**College code:**9133

**Course:**Internet of things

**Phase 4:**Development part 2

**Project title:**Public Transport Optimization

**Team members:**

M.Jothy meena

   (Reg no:913321106014)

   K.Nithyaindhuja

   (Reg no:913321106026)

   G.Pandiselvi

   (Reg no:913321106029)

   T.Pavithra

   (Reg no:913321106030)

   R.Srinithi

   (Reg no:913321106045)

**Topic:**

In this technology you will continue building your project by selecting a machine learning algorithm, training the model , and evaluating its performance. Perform different analysis as needed. After performing the relevant activities create a document around it and share the same for assessment.

**Create the data:**

The following function creates the data for the problem.

Def create\_data\_model():

“””Stores the data for the problem.”””

Data = {}

Data[“distance\_matrix”] = [

# fmt: off

[0, 548, 776, 696, 582, 274, 502, 194, 308, 194, 536, 502, 388, 354, 468, 776, 662],

[548, 0, 684, 308, 194, 502, 730, 354, 696, 742, 1084, 594, 480, 674, 1016, 868, 1210],

[776, 684, 0, 992, 878, 502, 274, 810, 468, 742, 400, 1278, 1164, 1130, 788, 1552, 754],

[696, 308, 992, 0, 114, 650, 878, 502, 844, 890, 1232, 514, 628, 822, 1164, 560, 1358],

[582, 194, 878, 114, 0, 536, 764, 388, 730, 776, 1118, 400, 514, 708, 1050, 674, 1244],

[274, 502, 502, 650, 536, 0, 228, 308, 194, 240, 582, 776, 662, 628, 514, 1050, 708],

[502, 730, 274, 878, 764, 228, 0, 536, 194, 468, 354, 1004, 890, 856, 514, 1278, 480],

[194, 354, 810, 502, 388, 308, 536, 0, 342, 388, 730, 468, 354, 320, 662, 742, 856],

[308, 696, 468, 844, 730, 194, 194, 342, 0, 274, 388, 810, 696, 662, 320, 1084, 514],

[194, 742, 742, 890, 776, 240, 468, 388, 274, 0, 342, 536, 422, 388, 274, 810, 468],

[536, 1084, 400, 1232, 1118, 582, 354, 730, 388, 342, 0, 878, 764, 730, 388, 1152, 354],

[502, 594, 1278, 514, 400, 776, 1004, 468, 810, 536, 878, 0, 114, 308, 650, 274, 844],

[388, 480, 1164, 628, 514, 662, 890, 354, 696, 422, 764, 114, 0, 194, 536, 388, 730],

[354, 674, 1130, 822, 708, 628, 856, 320, 662, 388, 730, 308, 194, 0, 342, 422, 536],

[468, 1016, 788, 1164, 1050, 514, 514, 662, 320, 274, 388, 650, 536, 342, 0, 764, 194],

[776, 868, 1552, 560, 674, 1050, 1278, 742, 1084, 810, 1152, 274, 388, 422, 764, 0, 798],

[662, 1210, 754, 1358, 1244, 708, 480, 856, 514, 468, 354, 844, 730, 536, 194, 798, 0],

# fmt: on

]

Data[“num\_vehicles”] = 4

Data[“depot”] = 0

Return data

**The data consists of:**

* distance\_matrix: An array of distances between locations on meters.
* num\_locations: The number of locations.
* num\_vehicles: The number of vehicles in the fleet.
* depot: The index of the depot, the location where all vehicles start and end their routes.

### **The distance callback:**

As in the [TSP example](https://developers.google.com/optimization/routing/tsp#create_the_distance_callback), the following function creates the distance callback, which returns the distances between locations, and passes it to the solver. It also sets the arc costs, which define the cost of travel, to be the distances of the arcs.

def distance\_callback(from\_index, to\_index):  
    """Returns the distance between the two nodes."""  
    # Convert from routing variable Index to distance matrix NodeIndex.  
    from\_node = manager.IndexToNode(from\_index)  
    to\_node = manager.IndexToNode(to\_index)  
    return data["distance\_matrix"][from\_node][to\_node]  
  
transit\_callback\_index = routing.RegisterTransitCallback(distance\_callback)  
routing.SetArcCostEvaluatorOfAllVehicles(transit\_callback\_index)

**Add a distance dimension:**

The following code creates the distance dimension, using the solver’s AddDimension method. The argument transit\_callback\_index is the index for the distance\_callback.

dimension\_name = "Distance"  
routing.AddDimension(  
    transit\_callback\_index,  
    0,  # no slack  
    3000,  # vehicle maximum travel distance  
    True,  # start cumul to zero  
    dimension\_name,  
)  
distance\_dimension = routing.GetDimensionOrDie(dimension\_name)  
distance\_dimension.SetGlobalSpanCostCoefficient(100)

### **Add the solution printer:**

### The function that prints the solution is shown below.

def print\_solution(data, manager, routing, solution):  
    """Prints solution on console."""  
    print(f"Objective: {solution.ObjectiveValue()}")  
    max\_route\_distance = 0  
    for vehicle\_id in range(data["num\_vehicles"]):  
        index = routing.Start(vehicle\_id)  
        plan\_output = f"Route for vehicle {vehicle\_id}:\n"  
        route\_distance = 0  
        while not routing.IsEnd(index):  
            plan\_output += f" {manager.IndexToNode(index)} -> "  
            previous\_index = index  
            index = solution.Value(routing.NextVar(index))  
            route\_distance += routing.GetArcCostForVehicle(  
                previous\_index, index, vehicle\_id  
            )  
        plan\_output += f"{manager.IndexToNode(index)}\n"  
        plan\_output += f"Distance of the route: {route\_distance}m\n"  
        print(plan\_output)  
        max\_route\_distance = max(route\_distance, max\_route\_distance)  
    print(f"Maximum of the route distances: {max\_route\_distance}m")

The function displays the routes for the vehicles and the total distances of the routes.

### **Running the programs**

### When you run the programs, they display the following output:

Route for vehicle 0:

0 -> 8 -> 6 -> 2 -> 5 -> 0

Distance of route: 1552m

Route for vehicle 1:

0 -> 7 -> 1 -> 4 -> 3 -> 0

Distance of route: 1552m

Route for vehicle 2:

0 -> 9 -> 10 -> 16 -> 14 -> 0

Distance of route: 1552m

Route for vehicle 3:

0 -> 12 -> 11 -> 15 -> 13 -> 0

Distance of route: 1552m

Total distance of all routes: 6208m

**Using the Google distance Matrix API:**

The section shows how to use the Google Distance Matrix API to create the distance matrix for any set of locations defined by addresses, or by latitudes and longitudes. You can use the API to calculate the distance matrix.

### Example

As an example, we'll walk through a Python program that creates the distance matrix for a set of 16 locations in the city of Memphis, Tennessee. The distance matrix is a 16 x 16 matrix whose i, j entry is the distance between locations i and j. Here are the addresses for the locations.

data['addresses'] = ['3610+Hacks+Cross+Rd+Memphis+TN', # depot  
                     '1921+Elvis+Presley+Blvd+Memphis+TN',  
                     '149+Union+Avenue+Memphis+TN',  
                     '1034+Audubon+Drive+Memphis+TN',  
                     '1532+Madison+Ave+Memphis+TN',  
                     '706+Union+Ave+Memphis+TN',  
                     '3641+Central+Ave+Memphis+TN',  
                     '926+E+McLemore+Ave+Memphis+TN',  
                     '4339+Park+Ave+Memphis+TN',  
                     '600+Goodwyn+St+Memphis+TN',  
                     '2000+North+Pkwy+Memphis+TN',  
                     '262+Danny+Thomas+Pl+Memphis+TN',  
                     '125+N+Front+St+Memphis+TN',  
                     '5959+Park+Ave+Memphis+TN',  
                     '814+Scott+St+Memphis+TN',  
                     '1005+Tillman+St+Memphis+TN'  
                    ]

### **API requests:**

Here's the response to the request.

{

"destination\_addresses" : [ "1921 Elvis Presley Blvd, Memphis, TN 38106, USA" ],

"origin\_addresses" : [ "3610 Hacks Cross Rd, Memphis, TN 38125, USA" ],

"rows" : [

{

"elements" : [

{

"distance" : {

"text" : "15.2 mi",

"value" : 24392

},

"duration" : {

"text" : "21 mins",

"value" : 1264

},

"status" : "OK"

}

]

}

],

"status" : "OK"

}

The response contains the travel distance (in miles and meters), and the travel duration (in minutes and seconds), between the two addresses.

### **Compute the distance matrix:**

To compute the distance matrix, we would like to send a single request containing all 16 addresses as both the origin and destination addresses. However, we can't because this would require 16x16=256 origin-destination pairs, while the API is restricted to 100 such pairs per request. So we need to make multiple requests.

The following code computes the distance matrix as follows:

* Divide the 16 addresses into two groups of six addresses and one group of four addresses.
* For each group, build and send a request for the origin addresses in the group and all destination addresses. See [Build and send a request](https://developers.google.com/optimization/routing/vrp#send_request).
* Use the response to build the corresponding rows of the matrix, and concatenate the rows (which are just Python lists). See [Build rows of the distance matrix](https://developers.google.com/optimization/routing/vrp#create_the_data).

Def create\_distance\_matrix(data):

Addresses = data[“addresses”]

API\_key = data[“API\_key”]

# Distance Matrix API only accepts 100 elements per request, so get rows in multiple requests.

Max\_elements = 100

Num\_addresses = len(addresses) # 16 in this example.

# Maximum number of rows that can be computed per request (6 in this example).

Max\_rows = max\_elements // num\_addresses

# num\_addresses = q \* max\_rows + r (q = 2 and r = 4 in this example).

Q, r = divmod(num\_addresses, max\_rows)

Dest\_addresses = addresses

Distance\_matrix = []

# Send q requests, returning max\_rows rows per request.

For I in range(q):

Origin\_addresses = addresses[I \* max\_rows: (I + 1) \* max\_rows]

Response = send\_request(origin\_addresses, dest\_addresses, API\_key)

Distance\_matrix += build\_distance\_matrix(response)

# Get the remaining remaining r rows, if necessary.

If r > 0:

Origin\_addresses = addresses[q \* max\_rows: q \* max\_rows + r]

Response = send\_request(origin\_addresses, dest\_addresses, API\_key)

Distance\_matrix += build\_distance\_matrix(response)

Return distance\_matrix

**Build and send a request:**

The following function builds and sends a request for a given set of origin and destination addresses.

Def send\_request(origin\_addresses, dest\_addresses, API\_key):

“”” Build and send request for the given origin and destination addresses.”””

Def build\_address\_str(addresses):

# Build a pipe-separated string of addresses

Address\_str = ‘’

For I in range(len(addresses) – 1):

Address\_str += addresses[i] + ‘|’

Address\_str += addresses[-1]

Return address\_str

Request = ‘https://maps.googleapis.com/maps/api/distancematrix/json?units=imperial’

Origin\_address\_str = build\_address\_str(origin\_addresses)

Dest\_address\_str = build\_address\_str(dest\_addresses)

Request = request + ‘&origins=’ + origin\_address\_str + ‘&destinations=’ + \

Dest\_address\_str + ‘&key=’ + API\_key

jsonResult = urllib.urlopen(request).read()

response = json.loads(jsonResult)

return response

The sub-function build\_address\_string concatenates addresses separated by the pipe character, |.

The remaining code in the function assembles the parts of the request described above, and sends the request. The line

Response = json.loads(jsonResult)

converts the raw result to a Python object.

**Run the program:**

The following code in the main function runs the program

Def main():

“””Entry point of the program”””

# Create the data.

Data = create\_data()

Addresses = data[‘addresses’]

API\_key = data[‘API\_key’]

Distance\_matrix = create\_distance\_matrix(data)

Print(distance\_matrix)

When you run the program, it prints the distance matrix, as shown below

[[0, 24392, 33384, 14963, 31992, 32054, 20866, 28427, 15278, 21439, 28765, 34618, 35177, 10612, 26762, 27278],

[25244, 0, 8314, 10784, 6922, 6984, 10678, 3270, 10707, 7873, 11350, 9548, 10107, 19176, 12139, 13609],

[34062, 8491, 0, 14086, 4086, 1363, 11008, 4239, 13802, 9627, 7179, 1744, 925, 27994, 9730, 10531],

[15494, 13289, 13938, 0, 11065, 12608, 4046, 10970, 581, 5226, 10788, 15500, 16059, 5797, 9180, 9450],

[33351, 7780, 4096, 11348, 0, 2765, 7364, 4464, 11064, 6736, 3619, 4927, 5485, 20823, 6170, 7076],

[32731, 7160, 1363, 12755, 2755, 0, 9677, 3703, 12471, 8297, 7265, 2279, 2096, 26664, 9816, 9554],

[19636, 10678, 11017, 4038, 7398, 9687, 0, 9159, 3754, 2809, 7099, 10740, 11253, 8970, 5491, 5928],

[29097, 3270, 4257, 11458, 4350, 3711, 9159, 0, 11174, 6354, 10160, 5178, 5258, 23029, 10620, 12419],

[15809, 10707, 13654, 581, 10781, 12324, 3763, 10687, 0, 4943, 10504, 15216, 15775, 5216, 8896, 9166],

[21831, 7873, 9406, 5226, 6282, 8075, 2809, 6354, 4943, 0, 6967, 10968, 11526, 10159, 5119, 6383],

[28822, 11931, 6831, 11802, 3305, 6043, 7167, 10627, 11518, 7159, 0, 5361, 6422, 18351, 3267, 4068],

[35116, 9545, 1771, 15206, 4648, 2518, 10967, 5382, 14922, 10747, 5909, 0, 1342, 29094, 8460, 9260],

[36058, 10487, 927, 16148, 5590, 2211, 11420, 9183, 15864, 11689, 6734, 1392, 0, 30036, 9285, 10086],

[11388, 19845, 28838, 5797, 20972, 27507, 8979, 23880, 5216, 10159, 18622, 29331, 29890, 0, 16618, 17135],

[27151, 11444, 9719, 10131, 6193, 8945, 5913, 10421, 9847, 5374, 3335, 8249, 9309, 16680, 0, 1264],

[27191, 14469, 10310, 9394, 7093, 9772, 5879, 13164, 9110, 6422, 3933, 8840, 9901, 16720, 1288, 0]]

**Using the distance matrix:**

change the value of the maximum\_distance parameter in the distance dimension to 70000. When you run the modified program, it returns the following output.

Route for vehicle 0:

0 -> 1 -> 7 -> 5 -> 4 -> 8 -> 0

Distance of route: 61001m

Route for vehicle 1:

0 -> 0

Distance of route: 0m

Route for vehicle 2:

0 -> 3 -> 2 -> 12 -> 11 -> 6 -> 0

Distance of route: 61821m

Route for vehicle 3:

0 -> 13 -> 9 -> 10 -> 14 -> 15 -> 0

Distance of route: 59460m

Total distance of all routes: 182282m

**Design thinking for mobile app development:**

Design thinking has revolutionised the way we approach problem-solving and innovation. It is a problem-solving methodology that prioritises empathy, experimentation, and iteration to create human-centered solutions. In the world of app development, design thinking is an invaluable tool for creating features that meet the needs of users in a way that is both efficient and effective.



**Stage 1: Empathise**

The first stage of any design thinking process involves designers seeking to understand and empathise with the needs of the users by conducting research, interviewing users and gathering data to gain insights into the users’ needs, challenges and behaviour.

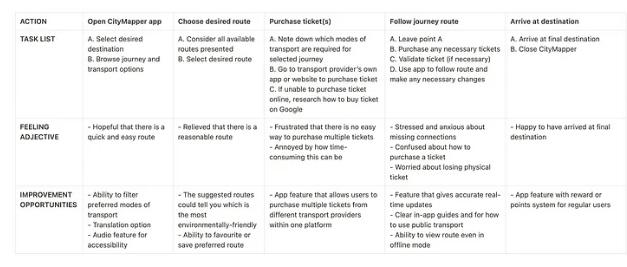
**User Story**

I first created a user story to help me stay focused on the user’s needs:

As a busy working professional, I want to be able to plan my journey and purchase all required tickets in one app, so that I can save time and not feel stressed since I will have everything I need beforehand.

**User Journey Map**

Next, I created a user journey map to get a clearer picture of all the steps a user currently takes to plan a journey from A to B.This was a useful exercise as it allowed me to get into the mind of the user: I thought about how they would be feeling at each stage of the journey and what friction points there might be. This also highlighted additional improvement opportunities.



**Stage 2: Define**

The second step in the design thinking process is the define stage.Users of journey planning apps need the ability to purchase different public transport tickets from different companies all in one place so that they can efficiently plan their journey, save time and money, and have a stress-free travel experience.

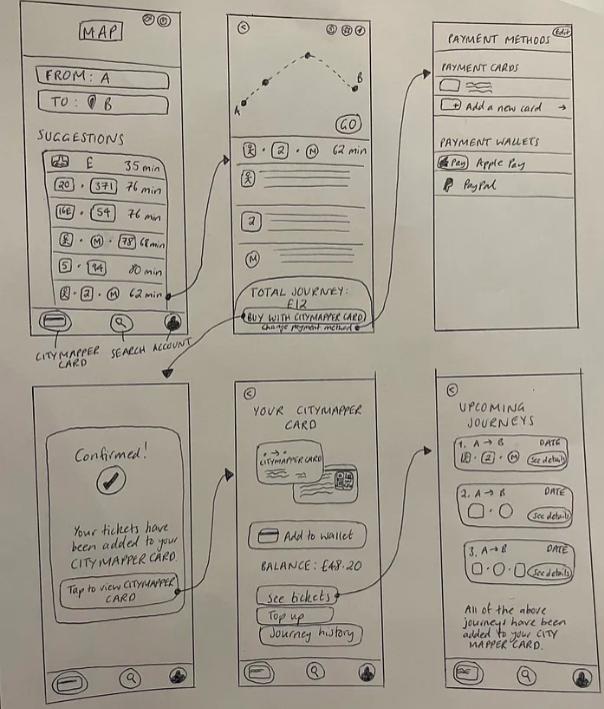
**Stage 3: Ideate**

The ideate stage involves brainstorming as many ideas as possible to solve this particular problem. Below are some of my ideas:

* Purchase all tickets in-app using their virtual travel card (or using other payment methods)
* Store all virtual tickets on the virtual travel card, needing only to scan the virtual travel card’s QR code.
* Add the virtual travel card to the user’s mobile wallet, removing the need to go into the app each time to show tickets

**Stage 4: prototype**

The penultimate stage of the design thinking process is where designers create low-fidelity prototypes to test and refine ideas, using these to gather feedback from users and stakeholders.I used six screens to show how this new virtual card feature would work on our project and what the user flow would be. You can see the below wireframes:

**Stage 5: Test**

The fifth and final stage of the design thinking process is testing. This is when designers test their prototypes with users to gather as much feedback and insights as possible to refine and improve their ideas.

**Mobile app development:**

Mobile Apps:

* Java (for Android) and Kotlin: For native Android app development.
* Swift (for iOS): For native iOS app development.
* React Native, Flutter, or Xamarin: Cross-platform frameworks if you want to target both Android and iOS with a single codebase.

Backend Development:

* Node.js: A popular choice for server-side development.
* Python (Django, Flask): For building RESTful APIs and data processing.
* Ruby on Rails: If you’re more comfortable with Ruby for backend development.
* Java (Spring Boot): If you prefer a Java-based backend.

Database Management:

* PostgreSQL, MySQL, or SQLite: For storing application data.
* Redis: Useful for caching and real-time data.

RESTful APIs:

* Use RESTful web services to facilitate communication between the mobile app and the server.

UI/UX Design:

* React Native Paper, Material-UI, or UIKit: UI component libraries for consistent and attractive design.
* Sketch, Figma, or Adobe XD: For designing app interfaces.



**Conclusion:**

The limitations around processing power, connectivity costs, battery needs, and ongoing maintenance should be factored in. Extra care needs to be taken with security and data privacy as well.The ESP32 provides a flexible and scalable IoT platform to build this tracking and optimization system at relatively low cost. But the tradeoffs need to be evaluated to determine if the capabilities match the needs and constraints of the specific public transit agency and use case. With good planning, limitations can be mitigated to take advantage of the benefits this platform provides.