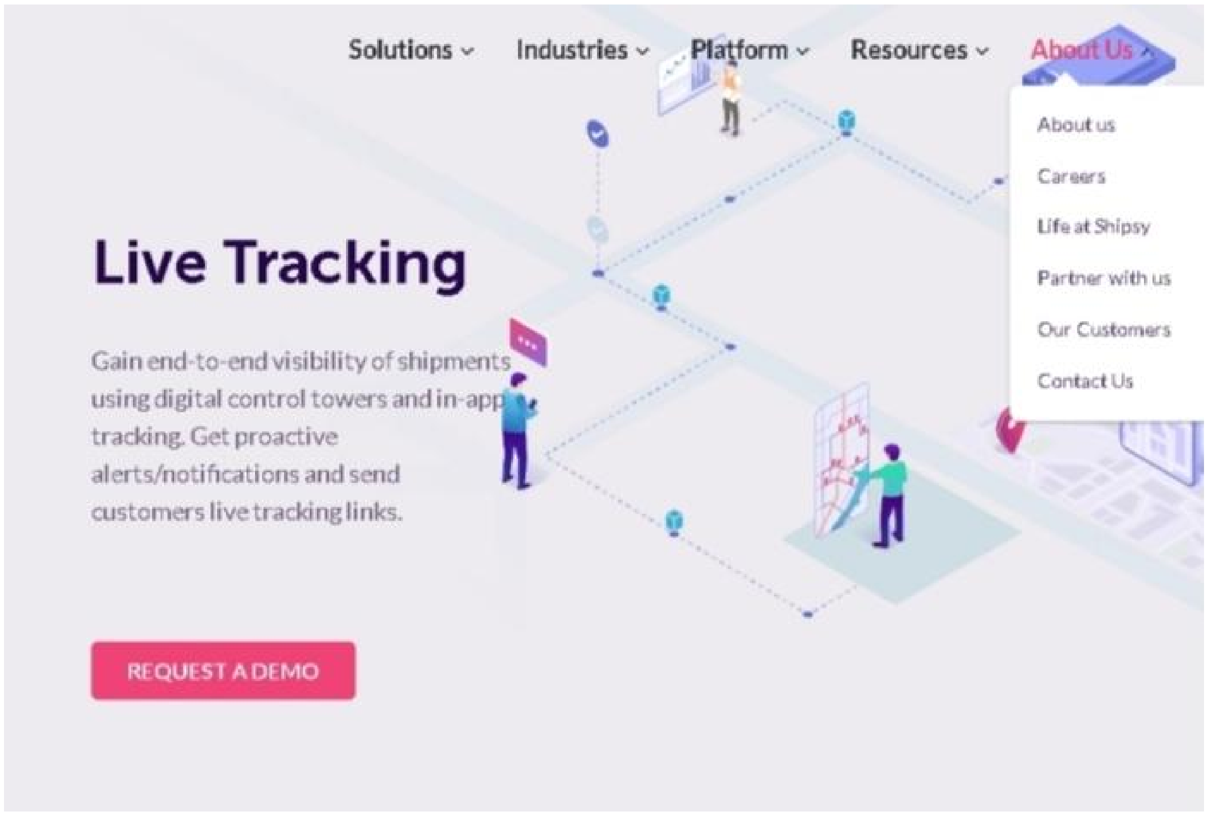
Live Tracking

Gain end-to-end visibility of shipments using digital control towers and in-app tracking. Get proactive alerts/notifications and send customers live tracking links.



Live Tracking

Gain end-to-end visibility of shipments using digital control towers and in-app tracking. Get proactive alerts/notifications and send customers live tracking links.



Live Tracking

Gain end-to-end visibility of shipments using digital control towers and in-app tracking. Get proactive alerts/notifications and send customers live tracking links.

ADVANTAGES OF OUR WEBSITE:-

- Low-Code Integrations
- Third-Party Connectors
- No Upfront Costs
- Quick Time to Implement
- Future Ready Platform
- Consultative Approach

CONCLUSION:-

Those who adapt themselves to this website will find it easy for both ovesease and local transportatio