
SHELL

جامعة شعيب الدكالي
Université Chouaib Doukkali

Royaume du Maroc
Ministère de l'Education Nationale, de la Formation Professionnelle
de l'Enseignement Supérieur et de la Recherche Scientifique

Université Chouaib Doukkali,
Ecole Nationale des Sciences Appliquées d'El Jadida (ENSA)
2021-2022

ENSA
EL JADIDA

★ Projet python : ★

Classe : **Ingenierie informatique et technologie émergente 1.**

Module : **Algorithmique avancée et Python.**

Groupe : **1**

Réalisé par :

Zakaria el harchi **-Ilyas Bennane**

Ayman Cherkani **-Safiya Daoudi** **-Hajar Machmoum**

Shell

2022-2023

SOMMAIRE :

- **Code sous python de la partie du plus court chemin**
- **Explication de Travelling Salesman Problem**
- **Code sous python de traçage du chemin dans Map**

VII. Conclusion :

INTRODUCTION :

En raison des nombreuses étapes et des parties impliquées , la distribution de gaz aux diverses stations de service peut être une procédure difficile et compliquée . Le transport du gazoil peut être l'un des facteurs qui rend difficile cette distribution.En effet, le gazoil doit être transporté de raffineries ou de terminaux de stockage jusqu'aux stations de gaz, ce qui peut être coûteux en raison des frais de transport et de logistique.En outre, Il y a plusieurs moyens de transporter le gazole, tels que par camion, par train ou par pipeline. Mais, ce qui nous intéresse dans notre mini-projet, c'est la distribution du gazoil par camion en utilisant une bonne stratégie en minimisant le cout du transport,c-à-d on va travailler à écrire un code python qui va permettre le passage d'un camion d'une station de départ choisie vers tous les autres stations (shell) en utilisant le plus court chemin

Pseudocode :

Algorithme :Plus_court_chemin

Variables :matrix,S

Début :

Ecrire(« Bienvenue,veuillez choisir la station de départ »)

Lire(S)

Si S est dans (liste des stations) :

Solve_salesman_problem_function(matrix,S)

Retourner liste

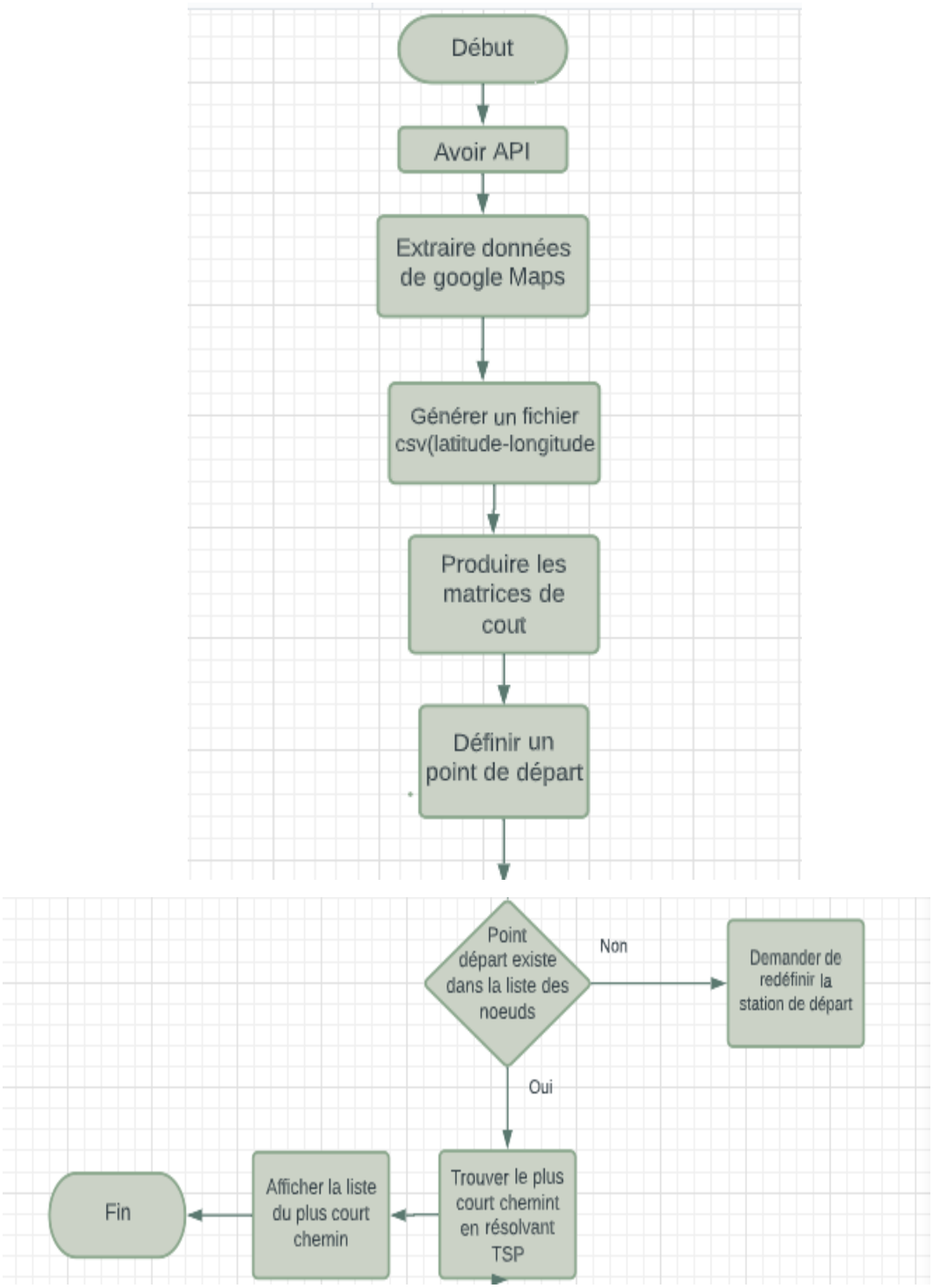
Afficher map qui contient le plus court chemin

Sinon

Ecrire(« la station que vous avez entré est invalide,veuiller saisir le point de départ une autre fois »)

Fin

Flowchart :


Début
Avoir API
Extraire données de google Maps
Générer un fichier csv(latitude-longitude
Produire les matrices de cout
Définir un point de départ
Point départ existe dans la liste des noeuds
Non
Demander de redéfinir la station de départ
Oui
Trouver le plus court chemint en résolvant TSP
Afficher la liste du plus court chemin
Fin

Explications des principaux étapes de notre projet :

➢ Extraction des données :

L'extraction de données de **Google Maps** peut être très utile dans de nombreux domaines, tels que la cartographie, l'analyse de données, la géolocalisation et la planification de trajets. Elle peut permettre de récupérer des informations précieuses sur les lieux, les itinéraires et les distances, ainsi que sur les caractéristiques des régions et des villes.

Il existe plusieurs façons d'extraire des données de Google Maps. L'une des façons les plus simples est d'utiliser l'API de Google Maps, qui offre un accès programmatique aux données de Google Maps. Cela permet de récupérer des informations telles que les coordonnées géographiques, les itinéraires et les distances, ainsi que des données sur les points d'intérêt et les événements.

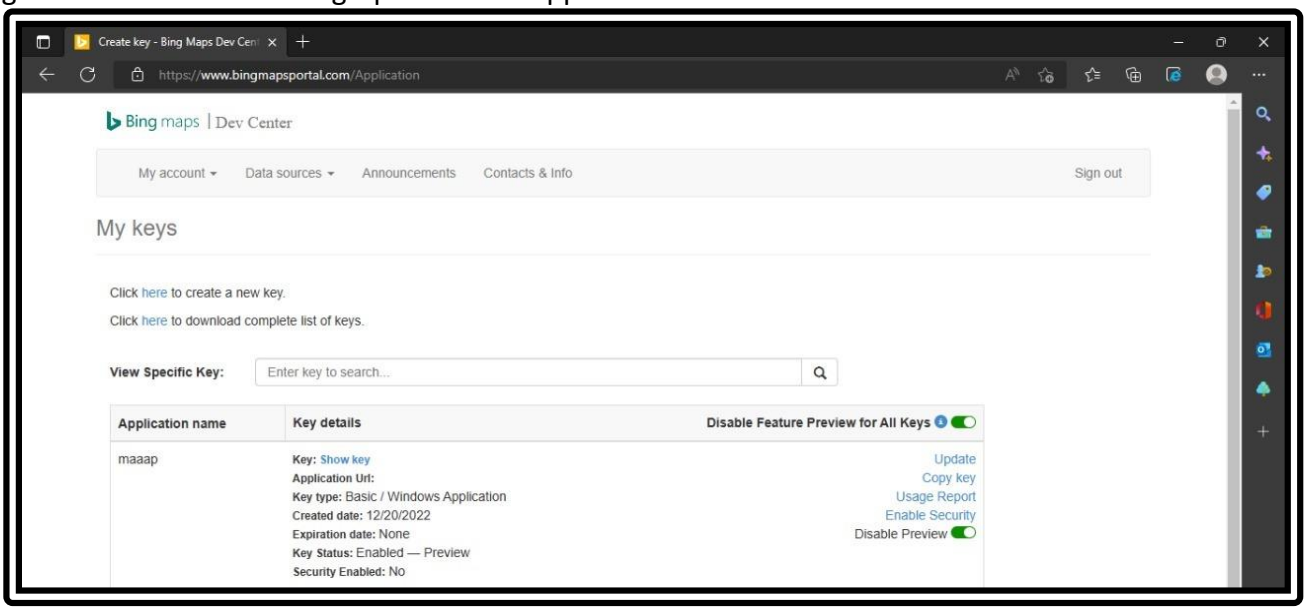
Il est également possible d'extraire des données de Google Maps en utilisant des outils de scraping de données Web. Ces outils permettent de récupérer automatiquement des données à partir de sites Web en utilisant des scripts ou des programmes. Cependant, il est important de se rappeler que l'utilisation de ces outils peut être soumise à des restrictions et à des conditions d'utilisation strictes, et il est important de respecter les termes d'utilisation de Google Maps.

Puisque l'API est un outil payant ,on a trouvé après plusieurs recherches qu'on peut utiliser un moyen plus adaptable et gratuit ,c'est le **Bing Maps**

Bing Maps est un service de cartographie en ligne développé par Microsoft. Il permet aux utilisateurs de visualiser, d'explorer et de planifier des itinéraires sur une carte en ligne, ainsi que de rechercher des points d'intérêt et des adresses. Bing Maps propose également des

images aériennes et des vues de rue à 360 degrés, ainsi que des données sur la circulation et les trajets en transport en commun.

Bing Maps est disponible gratuitement et peut être utilisé sur différents appareils, tels que les ordinateurs, les téléphones mobiles et les tablettes. Il est également possible de l'intégrer à des applications et à des sites Web en utilisant l'API (Application Programming Interface) de Bing Maps. L'API de Bing Maps permet aux développeurs de créer des applications qui utilisent les données et les fonctionnalités de Bing Maps.En général, c'est un service de cartographie en ligne complet et facile à utiliser qui peut être utile pour visualiser, explorer et planifier des itinéraires, ainsi que pour rechercher des points d'intérêt et des adresses. Il peut être particulièrement utile pour les entreprises et les organisations qui souhaitent intégrer les données de cartographie à leurs applications et sites Web.



Ce service Bing maps permet de générer un Bing key (Bing API) ,ce qui va nous aider à extraire longitude et la latitude de chaque station **shell**

➢ Génération d'un fichier csv :

Après l'étape précedente,c-à-d après la génération d'un Bing API .on utilise le web scraping ,ce qui consiste à récupérer automatiquement des données sur le web à partir de sites web.Dans notre cas, on va utiliser cet outil informatique pour produire un fichier csv contenant tous les stations désirées avec la latitude ainse que la longitude de chaque point (chaque station shell) , on a utilisé un site nommé Selenium qui a déjà une bibliothèque convenable dans python

On convertit fichier csv généré en format Excel puisque l'API peut manipuler et exploiter juste les fichiers Excel

Et dans ce fichier,on trouve toutes les informations concernant chaque station du MAP (nom de station,adresse,catégorie,longitude,latitude…)

Name	Fulladdress	Street	Municipality	Categories	Phone	Phones	Review Count	Average Rating	Review URL	Google Maps URL	Latitude	Longitude
Shell	Rte d'El Jadida, Casablanca 20000	Rte d'El Jadida	Casablanca 20000	Gas station,ATM,Car wash,Convenience store	05229-20020	05229-20020, +212 5229-200						
Station Shell Hassania, Hay Hassani.	Croisement Boulevard Sidi Abderrahmane et, Bd Ibnou Sina, Casablanca 20204	Croisement Boulevard Sidi Abderrahmane et	Bd									
Shell	Avicenne / Alexandre 1Er, Boulevard Abdelkrim Al Khattabi, Casablanca 20250	Avicenne / Alexandre 1Er	Boulevard Abdelkrim Al Khattabi	Gas station,ATM,C								
Station Shell	Route de Rabat Km 9 400 Route De Mohamadia Ain Sebaa, N1, Casablanca 20250	Route de Rabat Km 9 400 Route De Mohamadia Ain Sebaa	N1	Gas stat								
Shell Station Bd Alexandrie	Boulevard Alexandrie Sis 2‹ Angle Av. 2 Mars, Casablanca 20250	Boulevard Alexandrie Sis 2	Angle Av. 2 Mars	Gas station,Car wash,Conv								
Shell	Rue Lice d Anfa, Bd Kennedy, Casablanca 20250	Rue Lice d Anfa	Bd Kennedy	Gas station,ATM,Car wash,Convenience store	05229-20020	05229-20020, +212						
Shell Station Service Flahi	HCFW+G8X, Casablanca 20250	HCFW+G8X	Casablanca 20250	Gas station			169	3.9	https://search.google.com/local/reviews?placeid			
Shell	Rue du Sergent Driss Chbakou Ibn Tachfine‹ Boulevard Emile Zola, Casablanca 20000	Rue du Sergent Driss Chbakou Ibn Tachfine	Boulevard Emile Zola	Gas sta								
Shell	Av. 2 Mars, Casablanca 20250	Av. 2 Mars	Casablanca 20250	Gas station,Car wash	05229-20020	05229-20020, +212 5229-20020	175	3.8	https://search.googl			
Shell	Boulevard Mohammed VI‹ Route Mediouna, Casablanca 20000	Boulevard Mohammed VI	Route Mediouna	Gas station,Car wash	05229-20020	05229-20020						
station shell Grande Route	P3006, Casablanca 20250	P3006	Casablanca 20250	Gas station	0676-455438	0676-455438, +212 676-455438	167	4	https://search.goo			
Shell	19 Av. Pasteur, Casablanca 20250	19 Av. Pasteur	Casablanca 20250	Gas station,Car wash,Convenience store	05229-20020	05229-20020, +212 5229-20020	6					
Shell station	Bd Oqba Bnou Nafii, Casablanca 20250	Bd Oqba Bnou Nafii	Casablanca 20250	Gas station	05229-20020	05229-20020, +212 5229-20020	12	3.8	https			
Shell	Ait Ishaq‹ Via Pian della Genna‹ Boulevard Aïn Taoujtate, Casablanca 20250	Ait Ishaq	Via Pian della Genna	Gas station	05229-20020	05229-20020, +212 522						
Shell	Driss 1, Bd Abdelmoumen, Casablanca 20250	Driss 1	Bd Abdelmoumen	Gas station,Car wash,Restaurant	05229-20020	05229-20020, +212 5229-20020	9	3.6				
Station Shell	Bd Mohamed Zerktouni, Casablanca 20250	Bd Mohamed Zerktouni	Casablanca 20250	Gas station,Car wash	05229-20020	05229-20020, +212 5229-20						
Shell	Rte de Taddart, Casablanca 20000	Rte de Taddart	Casablanca 20000	Gas station,Car wash	05229-20020	05229-20020, +212 5229-20020	92	3.9	https://searc			
Shell	296 Bd El Fida, Casablanca 20250	296 Bd El Fida	Casablanca 20250	Gas station,Car wash	05229-20020	05229-20020, +212 5229-20020	25	4	https://search.go			
Shell Oasis	H984+J4Q, Rte d'El Jadida, Casablanca 20250	H984+J4Q	Rte d'El Jadida	Gas station	05222-50612	05222-50612, +212 5222-50612	157	3.8	https://searc			
Shell	Bd Moulay Youssef, Casablanca 20100	Bd Moulay Youssef	Casablanca 20100	Gas station,Car wash	05229-20020	05229-20020, +212 5229-20020	35	3.8	http			

➢ Génération des matrices :

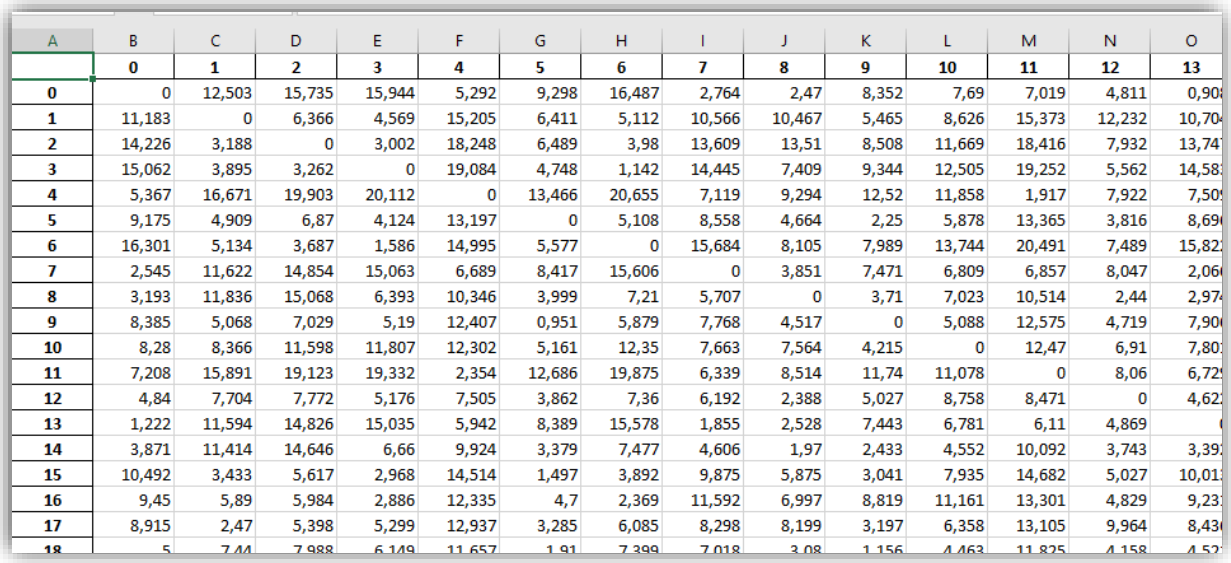
Après avoir le fichier csv, on va exploiter ce dernier dans l'API en utilisant une bibliothèque de lecture de CSV, comme csv pour Python pour générer deux matrices :une matrice qui définit le temps,et l'autre définit la distance entre tous les stations shell .Les deux matrices sont en format Excel comme le montre les deux figures suivantes

Matrice de temps :

A	B	C	D	E	F	G	H	I	J	K	L	M	N
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
0	0	16,0167	22,0167	23,95	11,3	15,8167	25,85	8,0667	6,7333	13,2333	14,9167	10,6833	13,3
1	12,8167	0	12,4667	13,6333	16,1333	10,2667	15,55	12,7833	13,1667	7,6833	15,2333	14,7667	17,25
2	20,0333	9,1833	0	7,7	23,35	17,2333	10,9	20	20,3833	14,9	22,45	21,9833	21,7833
3	22,8667	11,6	7,4667	0	26,1833	14,5167	3,8833	22,8333	22,05	17,7333	25,2833	24,8167	16,35
4	12,75	20,6333	26,6333	28,55	0	20,4333	30,4667	13,7833	17,2667	17,85	19,5333	6,8333	17,4
5	15,2167	12	16,8333	13,0333	18,55	0	16,9833	15,1833	13,1167	5,9167	17,2167	17,1667	10,1667
6	26,75	15,4833	8,4167	6,0167	30,4667	18,4	0	26,7167	24,05	21,7833	29,1667	28,7	16,8333
7	7,8	14,2333	20,2333	22,1667	12,7333	14,0333	24,0667	0	10,2167	11,45	13,1333	11,35	16,1667
8	9,4667	15,95	21,95	20,85	15,6167	12	23,2333	12,2667	0	11,3667	14,85	14,25	8,0333
9	14,5	12,2833	16,7	13,9833	17,8167	3,3667	17,0667	14,4667	13,4	0	16,5	16,45	12,8167
10	15,6333	13,0833	19,0833	21,0167	18,95	12,9	22,9333	15,6	15,9833	10,3167	0	17,5833	18,9333
11	13,3167	19,1	25,0833	27,0167	5,65	18,9	28,9333	12,25	15,7333	16,3167	18	0	17,1833
12	11,75	20,5667	22,2333	16,7667	13,6167	11,3667	16,9667	15,9667	7,15	15,2667	19,1333	17,0833	0
13	4,45	13,4	19,4	21,3333	9,45	13,2	23,2333	5,45	6,8833	10,6167	12,3	8,0667	13,45
14	9,65	15,5833	21,5833	20,8667	15,25	10,3	23,25	11,75	5,9333	7,7	12,9333	13,8833	11,3667
15	15,9333	9,8333	14,1	9,7833	19,25	4,75	13,0667	15,9	16,45	7,5667	18,35	17,8833	13,5
16	21,15	17,75	15,2333	9,05	22,5333	15,4833	7,7167	24,3	15,5833	20,4333	26,2333	26	8,9167
17	11,6	6,9833	13,5667	14,5167	14,9167	8,1	17,2	11,5667	11,95	6,4667	14,0167	13,55	16,0167
18	12,4333	14,75	20,1333	17,4167	17,4833	7,1	22,4333	14,1167	9	4,7	13,8333	16,1	12,2
19	17,5667	6,3	9,4667	9,2667	20,8833	12,0167	11,1833	17,5333	17,9167	12,4333	19,9833	19,5167	18,5
20	21,3667	16,0167	15,6	10,1333	22,7667	14,2333	8,9167	24,5333	15,8	19,0167	26,45	26,2167	9,1333
21	4,7667	18,7333	24,7333	23,05	8,8167	14,4	23,25	10,0167	5,7	15,95	17,6333	13,2667	8,4833

Cette matrice identifie la durée entre chaque station shell de Map avec tous les autres stations restantes

Matrice de distance :



A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
0	0	12,503	15,735	15,944	5,292	9,298	16,487	2,764	2,47	8,352	7,69	7,019	4,811	0,90
1	11,183	0	6,366	4,569	15,205	6,411	5,112	10,566	10,467	5,465	8,626	15,373	12,232	10,70
2	14,226	3,188	0	3,002	18,248	6,489	3,98	13,609	13,51	8,508	11,669	18,416	7,932	13,74
3	15,062	3,895	3,262	0	19,084	4,748	1,142	14,445	7,409	9,344	12,505	19,252	5,562	14,58
4	5,367	16,671	19,903	20,112	0	13,466	20,655	7,119	9,294	12,52	11,858	1,917	7,922	7,50
5	9,175	4,909	6,87	4,124	13,197	0	5,108	8,558	4,664	2,25	5,878	13,365	3,816	8,69
6	16,301	5,134	3,687	1,586	14,995	5,577	0	15,684	8,105	7,989	13,744	20,491	7,489	15,82
7	2,545	11,622	14,854	15,063	6,689	8,417	15,606	0	3,851	7,471	6,809	6,857	8,047	2,06
8	3,193	11,836	15,068	6,393	10,346	3,999	7,21	5,707	0	3,71	7,023	10,514	2,44	2,97
9	8,385	5,068	7,029	5,19	12,407	0,951	5,879	7,768	4,517	0	5,088	12,575	4,719	7,90
10	8,28	8,366	11,598	11,807	12,302	5,161	12,35	7,663	7,564	4,215	0	12,47	6,91	7,80
11	7,208	15,891	19,123	19,332	2,354	12,686	19,875	6,339	8,514	11,74	11,078	0	8,06	6,72
12	4,84	7,704	7,772	5,176	7,505	3,862	7,36	6,192	2,388	5,027	8,758	8,471	0	4,62
13	1,222	11,594	14,826	15,035	5,942	8,389	15,578	1,855	2,528	7,443	6,781	6,11	4,869	
14	3,871	11,414	14,646	6,66	9,924	3,379	7,477	4,606	1,97	2,433	4,552	10,092	3,743	3,39
15	10,492	3,433	5,617	2,968	14,514	1,497	3,892	9,875	5,875	3,041	7,935	14,682	5,027	10,01
16	9,45	5,89	5,984	2,886	12,335	4,7	2,369	11,592	6,997	8,819	11,161	13,301	4,829	9,23
17	8,915	2,47	5,398	5,299	12,937	3,285	6,085	8,298	8,199	3,197	6,358	13,105	9,964	8,43
18	5	7,44	7,988	6,149	11,657	1,91	7,399	7,018	3,08	1,156	4,463	11,825	4,158	4,52

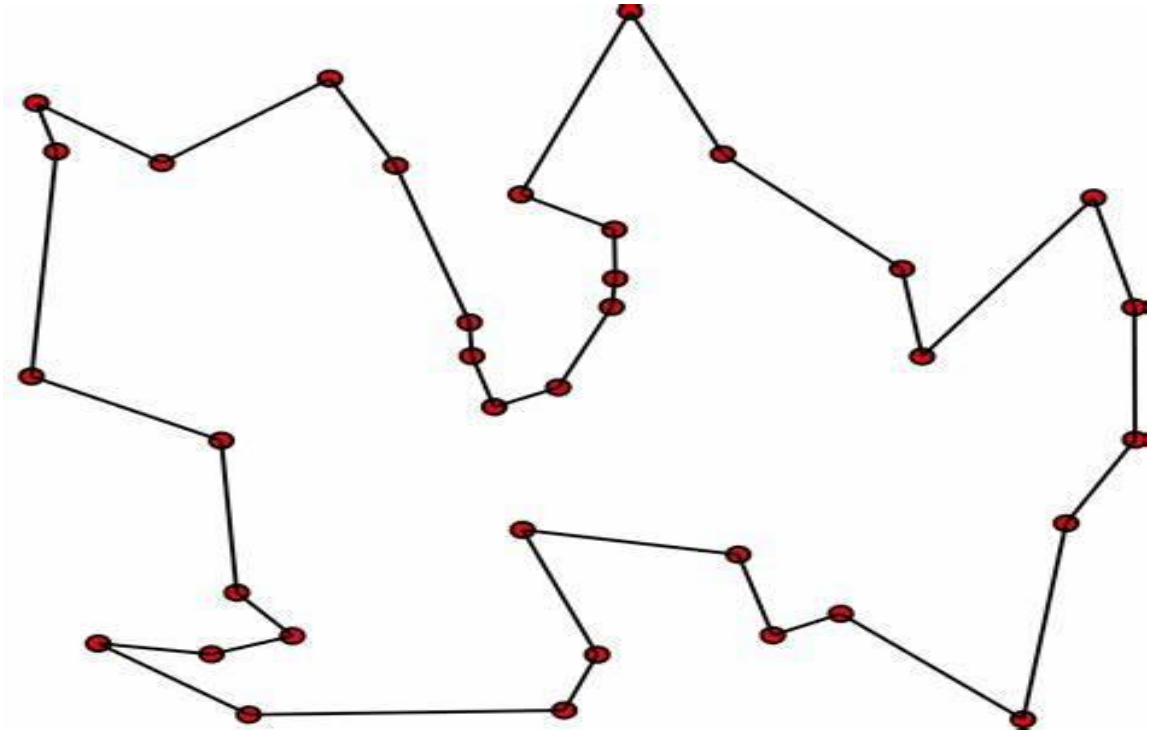
Cette matrice identifie la distance entre chaque station de map avec tous les autres stations shell restantes .

➢ Trouver le chemin optimal :

Pour trouver le plus court chemin qui va nous permettre de passer par tous les stations shell de Casablanca ,plusieurs algorithmes ont été proposé .En revanche ,on a remarqué que notre problème correspond au TSP (Traveling Salesman Problem) c'est un problème d'optimisation classique en informatique qui consiste à trouver l'itinéraire le plus court possible qui visite un ensemble donné d'emplacements, à revenir au point de départ et à visiter chaque emplacement exactement une fois.

Pour la résolution de ce problème ,on peut utiliser la biliothèque tsp existante dans python ou bien pytsp…,où on peut utiliser comme entrées les matrices générées précédement afin de trouver le plus court chemin en visitant tous les stations

Voici un exemple de fonctionement :



Explication de GUI :

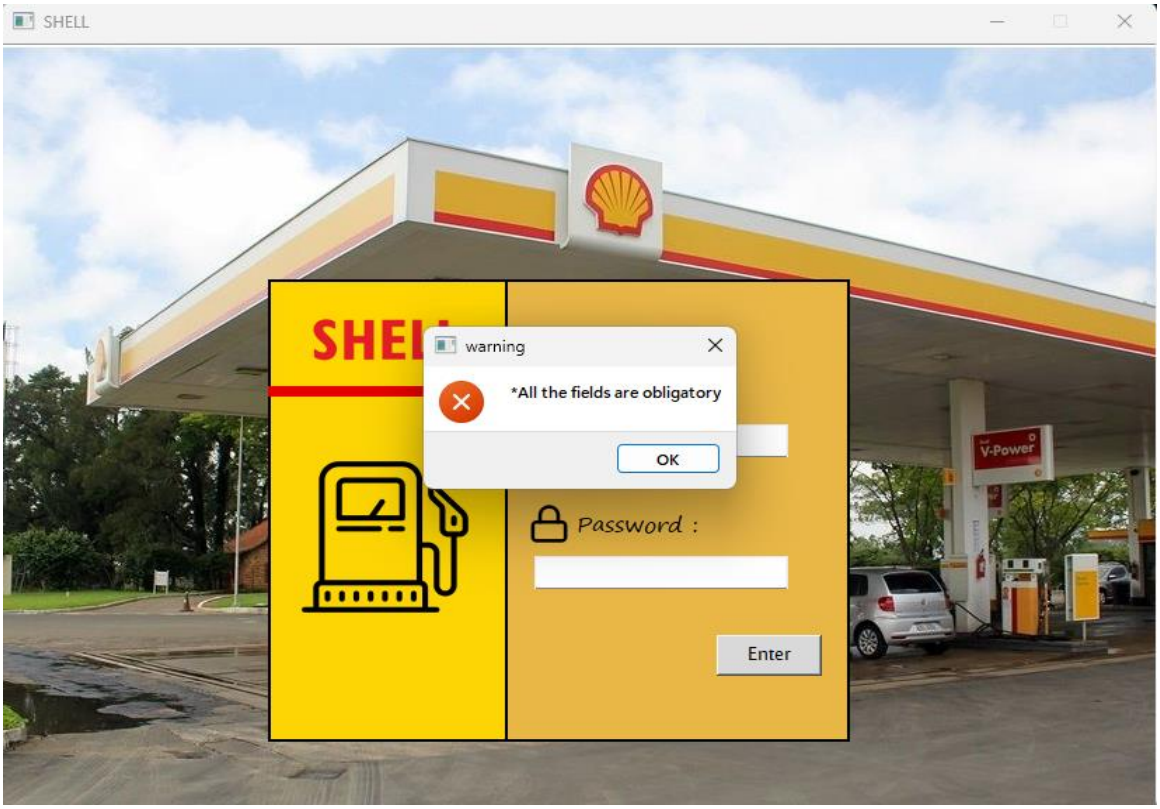
1ère fenêtre :

Cette fenêtre permet à l'utilisateur (Chauffeur du camion) de se connecter à l'interface en entrant le login et le mot de passe

Cette fenêtre possède un boutton « Enter » pour permettre à l'utilisateur d'entrer et poursuivre les prochaines étapes

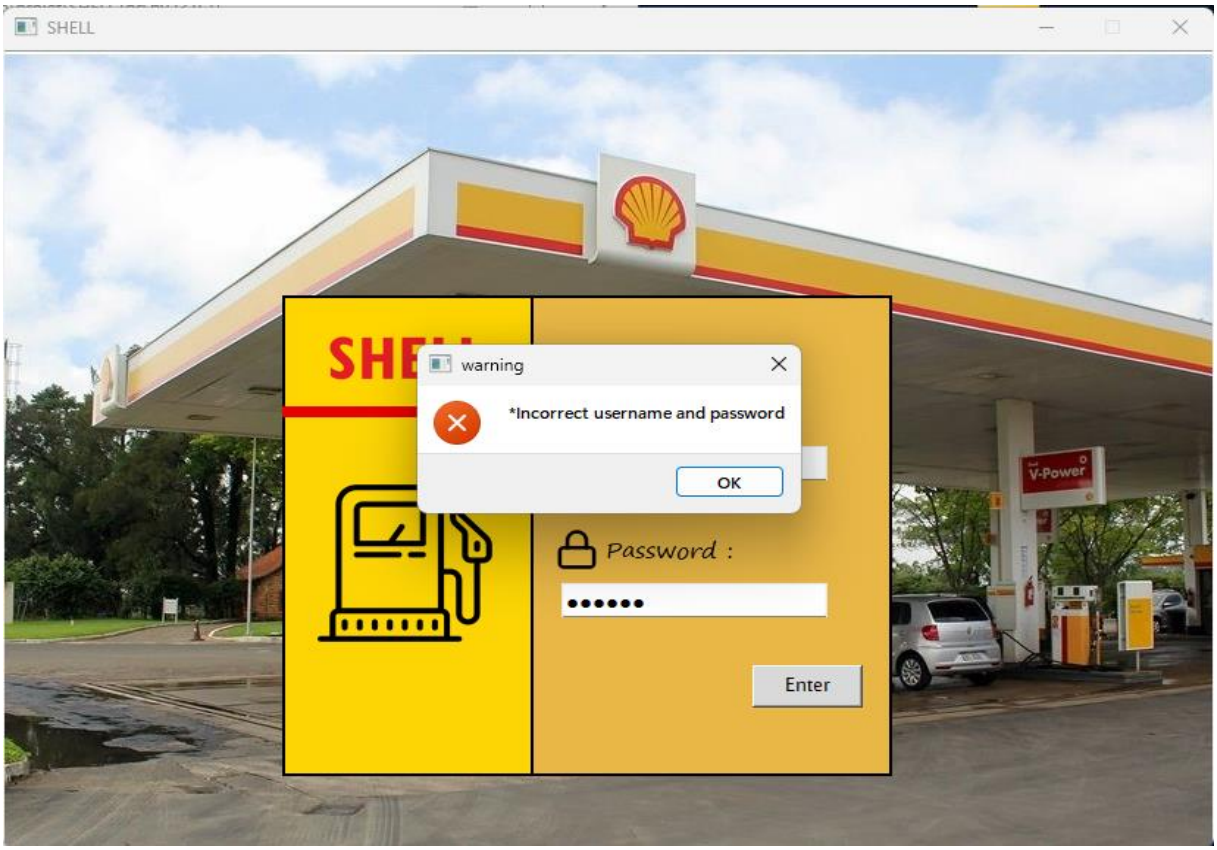
Affichage d'un message d'erreur :

1er cas :

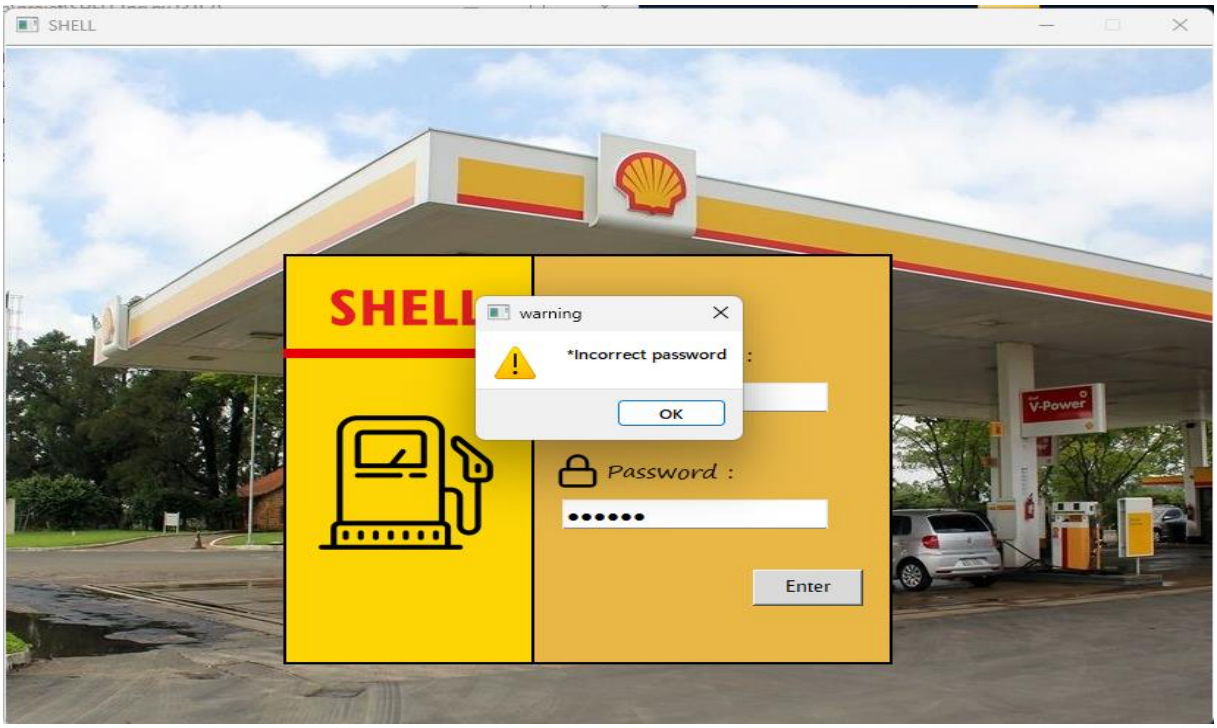


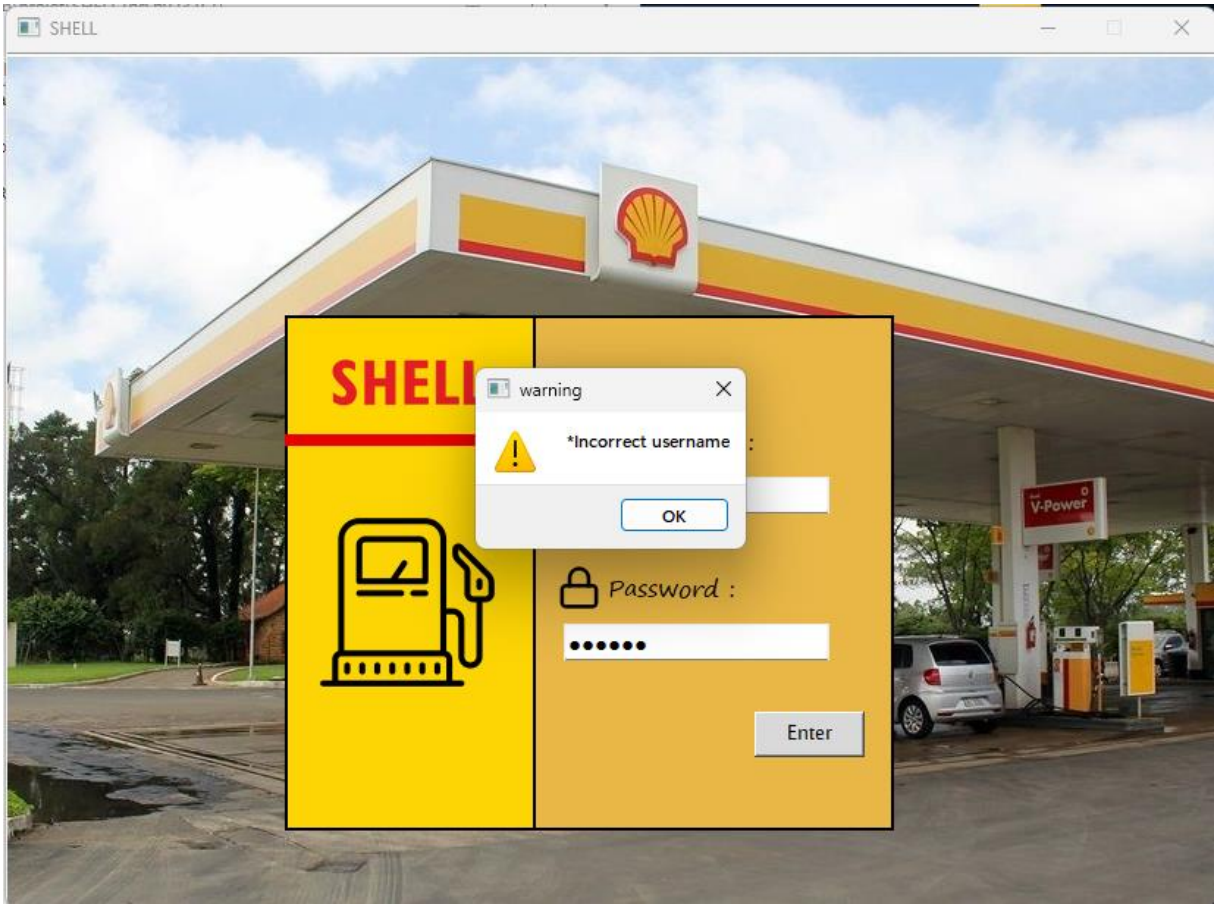
Si on remplit pas les valeurs demandés ,un message d'erreur s'affichera comme quoi on est obligé de remplir tous les cases.

2ème cas :



Si on entre pas les valeurs correctes ,un message d''erreur s'affichera ,par exemple dans le cas où on entre un login et un mot de passe invalide ou bien l'un des deux est incorrecte comme le montre les captures ci-dessous.



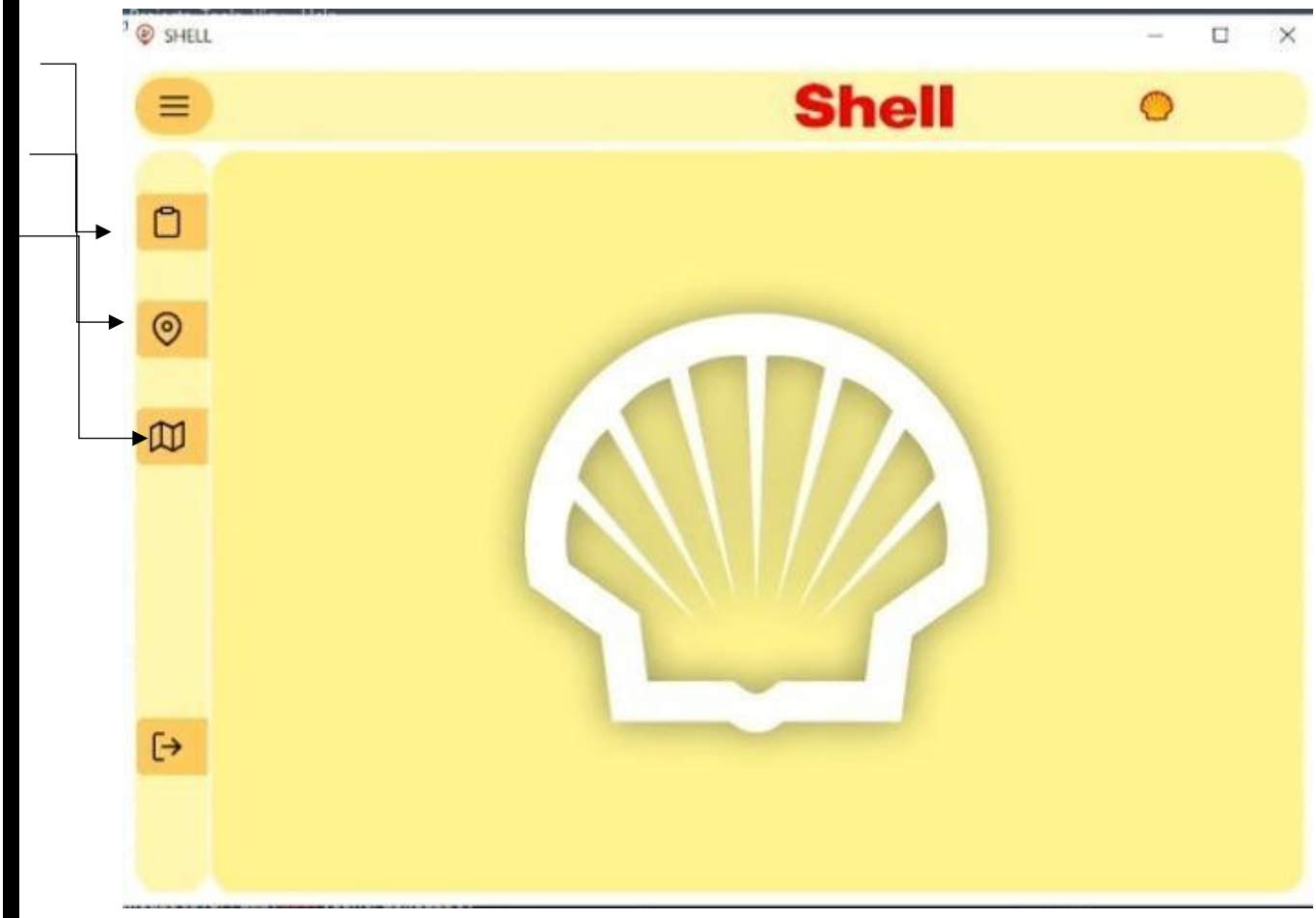

SHELL
warning
*Incorrect username
OK
Password :
Enter
V-Power

3ème cas :


SHELL
success
Login successful
OK
Cancel
Password :
Enter
V-Power

Ce dernier cas est le cas de 'Tout va bien',c-à-d :Lorsqu'on entre un nom d'utilisateur et un mot de passe correcte.Alors ,là le message « Login Successful » s'affichera ,donc l'utilisateur peut passer à la deuxième fenetre .

2ème fenêtre :



Lorsqu'on clique sur les symboles indiquées sur la figure ,une explication des symboles s'affichera sur l'interface

Par exemple si on clique sur le symbole « Menu » :

Ça donne :

C-àd,l'utilisateur peut effectuer que les options suivantes :

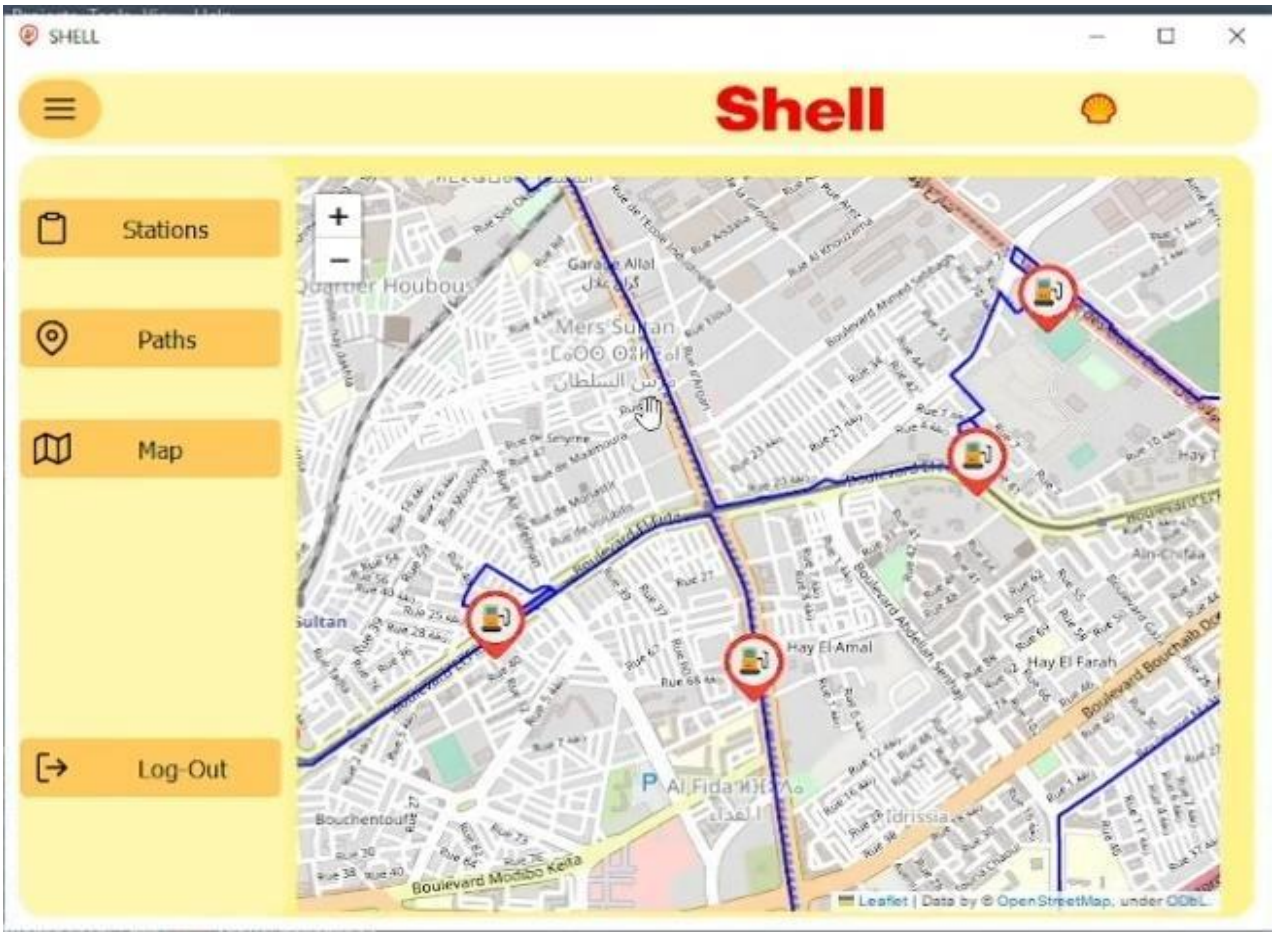
- Consulter les stations shell de casablanca

- Cliquer sur « Paths » pour voir le plus court chemin liant tous les stations à partir d'un point de départ bien précis

- Si on clique sur le boutton « Map », On remarque alors l'affichage d'une zone géographique 'MAP' qui contient tous les stations shell qui se trouvent dans Casablanca ,comme le montre la figure ci-dessous.

On remarque que l'utilisateur peut controler l'image comme il veut c-à-d ,il peut effectuer des zooms ……

Maintenant,si on veut voir le plus court chemin permettant de visiter tous les stations **shell** en minimisant la distance ,on clique sur le boutton « Paths »



➡ On remarque ainsi qu'on peut se déconnecter en tappant sur le boutton « Log-Out »

Codes sous python de GUI :

Pour réaliser cette interface ,on a pensé a utiliser Qt-designer

Pour la 1ère interface :

Parmi les principaux bibliothèques utilisées,on trouve :

```
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtWidgets import QDialog,QApplication
from PyQt5.QtWidgets import QMessageBox
```

QtCore, QtGui et QtWidgets sont des modules (c'est-à-dire des bibliothèques) du framework Qt qui fournissent respectivement des fonctionnalités de base non graphiques et graphiques.

QtCore fournit un ensemble de classes non GUI, y compris les types de données et la gestion des fichiers.

QtGui fournit des classes pour l'intégration du système de fenêtrage, les graphiques 2D et l'imagerie.

QtWidgets est un module qui fournit un ensemble de widgets GUI (c'est-à-dire des contrôles), tels que des boutons, des cases à cocher et des curseurs.

Des parties du code

```
class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(824, 583)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.horizontalLayout = QtWidgets.QHBoxLayout(self.centralwidget)
        self.horizontalLayout.setContentsMargins(0, 0, 0, 0)
        self.horizontalLayout.setSpacing(0)
        self.horizontalLayout.setObjectName("horizontalLayout")
        self.background = QtWidgets.QFrame(self.centralwidget)
        self.background.setStyleSheet("#background{\n"
"    \n"
"    background-image: url(:/photos/SHELL.jpg);\n"
"}")
        self.background.setFrameShape(QtWidgets.QFrame.StyledPanel)
        self.background.setFrameShadow(QtWidgets.QFrame.Raised)
        self.background.setObjectName("background")
        self.frame1 = QtWidgets.QFrame(self.background)
        self.frame1.setGeometry(QtCore.QRect(190, 180, 415, 351))
        self.frame1.setStyleSheet("#frame1{\n"
"background-color:#E8b746;\n"
"}")
        self.frame1.setFrameShape(QtWidgets.QFrame.WinPanel)
        self.frame1.setFrameShadow(QtWidgets.QFrame.Plain)
        self.frame1.setLineWidth(10)
        self.frame1.setMidLineWidth(10)
        self.frame1.setObjectName("frame1")
        self.frame2 = QtWidgets.QFrame(self.frame1)
        self.frame2.setGeometry(QtCore.QRect(0, 0, 171, 351))
        self.frame2.setStyleSheet("#frame2{\n"
"background-color:#fdd504;\n"
"}")
```

- ✓ Ce code définit une classe appelée Ui_MainWindow, qui est utilisée pour configurer l'interface utilisateur d'une fenêtre principale dans une application Qt. La setupUiméthode est utilisée pour configurer la fenêtre principale et ses widgets
- ✓ La fenêtre principale est créée avec une taille par défaut de 824x583 pixels et reçoit le nom d'objet "MainWindow". Un widget central est créé et ajouté à la fenêtre principale. Ce widget contiendra les autres widgets qui composent l'interface utilisateur.
- ✓ Un cadre d'arrière-plan est créé et ajouté à la mise en page. Ce cadre reçoit une feuille de style qui définit son image d'arrière-plan sur un fichier spécifique. Le cadre est également doté d'une forme de cadre de panneau stylé et d'une ombre en relief.

```
self.enter.setObjectName("enter")
self.username = QtWidgets.QLineEdit(self.frame1)
self.username.setGeometry(QtCore.QRect(190, 110, 181, 25))
self.username.setStyleSheet("font: 11pt \"Sitka Display\";")
self.username.setObjectName("username")
self.password = QtWidgets.QLineEdit(self.frame1)
self.password.setGeometry(QtCore.QRect(190, 210, 181, 25))
self.password.setStyleSheet("font: 11pt \"Sitka Display\";")
self.password.setObjectName("password")
self.password.setEchoMode(QtWidgets.QLineEdit.Password)
self.horizontalLayout.addWidget(self.background)
MainWindow.setCentralWidget(self.centralwidget)

self.retranslateUi(MainWindow)
QtCore.QMetaObject.connectSlotsByName(MainWindow)

self.enter.clicked.connect(self.show_popup)
```

- ✓ Le code crée un bouton "entrer" et deux widgets d'édition de ligne, un pour le nom d'utilisateur et un pour le mot de passe. Les widgets d'édition de ligne reçoivent une taille et une position fixes dans le widget "frame1" et un style de police via une feuille de style.
- ✓ L'édition de la ligne de mot de passe a son mode d'écho défini sur "Mot de passe", ce qui signifie que les caractères saisis par l'utilisateur seront remplacés par des astérisques pour masquer le mot de passe.
- ✓ Enfin, une connexion est établie entre le signal cliqué du bouton "entrer" et une méthode appelée "show_popup", de sorte que la méthode "show_popup" sera appelée chaque fois que le bouton est cliqué

Pour la deuxième interface :

Les principaux bibliothèques utilisées :

```
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtWidgets import QDialog,QApplication
from PyQt5.QtWidgets import QMessageBox
from interface2 import MainWindow_Ui
from PyQt5.QtGui import QPixmap, QIcon
```

- QPixmap est une classe utilisée pour afficher des images dans des widgets. Il est optimisé pour le dessin et peut être utilisé pour afficher des images à l'écran.
- QIcon est une classe utilisée pour représenter une icône dans une interface graphique. Il peut être utilisé pour créer une petite représentation graphique d'un objet ou d'un concept, tel qu'un fichier, un dossier ou une application.

Des parties du code :

```
self.frame2.setFrameShape(QtWidgets.QFrame.WinPanel)
self.frame2.setFrameShadow(QtWidgets.QFrame.Plain)
self.frame2.setLineWidth(10)
self.frame2.setMidLineWidth(10)
self.frame2.setObjectName("frame2")
self.label = QtWidgets.QLabel(self.frame2)
self.label.setGeometry(QtCore.QRect(10, 10, 151, 71))
self.label.setStyleSheet("background-image: url(:/newPrefix/shell-vertical.png);")
self.label.setText("")
self.label.setPixmap(QtGui.QPixmap("shell-vertical.png"))
self.label.setScaledContents(True)
self.label.setObjectName("label")
self.label_2 = QtWidgets.QLabel(self.frame2)
self.label_2.setGeometry(QtCore.QRect(20, 40, 131, 311))
self.label_2.setStyleSheet("image: url(:/logos/pngwing.com.png);")
self.label_2.setText("")
self.label_2.setObjectName("label_2")
self.line = QtWidgets.QFrame(self.frame2)
self.line.setGeometry(QtCore.QRect(0, 70, 171, 31))
self.line.setStyleSheet("color: rgb(230, 0, 0);")
self.line.setFrameShadow(QtWidgets.QFrame.Plain)
self.line.setLineWidth(8)
self.line.setFrameShape(QtWidgets.QFrame.HLine)
self.line.setObjectName("line")
self.user = QtWidgets.QLabel(self.frame1)
self.user.setGeometry(QtCore.QRect(220, 70, 121, 31))
self.user.setStyleSheet("font: 12pt \"Segoe Print\";\n"
```

- ✓ Le premier cadre, "frame2", reçoit une forme de cadre "WinPanel" et une ombre unie. On lui donne également une largeur de ligne et une largeur de ligne médiane de 10 pixels.

```
def retranslateUi(self, MainWindow):
    _translate = QtCore.QCoreApplication.translate
    MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow"))
    self.pushButton.setText(_translate("MainWindow", "MENU"))
    self.pushButton_2.setText(_translate("MainWindow", "Stations"))
    self.pushButton_3.setText(_translate("MainWindow", "Paths"))
    self.pushButton_4.setText(_translate("MainWindow", "Map"))
    self.pushButton_5.setText(_translate("MainWindow", "Log-Out"))
```

- ✓ Cette partie consiste à définir le texte affiché sur chaque bouton. Le texte de chaque bouton est spécifié en tant qu'argument de chaîne à la méthode setText et est passé par la QCoreApplication.translateméthode avant d'être défini.

```
self.centralwidget.setObjectName("centralwidget")
self.verticalLayout = QtWidgets.QVBoxLayout(self.centralwidget)
self.verticalLayout.setObjectName("verticalLayout")
self.header = QtWidgets.QFrame(self.centralwidget)
self.header.setMinimumSize(QtCore.QSize(0, 50))
self.header.setMaximumSize(QtCore.QSize(16777215, 50))
self.header.setStyleSheet("")
self.header.setFrameShape(QtWidgets.QFrame.StyledPanel)
self.header.setFrameShadow(QtWidgets.QFrame.Raised)
self.header.setObjectName("header")
self.horizontalLayout_2 = QtWidgets.QHBoxLayout(self.header)
self.horizontalLayout_2.setContentsMargins(0, 0, 0, 0)
self.horizontalLayout_2.setSpacing(0)
self.horizontalLayout_2.setObjectName("horizontalLayout_2")
self.frame = QtWidgets.QFrame(self.header)
self.frame.setFrameShape(QtWidgets.QFrame.StyledPanel)
self.frame.setFrameShadow(QtWidgets.QFrame.Raised)
self.frame.setObjectName("frame")
self.horizontalLayout_4 = QtWidgets.QHBoxLayout(self.frame)
self.horizontalLayout_4.setContentsMargins(0, 0, 0, 0)
self.horizontalLayout_4.setSpacing(0)
self.horizontalLayout_4.setObjectName("horizontalLayout_4")
self.pushButton = QtWidgets.QPushButton(self.frame)
self.pushButton.setMinimumSize(QtCore.QSize(0, 30))
self.pushButton.setMaximumSize(QtCore.QSize(16777215, 30))
self.pushButton.setStyleSheet("font: 75 14pt \"Alef\";")
icon = QtGui.QIcon()
```

Explication des codes sous-python :

Partie de l'extraction des données avec l'API et la production d'une matrice de cout:

Les bibliothèques utilisées :

Requests : est une bibliothèque pour faire des requêtes HTTP en Python. Il vous permet d'envoyer des requêtes HTTP à un serveur Web et de recevoir la réponse du serveur. Ceci est utile pour interagir avec les API Web, télécharger des données depuis Internet ou tester les performances d'un serveur Web.

Panda : est une bibliothèque d'analyse et de manipulation de données pour Python. Elle fournit des structures de données rapides, flexibles et expressives conçues pour rendre le travail avec des données « relationnelles » ou « étiquetées » à la fois facile et intuitif. C'est un excellent outil pour gérer et analyser de grands ensembles de données.

Le code sous python :

Première partie :

```
import pandas
import requests
import numpy as np
from array import *


fichier = pandas.read_excel('60-shell-station.xlsx')
noms = list(fichier.Name)
lat = list(fichier.Latitude)
long=list(fichier.Longitude)
#-----------------------------------------------------------------
```

La fonction **pandas.read_excel** lit le fichier Excel '60-shell-station.xlsx'et extrait les valeurs des colonnes (Name, Latitude et Longitude). Ces valeurs sont stockées dans les variables noms, lat et long, respectivement. Donc,après cette partie on va avoir 3 listes :Une contient

les noms des différents stations,l'autre possède la latitude de chaque station,et la dernière contient la longitude de chaque station shell

Deuxième partie :

```
data = []
for i in range(len(noms)):
    station = noms[i] +  '//' + str(lat[i]) + ',' + str(long[i])
    data.append(station.split('//'))
```

Cette partie permet de créer une liste vide appelée 'data'.Ainsi ,de itérer sur la liste des noms et combiner les valeurs correspondantes dans lat et long dans une chaîne au format 'name//latitude,longitude'. Chaque chaîne est ensuite divisée selon le délimiteur '//'

Troisième partie :

```
m = array('f',[])
for i in range(60):
    for j in range(60):
        if i == j :
            m.append(0)
        else:
            url = 'https://dev.virtualearth.net/REST/v1/Routes/DistanceMatrix?origins='+data[i][1]+'&destinations='+data[j][1]+'&travelMode=driving&
            r = requests.get(url)
            d = r.json()
            c=float(d['resourceSets'][0]['resources'][0]['results'][0]['travelDuration'])
            print(c)
            m.append(float(d['resourceSets'][0]['resources'][0]['results'][0]['travelDuration']))


m=np.array(m)
m = m.reshape(60,60)
print(m)
```

Cette partie du code consiste à :

- Créer un tableau m initialement vide
- Répeter la boucle 'For' 60 fois pour les deux indices i et j
- Si i est égal à j, on ajoute la valeur 0 au tableau 'm'. Cela représente la distance entre un emplacement et lui-même, qui est toujours 0.
- Si i n'est pas égal à j, le code envoie une requête à « **l'API Bing Maps Distance Matrix** » à l'aide de la bibliothèque requests. La requête API utilise les i et j éléments de la liste data générée de la partie précédente comme origine et destination, respectivement. La réponse de l'API est stockée dans une variable d.
- Le code extrait la valeur travelDuration de la réponse de l'API et la convertit en flottant. Cette valeur est ensuite ajoutée au m tableau.
- Le tableau m est converti en un tableau NumPy et remodelé en une matrice 60x60.

Quatrième partie :

```
df = pandas.DataFrame(m).T
df.to_excel(excel_writer = "matrice_duration.xlsx")
```

- Le DataFrame aura 60 lignes et 60 colonnes, chaque élément du DataFrame représentant la durée du trajet entre deux stations. Les lignes et les colonnes seront étiquetées avec les indices de la liste data, qui contiennent les noms et les coordonnées des stations shell.
- To_excel() une fonction Pandas qui vous permet d'écrire un DataFrame dans un fichier Excel. Le excel_writer paramètre spécifie le nom du fichier Excel dans lequel le DataFrame sera écrit. Si le fichier n'existe pas, il sera créé. Si le fichier existe déjà, le DataFrame sera ajouté au fichier existant.
- Le DataFrame transposé est écrit dans un fichier Excel appelé 'matrice_duration.xlsx'.
- D'où la production d'une matrice de temps

Partie de plus court chemin :

```python
import pandas as pd
from pyroutelib3 import Router
import folium

class TravellingSalesmanProblem:
    def __init__(self, graph):
        self.graph = graph
        self.n = len(graph)
        self.visited = [False for _ in range(self.n)]
        self.visited[0] = True
        self.hamiltonian_cycles = []
        self.path = [0 for _ in range(self.n)]

    def is_valid(self, vertex, actual_position):
        if self.visited[vertex]:
            return False
        if self.graph[actual_position][vertex] == 0:
            return False

        return True

    def tsp(self, actual_position, counter, cost):
        if counter == self.n and self.graph[actual_position][0]:
            self.path.append(0)
            self.hamiltonian_cycles.append(cost+self.graph[actual_position][0])
            self.path.pop()
            return
        for i in range(self.n):
            if self.is_valid(i, actual_position):
                self.visited[i]=True
                self.path[counter] = i
                self.tsp(i, counter+1, cost + self.graph[actual_position][i])
                self.visited[i]=False
```

Les principaux bibliothèques utilisées :

- **Panda** ,c'est une bibliothèque Python consiste à travailler avec les données. Elle fournit des outils pour lire et écrire des données de différents formats, ainsi que des outils pour manipuler et analyser les données.Comme dans notre cas , on va utiliser panda pour lire les données qui existent dans la matrice de distance qui est en format **Excel**

Les parties de code :

1ère partie :

```python
class TravellingSalesmanProblem:
    def __init__(self, graph):
        self.graph = graph
        self.n = len(graph)
        self.visited = [False for _ in range(self.n)]
        self.visited[0] = True
        self.hamiltonian_cycles = []
        self.path = [0 for _ in range(self.n)]

    def is_valid(self, vertex, actual_position):
        if self.visited[vertex]:
            return False
        if self.graph[actual_position][vertex] == 0:
            return False

        return True
```

⟶ Cette partie du code permet de créer une classe nommée 'TravellingSalesmanProblem' .dans laquelle on définit :

- self.n qui représente est le nombre de noeuds dans le graphe, qui est déterminé en prenant la longueur du graphe.
- Self.visited est une liste de valeurs booléennes, avec un élément pour chaque nœud du graphe. La liste est initialisée à toutes les valeurs '**False**' , à l'exception du premier élément, qui est défini comme ' **True'**.
- self.hamiltonian_cycles :est une liste vide qui sera utilisée pour stocker les cycles hamiltoniens trouvés dans le graphe ,càd les cycles qui passent par tous les points une et une seule fois.
- self.path est une liste d'entiers, avec un élément pour chaque nœud du graphe. La liste est initialisée à 0 toutes les valeurs.

2ème partie :

```python
def tsp(self, actual_position, counter, cost):
    if counter == self.n and self.graph[actual_position][0]:
        self.path.append(0)
        self.hamiltonian_cycles.append(cost+self.graph[actual_position][0])
        self.path.pop()
        return
    for i in range(self.n):
        if self.is_valid(i, actual_position):
            self.visited[i]=True
            self.path[counter] = i
            self.tsp(i, counter+1, cost + self.graph[actual_position][i])
            self.visited[i]=False
```

Cette partie permet de résoudre le problème de ‘ Travelling Salesman Problem ‘ en utilisant la fonction tsp qui prend comme arguments :

- actual_position: un entier représentant la position actuelle du chauffeur dans le graphe.
- counter: un entier représentant le nombre de stations visitées jusqu'à présent .
- cost: un entier représentant le coût total du chemin parcouru par le chauffeur jusqu'à présent.
- self: l'objet sur lequel la fonction est appelée

La fonction vérifie d'abord si le chauffeur a visité toutes les stations (counter == self.n) et s'il est possible de revenir à la station de départ (self.graph[actual_position][0]). Si ces deux conditions sont vraies, cela signifie que le chauffeur a terminé un cycle hamiltonien (un cycle qui visite chaque station exactement une fois) et la fonction ajoute la station de départ (0) à la liste des stations visitées (self.path), met à jour la liste des Cycles hamiltoniens avec le coût total du cycle (self.hamiltonian_cycles.append(cost+self.graph[actual_position][0])), et supprime la station de départ de la liste des stations visitées (self.path.pop()).

Si les conditions de réalisation d'un cycle hamiltonien ne sont pas remplies, la fonction itère sur toutes les noeuds (for i in range(self.n)) et vérifie s'il est possible pour le chaufeur de visiter la station (self.is_valid(i, actual_position)). Si la station est valide, la fonction la marque comme visitée (self.visited[i]=True), l'ajoute à la liste des points visités (self.path[counter] = i) et s'appelle de manière récursive avec les arguments mis à jour : i comme la nouvelle position réelle, counter+1 comme le compteur mis à jour et cost + self.graph[actual_position][i]comme le Coût. Enfin, la fonction marque la ville comme non visitée (self.visited[i]=False) avant de passer à l'itération suivante ou de revenir, selon les conditions

Remarque :

Cette fonction est une implémentation récursive du problème TSP, ce qui signifie qu'elle résout le problème en le décomposant en sous-problèmes plus petits et en les résolvant de manière récursive. La fonction garde une trace des points visitées jusqu'à présent et du coût total du chemin emprunté, et utilise ces informations pour déterminer si un cycle hamiltonien a été complété. Il utilise une approche de retour en arrière pour explorer tous les chemins possibles et trouver la solution optimale, qui est le cycle hamiltonien avec le coût minimum.

3ème partie :

```
df = pd.read_excel(matrice_distance)
matrice_distance = df.values.tolist()
```

Cette partie permet de convertir un fichier excel en une matrice en utilisant la bibliothèque panda

4ème partie :

```
excel = pd.read_excel(stations).values.tolist()
stations_coordinates = []
for i in range(len(excel)):
    lat,lon=excel[i][11],excel[i][12]
    lat,lon=float(lat),float(lon)
    stations_coordinates.append([lat,lon])
```

Cette partie permet d'importer tous les coordonnées des stations en utilisant le fichier excel

5ème partie :

```
tsp = TravellingSalesmanProblem(matrice_distance)
tsp.tsp(0,1,0)



distance_optimal = tsp.hamiltonian_cycles[-1]
# ce chemin contient les indices des stations (non pas les coordonnees)
chemin_optimal = tsp.path
```

Cette partie consiste à lancer le tsp et afficher le résultat de l'algorithme

```
wayPoints = []
for i in chemin_optimal:
    wayPoints.append(stations_coordinates[i])
```

Et celle-ci permet de remplacer les indices par les coordonnées

Partie de création du MAP :

```
itineraire_coordonnees = []
j = 1
router = Router("car")
carte= folium.Map(location=[33.57112356277723, -7.595921865804852],zoom_start=15)

for i in range(len(waypoints)-1):

    departi = waypoints[i]
    arriveei = waypoints[i+1]
    ptdepart = router.findNode(departi[0],departi[1])
    ptarrivee = router.findNode(arriveei[0],arriveei[1])
    status, itineraire = router.doRoute(ptdepart, ptarrivee)
    itineraire_coordonnees.extend(list(map(router.nodeLatLon, itineraire)))
    marker = folium.Marker(departi, tooltip=f"Way Point :{j}", icon=folium.CustomIcon(logo_url, icon_size=(50, 50)))
    marker.add_to(carte)
    j += 1

folium.Marker(arriveei,tooltip=f"Way Point :{j}", icon=folium.CustomIcon(logo_url, icon_size=(50, 50))).add_to(carte)
folium.PolyLine(itineraire_coordonnees, color="orange", weight=2.5, opacity=1).add_to(carte)
```

Parmi les bibliothèques utilisées dans cette partie ,on a :

- Folium : est une bibliothèque Python qui permet de créer des cartes interactives à l'aide de la bibliothèque Leaflet.js. Avec Folium, vous pouvez créer des cartes de différentes régions du monde et ajouter des marqueurs, des polygones et d'autres données géospatiales à la carte.
- Router : est une bibliothèque qui permet de créer des routes dans une application web en utilisant Python. Elle fonctionne en associant une URL à une fonction de traitement de la requête, ce qui permet de gérer les différentes actions que l'on souhaite effectuer en fonction de l'URL demandée.

Explication du code :

- La première ligne déclare une liste vide qui sera utilisée pour stocker les coordonnées de l'itinéraire trouvé par la bibliothèque Router.
- La variable "j" est initialisée à 1 et sera utilisée pour ajouter un marqueur sur la carte pour chaque point de passage.
- La variable "router" est initialisée avec une chaîne de caractères "car", ce qui indique que l'itinéraire sera calculé en utilisant le mode de transport "voiture".
- La variable "carte" est initialisée avec un objet de type Map de la bibliothèque Folium, centré sur les coordonnées [33.57112356277723, -7.595921865804852] et affichant un niveau de zoom de 15

- Le code suivant utilise une boucle "for" pour parcourir tous les points de passage de la liste "waypoints" (sauf le dernier). Pour chaque point de passage, le code effectue les étapes suivantes :

- Il récupère les coordonnées du point de départ et du point d'arrivée dans les variables "departi" et "arriveei" respectivement.

- Il utilise la méthode "findNode" de la bibliothèque Router pour trouver les points correspondants aux coordonnées du point de départ et du point d'arrivée. Ces points sont stockés dans les variables "ptdepart" et "ptarrivee" respectivement.

- Il utilise la méthode "doRoute" de la bibliothèque Router pour calculer l'itinéraire entre les points "ptdepart" et "ptarrivee". Le résultat est stocké dans les variables "status" et "itineraire".

- Il utilise la méthode "nodeLatLon" de la bibliothèque Router pour convertir chaque point de l'itinéraire en coordonnées latitude et longitude. Les coordonnées sont ajoutées à la liste "itineraire_coordonnees".

- Il crée un marqueur avec la bibliothèque Folium à l'emplacement du point de départ et l'ajoute à la carte. La variable "j" est incrémentée de 1 pour passer au point de passage suivant.

- Enfin, le code ajoute un marqueur à la carte pour le dernier point de passage de la liste "waypoints" et utilise la bibliothèque Folium pour tracer une ligne sur la carte entre chaque point de l'itinéraire grâce à la liste "itineraire_coordonnees".

Conclusion :

Dans ce projet ,on a amélioré nos compétences dans la recherche,et dans l'extraction de toutes les données qu'on veut de differents sites web possibles(API) et les exploiter par la suite . Ainsi,on a découvert des nouveaux outils et moyens dans Qtdesigner ,et tout cela c'est grace a python qui a rendu

plusieurs projets possibles .En outre, sans Python, ce projet n'aurait pas été possible .

Nous sommes vraiment reconnaissants envers Python pour sa simplicité et sa facilité d'utilisation. Cela nous a permis de nous concentrer sur la résolution du problème au lieu de nous perdre dans les détails techniques du langage de programmation.