

AVRIL 2021



Le projet : Tchou - Tchou

PROPOSÉ PAR

GAUDILLAT Eliott, BERNAT Loan

SOMMAIRE

1 - Organisation de Travail

- 1.1 - Brainstorming & Définition des objectifs
- 1.2 - Répartition de la charge de Travail
- 1.3 - Mode de fonctionnement

2 - Écriture du code

- 2.1 - Structure du Programme
- 2.2 - Eliott
- 2.3 - Loan

3 - Mode d'emploi du Programme

- 3.1 - Le menu des admins
- 3.2 - Le menu des contrôleurs
- 3.3 - Le menu des clients

4 - Annexes

- 4.1 - Notre Avis sur le Projet & Améliorations
- 4.2 - La C-Doc du projet
- 4.3 - Liens & Droits & Source

1 - Organisation de Travail

1.1 - Brainstorming & Définition des objectifs

Lors de la remise du sujet et suite à sa lecture, nous avons émis de nombreuses idées (interface graphique via SDL, Graphes, ...). Nous nous sommes revus la semaine du 22 février pour commencer à poser réellement sur papier les axes sur lesquels nous allions travailler. Nous avons donc écarté l'interface graphique par manque de temps, et nous avons posé des "préfaces" de structures dont nous aurions besoin pour représenter en mémoire cet ensemble routier. On s'est donc fixé les objectifs suivants : rendre une documentation complète à la manière d'une javadoc, tenir à jour le Github régulièrement, remplir la quasi-totalité des objectifs fixé sur la fiche de projet.

1.2 - Répartition de la charge de Travail

Nous nous sommes réparti les tâches de façon à travailler en collaboration sans pourtant avoir besoin d'intervenir dans les fonctions de l'autre. Eliott s'est concentré principalement sur l'interface console et la gestion des mots de passe, des comptes administrateurs et contrôleurs ainsi que de la navigation générale dans l'application ! On peut associer cela au "front-end" de l'application. Quant à Loan, il s'est occupé de la partie plutôt "back-end", soit le stockage en mémoire du réseau complet, les interactions avec ce dernier et la sauvegarde entre chaque utilisation.

1.3 - Mode de Fonctionnement

Nous avons fait le choix de fonctionner avec GitHub pour nous partager les avancées du projet de manière régulière. [Le lien du GitHub](#) est disponible en annexe, nous vous invitons à aller jeter un œil pour avoir une version avec des tests déjà effectués en mémoire (plus de gares, plus de trajets, déjà des voyageurs en mémoire...). Nous évoquions vocalement les points à corriger ou à ajouter pour nous tenir à jour régulièrement mutuellement, que cela soit via Discord ou à la Fac. Chacun travaillait sur sa branche ou parfois directement sur le main lors de petits correctifs ou améliorations possibles.

2 - Écriture du Code

2.1 - Structure du Programme

Le programme contient :

- un fichier main.c
- 8 modules composés respectivement d'un fichier .h et un fichier .c contenant nos structures et nos fonctions. Ainsi que 3 accesseurs.
- 9 fichiers .txt qui servent de fichier de sauvegardes.
- 2 fichier .json pour exporter les données

```
int main(int argc, char const *argv[])
{
    /* initialisation du reseau*/
    Reseau reseau = initReseau();

    int event = 1 ;
    long choixMenu;
    while(event != -1){
        afficheMenuPrincipal();
        choixMenu=lireLong();
        printf("\n");
        switch(choixMenu)
        {
            case 1:
                menuAdminVerification(reseau);
                break;
            case 2:
                menuControleurVerification(reseau);
                break;
            case 3:
                menuClient(reseau);
                break;
            case 4:
                afficheMessageQuitter();
                event = -1;
                break;
            default:
                afficheErreurMenu();
                break;
        }
        printf("\n");
    }
    //Sauvegarde et Fermeture du Reseau (liberation de la memoire)
    sauvReseau(reseau);
    closeReseau(reseau);
    return 0;
}
```

Notre Fonction main se compose en trois parties. 1ere phase on initialise notre Réseau ferroviaire en mémoire à partir des informations des fichiers .txt .

Ensuite, vient la partie principale de la fonction. Celle-ci est composée d'une pseudo boucle événementielle. En effet, notre boucle affiche sur la console le menu principal de notre programme et va attendre que l'utilisateur réalise un événement, ici la saisie d'un chiffre sur le clavier. Si l'utilisateur saisit la valeur ' 4 ' cela modifie la valeur de la variable event et permet de sortir de la boucle afin de terminer l'exécution du programme.

La dernière partie du main survient après la boucle while et avant la fermeture du programme. On va venir sauvegarder notre programme dans les fichiers .txt puis libérer la mémoire.

De plus, on peut voir la structure de notre programme comme un programme "à tiroirs". En effet, dans la boucle while de notre main l'utilisateur va par saisie d'un chiffre sélectionner le menu qui l'intéresse parmi 3 menus :

- Administrateur
- Contrôleur
- Client

Ce qui va appeler la fonction du menu associé elle-même composé d'une boucle événementielle similaire ainsi de suite. On peut donc voir cela comme quelqu'un face à une armoire qui ouvre un tiroir pour en voir le contenu.

On a choisi cette structure de programme, car c'est celle qui paraissait la plus évidente pour représenter un menu en console et permettre une navigation plutôt efficace entre les différents menus.

Une des difficultés était de sécuriser la saisie de l'utilisateur si celui-ci tape une lettre au lieu d'un chiffre par exemple.

Et cette structure amène comme contrainte d'avoir notre réseau comme paramètre dans la plupart de nos fonctions.

2.2 - Eliott

2.2.1 : Représentation administrateur/Contrôleur

Pour stocker les informations concernant l'administrateur et les Contrôleurs j'ai décidé d'écrire cela dans un fichier .txt par personne selon un modèle précis :

```
cctrlsd
*T\C_h<0x19>1NHT]E4[]<0x1e>
Yannick
Chevalier
```

- identifiant
- mot de passe crypté
- prénom
- nom

Ce choix d'écriture amène des avantages et des inconvénients.

En effet, cela a permis par la suite dans les fonctions du module pwd (contient les fonctions liées à la gestion des données des contrôleurs et de l'administrateur) de rendre le travail plus simple. Par exemple, dans la vérification des identifiants et du mot de passe pour accéder aux menus sécurisés, étant données que les fichiers sont écrits d'une certaine manière, il m'était plus facile d'accéder au mot de passe pour le comparer avec le mot de passe saisi par l'utilisateur.

Cependant, cela a comme inconvénient que cela rend statique le nombre de contrôleurs, Les fonctions de modifications des données sont assez spécifiques au programme et peu réutilisables.

J'aurai pu choisir de mettre toutes les données des contrôleurs dans un seul et même fichier pour permettre l'ajout ou la suppression d'un contrôleur rendant le tout dynamique. Mais cela n'est pas précisé dans les contraintes de programmation.

Donc le faire de manière plutôt statique semblait la meilleure solution pour ce projet.

Et cela peut être une amélioration à apporter à notre programme.

2.2.2 : Les menus

Les fonctions du module menu.c vont jouer le rôle de l'interface graphique.

Elles vont faire le lien entre l'utilisateur et les autres fonctions du programme.

Comme la fonction main, les fonctions de menu.c vont dans un premier temps afficher sur la console des informations et des instructions à l'utilisateur puis demander une saisie à l'utilisateur et enfin envoyer les informations saisies à des fonctions d'autres modules.

Comme on peut le voir sur l'exemple ci-dessous avec le menuModificationVoyage, on va afficher l'instruction de saisir un numéro client.

```

int menuModificationVoyage(Reseau r){
    long choix;
    char numClient[10];
    printf("\n");
    printf("#####\n");
    printf("# Indiquez votre numero client #\n");
    printf("#####\n");
    printf("\n");
    scanf("%s", numClient);
    fflush(stdin);
    rechercheVoyageur(r,numClient);
    int event = 1;
    printf("\n");
    printf("#####\n");
    printf("# 1- Modifier #\n");
    printf("# 2- RETOUR #\n");
    printf("#####\n");
    printf("\n");
    while(event != -1){
        choix = lireLong();
        switch (choix) {
            case 1:
                modifVoyageur(r, numClient);
                event = -1;
                break;
            case 2:
                event = -1;
                break;
            default :
                afficheErreurMenu();
                break;
        }
    }
    return 0;
}

```

l'utilisateur va saisir son numéro de client et ce numéro de client est envoyé à la fonction `rechercheVoyageur` pour vérifier l'existence ou non de ce voyageur.

2.2.3 : La sécurité

Dans cette partie on va parler du module `pwd.c`. Ce module est composé des fonctions en lien avec la sécurité. C'est-à-dire sécuriser la saisie de l'utilisateur. Vérifier les accès aux menus sécurisés et crypter les mots de passe.

la fonction `lire` était nécessaire pour lire sur `stdin` de manière sûr.

La fonction `lireLong` est une fonction dérivée de `lire`. On l'utilise dès lors que l'on veut naviguer dans un menu, étant donné qu'on souhaite un chiffre pour faire un choix on vérifie que l'utilisateur en a bien saisi un sinon on lui redemande.

L'un des enjeux au niveau de la sécurité était de restreindre l'accès au menu Administrateur et contrôleur aux personnes autorisées. c'est pour cela qu'on a deux fonctions qui demandent à l'utilisateur un identifiant et un mot de passe puis les compare à ceux qu'on a sauvés dans les fichiers `.txt`.

l'une des difficultés était de savoir quel fichier ouvrir pour vérifier le mot de passe.

Pour cela j'ai tenu compte de la première lettre de l'identifiant 'a' correspond au contrôleur 1, b au second et c au dernier. Comme dit plus haut, cela est un des inconvénients d'avoir choisi un fichier par contrôleur. Pour l'administrateur, c'est une fonction à part.

Enfin l'une des contraintes du projet est d'avoir un mot de passe crypté.

Ceci a amené pas mal de complications, car souhaitant un système de cryptage d'assez bon niveau, j'ai passé pas mal de temps sur les forums les documentations de librairies de cryptage. Cependant, rien de concluant. J'ai voulu utiliser la fonction `base_crypt()` implémenter dans C hélas suite à des problèmes avec MinGW la librairie n'était pas reconnu. J'ai donc envisagé une librairie extérieur OpenSSL qui possède une bonne base d'outils de cryptage cependant la documentation très mal expliqué peu intuitive et apprendre l'utilisation d'une librairie pour seulement une fonction m'aurait pris énormément de temps pour peu de choses à la fin. J'ai donc décidé d'écrire ma propre fonction de cryptage. J'ai donc utilisé le principe de la clé de Vigenère pour crypter le mot de passe. Cependant j'ai agrémenté cette dernière en ajoutant un 'salt' avant et après le mot de passe. Cela à pour

but de salir le mot de passe. Je suis bien conscient que le niveau de protection est loin d'être optimal, mais suffisant dans le cadre de ce projet.

2.2.4 : JSON

le module gestionjson.h comporte les fonctions liées à la conversion de fichier.txt en fichier.json

Les fonctions voyageurJson() et trajetJson() permettent d'enregistrer en format JSON les informations concernant les voyageurs et les trains contenus sur le réseau.

Une fois convertit les informations se présentent de cette manière :

voyageur:		▼ 0:	
▼ 0:		▼ 0:	
numero client:	"A001"	numero train:	"A1"
nom:	"Gaudillat"	morceau de trajet n°1:	"Paris-Orleans"
prenom:	"Eliott"	morceau de trajet n°2:	"Orleans-Limoges"
Gare de depart:	"Toulouse"	morceau de trajet n°3:	"Limoges-Brive"
Gare arrivee:	"Nice"	morceau de trajet n°4:	"Brive-Montauban"
		morceau de trajet n°5:	"Montauban-Toulouse"
		▼ 1:	

Pour les voyageurs

Pour les trains.

J'ai rencontré des difficultés pour saisir comment écrire un fichier JSON. Les fonctions sont assez basiques elles demandent à être améliorées pour extraire des choses plus précises comme les données d'un seul voyageur alors que pour le moment on obtient tous les voyageurs en même temps.

De plus, on aurait peut-être dû s'intéresser au format JSON dès le début du projet, car on aurait gagné du temps à écrire tous nos fichiers en format JSON, mais cela aurait apporté une difficulté en plus pour l'initialisation et la sauvegarde du réseau.

2.3 - Loan

2.3.1 : Représenter le réseau



IMPORTANT : Nous avons décidé d'appeler "*Trajet*" les tronçons séparant chaque gare. Les trajets 01,02... définis dans le sujet sont appelés "*Itineraire*" et sont définis directement dans les trains.

Pour représenter le réseau dans son ensemble, j'ai utilisé plusieurs structures (implémenté de manière dynamique et déclaré en opaque) pour représenter chaque aspect du réseau routier :

- Reseau : struct s_reseau*
- Gare : struct s_gare*
- Trajet : struct s_trajet*
- Train : struct s_train*
- Place : struct s_place*

- Voyageur : struct s_voyageur*

La structure **Reseau** représente le point d'accès à l'ensemble des informations du programme. C'est grâce à cette variable que nous pouvons accéder à toutes les autres simplement en parcourant la mémoire selon le format que nous avons défini.

```
struct s_reseau{
    Gare head; //Indice de
    Gare tail;
    int size; //taille du r
    Train headTrain; //Indi
    Train tailTrain;
    int nbTrain; //nb de tr
    Voyageur headVoyageur;
    Voyageur tailVoyageur;
    int nbVoyageur; //nb de
};
```

Elle est composée de 3 listes chaînées : La première liste donne accès aux informations des gares et de leurs trajets, la seconde les informations des trains, de leurs places et des itinéraires, enfin la troisième indique une liste des voyageurs actuels de notre réseau avec leur itinéraire complet.

La structure **Gare** représente une gare en mémoire. C'est un élément de la première liste doublement chaînée du réseau. Elle donne accès aux informations d'une gare : son nom, le nombre de trajets de la gare et les trajets qui partent d'elles sous forme d'une seconde liste chaînée. On retrouve également, les liens vers la Gare suivante via la première liste du réseau.

```
struct s_gare{
    char nomGARE[30]; //Le nom
    int nbTrajet; //Le nombre
    Trajet headListeTrajet; //
    Trajet tailListeTrajet; //
    Gare next; //La gare suiva
    Gare previous; //La gare p
};
```

La structure **Trajet** symbolise le tronçon séparant deux gares, on y retrouve les informations suivantes : La gare d'arrivée et le temps de parcours du trajet. Également, le lien vers le trajet suivant de la gare s'il existe.

J'ai fait le choix de représenter très simplement un trajet en mémoire avec le strict minimum d'information, ceux qui, par la suite, va m'imposer un traitement un peu plus lourd. En effet, je ne précise pas dans le trajet, sa gare de départ. Nous n'en avons pas fondamentalement besoin, car chaque trajet est lié à sa gare de départ via la liste chaînée, donc on peut

```
struct s_trajet{
    int ponderation;
    Gare gArrive; //L
    Trajet next; //In
};
```

remonter à l'information assez simplement. Mais cela aurait pu faciliter le traitement par la suite des itinéraires. Ici une liste simplement chaînée suffit, car une gare n'a pas énormément de trajet à mettre en mémoire et on n'a pas besoin de parcourir les trajets dans tous les sens.

La Structure **Train** représente un train en mémoire : il possède son propre itinéraire, un tableau de 10 places et un numéro d'identification. Ce sont des éléments de la seconde liste chaînée du réseau.

La Structure **Place** nous permet de gérer l'ensemble des voyageurs assis sur une Place. Simple d'usage, les voyageurs assis sont contenus dans une liste chaînée. Chaque place est repérée par un identifiant !

Enfin la Structure **Voyageur**, permet de gérer un voyageur ou “*morceaux de voyageur*”, j’y reviendrai plus en détail dans la section [2.3.3](#). Un voyageur est constitué d’un identifiant, un nom et un prénom ainsi que son itinéraire. Ils sont présents dans la 3^e liste du réseau.

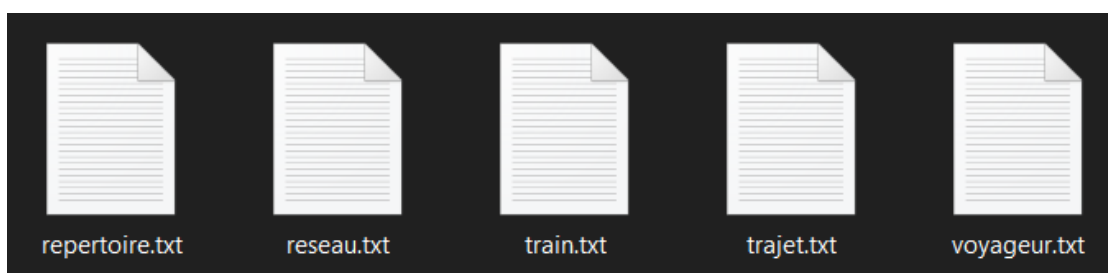
```
struct s_train {  
    char num[3]; //num d'  
    Itineraire chemin; //  
    Place place[10];  
    Train next; //le trai  
    Train previous; //le  
};
```

```
struct s_place {  
    Voyageur head;  
    Voyageur tail;  
    int nbVoyageur;  
    char numPlace[5];  
};
```

```
struct s_voyageur {  
    char id[5]; //le num  
    char nom[30]; //le no  
    char prenom[20]; //le  
    Itineraire voyage; //  
    Voyageur next;  
};
```

Pour permettre la **Sauvegarde** entre chaque utilisation, j’utilise un total de 5 fichiers.txt. J’aurai pu en utiliser seulement 1 ou 2, mais pour maintenir un code plus lisible et plus pratique à déboguer, j’ai choisi de répartir au maximum les informations.

Tous les noms, numéro, d’identification répondent à un format précis : Les trains sont numérotés par une lettre et un chiffre “A9”, les places de ce train reprennent ces deux caractères et y ajoutent les deux chiffres signifiant leur rang de “A901” à “A910”. Les voyageurs eux ont une lettre et 3 chiffres “A145”, permettant d’avoir un nombre de combinaisons possible très importante.



Repertoire.txt est un annuaire de tous les voyageurs présents dans le réseau, il contient leur nom, prénom, numéro d’identification, gare de départ et gare d’arrivée.

Reseau.txt est un annuaire des gares ! Il contient le nom de toutes les gares présente dans le réseau

Train.txt est un annuaire des trains, il contient leurs identifiants et leurs itinéraires.

Trajet.txt est un annuaire pas très lisible à première vue de l’ensemble des trajets du réseau. Ils sont écrits par ordre d’appartenance. Cet ordre est défini par l’ordre des gares dans reseau.txt. Les réseaux d’une même gare sont encadrés par des “/”.

Voyageur.txt regroupe l’ensemble des places de l’ensemble des trains ainsi que les voyageurs assis dans ces dernières. Nous verrons dans la partie 2.1.3, qu’en réalité ce sont des bouts de voyageurs qui sont indiqués dans ce fichier.

Au début du programme, la fonction *initReseau* est appelée et elle permet en lisant selon un format préétabli les informations dans ces fichiers qu'elle initialise par groupe à tour de rôle. Enfin elle nous renvoie uniquement l'adresse principale du réseau, dont nous pouvons récupérer l'ensemble des informations en parcourant la mémoire via d'autres fonctions présentées dans le 2.3.2.

Enfin, à la fermeture du programme, on appelle les fonctions *sauvReseau()* et *closeReseau()* fonctionnant ensemble, l'une écrit tout le réseau en mémoire en détruisant partiellement le réseau. La seconde vide complètement la mémoire allouée de ce dernier. La fonction *sauvReseau()*, fonctionne de manière diamétralement opposée à *initReseau()* en écrivant elle dans les fichiers en respectant le format de lecture. Format disponible en Annexe.

2.3.2 : Parcourir le réseau

Afin de parcourir le réseau, de stocker des itinéraires, permettre la réservation etc... j'ai décidé de centrer toute cela autour de la structure **Itineraire** :

```
struct s_itineraire{
    Gare depart; //gare
    Gare arrive; //gare
    int temps; //temps
    Trajet liste[30]; /
    int nbEtape;
};
```

Toutes données insinuant un déplacement est un itinéraire. Précédemment je vous ai présenté les structures **Trains** et **Voyageurs** qui possèdent toutes deux un attribut de type **Itineraire**. Pour les trains, cet itinéraire correspond au trajet du train (Les trajets définis du sujet), pour les voyageurs ceux-ci représentent leur déplacement.

Itineraire se compose d'un pointeur vers la gare de départ et celle d'arrivée, ainsi que le temps total de l'itinéraire, la liste des **Trajets** (tronçons) qui composent notre itinéraire et le nombre de Trajets présents dans la liste !

Plusieurs fonctions permettent alors de parcourir le graphe afin de construire un itinéraire soit demandé lors de la création d'un nouveau train, la modification d'un ancien, la réservation d'un client ou la modification d'une ancienne réservation. La “**super fonction**” qui articule tout ça est *rechercheItineraire*. Elle est construite sur le modèle de l'algorithme de Djiskra qui me semblait le plus “simple” et rapide à mettre en place : en effet, pas d'arcs négatifs et un graphe pas trop complexe. J'ai donc décidé de penser le réseau comme un graphe, et pour ce faire j'ai eu besoin de deux structures servant de base de travail pour notre fonction : **Sommet & Ensemble**.

```
struct s_sommet{
    Gare gare; //A quelle gare correspond le sommet
    Sommet pere; //la gare precedente
    int etat; //0 si jamais fait, 1 si deja fait,
    int distance; //distance a la gare de depart (-1) si infini
    Sommet next;
    Sommet previous;
};
```

```
struct s_ensemble{
    int nbSommets; //
    Sommet head; //so
    Sommet tail; //so
};
```

L'algorithme fonctionne de la manière suivante : on initialise un graphe autour d'un sommet père (ici la gare de départ) en initialisant ses premiers voisins avec la distance égale au temps de trajets entre le voisin et la gare de départ. L'attribut état du sommet contenant la gare de départ est passé à 1. Puis on fait une boucle, qui tourne tant que le sommet contenant la gare d'arrivée n'est pas créé et que son état vaut 0. La boucle peut s'arrêter seule avec un return NULL si elle n'arrive pas à parvenir jusqu'à la gare arrivée (si la gare fait partie d'un autre composant connexe suite à une suppression de trajet).

Quand on passe un sommet, on regarde tous ces voisins : Si le voisin est déjà en mémoire on regarde la distance qui le sépare à la gare de départ, si elle est plus petite que celle actuellement en mémoire, on la remplace et on change le père de ce voisin par le sommet que l'on regarde. Si le voisin n'était pas en mémoire, on l'initialise par défaut avec un état 0. Une fois tous les voisins testés, on passe le sommet à 1.

Une fois cette grande boucle finit on récupère alors le dernier sommet (qui est celui de la gare arrivée), et on remonte petit à petit, à l'envers le chemin via l'attribut père qui ramène au sommet précédent, et ceux jusqu'à retourner au sommet avec la gare Gdep. On a alors une liste de gare à l'envers et le nombre d'étapes à faire. On a plus qu'à construire notre liste de Trajet et le tour est joué ! On libère à la fin, la mémoire prise par les deux structures, pour conserver uniquement l'itinéraire ainsi construit.

Toutes les autres fonctions qui parcourent : recherche d'une gare/trajet/train dans le réseau, (par exemple) utilisent des boucles for afin de passer pour chaque gare, pour tous les trajets. Cela évite à chaque fois de mettre en place tout un graphe complexe pour simplement retourner une gare.

J'ai également réalisé une multitude de fonctions présentes dans le fichier ***parcoursGraphe.c*** qui me permettent de changer manuellement les itinéraires lors de petite modification ou d'ajout. Comme on utilise une liste, je manipule les rangs et le nombre total d'éléments pour par exemple retirer, un trajet en milieu d'un itinéraire. Vous pouvez retrouver individuellement toutes ces fonctions en **Annexes**.

2.3.3 : Manipuler le réseau

Nous avons notre réseau et toutes ces données implémentés en mémoire et également, des fonctions permettant de parcourir ce réseau... Il ne manque plus que les fonctions de manipulations.

La quasi-totalité des données du réseau sont manipulables et donc modifiables. Cela va d'un trajet à la réservation d'un client.

La majeure partie des données sont représentées dans des listes chaînées ou doublement chaînées. Pour retirer un élément, on raccorde simplement les liens et libérant la mémoire occupée par la donnée ainsi décrochée. Voici quelques précisions :

- **Les Trajets :**

Il est possible d'ajouter ou supprimer un trajet, en respectant quelques contraintes : il ne peut y avoir qu'un seul trajet entre deux gares, un trajet est forcément aller-retour. L'ajout est géré par la fonction *ajouterUnTrajet()*, qui rajoute simplement en mémoire, un nouveau trajet, à la queue de la liste de trajet de la Gare. On le fait également, pour la gare d'arrivée vers le départ. *retirerUnTrajet()* est plus complexe, en effet on doit potentiellement modifier les trajets de trains passant habituellement par le trajet que l'on veut supprimer.



IMPORTANT : Le trajet ou la Gare ne pourront être supprimé si un train passe par l'un d'eux et qu'il y a des passagers dedans ! il en est de même pour le train, s'il y a des passagers dedans.

Pour raccorder potentiellement le trajet du train, on manipule la liste de trajet de son itinéraire. On détecte à quelle position est situé la gare ou le trajet à retirer : si son rang est inférieur à la moitié du nombre total de trajets on garde la seconde partie de l'itinéraire et on décale donc tous les trajets de tels sorte que le rang 0 correspondent au trajet suivant de celui qui est supprimé, on change également la gare de départ ; si le rang est supérieur à la moitié, nous avons juste à changer la gare d'arrivée, en y mettant désormais la gare de départ du trajet que l'on supprime. Dans tous les cas, on réduit ensuite le nombre d'étapes. La fonction qui effectue cette modification s'appelle *modifItineraireTrainTrajet()*. Petite particularité, en réalité la fonction *retirerUnTrajet()* appelle la fonction *suppTrajetDansTrain()* qui supprime le train si elle possède moins de 3 étapes et sinon appelle *modifItineraireTrainTrajet()*.



IMPORTANT : L'ajout d'un trajet ne crée pas un Train qui passe par ce dernier. Il est nécessaire pour l'administrateur, d'ajouter manuellement, le train dans le sens qu'il veut.

- Les Gares :

Cela fonctionne sur le même principe que les Trajets, sauf que lors de la suppression d'une gare en mémoire on supprime également tous les trajets où cette gare est soit le départ, soit l'arrivée. On effectue également, une modification des trajets de trains contenant la gare en question, sur le même principe que précédemment. La fonction *retirerUneGare()* appelle alors *suppGareDansTrain()* qui supprime, cette fois-ci, les trains avec moins de 4 étapes, et sinon appelle *modifItineraireTrain()*.

- Les Trains :

L'ajout d'un train en mémoire est un peu différent du principe précédent. En réalité, c'est comme un PUZZLE, le train a besoin d'une gare de départ et d'arrivée, et d'un trajet entre. Une fois ces informations tapées, on boucle la question autant de fois que l'utilisateur souhaite ajouter des trajets à l'itinéraire du train. Et à chaque nouvelle gare arrivée, on insère le trajet reliant l'ancienne à la nouvelle à la suite du premier dans la liste de l'itinéraire. Le train est ensuite rajouté en queue de la liste doublement chaînée de train du réseau. Pour supprimer un train, il y a toujours la contrainte de ne pas avoir de voyageurs dessus, sinon cela consiste juste à retirer le train de la liste en raccordant la liste

correctement, et de libérer la mémoire occupée par le train. Il est également possible de modifier le trajet d'un train de deux façons : Ajouter des trajets à la fin (le même principe que lors de la création), ou raccourcir le Trajet. Pour ce dernier, on affiche alors l'ensemble des gares par lesquelles passe le Train et l'utilisateur choisit une des gares qui devient alors la gare d'arrivée et on récupère le rang de cette dernière pour réduire le nombre d'étapes de la liste de Trajets.



IMPORTANT : Un train ajouté n'est pas aller-retour. Vous pouvez soit créer un second train (conseillé), soit bouclé les trajets de votre propre train.

- La Réservation & les Voyageurs :


C'est probablement, les deux données demandant la gestion la plus complexe de l'ensemble du projet. La réservation récupère donc l'itinéraire demandé (issu de la fonction de recherche décrite dans le 3.2.2) pour créer un voyageur.

L'utilisateur saisit donc le nom et le prénom du voyageur, son id est tiré via la fonction *tirerNumVoyageur()* qui compte le nombre de voyageurs en mémoire et rajoute 1 pour donner l'identifiant via une conversion avec la fonction *atoi()* (Ps : si l'identifiant existe déjà, elle boucle sur elle-même pour avoir le double comme numéro, etc...). On vérifie bien que pour chaque trajet de l'itinéraire il y a un train qui est associé avec *rechercheTrainCorres()* ! Si ce n'est pas le cas, on renvoie une erreur. On créait alors deux listes, qui vont nous permettre de stocker en mémoire les changements de trains et les gares où il y a correspondance. Pour chaque changement de train et donc de place, on créait en fait, un bout de voyageurs ! Avec les mêmes données (nom, prénom, id) mais l'itinéraire correspond uniquement aux trajets où il restera dans le même train. C'est ce bout de voyageur qui sera ajouté dans la liste chaînée des places via *mettreSurUnePlace()*. Le voyageur de base avec l'itinéraire complet est quant à lui ajouté dans la liste générale des voyageurs (directement relié à réseau). Pour éviter une redondance d'information, l'itinéraire entier n'est pas sauvegardé, mais uniquement, la gare de départ et d'arrivée.

La fonction *rechercheVoyageur()*, nous permet donc de recomposer les informations d'un seul voyageur en recherchant à la fois dans les places et dans le répertoire (la liste de base). J'ai fait cela par souci de "simplicité" d'utilisation. En effet, même si cette implantation est un peu plus complexe et coûteuse, on gagne du temps, lors du remplissage des places.

Ce dernier fonctionne sur le principe suivant : on essaiera toujours de placer un voyageur sur une place déjà occupée. On va donc regarder à chaque place occupée s'il est possible d'emboîter le nouveau voyageur par rapport à l'ancien. Si ce n'est pas le cas, on continue à regarder la place suivante. Par défaut, le voyageur est placé à la dernière place vacante.

Enfin, la modification de la réservation est un peu plus simple grâce à cette implantation, on récupère les données de base du voyageur depuis le répertoire, et il a deux possibilités soit supprimer totalement sa réservation avec *suppVoyageur()* qui libère les places et supprime du répertoire. Ou il peut simplement changer sa gare d'arrivée, cela supprime en réalité sa



réserveation en conservant ses données et cela recrée un itinéraire entre ces deux gares et on réimplémente ces nouvelles données. À vrai dire, la modification n'est qu'une suppression pour recréer par-dessus. On pourrait probablement alléger ceci, mais par manque de temps, je me suis contenté de ceci.

3 - Mode d'emploi du Programme



IMPORTANT : Il ne faut pas mettre de ' : ' dans les entrées claviers. En effet, ces caractères nous servent pour délimiter les informations dans les fichiers de sauvegarde.

```
#####
#      BIENVENUE SUR LE RESEAU :      #
#      CUPGE EXPRESS                   #
#      pour naviguer dans les menus    #
#      il faut saisir le chiffre correspondant #
#####
#                                     #
#      1:ADMINISTRATEUR                #
#                                     #
#      2:CONTROLEUR                   #
#                                     #
#      3:CLIENT                       #
#                                     #
#      4:QUITTER                      #
#                                     #
#      choisissez un menu              #
#####
```

Le programme se présente comme ceci lors de l'exécution. C'est le menu principal du logiciel. À partir de lui on peut quitter l'application proprement et naviguer dans les 3 grands menus : Administrateurs, Contrôleurs et clients.

Pour exécuter le programme, il faut être soit sur une console de commande à l'emplacement du dossier et exécuter l'exécutable prog.exe. Soit en double cliquant directement sur le fichier prog.exe

Si vous souhaitez recompiler le programme il faut exécuter la commande suivante avec gcc:
gcc src/* -o prog -I include -L lib -lmingw32



IMPORTANT : Si vous quittez l'application d'une autre manière que via ce menu (Crash, fermeture Console...), vos changements ne seront pas sauvegardés ! Il est donc important de potentiellement fermer le programme entre chaque phase d'utilisation différente ! Cependant, les changements sont faits en mémoire, si vous ajoutez une gare en mémoire vous pouvez directement lui ajouter des trajets, pas besoin de quitter pour que cela soit possible.

3.1 - Le menu des admins

On accède au menu des administrateurs après avoir saisi ' 1 ' dans le menu principal avant d'avoir accès au menu admin on a une vérification du droit d'accès dont voici les codes :

- **identifiant** : jesuisadmin
- **mot de passe** : cryptolol

après vérification on arrive sur le menu administrateur :

Toute autre saisie affiche un message d'erreur et redemande une saisie valide.

```
#####
# Bienvenue sur le menu ADMINISTRATEUR #
#                                     #
# 1: GESTION DE TRAJET               #
#                                     #
# 2: EXPORTATION DONNEES              #
#                                     #
# 3: ADMINISTRATION                  #
#                                     #
# 4: RETOUR MENU PRINCIPAL            #
#                                     #
# choisissez un menu                 #
#####
```

3.1.1 - Le menu de Gestion des Trajets

```
#####
# Que voulez vous faire ?           #
#                                     #
# 1- Ajouter une Gare                #
# 2- Supprimer une Gare               #
# 3- Ajouter un trajet aller-retour  #
# 4- Supprimer un trajet aller-retour #
# 5- Ajouter un Train                #
# 6- Supprimer un Train               #
# 7- Modifier un Train               #
# 8- RETOUR                          #
#####
```

Pour lancer une action, tapez le numéro adéquat !

Ajouter Une Gare :

Pour ajouter une gare, il vous suffit de taper le nom de la Gare que vous voulez ajouter. **Elle sera ajoutée mais ne possèdera pas de Trajets.**

Supprimer une Gare :

Lors du lancement de l'action, le programme vous affichera l'ensemble des gares existantes

```
#####
# Voici la liste des Gares           #
#                                     #
# Paris | Orleans | Limoges | Lyon | Brive | Tours | Bordeaux | Agen | Montauban | Toulouse | Narbonne | Montpellier | Mar
# seille | Nice | Biarritz |
#####
```

vous n'avez qu'à taper le nom de la gare à supprimer. Si vous tapez une gare qui n'existe pas, on vous renvoie au menu de gestion de Trajet. **La gare ne pourra pas être supprimée si elle fait partie de l'itinéraire d'un Train et qu'il y a des passagers sur ce dernier.**

Ajouter un trajet

```
#####
#      Voici la liste des Gares      #
#####

Paris | Orleans | Limoges | Lyon | Brive | Tours | Bordeaux
seille | Nice | Biarritz |

#####
#      Tapez le nom de la Gare de Depart      #
#####

Paris
#####
#      Tapez le nom de la Gare d'Arrivee      #
#####

Biarritz

#####
#      Combien de temps de trajet ? (en min)      #
#####

150
```

En cas de saisie erronée ou si le trajet existe déjà, vous êtes renvoyé au menu précédent ! **Le trajet est ajouté aller-retour et AUCUN TRAIN NE PARCOURT LE TRAJET PAR DÉFINITION, vous devez soit ajouter un train, soit modifier un existant pour pouvoir permettre aux clients de l'utiliser.**

Supprimer un Trajet

```
#####

Paris | Orleans | Limoges | Lyon | Brive | Tours | B
seille | Nice | Biarritz |

#####
#      Tapez le nom de la Gare de Depart      #
#####

Paris
#####
#      Voici la liste des Trajets de la Gare      #
#####
Paris-Orleans en 60 minutes
Paris-Lyon en 90 minutes
Paris-Tours en 60 minutes
Paris-Biarritz en 150 minutes

#####
#      Tapez le nom de la Gare d'arrivee      #
#####

Biarritz

#####
#      Le trajet a bien ete Supprime      #
#####
```

En cas de saisie erronée, vous êtes ramené au menu précédent. **Le trajet ne sera pas supprimé s'il est parcouru par un train avec des passagers !**

Ajouter Un Train :

```
#####
#      Voici une liste de Trajets      #
#      non parcouru par un train      #
#####

Le trajet Limoges-Tours
Le trajet Tours-Limoges
```

Lors du lancement de l'action, on vous présente les trajets parcourus par aucun train. Il serait utile de créer un train passant par eux. **Le train que vous ajoutez n'est pas aller-retour !** Vous pouvez soit boucler votre train, soit créer un second train faisant le chemin inverse (conseillé).

Après avoir indiqué la gare de départ, vous allez pouvoir ajouter autant de trajets que vous voulez. Une fois que vous avez fini, tapez 2 pour être renvoyé au menu précédent.

```
#####
#      Tapez le nom de la Gare de Depart      #
#####

Paris

#####
#      Voici la liste des gares disponible      #
#####

Paris-Orleans en 60 minutes
Paris-Lyon en 90 minutes
Paris-Tours en 60 minutes

#####
#      Tapez le nom de la Gare Suivante      #
#####

Tours

#####
#      Ajoutez un Trajet supplementaire ?      #
#      1- OUI      #
#      2- NON      #
#####
```

```
#####
Ajoutez un Trajet supplementaire ?      #
1- OUI      #
2- NON      #
#####

in que vous venez de creer : A9
```

Une fois fini, le programme vous indique le numéro d'identification du train que vous venez de créer.

Supprimer un Train :

```
#####
#           Indiquez le numero du           #
#           train que vous voulez supp       #
#####
A9
Le train A9 a bien etait supprime
```

Pour supprimer un train, vous avez juste à indiquer son numéro d'identification ! **Il sera supprimé s'il n'y a pas de passager dedans !**

Modifier un Train :

```
#####
#           Indiquez le numero du           #
#           train que vous voulez modifier   #
#####
A8
#####
#           Comment voulez vous modifier ?   #
#           1- Ajouter des Trajets a la fin   #
#           2- Raccourcir le Trajet           #
#####
```

Une fois que vous lancez l'action et indiquez le numéro du train à modifier, vous avez le choix entre deux modes : ajouter des trajets à la fin ou raccourcir le trajet.

Pour ajouter des trajets à la fin, c'est la même chose que lors de la création d'un train.

Pour raccourcir le train, vous avez à choisir la gare d'arrivée parmi celle affichée. En cas d'erreur, vous êtes renvoyé au menu précédent.

```
#####
#  Voici la liste des etapes que vous faites  #
#####
Biarritz | Bordeaux | Tours | Paris
#####
#           Ou vous voulez vous arreter ?     #
#####
Tours
#####
#           La gare d'arivee a bien etait changee   #
#####
```

3.1.2 - L'exportation

```
#####
#      choisissez donnees a exporter      #
#                                         #
#      1- Voyageurs                       #
#      2- Trains                         #
#      3- RETOUR                         #
#####
```

Pour l'exportation vous avez juste à choisir 1 ou 2 pour convertir les données des voyageurs ou des trains en fichier JSON



IMPORTANT : Si vous avez ajouté des trains et/ou des voyageurs pendant votre utilisation il faudra fermer le programme puis le relancer afin d'avoir les fichiers à jour. Sinon vous exportez les données sans vos modifications

3.1.3 - L'Administration

```
#####
# De qui voulez vous modifier les informations #
#                                         #
#      1- controleur 1                   #
#      2- controleur 2                   #
#      3- controleur 3                   #
#      4- administrateur                 #
#      5- RETOUR                         #
#####
```

Le menu administration vous demandera de saisir le numéro correspondant à la personne dont vous souhaitez modifier les données.
une fois cette première saisie effectuée le programme vous affichera le message ci-dessous

```
#####
      voici vos donnees personnelles
actrlad
*T\C_jVw\?eY7Rf0J,►

Philippe
Truillet
#####

#####
#          Que voulez vous faire ?          #
#                                           #
#      1- Modifier le mot de passe         #
#      2- Modifier prenom                  #
#      3- Modifier nom                    #
#      4- RETOUR                          #
#####
```

Il contient les données de la personne sélectionnée. et Vous devrez saisir le chiffre correspondant à l'information que vous souhaitez modifier(1-mot de passe , 2-prenom, 3-nom).

Ensuite les 3 choix se présentent de la même façon

```
#####
#veuillez saisir votre nouveau mot de passe#
#          19 caracteres maximum          #
#####

upssitech
#####
#          Modification enregistree        #
#####
```

Vous saisissez le nouveau mot de passe, prénom ou nom et le programme se charge de faire la modification.

Après cela vous revenez au menu du choix de la donnée à modifier si vous souhaitez modifier autre chose.

3.2 - Le menu des contrôleurs

On accède au menu des contrôleurs après avoir saisi ' 2 ' dans le menu principal avant d'avoir accès au menu contrôleur on a une vérification du droit d'accès dont voici les codes :

- **Contrôleur 1:** - identifiant : actrlad
 - mot de passe : upssitech
- **Contrôleur 2:** - identifiant : bctrlqd
 - mot de passe : java>c
- **Contrôleur 3:** - identifiant : cctrlsd
 - mot de passe : s3Merci

après vérification on arrive sur le menu contrôleur :

```
#####
#                               #
#   Bienvenue sur le menu CONTROLEUR   #
#                               #
#       1: RECHERCHE                   #
#       2: ADMINISTRATION               #
#       3: RETOUR MENU PRINCIPAL       #
#                               #
#   choisissez un menu                 #
#                               #
#####
```

saisir '1' amène au menu de recherche il permet de vérifier le trajet d'un client en saisissant son numéro de client

saisir '2' amène au menu d'administration qui permet de modifier ses données personnelles tel que :

- mot de passe
- prénom
- nom

Nous avons une contrainte de 19 caractères pour les différentes données.

saisir '3' ramène au menu principal

Toute autre saisie affiche un message d'erreur et redemande une saisie valide.

3.2.1 - La Recherche Contrôleur

```
#####
#                               #
#   Indiquez le numero client         #
#   que vous recherchez               #
#                               #
#####
A001
#####
LE VOYAGEUR Eliott Gaudillat AU NUMERO A001 :
il part de Toulouse et arrivera a Nice

Prend le train A2, il sera assis
a la place numero A210 de Toulouse a Nice
#####
#                               #
#   1- Recherche un autre client     #
#   2- RETOUR                         #
#                               #
#####
```


Indiquez le numéro du client et le programme vous indique les informations de voyage de ce client. Vous pouvez relancer une recherche en tapant 1 ou revenir en arrière en tapant 2.

3.2.2 - L'Administration Contrôleur

Comme l'administration pour l'administrateur

le programme vous affiche vos données personnelles et vous demande la donnée à modifier taper 1 pour le mot de passe, 2 pour le prénom , 3 pour le nom et 4 pour revenir au menu précédent

```
#####
#veuillez saisir votre nouveau mot de passe#
#          19 caracteres maximum          #
#####
upssitech
#####
#          Modification enregistree        #
#####
```

Ensuite les 3 choix de modifications sont semblables. On vous demande de saisir votre

```
#####
#          voici vos donnees personnelles  #
actrlad
#T\C_jVw\?eY7Rf0J,►
Philippe
Truillet
#####
#####
#          Que voulez vous faire ?        #
#                                          #
#          1- Modifier le mot de passe    #
#          2- Modifier prenom              #
#          3- Modifier nom                #
#          4- RETOUR                      #
#####
```

nouveau mot de passe,nom ou prénom et le programme ce charge de le modifier dans le fichier correspondant. et vous informe du bon déroulement de la procédure.

Une fois fait vous revenez au menu du choix de modification si vous souhaitez modifier quelque chose d'autre

3.3 - Le menu des clients

On accède au menu des contrôleurs après avoir saisi ' 3 ' dans le menu principal:

```
#####
#                               #
#   Bienvenue sur le menu CLIENT   #
#                               #
#   1: RECHERCHE                   #
#                               #
#   2: RESERVATION                 #
#                               #
#   3: MODIFICATION               #
#                               #
#   4: RETOUR MENU PRINCIPAL      #
#                               #
#   choisissez un menu           #
#                               #
#####
```

saisir '1' amène au menu recherche d'un trajet il permet au client de trouver les informations sur un trajet entre 2 gares

saisir ' 2 ' amène au menu de réservation il permet au client de réserver une place sur un trajet en renseignant sa gare de départ celle d'arrivée son nom et son prénom. Le client obtient un numéro de réservation.

saisir '3' amène au menu de modification qui permet au client avec son numéro de réservation de modifier son trajet.

saisir '4' ramène au menu principal

Toute autre saisie affiche un message d'erreur et redemande une saisie valide.

3.3.1 - La Recherche Client

```
#####
#   Tapez le nom de la Gare de Depart   #
######
Marseille
#####
#   Tapez le nom de la Gare d'Arrivee   #
######
Biarritz
#####
Depart de la gare Marseille via le train A7 direction Lyon
Correspondance a la gare Paris avec le train A4 direction Tours
Arrive a la gare Biarritz en 390 minutes
#####
```

Indiquez la gare de départ, ainsi que celle d'arrivée. Le programme vous indique le trajet le plus court entre ces deux, avec les potentielles correspondances. Ainsi que le temps de trajet !

3.3.2 - La Réservation

Via la réservation vous entrez 4 informations : Gare de départ, d'arrivée, votre nom et votre prénom. Le programme vous renvoie votre numéro de passagers.

```
#####
#      Tapez le nom de la Gare de Depart      #
#####
Lyon

#####
#      Tapez le nom de la Gare d'Arrivee      #
#####
Agen

#####
#      Indiquez votre Nom                      #
#####
BERNAT

#####
#      Indiquez votre Prenom                   #
#####
Loan

Voici votre identifiant voyageur : A001
```

3.3.3 - La Modification

```
#####
#      Indiquez votre numero client           #
#####
A001
LE VOYAGEUR Loan BERNAT AU NUMERO A001 :
il part de Lyon et arrivera a Agen

Prend le train A2, il sera assis
a la place numero A210 de Bordeaux a Agen
Prend le train A4, il sera assis
a la place numero A410 de Paris a Bordeaux
Prend le train A7, il sera assis
a la place numero A710 de Lyon a Paris

#####
#      1- Modifier                           #
#      2- RETOUR                             #
#####
```

En entrant votre numéro de passager, le programme vous montre votre itinéraire. Vous pouvez le modifier en tapant 1. Vous pouvez alors soit annuler votre réservation, soit modifier votre gare d'arrivée.

```
#####  
#      Quel modif voulez vous faire ?      #  
#      1- Changer la gare d'arrivee         #  
#      2- Annuler la reservation            #  
#####
```



IMPORTANT : En cas de crash, ou d'erreur de lancement, vous avez à votre disposition un dossier restore contenant les fichiers de sauvegarde vierge pour reboot le programme et ses données !

4 - Annexe

4.1 - Notre Avis sur le Projet & Améliorations

4.1.1 : Avis

Nous avons tous les deux trouvé le projet très intéressant à réaliser. Il nous a permis de progresser dans notre manière de coder, mais aussi de pratiquer régulièrement et hors du cadre scolaire (TP) de manière autonome.

De plus, cela permet de développer plusieurs compétences comme travailler en équipe, gérer un délai.

4.1.2 : Les Difficultés

Avis Loan

Finalement le projet s'est avéré plus complexe que ce que je pensais. En effet, j'ai essayé de prévoir la quasi-totalité des possibilités que l'utilisateur pouvait entrer mais, il peut toujours exister une faille qui m'aurait échappé à travers toutes mes fonctions de manipulation. La première difficulté rencontrée était pour les includes ! En effet, certaines fonctions utilisaient des structures définies dans d'autres .c, j'ai donc dû faire des accesseurs, et par moment certaines fonctions de manipulation simple (par exemple : ajouter 1 au nb de train), me demandait de définir hors de train.c, cette fonction pour qu'elle puisse utiliser la structure Réseau sans problème. J'ai donc dû "tricher" un peu, en définissant des fonctions dans des .c et leurs signatures dans un autre .h ! La seconde a été de garder à l'esprit la méthode de programmation. Heureusement, elle a peu évolué, mais j'ai dû, en cours de route, reprendre des anciennes fonctions qui par exemple ne remettait pas à NULL un pointeur après leur utilisation etc... Dans l'ensemble, cela a été un travail de précision en grande quantité. J'ai dû je pense consacrer au moins 30h dans le projet dont au moins 5 h sur papier pour réfléchir et anticiper les problèmes. J'ai dû poser quelques contraintes pour simplifier l'écriture du code : la recherche ne permet pas de choisir entre plusieurs itinéraires, 20 caractères max pour les noms, prénoms, gares ou encore le fait que l'on peut insérer des chiffres, symboles dans les noms, prénoms et gares.

Pour ma part, comme évoqué plus tôt dans ce rapport, la difficulté se trouve dans la partie de cryptage et de JSON. En effet, la documentation et la recherche d'information prennent pas mal de temps lorsqu'on débute de par l'anglais qui aide pas tout le temps à la compréhension. Et de part le manque de clarté de certaines documentations.

Cela fait prendre conscience aussi de l'utilité de lier par moment plusieurs langages de programmation. Hors ici nous devons utiliser uniquement du C ce qui nous empêche de contourner certains de ses inconvénients.

Ceci me fait rebondir sur une autre difficulté rencontrée liées à ces inconvénients. le fait qu'on ne puisse pas définir un tableau de char et plus loin l'initialiser avec une chaîne directement. Cela à été contraignant pour manipuler les noms de fichier en fonctions des différents contrôleurs

Je pense qu'on pourrait chiffrer ce travail autour de 25h de travail pur. Sans compter le temps de réunion etc.

4.1.3 : Améliorations

Notre programme est loin d'être parfait et voici une liste d'améliorations que l'on pourrait potentiellement ajouté au programme avec des délais supplémentaires :

- Rendre le code plus lisible/propre
- Supprimer les duplications de codes
- Faire une interface graphique
- Pouvoir proposer plusieurs trajets aux clients lors de leurs recherches
- Rendre l'aspect des contrôleurs plus dynamiques
- Mettre une petite musique de fond
- Mettre en place des horaires de trains
- Limiter tentative connexion zone sécurisée.
- améliorer la partie sur l'exportation JSON
- lors de l'affichage des données d'un contrôleur cacher le mot de passe avec des * même si ce dernier est crypté.

4.2 - La C-Doc du projet

Joint dans le ZIP

4.3 - Liens & Droits & Source

Lien du GitHub : <https://github.com/loanBRNT/TCHOU-TCHOU>

Les images exposées dans ce document proviennent de notre création personnelle.