Projet Fin Etude

Serious Person 1, Serious Person 2

1er mai 2018

Abstract

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction générale

1.1 Motivation

La population mondiale connaît une très forte croissance depuis 1900. Il y avait 1,5 milliard d'hommes au début du siècle, 2,5 milliards en 1950, et aujourd'hui en 2018 : 7,4 milliards. Les villes se sont développées, au cours des 19e et 20e siècles, sur des carrefours de communication, des fleuves ou sur des littoraux. Des banlieues se sont également développées autour des grandes villes. Avec ce développement, un réseau de transport public a été nécessaire pour permettre de lier les éléments essentiels de ces villes. Mais cette demande croissante de transport a engendré de nouveaux problèmes. Parmi eux, le fait que les usagers nécessitent souvent un moyen pour s'orienter et se renseigner.

L'informatique a apporté plusieurs solutions à ce problème. Il existe en effet plusieurs moyens de visualisation cartographique. On cite OpenStreetMap et Google Maps comme exemple, qui permettent la manipulation et visualisation de données géographiques du monde entier, de tracer des itinéraires entre deux points et d'aider les utilisateurs à marquer et trouver leurs magasins, hôtel ou un autre lieu favori. Cependant, ces outils ne proposent généralement qu'un seul moyen de transport. De plus, certaines de ces fonctionnalités ne sont pas adaptées pour tous les pays, voire non disponibles. Ajouté à ça un nombre immense de lignes de bus, par exemple, avec des noms très peu significatifs et une organisation aléatoire des lignes qui changent fréquemment : des facteurs contribuant encore plus à la confusion des usagers.

Ces raisons font que trouver un chemin optimal faisant bon usage de différents moyens de transport pour une personne est une tâche bien difficile, car ça demande principalement une connaissance parfaite de toute la ville, chose qui est impossible notamment pour un touriste. Le choix du chemin dépend aussi de plusieurs facteurs : la situation et les préférences de chaque personne, ainsi que le jour du trajet, l'heure et la météo. Par exemple, un usager peut choisir habituellement de marcher vers un arrêt de tramway et puis le prendre jusqu'à sa destination, mais préfère, en jour de pluie ou suite à défaut de pouvoir marcher, de prendre deux bus avec un chemin plus long.

Notre but est donc de créer un outil qui aide à cette décision, qui sera ensuite adapté à la ville d'Oran.

1.2 Intérêt et avantages

Offrir un guide de transport évolué a pour apport :

- Gain considérable de temps, de transports et réduction de dépenses.
- Assurer un meilleur respect d'horaire et d'itinéraires de la part des entreprises de transport, et ainsi avoir plus de confiance de la part des utilisateurs.
- Encourager plus de citoyens à utiliser les transports communs.
- Améliorer la circulation en ville en diminuant le nombre d'utilisateurs de véhicules personnels et ainsi réduire les embouteillages.
- Meilleure expérience aux touristes et visiteurs.

1.3 Problématique

La création d'un outil qui aide à cette décision présente plusieurs problématiques, avant de commencer le projet, nous sommes amenés à répondre aux questions suivantes :

- Quels sont ces différents facteurs et critères à considérer lors de ce choix ?

- Quels sont les différents usagers des transports communs, et quel est le besoin de chaque profile ?
- Quelles sont les différentes difficultés et contraintes qui peuvent affecter ce choix ?
- Peut-on offrir une solution informatique évoluée pour assister à ce choix ? Si oui :
  - Comment représenter un réseau routier d'une ville comme Oran, et contenant plusieurs moyens de transport ?
  - Comment calculer un chemin sur ce réseau, et comment déterminer un chemin optimal ?
  - Comment inclure les différentes préférences et circonstances de chaque utilisateur ?
  - Comment présenter cette solution aux usagers de manière simple et efficace ?

1.4 Applications similaires

1.4.1 RATP

L'application RATP a été développée 2010 dans le but d'injecter du réseau social dans le réseau souterrain. Aujourd'hui c'est le compagnon de transport pour Paris et l'ile de France. Parmi les services qu'elle propose, on cite :

- La possibilité de consulter les horaires de passages des différents moyens de transport public desservant Paris et ses environs.
- Permet de retrouver les stations où les arrêts autour de l'endroit où se trouve l'utilisateur grâce à la géolocalisation.

Un avantage particulier avec RATP est le fait qu'il peut être paramétré de manière à émettre des alertes en cas de perturbations ou éventuels retards sur une ligne particulière.

1.4.2 CityMapper

CityMapper est une application de transports en commun qui a été lancé à Londres en 2011. Elle propose 3 modes d'utilisation :

- Découvrir de tous les modes de transports en commun dans cette ville.
- Obtenir les différents moyens pour arriver à la destination souhaitée.
- Comparer le temps de chaque trajet.

CityMapper couvre en ce moment 36 villes dont Londres, Berlin, Tokyo, Paris et New York...etc. Un de ses avantages est la possibilité de prévoir un trajet en avance et de le télécharger pour qu'il soit disponible hors connexion.

1.5 Solutions proposée

Au moment de rédiger ce mémoire, aucune application n'offre ce service en Algérie. ***

De ce fait, nous proposons de développer une application Web full-REST qui proposera les fonctionnalitées suivantes :

- Calculer un chemin entre deux lieux saisis par l'utilisateur.
- Proposer un chemin optimal faisant usage de plusieurs moyens de transport public, tout en donnant la main à l'utilisateur pour choisir les moyens désirés.
- Prendre en compte la possibilité de marche dans les chemins suggérés.

- Se baser sur différents critères dans ce choix, nous prendrons les 04 critères suivants :
  - Minimum de temps.
  - Minimum de marche.
  - Minimum de correspondance.
  - Minimum de cout.

Ce projet sera composé de 04 parties :

Conception et stockage des données : nous étudierons le problème pour enfin proposer une bonne représentation des données que nous pourrons facilement utiliser dans la recherche du chemin.

Service Web : en ce moment, les Services Web sont une tendance sur le Web, presque indispensable dans toute application non triviale : nous commencerons donc par créer une API REST complète de l'application afin qu'elle soit facilement extensible et portable sur différentes plateformes par la suite.

Interface Administratrice : implémenter une application indépendante qui va faciliter la communication avec l'API pour l'ajout et modification des lignes de transport et autres informations nécessaires à l'application.

Interface Web : implémenter une Application Web client afin de visualiser les fonctionnalités de l'API.

Vu le grand nombre de fonctionnalités et la complexité de cette problématique, nous limiterons la version initiale de l'application à un premier algorithme de recherche simple mais applicable sur la ville d'Oran.

L'architecture de l'application devra être suffisamment flexible pour pouvoir améliorer la recherche du chemin, et ajouter progressivement des améliorations ou de nouveaux critères.

1.6 Plan du rapport

Après ce premier chapitre où nous avons présenté les objectifs du projet et la solution proposée ;

Nous présenterons tout au long du Chapitre 2 ce qu'est un service Web, ses caractéristiques et avantages, nous explorerons ensuite deux manières concurrentes pour implémenter des Services Web : SOAP et REST pour débattre brièvement les points forts de chacune afin de justifier notre décision d'utiliser REST.

Le chapitre 3 présentera en première partie les définitions relatives aux graphes et recherche de chemins, et traitera ensuite les différents détails concernant la représentation et stockage des données de l'application.

(À faire après chapitre 4)
Nous présenterons ensuite dans le chapitre 4 les différentes technologies qu'on a utilisé, ainsi qu'un aperçu des représentations utilisées et des détails implémentations. ***qui seront accompagnées par des captures d'écran décrivant le fonctionnement de l'application.** Nous parleront enfin dans le chapitre 5 sur les différentes perspectives et futur stuff

Chapitre 2

Les Services Web

2.1 Introduction

(Laisser l'introduction et conclusion à la fin)

- Web passé de centré interaction utilisateur à plus d'interaction entre applications..
- avant les services web

2.2 Avant les services web

Parler brièvement des prédécesseurs (COBRA, DCOM) middlewares et protocoles, XML-RPC.

Vers les années 90s, les technologies Web et les technologies des systèmes distribués ont commencé à gagner en popularité. Les technologies Web étaient principalement conçus pour envoyer des informations d'utilisateurs à un autre, en HTML par exemple. Les technologies de systèmes distribués, cependant, étaient principalement conçus pour connecter des ordinateurs, ou plus spécifiquement des applications exécutées sur ces machines, et donc leur permettre de s'échanger des informations ou collaborer sur des tâches. Chacune de ces deux technologies avait des objectifs différents, et évoluait vers un chemin différent. Ainsi, les Services Web sont apparu avec but de rassembler ces deux technologies sur un seul objectif commun reliant les deux.[5]

2.3 Définition

Un Service Web est un programme dynamique qui utilise le schéma Client-Serveur pour permettre la communication et l'échange de données entre des applications à travers un réseau (Internet par exemple). Il utilise un système de messagerie standard [1] comme XML mais n'est lié à aucun système d'exploitation ou langage de programmation.

Quelques définitions officielles :

1. W3C : un Service Web est un composant logiciel conçu pour permettre l'interaction interopérable entre machines à travers un réseau. Il a une interface décrite en un format compris par une machine (Spécifiquement WSDL). Les autres systèmes interagissent avec le Service Web d'une manière définie par sa description en utilisant des messages SOAP, généralement transmis en utilisant HTTP avec un XML sérialisé en parallèle avec d'autres Standard du Web.[8]
2. Wikipedia : un service web (ou service de la toile) est un protocole d'interface informatique de la famille des technologies web permettant la communication et l'échange de données entre applications et systèmes hétérogènes dans des environnements distribués. Il s'agit donc d'un ensemble de fonctionnalités exposées sur internet ou sur un intranet, par et pour des applications ou machines, sans intervention humaine, de manière synchrone ou asynchrone. Le protocole de communication est défini dans le cadre de la norme SOAP dans la signature du service exposé (WSDL). Actuellement, le protocole de transport est essentiellement HTTP(S).[10]

1. Un protocole standard est un protocole formalisé (fixé) accepté par une autorité (comme la W3C) ou la plupart des parties qui l'implémentent.

2.4 Principe et objectifs

2.4.1 Pourquoi les services Web ?

Les services web ont été conçus pour permettre l'interaction interopérable entre machines à travers un réseau, en d'autres termes, permettre à différents systèmes de fonctionner et communiquer entre eux.
Puisque différentes applications peuvent être écrites en différents langages et architectures, et sont souvent ramenées à subir plusieurs changements, l'interaction directe entre ces applications est souvent difficile voire pas possible, un service web implémenté pour une application ou ressource offre donc un moyen compréhensible par les autres machines d'accéder au service de cette application et ce, indépendamment de la technologie implémentée par cette application. [9]

2.4.2 Fonctionnement général d'un Service Web

Un service web est un agent réalisé par un fournisseur de services, cet agent joue le rôle d'un intermédiaire entre ce service et les demandeurs extérieurs, ou clients, qui communiquent à travers leur *agent de requête*.
Un service web permet donc de recevoir des messages d'un agent de requête, qu'il traduit ou transmit au service dans un format compris par le fournisseur du service.
La figure **??** résume le rôle de ces différents acteurs.

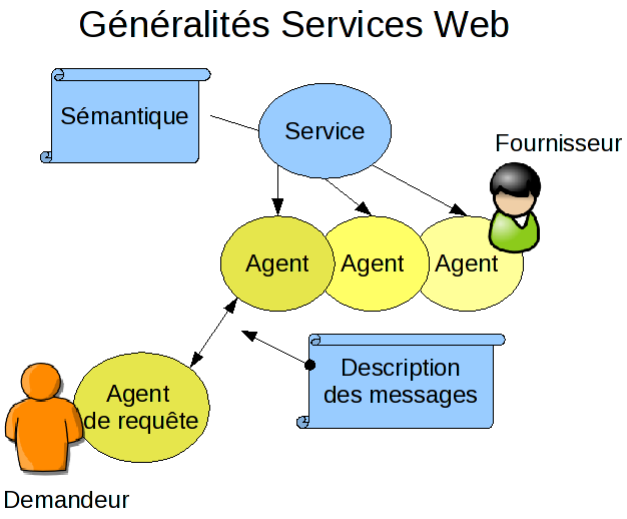

Figure 2.1 – Generalités Services Web

Dans la figure 2.1, l'agent de requête et l'agent du service utilisent le même format de message, basé sur l'ensemble de standards et de protocoles ouverts déjà existants, tel que le XML. Ces protocoles sont implémentés par défaut (ou faciles à intégrer) dans la majorité des technologies, ce qui permet la communication entre différentes applications sans contraintes majeures.

Ce système est réalisé grâce à plusieurs technologies, les plus connus étant les Services Web type SOAP et les Services Web type REST, détaillés dans les parties 2.5 et 2.6 respectivement.

2.4.3 L'architecture orientée services

L'architecture orientée services (Service Oriented Architecture, SOA) est style architectural qui vise à concevoir des services web extensibles et hautement réutilisables, et les rendre ensuite visibles et accessibles aux consommateurs.

On trouve trois acteurs majeurs dans une architecture SOA de service web :

- **Service Provider** : c'est le fournisseur du service. il implémente le service et le rend disponible sur le réseau.
- **Service requester** : c'est le consommateur (Client) de ce service. Il utilise un service web existant en ouvrant une connexion réseau et en envoyant des requêtes (par exemple des messages en XML).
- **Annuaire service registry** : c'est un annuaire de services. ce registre fournit un endroit central où les développeurs peuvent publier leur nouveau service web, ou trouver d'autres existants.

Les services Web emploient un ensemble de technologies qui ont été conçues en respectant une structure en couches sans être dépendantes entre elles. Cette pile, appelée "Pile de protocoles de services web", peut évoluer, mais elle est principalement formée de quatre couches : [9]

Couche transport : cette couche est responsable du transport des messages échangés entre les applications, cette couche utilise généralement le protocole HTTP, mais d'autres tels que SMTP, FTP ou BEEP peuvent être utilisés.

Couche communication (Messaging) : cette couche est responsable de la représentation et formatage des messages échangés entre applications, et ceci afin d'assurer qu'ils soient compris par chaque extrémité. Cette couche utilise principalement XML (ou un dérivé de ce dernier comme XML-RPC et SOAP), mais peuvent aussi utiliser d'autres formats standard suivant l'architecture, les services REST peuvent envoyer en simple texte, JSON ou XML par exemple.

Couche description de service : cette interface est responsable de la description de l'interface publique du Service Web, cette description est réalisée en utilisant le WSDL.

`WSDL : (Web Services Description Language)` : c'est un langage de description standard basé sur XML, développé en jointure par Microsoft et IBM. Il permet de décrire différentes informations sur le Service Web tel que la liste de ses fonctions, comment interagir avec et les invoquer, les ports et protocoles utilisés...etc.

Couche découverte de service : cette couche est responsable de centraliser les services dans un registre commun, et fournir des fonctionnalités de recherche/publication de services web. Ce service est assuré par l'annuaire UDDI.

`UDDI (Universal Description, Discovery and Integration)` : c'est un annuaire de services web, ce protocole, basé sur XML, comporte deux sections en WSDL : les descriptions de ces services et des définitions de ports pour manipuler et rechercher dans l'annuaire.

2.4.4 Caractéristiques d'un Service Web

Dans un Service Web complet on trouve les caractéristiques suivantes :[9]

- Il ne dépend d'aucun système ou langage.
- Il est accessible par un réseau (Internet/Intranet).

- Il dispose d'une interface publique permettant aux clients d'invoquer des procédures ou méthodes sur des objets distants, l'annuaire du service contient la description de ces fonctionnalités et comment les utiliser.
- Utilise un système de messagerie standard (souvent basé sur XML).
- Les fonctionnalités utilisées par le service sont regroupées en un nombre limité de *gros grains* qui sont exposés ensuite au client et ainsi permettre une interaction plus simple avec le service. Par exemple, regrouper des fonctions de bas niveau (petites graines) en une seule fonction de haut niveau, et exposer celle-ci sur le réseau.
- Son système est *faiblement couplé* : le consommateur de ce service n'est pas directement lié au service, ce service peut changer au fil du temps sans perturber le client, ou peut même disparaitre et le client trouvera un autre service en cherchant dans son annuaire.

2.5 Le protocole HTTP

Dans le modèle client-serveur, les clients et les serveurs échangent des messages dans un modèle de messagerie de type requête-réponse : le client envoie une requête et le serveur renvoie une réponse. Pour gérer ces messages et définir un format commun entre eux, on utilise le protocole HTTP.

2.5.1 HTTP (Hyper Text Transfer Protocol)

C'est un protocole qui décrit le contenu et la mise en forme des requêtes et des réponses échangées entre le client web et le serveur web. Lorsqu'un client, généralement un navigateur web, envoie une requête à un serveur web, il utilise un URL combiné avec un "verbe" HTTP pour spécifier le type de message et l'action que doit prendre le serveur sur une ressource.

2.5.2 Méthodes HTTP

HTTP propose plusieurs verbes (ou méthodes), mais les plus importants sont :

- **GET** : obtenir des données, un navigateur web envoie une requête GET au serveur web pour demander des pages HTML par exemple.
- **POST** : envoyer des fichiers ou des données vers le serveur web, comme des données de formulaires.
- **PUT** : envoyer des ressources ou du contenu vers le serveur web, similaire à POST mais utilisé principalement pour mettre à jour une ressource, PUT doit être idempotente : c'est-à-dire qu'envoyer plusieurs fois la même requête produit le même résultat que la première.
- **DELETE** : c'est une requête qui supprime la ressource indiquée.

2.5.3 HTTPS

Le protocole HTTP est certes extrêmement flexible, mais il n'est pas sécurisé. Les messages de demande transmettent au serveur des informations en texte brut pouvant être interceptées et lues. Les réponses du serveur, généralement des pages HTML, ne sont pas chiffrées. Pour une communication sécurisée via Internet, le protocole HTTPS (HTTP Secure) est utilisé. HTTPS utilise l'authentification et le chiffrement pour sécuriser les données pendant leur transfert entre le client et le serveur. HTTPS utilise le même processus de demande client-réponse serveur que le protocole HTTP, sauf que le flux de données est chiffré avec le protocole SSL (Secure Socket Layer) avant d'être transporté sur le réseau.

2.6 Services web SOAP

2.6.1 Le protocole SOAP

SOAP (Simple object Access Protocol) est un protocole de communication standardisé par le W3C. Il définit un format pour l'envoi des messages (basé sur XML) sous forme d'enveloppe et de son contenu, qui peuvent être envoyées par divers protocoles de transport tel que HTTP et SMTP.

SOAP s'appuie donc sur des standards très connus, ce qui lui a donné une grande portabilité et interopérabilité comparée à ses prédécesseurs (COBRA, DCOM, JAVA RMI...etc).

2.6.2 Structure de message SOAP

Un message SOAP est un document XML contenant les éléments suivants : [6]

- **Enveloppe** : définit le début et fin du message, elle contient contient la spécification des espaces de désignation (namespace [2]) et le codage des données. c'est un élément obligatoire.
- **Header (Entête)** : contient des informations ou options supplémentaires pour le traitement du message comme l'encodage, elle est optionnelle.
- **Body (Corps)** : contient les données XML du message à transporter, un élément obligatoire.
- **Fault (Gestion d'erreurs)** : fournit des informations sur les erreurs qui peuvent se produire au traitement de l'information, un élément qui peut être optionnel.

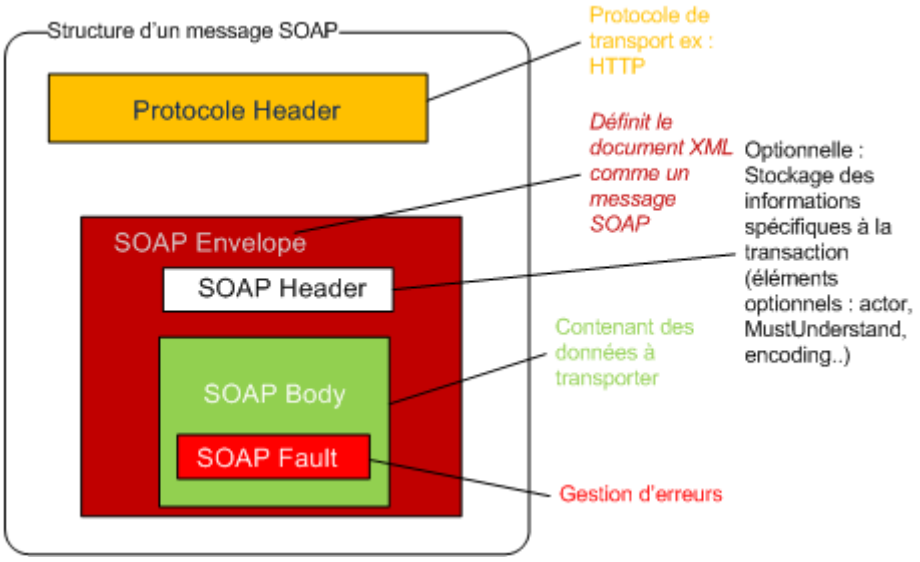

Figure 2.2 – Structure message SOAP

Un message SOAP est un peu lourd, mais reste simple à comprendre et à utiliser, sa structure offre une fiabilité par rapport au format de données, ainsi qu'un moyen de gérer les erreurs grâce au SOAP Fault.

2. Un namespace est un préfixe identifiant de manière unique la signification d'un terme pour éviter toute ambiguïté.

2.6.3 Inconvenients

Bien que SOAP soit une solution standardisée pour l'accès aux Services Web, on peut noter certains inconvénients :

- **Lourdeur :** l'utilisation d'XML et la structure des messages SOAP sont assez verbeuse, XML nécessite aussi d'être parsé [3] , ce qui peut rendre SOAP moins rapide comparé aux autres solutions lorsqu'il nécessite des communications répétées avec le serveur.
- **limité uniquement à XML :** les messages SOAP et WSDL (XML) sont fortement typés, bien que ce soit un point fort de XML, ce n'est pas très adapté pour des systèmes faiblement couplés, car changer le moindre paramètre (par exemple, changer un réel en entier) induit un changement de la signature du type et imposera donc à tous les clients de s'adapter.
- **Différents niveaux de support :** on peut trouver SOAP presque automatisé dans certains langages et IDEs (Java, PHP,...etc), mais beaucoup moins supporté dans d'autres, notamment dans les langages au typage dynamique tel que Python et Javascript.

Ainsi, plusieurs entreprises s'orientent vers REST, et autres ont abandonné leurs services SOAP en faveur de REST, on peut citer comme exemple Google qui a mit fin à son API SOAP en 2009.

3. Parser XML : Lire le document XML et identifier les fonctions de chaque pièce de ce document, tout en vérifiant sa syntaxe.

2.7 REST

Une autre alternative à SOAP est REST, un ***Presenter REST comme une alternative aux Services Web qui utilisent SOAP

2.7.1 Définition

REST est un terme signifiant **REpresentational State Transfer**, provenant de la thèse de doctorat de Roy Fielding, publiée en 2000.

REST n'est pas exactement une architecture, mais un ensemble de contraintes qui, lorsqu'elles sont appliquées à la conception d'un système, créent un style architectural logiciel qui délimite les rôles des ressources et des représentations et la façon dont le protocole de transport (souvent HTTP) est utilisé. Un serveur qui implémente REST fournit et contrôle l'accès à des ressources (pièces d'information) et les présente aux clients.

Un système basé sur REST peut être implémenté dans n'importe quel réseau et architecture disponible en minimisant la complexité de l'implémentation, de la communication et de la distribution.[4]

2.7.2 Les ressources

Une ressource est du contenu identifié par des URI [4] qui peut être un fichier texte, page HTML, image, vidéos...etc. REST fournit au client l'accès aux ressources. Il utilise diverses représentations pour représenter une ressource comme le texte, JSON et XML. Aujourd'hui JSON est le format le plus populaire utilisé dans les services REST.

2.7.3 Service web full-REST

Les services Web RESTful sont des services web basés sur l'architecture REST, en d'autres termes des services qui respectent les contraintes de REST. Ces services sont conçus pour fonctionner au mieux sur le Web, ils sont légers, maintenables, évolutifs et facilement extensibles et sont donc souvent utilisés pour implémenter des APIs pour des applications web.[3]
En résumé, une API est qualifiée de RESTful si elle respecte les critères de REST.

API Web : c'est une interface côté serveur qui consiste en un ou plusieurs *points d'entrée* publiques (endpoints) spécifiant la location d'une ressource et comment y accéder, souvent à travers une URI vers laquelle des requêtes (HTTP) sont envoyées, et par laquelle des réponses serveur sont attendues. Elle définit ainsi un système de requête/réponses entre le client et le serveur à partir de ces points.

4. Uniform Resource Identifiers : une chaine de caractère servant à identifier une ressource en ayant une certaine hiérarchie, par exemple on peut représenter les livres de catégorie Informatique par "/livre/informatique/"

2.7.4 Critères REST

Une API RESTful complète doit vérifier six (06) critères :

- **Orientée client-serveur**.
- **Sans état** : le serveur ne doit avoir aucune idée de l'état du client entre deux requêtes. Du point de vue du serveur, chaque requête est une entité distincte et indépendante des autres, le service ne devrait donc pas avoir besoin de garder les sessions des utilisateurs.
- **Mise en cache** : un client doit être capable de garder en mémoire des informations sans avoir constamment besoin de demander tout au serveur, tel que les images, polices d'écritures, ...etc.
- **Interface uniforme** : permet à tout composant qui comprend le protocole HTTP d'interagir avec l'application, et avoir la possibilité de modifier l'implémentation côté serveur sans affecter l'interface.
- **Avoir un système de couche** : isolation des différents composants de l'application pour bien organiser leurs responsabilités en prenant en charge leurs éventuelles évolution. Chaque couche représente un système borné qui traite une problématique spécifique de l'application.
- **Fournir un code à la demande :** le serveur peut être capable d'étendre les fonctionnalités d'un client en transmettant une "logique" qu'il peut exécuter, comme des Applets Java ou des scripts en Javascript.

Ces contraintes ne dictent pas quel type de technologie utiliser ; Ils définissent seulement comment les données sont transférées entre les composants.

2.7.5 Architecture

Dans REST, les interactions entre clients et services sont améliorées par le recours à un nombre limité d'opérations. L'affectation aux ressources de leurs propres identifiants URI (Universal Resource Identifiers) uniques autorise une grande souplesse. L'architecture se résume en 5 règles à suivre dans implémentation.[2]

- **Règle 1** : l'URI comme identifiant des ressources
  Afin d'identifier une ressource, REST se base sur les URI. Une application se doit construire ses URI (et donc ses URL) de manière précise et statique, en tenant compte des contraintes REST et de futurs changements.
- **Règle 2** : les verbes HTTP comme identifiant des opérations
  Utiliser les verbes HTTP existants plutôt que d'inclure l'opération dans l'URI de la ressource. Ainsi, généralement pour une ressource, il y a 4 opérations possibles : Create, Read, Update et Delete, on utilisera les verbes correspondants : POST, GET, PUT et DELETE. Comme chaque verbe possède une signification, REST permet d'éviter toute ambiguïté.
- **Règle 3** : les réponses HTTP comme représentation des ressources
  Une ressource peut avoir plusieurs représentations dans des formats divers : HTML, XML, CSV, JSON, etc. C'est au client de définir quel format de réponse il souhaite recevoir via l'entête *Accept* de la requête HTTP, comme il est possible de définir plusieurs formats.
- **Règle 4** : les liens comme relation entre ressources
  Les liens d'une ressource vers une indiquent la présence d'une relation. Il est cependant possible de la décrire afin d'améliorer la compréhension du système. Un attribut rel doit être spécifié sur tous les liens.

- **Règle 5 :** un paramètre comme jeton d'authentification
  Pour authentifier une requête, les APIs non-triviales utilisent le jeton d'authentification [5] . Chaque requête est envoyée avec un jeton passé en paramètre ou dans l'entête. Ce jeton temporaire peut être obtenu en envoyant une première requête authentification puis en le combinant avec les requêtes suivantes.

2.7.6 Le format JSON

JSON (JavaScript Object Notation) est un format léger d'échange de données. Il est basé sur un sous-ensemble du langage de programmation. C'est un format texte indépendant de tout langage. Ces propriétés font de lui un langage d'échange de données idéales, qui est facile à lire ou à écrire pour des humains. Il est aisément analysable ou générable par des machines. JSON se base sur deux structures :

- Une collection de couples nom/valeur.
- Une liste de valeurs ordonnée.

Ces structures prennent les formes suivantes :

- Un objet, qui est un ensemble de couples nom/valeur non ordonnés.
- Un tableau est une collection de valeurs ordonnées.
- Une valeur peut être soit une chaîne de caractères entre guillemets, un nombre, un booléen, un objet ou un tableau. Ces structures peuvent être imbriquées.
  ** exemple de format JSON

2.7.7 Avantages

REST devient de plus en plus utilisé dans les entreprises aujourd'hui, les principales motivations pour ce choix sont :[1]

- Plus grande simplicité d'implémentation, aucun besoin d'outils pour interagir avec.
- Courbe d'apprentissage plus petite pour les développeurs.
- Utilisation de multiples formats pour l'échange de données (XML, JSON, HTML), notamment le format JSON qui est plus léger que HTML et plus adapté à certains langages (comme Javascript).
- Plus proche de la conception et de la philosophie initiale du Web (URI, GET, POST, PUT et DELETE).
- Pas de gestion d'états du client sur le serveur.
- Favorise la compatibilité au niveau de la couche de présentation.
- Peut être implémenté localement.
- Utilise l'URI comme représentation de ses ressources, ce qui lui donne une flexibilité d'évolution (changer l'implémentation sans changer l'URL et donc sans affecter le client).

2.7.8 Limites

- Difficile de respecter l'autorisation et la sécurité
- Ne convient pas pour gérer une grande quantité de données
- Moins sécurisé comparé à SOAP
- Il est sans état

5. Un jeton est généralement une suite de caractères uniques à l'utilisateur, par laquelle le serveur peut identifier et ainsi authentifier ses requêtes

- Ne peut pas être utilisé lorsque les interactions client et serveur sont essentielles
- Ne peut pas être utilisé pour les transactions impliquant plusieurs appels

2.7.9 REST vs SOAP

** points à mentionner :

- expliquer que REST est un style architectural, SOAP un protocole
- REST n'est pas lié à XML contrairement à SOAP
- SOAP est strictement typé et a une spécification standard plus stricte pendant que REST donne le concept et est moins restreint dans l'implementation
- SOAP a une gestion des erreurs par défaut (vu dans son enveloppe), expliquer utilité pour un exemple de transaction bancaire

conclusion : SOAP peut être utile pour des applications d'entreprises demandant une haute sécurité "unless you have a good reason, use REST"

2.8 Exemples et applications

Deux-Trois Api connues, exemple avec Google Maps

2.9 GraphQL et GRPC

2.10 Conclusion : pourquoi choisir REST

Après avoir exploré les deux technologies, nous avons choisi d'implémenter notre Service Web en utilisant REST qui est plus simple à mettre au point mais aussi plus flexible que SOAP, ce qui permettra d'améliorer ce projet plus rapidement.

Il existe encore certains principes qui n'ont pas été cités dans ce chapitre, bien qu'il n'existe pas réellement une manière absolue pour implémenter une API REST, il existe certaines *bonnes pratiques* que nous tâcherons d'implémenter dans ce projet. Le résultat ainsi que les détails d'implémentation seront décrits dans les chapitres suivants.

Chapitre 3

Représentation du réseau routier

Un graphe sert mieux à définir l'existence d'une relation entre objets tels qu'une ligne entre deux stations de métro, ce qui est la représentation optimale pour nos données. Dans ce chapitre, nous présenterons en première partie les définitions relatives aux graphes et recherche de chemins, ensuite nous discuterons les différentes approches de représentation prises en compte et les spécifications de nos données, enfin nous détaillerons la représentation choisie en donnant des exemples.

3.1 Généralités sur les graphes

La théorie des graphes est très probablement née en 1735 lorsque Leonhard Euler (1707 - 1783) résout le problème des sept ponts de Königsberg. L'énoncé de ce problème est : la ville de *Königsberg* est une ville autour d'un fleuve, elle compte quatre berges et sept ponts les reliant. Le but du jeu est de savoir s'il existe un chemin permettant d'emprunter tous les ponts une fois et une seule et revenir au point de départ. Le problème s'appelle désormais, de façon plus formelle, la recherche d'un cycle eulérien dans un graphe. Euler a démontré que ce problème n'avait pas de solution.

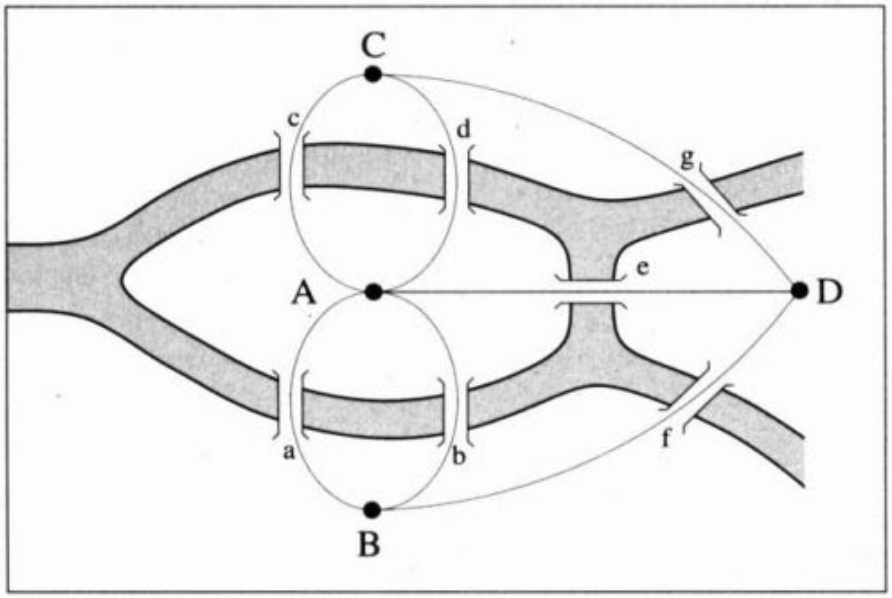

Figure 3.1 – Démonstration du problème des septs ponts de Königsberg

3.1.1 Définitions

Graphe : un graphe est composé de sommets (**vertices**) ou noeuds (**nodes**), et d'arcs (**edges**) ou d'arêtes (**links**) reliant certains de ces sommets ou noeuds. Un graphe G est défini de manière formelle par un couple (S,A) où :

- S est un ensemble fini d'éléments. Chacun de ces éléments est appelé sommet du graphe.
- A est un sous-ensemble (éventuellement nul) de SxS. Chacun de ces éléments de A est appelé arc ou arête.

Chaque arc est associé à un poids ou une étiquette qui le décrit. Par exemple, dans un réseau social il peut définir la nature de la relation (ami, famille, collègue) et dans un réseau routier la longueur d'une rue. Parfois le terme coût est utilisé.

Graphe connexe : un graphe est connexe si on peut atteindre n'importe quel sommet à partir d'un sommet quelconque en parcourant différentes arêtes.

Graphe directionnel : aussi appelé graphe orienté, digraphe ou un réseau dirigé, c'est un graphe où les sommets (nœuds) sont connectés ensemble, et tous les bords sont dirigés d'un sommet à l'autre. Les bords sont généralement des flèches dessinées indiquant la direction. Un graphe où les bords sont bidirectionnels est appelé un graphique non orienté.

Chemin : un chemin est une séquence finie et alternée de sommets et d'arcs, débutant et finissant par des sommets, tel que chaque arc sortant d'un sommet est incident au sommet suivant dans la séquence (cela correspond à la notion de chaîne *orientée*).

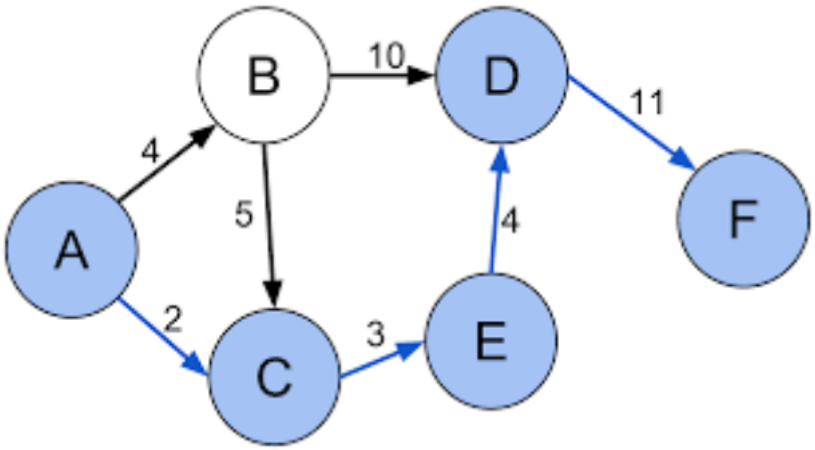

Figure 3.2 – Exemple de chemin orienté

3.2 Avantages d'utilisation d'un graphe

3.2.1 Domaines d'utilisation des graphes

Un graphe sert avant tout à manipuler des concepts, et à établir un lien entre ces concepts. N'importe quel problème comportant des objets avec des relations entre ces objets peut être modélisé par un graphe. Les graphes sont donc des outils très puissants et largement répandus qui se prêtent bien à la résolution de nombreux problèmes. Voici quelques-uns :

3.2.2 Recherche de chemins (PathFinding)

Un cas très fréquent. Chaque nœud représente une position et chaque arête est un chemin entre deux positions, ou en remplaçant les nœuds par des adresses et les arêtes par des routes, on obtient le graphe utilisé par les GPS ou Google Map par exemple. La recherche de chemins est aussi utilisée en biologique, communications (réseaux de télécommunications), réseau hydrographique...etc. Il est courant de chercher le chemin le plus court entre deux positions dans la plupart de ces domaines, nous nous intéressons particulièrement à ce cas d'utilisation, que nous discutons en détail dans la section suivante.

3.2.3 L'ordonnancement de tâches :

On peut représenter chacune des tâches à effectuer par un nœud, et les dépendances entre chacune de ces tâches par des arêtes. On cite l'exemple de l'ordonnancement des projets, les graphes permettent de planifier les différentes tâches d'un projet, détecter les taches pouvant être effectuées simultanément et estimer la durée totale du projet.

3.2.4 Les systèmes de recommandation :

C'est une forme spécifique de filtrage de l'information qui a pour but de présenter à un utilisateur des éléments qui sont susceptibles de l'intéresser, en se basant sur ses préférences et son comportement. Les moteurs de recommandation font usage des graphes pour représenter des individus ou objets et leurs différents liens, cet outil est très utilisé en sciences sociales, par exemple le graphe social de Facebook qui représente les associations entre des personnes ou le réseau LinkedIn qui est un graphe de relations entre des professionnels...etc.

Le but est de pouvoir identifier les communautés formées, les centres d'intérêt commun, en suggérant à l'utilisateur les choses qu'il est susceptible d'aimer, les personnes qu'il connaît peut-être, et avant tout (et surtout) pour créer des publicités ciblées adaptées à chacun.

3.3 Recherche de Chemin :

(Intro à revoir)

Il est difficile de retracer l'histoire du problème du plus court chemin. On peut imaginer que même dans les sociétés très primitives, trouver des chemins courts était essentiel. Comparé à d'autres problèmes d'optimisation combinatoire, comme l'algorithme de Kruskal, l'affectation et le transport, la recherche mathématique dans le problème du chemin le plus court a commencé relativement tard. Cela pourrait être dû au fait que le problème est élémentaire et relativement facile, ce qui est également illustré par le fait qu'au moment où le problème est apparu, plusieurs chercheurs ont développé indépendamment méthodes similaires. Les problèmes de chemin ont été étudiés au début des années 1950 dans le contexte de routage alternatif*, c'est-à-dire, trouver une deuxième route plus courte si la première est bloquée. À cette époque, les appels interurbains aux États-Unis étaient automatisés et il fallait trouver automatiquement d'autres itinéraires pour les appels téléphoniques sur le réseau téléphonique américain.*

3.3.1 Définition :

La recherche du plus court chemin est la capacité pour un système de déduire le chemin approprié autour des obstacles pour atteindre un point de destination tout en évitant les obstacles et en parcourant la distance la plus petite possible. Le choix de la méthode de l'analyse et sa complexité peuvent augmenter à mesure que d'autres circonstances doivent être analysées, en prenant en compte différentes contraintes :

- **Poids :** certains algorithmes n'acceptent que des arcs dont le poids est positif.
- **Chemins calculés :** il existe des algorithmes qui calculent le plus court chemin de nœud à nœud, entre toutes les paires de nœuds ou encore d'un nœud vers tous les autres.
- **Prise en compte d'informations externes :** l'utilisation d'une connaissance externe à la structure du graphe peut parfois accélérer la recherche.

3.3.2 Domaines d'utilisation :

Le problème du plus court chemin est parmi les problèmes les plus étudiés de la théorie des graphes ,on le retrouve dans beaucoup de domaines :

- **Economie :**problèmes d'investissement
- **Gestion :** gestion des projets
- **Optimisation des réseaux :** (réseaux routiers, de télécommunications, de distribution)
- **Réseaux informatiques et protocoles de routage :** (protocole OSPF : Open Shortest Path First)
- **En Biologie :** il est utilisé pour trouver le modèle de réseau dans la propagation d'une maladie infectieuse.
- **Cartographie :** pour mesurer le chemin le plus court entre deux lieux ou villes.
- **Jeux vidéo**.
- **En robotique**.

3.3.3 Les algorithmes de recherche de chemin

**deux types principalement : à fixation d'étiquettes (Djikstra) et correction (Bellman Ford)
(Parler de complexité)
Les algorithmes de calcul d'itinéraires origine-destination(s) sont de complexité polynomiale et on les répartit classiquement en deux familles : ceux à fixation d'étiquettes (algorithme de Dijkstra) et ceux à correction d'étiquettes (algorithme de Bellman-Ford). Malgré leur complexité polynomiale, ces algorithmes peuvent néanmoins engendrer des temps de calculs importants pour des graphes de grande taille, ce qui a suscité le développement des techniques d'accélération (visant souvent l'optimalité)comme l'algorithme A*, le parcours bidirectionnel, ainsi que les méthodes de pré traitement.

Nous nous interesserons (.... parcours bidirectionnel ..etc)

3.3.4 L'algorithme (utilisé)

- présenter l'algorithme
- Parler de complexité et pourquoi avoir choisi
- Code/algorithme de .. l'algorithme.

3.4 Données collectées

Afin de mieux tester notre application, nous avons tenté de collecter un maximum de données réelles, à commencer par contacter l'Entreprise de Transport d'Oran, puis ensuite traiter ces informations et ajouter d'autres que nous avons collectés par nous-mêmes. Le résultat de cette collecte est comme suit :

3.4.1 ETO (Entreprise de Transport d'Oran)

- Nous avons été accueillis par un des responsable de l'ETO, qui nous a fourni plusieurs informations sur (comment marche chaque ligne, fréquences, horaires, temps d'été/hiver...etc)
- Parler des données des lignes de l'ETO qu'il nous a fourni
- les stations ont des noms, donc besoin de stocker nom "commun" et adresse.
- Parler des inconvénients, adresses non complètes, informations mal classées,... donc besoin de traiter et de compléter
- Remerciements.

3.4.2 Données supplémentaires

Données qu'on a ajouté nous-même, parler de l'idée du crowd-sourcing.

3.4.3 Traitement de données

Comment on a utilisé ces données (exemple : trouver coordonnées, calculer distance avec les coordonnées), et les outils/programmes écrits pour automatiser.. si possible).

** Nous serons contraints d'utiliser les adresses données comme premiere version, vu l'absence de données GPS, nous (*ajouter une fonctionnalité pour pouvoir intégrer aisément les coordonnées GPS au dessus des données existantes.

3.5 Construction du graphe

3.5.1 Différentes approches

- **Outils Open Source (OpenTripPlanner** : Il existe plusieurs outils qui proposent ce service, en particulier OpenTripPlanner : un projet Open Source qui permet de créer un réseau routier à partir de données GTFS [1] , ces données seront ensuite intégrées avec OpenStreetMap et stockée sur le serveur, qui exposera une API REST pour questionner le serveur : recherche de Chemin, possibilité d'intégrer les horaires, ...etc.
  Vu la nature un peu particulière du réseau d'Oran, et la non-disponibilité des données (Données officielles des lignes et données (adresses) sur OpenStreetMap), cette solution ne sera pas envisagée. Cependant, l'application prendra une architecture flexible permettant d'intégrer, au futur, de tels outils rapidement au cas de nécessité.

1. GTFS (General Transit Feed Specification : ...

- **Représentation indépendante de chaque ligne** Une des approches considérées était de représenter chaque ligne indépendamment, l'algorithme du service aura à chercher des points de liaison entre ces lignes. L'avantage principal de cette approche est la facilité de manipulation de ces données de lignes, en ajoutant/supprimant des lignes sans conflit. Cette approche présente par contre un inconvénient majeur au niveau de performance, vu que l'utilisation d'un algorithme de Path-Finding requiert l'utilisation de plusieurs tables (lignes) à chaque requête. Ce qui nous a menés à l'approche suivante.
- **Représentation en graphe (Base de données orientée graphe** : nous avons finalement opté vers une représentation en graphe, qui est la plus intuitive dans notre cas, et qui permet aussi un accès plus rapide et simple en parcourant un graphe précalculé, tout en tirant les meilleures performances des différents algorithmes de recherche de chemins. On utilisera pour cela une base de données orientée graphe, qui permet de stocker en permanence le graphe, mais aussi de fournir plusieurs opérations pour créer, modifier et parcourir nos graphes.
  On a, cependant, noté certaines contraintes suivant cette représentation,
  *** lors de la modification, modification d'une station peut affecter plusieurs lignes, modification d'une ligne peut poser plusieurs conflits ou imposer la reconstruction du graphe...

3.5.2 Représentation choisie

- Présenter en détail la représentation
- Donner la structure du graphe (trouver une façon pour représenter proprement ces informations :)
  - Noeuds représentent station
  - Arcs representent un tronçon/chemin entre deux stations
  - Arcs étiquetés par le type du transport (BusSegment, TramSegment, WalkSegment. ..)
  - Repéesenter chaque bus séparémment (pour faciliter la recherche et filtre)
  - Noeuds (Stations) ont les attributs suivants :
    - Nom.
    - Adresse.
    - Tableau de coordonnées (pour chaque direction) avec indicateur de direction.
  - Arcs ont les attributs suivants :
    - Dist : distance entre les deux stations
    - Time : temps moyen pour passer de la première station à la deuxieme
    - Bus : (pour les arcs de type Bus) nom/id du bus. ** remarque : infos du bus (temps d'arrêt, prix ..etc) sont stockés sépareraient.

3.5.3 Résultat final

** Capture d'écran d'un graphe exemple.

3.5.4 Conclusion

Malgré les différentes contraintes (données manquantes, horraires non régulières...etc.), bla bla

Chapitre 4

Développement

4.1 Technologies utilisées

4.1.1 NodeJs

Pour l'API, parler des avantages : performances, moteur V8, flexibilités...

4.1.2 Neo4J

Pourquoi choisir une BDD orienté Graph, pourquoi Neo4J , ..etc. plus connue, big community Neo4J utilise l'algorithme

** **Cypher :** présenter le language de requete avec quelques exemples

4.1.3 MongoDB

4.2 Structure de l'application

4.2.1 L'architecture MVC (Model-View-Controller)

L'application suit une architecture MVC qui est un modèle d'architecture qui sépare une application en 3 composants logiques : **Model**, **View**, **Controller**. Chaque composant est construit pour gérer un aspect spécifique de l'application, on peut résumer le rôle de chacun comme suit :

1. **Models :** Ces composants correspondent à tout ce qui est relié aux données, un model peut représenter une classe d'objets ou une structure de données échangée entre Views et Controllers, ainsi que la logique manipulant ces données tel que des calculs de moyennes, les différentes opérations d'ajout, modification et suppression dans une base de données.
2. **Controller :** Le Controller joue le rôle d'une interface entre le Model et les Views afin de traiter les requêtes venant du View (Par exemple un clique de bouton), manipuler les données du Model et enfin les rendre au View pour qu'elles soient présentées.
3. **Views :** Les views sont utilisés pour l'interface utilisateur (UI) de l'application, ceci inclut les composants d'interface (Textes, boutons, formulaires...etc.), ainsi que certaines logiques de mise en forme et affichage des données.

MVC est une des architectures les plus utilisées pour créer des projets évolutifs et extensibles. Il existe plusieurs variantes du MVC et d'autres architectures similaires tel que le MVVM (Model-View-ViewModel) et le MVP (Model-View-Presenter). [7]

4.2.2 Composants de l'application

** Placer diagramme Here

1. **Models :**
   - Station :
   - Line :
     - Bus :
   - Path :
     - Step :
   - Models du graphe :
     - GrapheNode : Represente un noeud.
     - GrapheSegment : Represente un arc entre deux noeuds.

4.3 Implémentation de l'API

4.3.1 Ressources exposées

Notre API définit principalement 3 ressources :

1. **Station :** Ressource qui représente les différentes stations du réseau, elle accepte 3 requêtes différentes de type GET :
   - ***/api/station*** : Retourner toutes les stations.
   - ***/api/station/{id}*** : Retourner une station par identifiant.
   - ***/api/station ?match={name}*** : Retourner toutes les stations dont le nom contient le paramètre *match*, peut être utilisée pour la recherche lors du choix du chemin, par exemple.

   Cette ressource accepte aussi des requêtes de type POST (création), PUT (mise à jour) et DELETE (suppression) pour les client admin (authentifiés).
2. **Ligne : *api/line*** : Ressource qui représente les différentes lignes de transport entre stations, elle accepte des requêtes de type GET du coté client, aussi des requêtes POST, PUT et DELETE du coté admin pour gérer les lignes.
3. **Direction : *api/direction*** : Ressource représentant les chemins, elle accepte seulement des requêtes de type GET, les détails de ces requêtes et leurs réponses sont détaillés dans la section 4.4.1

4.3.2 Description des formats des ressources

1. **Station :**

```
{
  "id": 0,
  "name": "string",
  "address": "string",
  "coordLat": 0,
  "coordLon": 0
```

```
}
```

Listing 4.1 – Format JSON de Station

2. **Line :**

```
{
  "id": 0,
  "name": "string",
  "bus": {
    "name": "string",
    "price": 0,
    "frequence": 0,
    "avgWaitTime": 0
  },
  "lineStations": [
    {
      "stationID": 0,
      "distFromPrev": 0,
      "timeFromPrev": 0
    }
  ]
}
```

Listing 4.2 – Format JSON de Line

4.4 Recherche de chemin

L'implémentation de l'application du coté serveur permet de calculer un ensemble de chemins optimaux entre deux stations identifiées par un identifiant. Le calcul se déroule en plusieurs étapes :

4.4.1 Déroulement d'une requête

** Figure du diagramme de Sequence

1. L'API reçoit la requête de chemin, vérifie les paramètres et met en forme les données, pour les passer au module qui calcule le chemin (PathFinder).
2. Le module PathFinder formule la requête Cypher et l'envoi au serveur de la base de donnée Neo4j. La requête est comme suit :

```
MATCH p = AllShortestPaths((A:Station)-[*..20]->(B:Station))
WHERE A.name = {startParam} AND B.name = {endParam}
  AND ALL(rel IN relationships(p)
  WHERE type(rel) IN {transportsParam})

WITH p, RELATIONSHIPS(p) as segments
WITH EXTRACT (segment in segments| StartNode(segment)) AS startNodes,
EXTRACT (segment in segments| EndNode(segment)) AS endNodes,
RELATIONSHIPS(p) as segments

RETURN segments, startNodes, endNodes
```

Listing 4.3 – Requête Cypher des plus courts chemins

Nous pouvons depuis cette requête filtrer les différents transports à inclure dans le chemin facilement, puisque chaque moyen de transport est désigné par ses propres arcs étiquetés dans le graphe.

3. Neo4J retourne ainsi une liste de chemins optimaux, le module va ensuite évaluer chaque chemin :
   - Évaluer le temps et cout estimé de chaque chemin.
   - Trier les chemins selon le critère de la requête, en cas d'égalité considérer les autres critères.
   - Structurer chaque chemin en Objets Path, détaillé dans la section suivante.
   - Minimiser le chemin en regroupant les étapes intermédiaires (tel que deux arrêts utilisant le même Bus).
4. Retourner l'objet à l'API qui l'envoi sous forme de réponse HTTP.

4.4.2 Format des requêtes

4.4.3 Format des réponses

L'API retourne un tableau des chemins (Objets Path) en format JSON, ces objets contiennent les informations suivantes :

- Le temps, prix et distance totale du chemin.
- Les moyens de transport utilisés dans ce chemin.
- La liste des étapes à suivre dans ce chemin, chaque étape contenant les données suivantes :
  - La station de départ et la station d'arrivée.
  - Les stations intermédiaires entre station de départ et d'arrivée.
  - Le prix, temps estimé et distance à parcourir de l'étape.
  - Le transport à prendre (ou type de l'étape).

```
{
  "totalDist": 0,
  "totalPrice": 0,
  "totalTime": 0,
  "steps": [
    {
      "sourceStation": {
        "id": 0,
        "name": "string",
        "address": "string",
        "coordLat": 0,
        "coordLon": 0
      },
      "destStation": {
        "id": 0,
        "name": "string",
        "address": "string",
        "coordLat": 0,
        "coordLon": 0
      },
      "price": 0,
      "time": 0,
      "type": "string",
      "name": "string"
    }
```

```
26    ]
27 }
```

Listing 4.4 – Format JSON de Path (chemin)

4.5 Implémentation

Chapitre 5

Conclusion

Evaluer le résultat

Bibliographie

[1] Amine Benkirane. *Architecture logicielle : Web services SOAP ou REST ?* 2012. URL : http://amine.benkirane.over-blog.com/2012/05/architecture-logicielle-soap-vs.-rest-web-services.

[2] Nicolas Hachet. *L'architecture REST expliquée en 5 règles*. 2012. URL : https://blog.nicolashachet.com/niveaux/confirme/larchitecture-rest-expliquee-en-5-regles/.

[3] *RESTful Web Services*. Nov. 2017. URL : https://www.tutorialspoint.com/restful/.

[4] L. Richardson et S. Ruby. *RESTful Web Services*. O'Reilly Media, 2008. ISBN : 9780596554606. URL : https://books.google.dz/books?id=XUaErakHsoAC.

[5] George Scott. *The Road to Web Services*. 2001. URL : https://www.w3.org/2001/03/WSWS-popa/paper63.

[6] *SOAP*. Nov. 2017. URL : https://www.tutorialspoint.com/webservices/.

[7] tutorialspoint. *MVC Framework*. 2018. URL : https://www.tutorialspoint.com/mvc_framework/mvc_framework_introduction.htm.

[8] W3C. *Web Services Glossary : Web service*. 2004. URL : https://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/#webservice.

[9] *Web Services*. 2017. URL : https://www.tutorialspoint.com/webservices/.

[10] Wikipedia. *Service Web*. 2017. URL : https://wikipedia.org/wiki/Service_web.