

Rapport de projet:

Application client/Serveur

TCP/IP

Mis en œuvre par :

Mathieu Clusel

Jonathan Monbeig

Florian Pignard

Louis Reynaud

Sommaire

Table des matières :

Introduction	3
Le client	4
Présentation	4
Fonctionnement	4
Arborescence	5
Le serveur	6
Présentation	6
Fonctionnement	6
Arborescence	7
Fonctionnement du socket TCP côté serveur	8
La gestion du fichier de données	8
Présentation	8
"Trains.txt"	8
"Trains.c"	9
Structure de donnée utilisée pour les trajets	9
Les réductions ou suppléments	9
Architecture de l'application	10
Arborescence de l'application	10
Makefile	10
Documentation	10
Manuel d'utilisation	11
Comment lancer le serveur et le client	11
Démarche dans l'application	11
Exemples fonctionnels	13
Exemple local (deux terminaux sur la même machine)	13
Sans horaires, sans tri:	13
Sans horaires, trié selon la durée du trajet:	14
Exemples distants (deux machines distinctes)	15
Par tranche horaire et trié selon le prix:	15
Par horaire de départ:	16
Conclusion	17

Introduction

Ce rapport concerne une application qui nous a été donné de faire durant notre cursus de L3 MAGE. Notre groupe est composé de 4 personnes, Louis REYNAUD, Florian Pignard, Mathieu CLUSEL et Jonathan MONBEIG. Il nous a été donné une semaine afin de réaliser une application réseau.

Le but de ce projet était la création, en une semaine, d'une application client/serveur TCP/IP complète et fonctionnelle. Le sujet de l'application était la consultation du client d'une liste de trajets de trains, contenue dans le serveur qui lui transmettra les trajets en fonction de données que lui donne le client, tel que la ville de départ, celle d'arrivée, et, si le client le souhaite, une heure de départ ou une tranche horaire pour cette heure.

Dans ce rapport, nous allons vous expliquer comment fonctionne chaque partie de l'application séparément, l'application en elle-même, et présenter quelques exemples fonctionnels.

I. Le client

A. Présentation

Le client est l'application textuelle utilisée par l'utilisateur pour communiquer avec le serveur et recevoir les informations au sujet de trajets de trains.

B. Fonctionnement

A son lancement, le client va tenter de se connecter au serveur, si la connexion est établie, un message du serveur de bienvenue apparaît, sinon un message d'erreur de connexion est affiché.

Une fois la connexion faite, l'utilisateur va interagir avec le client sur un terminal en rentrant des entiers (ex: 0, 1, 2, 3) correspondants aux choix qu'il souhaitera faire concernant les critères de sélection des trajets de trains ainsi que la ville de départ et la ville d'arrivée. Le client va ensuite créer un message avec les informations récoltées selon un protocole défini :

villeDépart;VilleArrivée;HeureDébut;HeureFin;ModeDeTri
ex : Valence;Grenoble;8:30;0;1

Si un critère n'est pas renseigné, on insère "0" à la place dans le message.
Le message est ensuite envoyé au serveur afin qu'il soit traité.

Une fois que l'utilisateur ne veut plus consulter de trains, il l'indique lorsque le client lui demande si il veut consulter de nouveaux trains par un "0", à ce moment, le client envoie au serveur le message "KILL" pour dire qu'il termine la connexion avec lui puis ferme le socket de connexion.

C. Arborescence

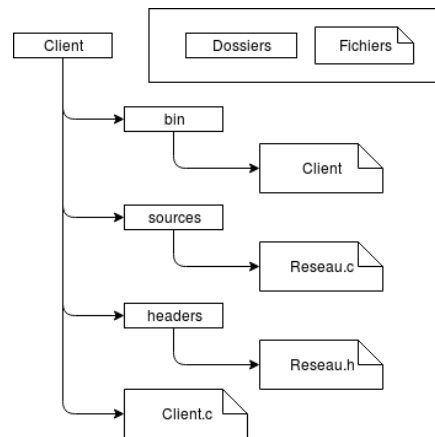


Image 1: Arborescence du client

Fonctionnement du socket TCP côté client.

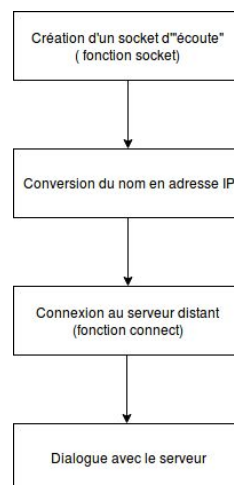


Image 2: schéma de fonctionnement du socket TCP côté client

II. Le serveur

A. Présentation

Le serveur est la “base de données” depuis laquelle le client recevra les trajets des trains. Il devra pouvoir supporter les requêtes de différents clients, et récupérer les données depuis un fichier nommé “Trains.txt” présent dans les sources du serveur. Il utilisera un socket TCP détaillé en C.

B. Fonctionnement

A son lancement, le serveur récupère tous les trajets de trains contenus dans le fichier Trains.txt : pour chaque trajet, il construit une nouvelle structure *trains* et l’ajoute dans un tableau de trains. Nous avons choisis d’utiliser un tableau de structure afin de pouvoir trier et manipuler plus facilement les trajets.

Le serveur attend une connexion d’un client. Dès qu’il établit une nouvelle connexion, il va créer un processus fils qui va s’occuper du nouveau client. Le fils va ensuite récupérer le message envoyé par le client, le décomposer pour traiter la demande de l’utilisateur et ensuite créer un nouveau message texte, formaté selon un protocole, contenant le résultat de la demande utilisateur.

Protocole du message : Numéro;VilleDébut;VilleArrivée;HeureDépart;HeureArrivée;Prix;\n

Exemple : 17459;Valence;Grenoble;6:10;7:10;19.00\n
17856;Valence;Grenoble;9:30;10:50;18.00\n
17945;Valence;Grenoble;14:40;16:00;19.00\n

Le fils envoie ensuite le message formaté au client puis attend un nouveau message de la part du client.

Si le message reçu est “KILL”, le fils se tue.

Nous avons décidé de n’envoyer qu’un seul message contenant tous les trajets correspondants à la demande afin de limiter les échanges entre le serveur et le client. Ceci nous impose donc une limite sur la quantité de données pouvant être lu sur le socket.

C. Arborescence

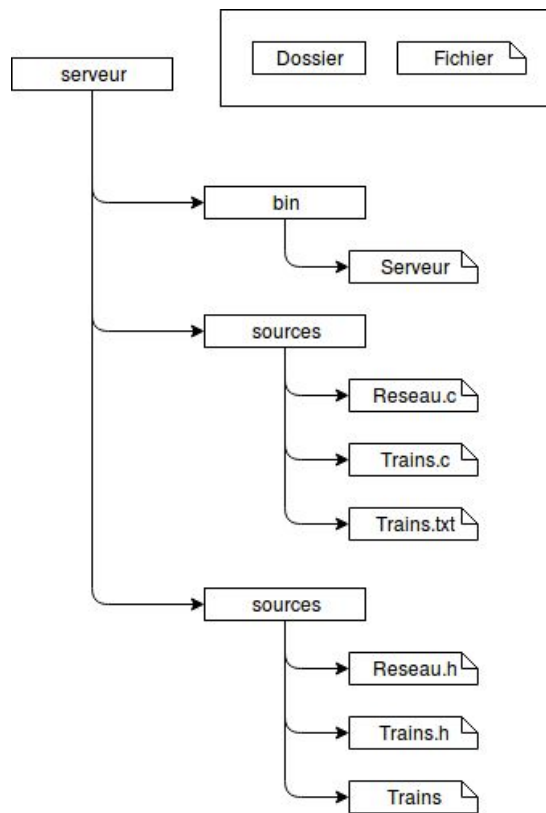


Image 3: Arborescence du client

D. Fonctionnement du socket TCP côté serveur

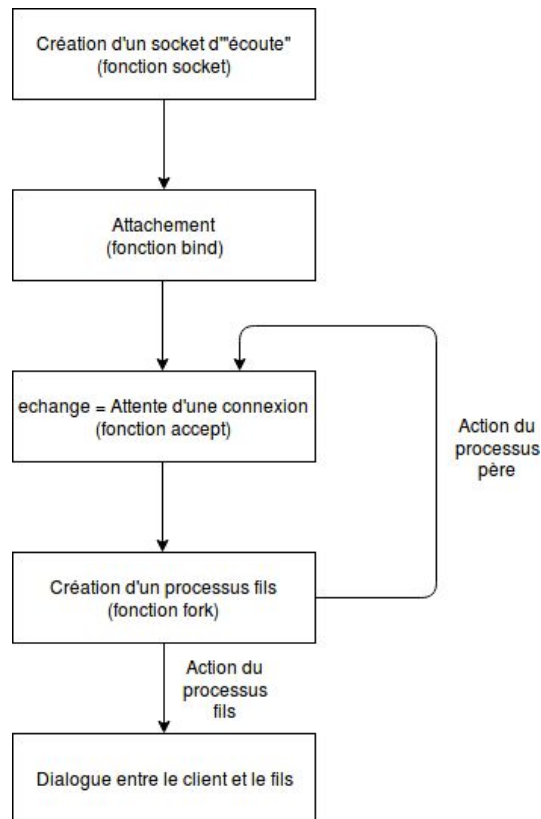


Image 4: schéma de fonctionnement du socket TCP côté serveur

III. La gestion du fichier de données

A. Présentation

L'application utilise des données depuis un fichier, "Trains.txt", qui est utilisé par plusieurs fonctions dans "Trains.c" et dans "Serveur.c". On peut constater que cette classe, "Trains.c", contenant les fonctions de gestion de données est utilisée par le serveur, cf. Image 2.

B. "Trains.txt"

Le fichier texte comporte plusieurs lignes, chaque ligne correspondant à un trajet que l'on admet récurrent. Chaque ligne est composée de plusieurs informations séparées par le caractère ";". Nous avons dans l'ordre le numéro du train, la gare de départ, celle d'arrivée, une heure de départ, une heure d'arrivée, le prix ordinaire du trajet, et un label indiquant si ledit prix est sujet à des modifications, une réduction (REDUC) ou un supplément (SUPPL).


```
17525;Grenoble;Valence;16:55;17:55;17.60;SUPPL  
17526;Grenoble;Valence;17:30;18:46;17.60;  
17528;Grenoble;Valence;18:30;19:45;17.60;REDUC
```

Image 5: Exemples de trajets

C. “Trains.c”

Le fichier contenant les fonctions permettant au serveur de parcourir les données du fichier texte présenté au-dessus est “Trains.c”, accompagné de son header “Trains.h”. Il contient des fonctions permettant au serveur de comparer les trajets, d’en sélectionner un certain nombre en fonction de données et d’appliquer des tries.

1) Structure de donnée utilisée pour les trajets

Pour structurer les données récupérées par le serveur depuis “Trains.txt”, nous avons opté pour un tableau de structures. Ces structures, nommées *Trains*, contiennent les champs correspondant à tous les éléments d’une ligne. Chaque objet de cette structure représente donc un trajet avec tous ses éléments. Ceci nous a permis de facilement pouvoir gérer l’ajout, le trie et la suppression de trains. Par contre, le fait d'utiliser un tableau nous a posé le problème de sa taille. Pour cela nous avons donc passé en paramètre la longueur du tableau, que l’on alloue dynamiquement via un *malloc* et le *free*.

2) Les réductions ou suppléments

Afin de gérer les labels situés en derniers éléments des lignes (REDUC ou SUPPL) du fichier de données, nous avons ajouté un attribut *evenement* à la structure *trains* qui contient 1 pour REDUC, 2 pour SUPPL et 3 pour rien afin de pouvoir facilement calculer le prix

IV. Architecture de l'application

A. Arborescence de l'application

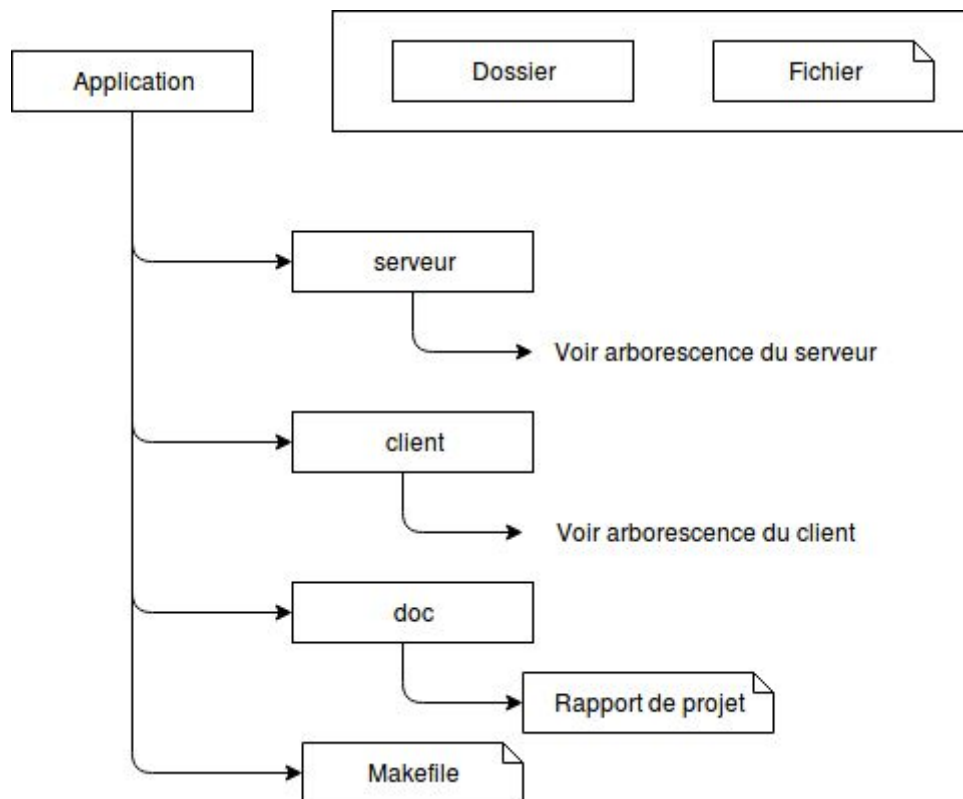


Image 6: Arborescence de l'application

B. Makefile

Nous avons créé un makefile général qui va chercher les différents éléments dont a besoin l'application pour fonctionner. Il crée un exécutable de serveur et de client dans le dossier bin de chaque côté de l'application.

C. Documentation

Nous avons utilisés doxygen afin de documenter le code. Une page est disponible dans `.../doc/html/index.html` afin de visualiser les commentaires et les différents éléments composant notre code (méthodes, structures de données, etc...).

V. Manuel d'utilisation

A. Comment lancer le serveur et le client

Il vous faudra au moins deux environnements différents. Soit plusieurs machines distinctes, soit des terminaux distincts sur la machine. Ou utiliser la boucle locale de sa machine.

Sur un terminal, nous allons faire tourner le serveur.

Voici un exemple de commande afin de lancer le serveur:

```
reynloui@ltsp23:~/Documents/UJFProjet_Reseau/Projet_Reseau$ serveur/bin/Serveur
1561 serveur/sources/Trains.txt
```

Image 7: Exemple de boot d'un serveur

Décomposons cette commande:

- "serveur/bin/Serveur" : Racine du fichier exécutable Serveur
- 1561 : N° de Port. Ce numéro, que vous choisirez, doit être supérieur à 1024 et devra être transmis à tout client souhaitant se connecter sur votre serveur.
- "serveur/sources/Trains.txt" : Fichier contenant les données des trajets de trains.

Ensuite, voyons un exemple de commande afin de lancer un client:

```
reynloui@ltsp23:~/Documents/UJFProjet_Reseau/Projet_Reseau$ client/bin/Client 127.0.0.1 1561
```

Image 8: Exemple de boot d'un client

Décomposons cette commande:

- "client/bin/Client: Racine du fichier exécutable Client
- 127.0.0.1: Adresse IP. 127.0.0.1 est l'adresse locale mais dans le cas d'une connexion distante entre deux machines, entrer l'adresse de la machine contenant le serveur.
- 1561: N° de Port. Numéro choisi par le créateur du serveur qui devra vous être fourni ainsi que l'adresse IP afin que vous vous connectiez au serveur.

B. Démarche dans l'application

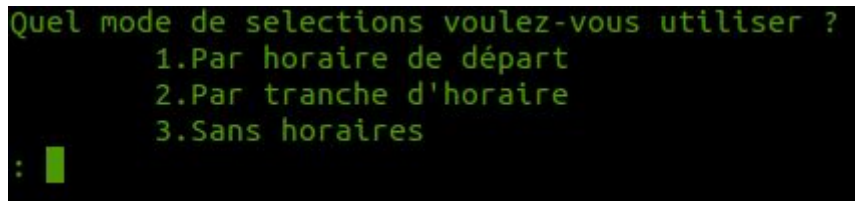
Une fois votre serveur lancé, effectuez la commande de boot client vue au-dessus. Vous vous retrouvez dans le client qui va dialoguer avec le serveur.

```
Bonjour, bienvenue sur le serveur.
Souhaitez-vous consulter les trains? (Oui = 1/ Non = 0) :
```

Image 9: premier contact avec le serveur.

En tapant "0", on quitte le serveur.

En tapant "1", on accède à ce dialogue:

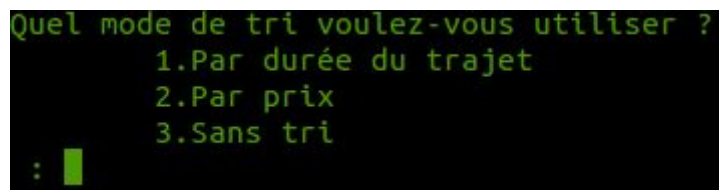


```
Quel mode de selections voulez-vous utiliser ?  
  1.Par horaire de depart  
  2.Par tranche d'heure  
  3.Sans horaires  
:  
█
```

Image 10: Choix des éléments à renseigner

- 1. Par horaire de départ: Il vous sera demandé une ville de départ, une ville d'arrivée, et un horaire de départ.
- 2. Par tranche horaire: Il vous sera demandé une ville de départ, une ville d'arrivée, et une tranche horaire (début et fin) pour l'horaire de départ.
- 3. Sans horaires: Il vous sera demandé une ville de départ et une ville d'arrivée.

Une fois avoir renseigné les différents éléments, le serveur vous demande:



```
Quel mode de tri voulez-vous utiliser ?  
  1.Par durée du trajet  
  2.Par prix  
  3.Sans tri  
:  
█
```

Image 11: choix du potentiel tri des données

Les intitulés sont explicites, soit on demande un tri par durée du trajet, du plus court au plus long, soit par prix, du moins cher au plus cher, soit sans tri.

Vous recevez alors le ou les trajets correspondant à vos critères, puis le serveur demande si vous voulez effectuer une nouvelle requête.

VI. Exemples fonctionnels

Voici quelques exemples fonctionnels du programme :

A. Exemple local (deux terminaux sur la même machine)

1) Sans horaires, sans tri:

```
Bonjour, bienvenue sur le serveur.

Souhaitez-vous consulter les trains? (Oui = 1/ Non = 0) : 1
Quel mode de selections voulez-vous utiliser ?
    1.Par horaire de départ
    2.Par tranche d'horaire
    3.Sans horaires
: 3
Saisissez une ville de départ : Valence
Saisissez une ville d'arrivée : Grenoble
Quel mode de tri voulez-vous utiliser ?
    1.Par durée du trajet
    2.Par prix
    3.Sans tri
: 3

*****
*****  RÉSULTAT DE LA RECHERCHE  *****
*****
*****      Numéro du train : 17564
*****      Ville de départ : Valence
*****      Ville d'arrivée : Grenoble
*****      Horaire de départ : 6:15
*****      Horaire d'arrivée : 7:31
*****      Prix : 17.60
***** -----
*****      Numéro du train : 17566
*****      Ville de départ : Valence
*****      Ville d'arrivée : Grenoble
*****      Horaire de départ : 6:45
*****      Horaire d'arrivée : 7:55
*****      Prix : 17.60
***** -----
*****      Numéro du train : 17568
*****      Ville de départ : Valence
*****      Ville d'arrivée : Grenoble
*****      Horaire de départ : 7:15
*****      Horaire d'arrivée : 8:32
*****      Prix : 17.60
***** -----
Souhaitez-vous consulter les trains? (Oui = 1/ Non = 0) : █
```

Image 12: Exemple d'une requête sans horaires et sans tri.

On constate ici que les données reçues ne sont pas triées, autre que par le numéro du train. Il s'agit de l'ordre dans lequel ces éléments sont présents dans le fichier "Trains.txt".

2) Sans horaires, trié selon la durée du trajet:

```

Souhaitez-vous consulter les trains? (Oui = 1/ Non = 0) : 1
Quel mode de selections voulez-vous utiliser ?
    1.Par horaire de départ
    2.Par tranche d'horaire
    3.Sans horaires
: 3
Saisissez une ville de départ : Paris Gare de Lyon
Saisissez une ville d'arrivée : Valence
Quel mode de tri voulez-vous utiliser ?
    1.Par durée du trajet
    2.Par prix
    3.Sans tri
: 1

*****
***** RÉSULTAT DE LA RECHERCHE *****
*****
*****
        Numéro du train : 9713
        Ville de départ : Paris Gare de Lyon
        Ville d'arrivée : Valence
        Horaire de départ : 10:07
        Horaire d'arrivée : 12:19
        Prix : 90.00
***** -----
        Numéro du train : 6035
        Ville de départ : Paris Gare de Lyon
        Ville d'arrivée : Valence
        Horaire de départ : 7:41
        Horaire d'arrivée : 10:11
        Prix : 113.00
***** -----
        Numéro du train : 6063
        Ville de départ : Paris Gare de Lyon
        Ville d'arrivée : Valence
        Horaire de départ : 10:07
        Horaire d'arrivée : 13:15
        Prix : 92.20
***** -----
Souhaitez-vous consulter les trains? (Oui = 1/ Non = 0) : █

```

Image 13: Exemple d'une requête sans horaires et tri sur la durée du trajet.

On constate ici que les trains ne sont plus triés par numéro de train (9713 → 6035 → 6063) mais par durée de trajet comme demandé (2:12 → 2:30 → 3:08).

B. Exemples distants (deux machines distinctes)

1) Par tranche horaire et trié selon le prix:

```

reynloui@ltsp23:~/Documents/UJFProjet_Reseau/Projet_Reseau$ client/bin/Client 193.48.34.21 1888
Bonjour, bienvenue sur le serveur.

Souhaitez-vous consulter les trains? (Oui = 1/ Non = 0) : 1
Quel mode de selections voulez-vous utiliser ?
    1.Par horaire de départ
    2.Par tranche d'horaire
    3.Sans horaires
: 2
Saisissez une ville de départ : Valence
Saisissez une ville d'arrivée : Paris Gare de Lyon
Saisissez un horaire de début : 15:00
Saisissez un horaire de fin : 17:00
Quel mode de tri voulez-vous utiliser ?
    1.Par durée du trajet
    2.Par prix
    3.Sans tri
: 2

*****
*****  RÉSULTAT DE LA RECHERCHE  *****
*****
*****      Numéro du train : 9862
*****      Ville de départ : Valence
*****      Ville d'arrivée : Paris Gare de Lyon
*****      Horaire de départ : 15:15
*****      Horaire d'arrivée : 17:49
*****      Prix : 87.60
*****      -----
*****      Numéro du train : 6194
*****      Ville de départ : Valence
*****      Ville d'arrivée : Paris Gare de Lyon
*****      Horaire de départ : 16:15
*****      Horaire d'arrivée : 19:15
*****      Prix : 92.00
*****      -----
*****
Souhaitez-vous consulter les trains? (Oui = 1/ Non = 0) : █

```

Image 14: Exemple d'une requête par tranche horaire et trié selon le prix.

En vérifiant sur le fichier Trains.txt :

```

9862;Valence;Paris Gare de Lyon;15:15;17:49;109.50;REDUC
6194;Valence;Paris Gare de Lyon;16:15;19:15;92.00;
6208;Valence;Paris Gare de Lyon;17:15;19:53;92.80;

```

On remarque que seuls les deux premiers trajets sont présents dans la réponse du serveur, car le dernier, partant à 17:15, dépasse la tranche horaire 15:00/17:00 définis dans les critères. De plus on constate que le label REDUC a bien été appliqué au trajet N° 9862, passant son prix de 109.50 à 87.60, devenant le moins cher trajet, il est alors présenté en premier.

2) Par horaire de départ:

```

Souhaitez-vous consulter les trains? (Oui = 1/ Non = 0) : 1
Quel mode de selections voulez-vous utiliser ?
    1.Par horaire de départ
    2.Par tranche d'heure
    3.Sans horaires
: 1
Saisissez une ville de départ : Grenoble
Saisissez une ville d'arrivée : Valence
Saisissez un horaire de départ : 16:50

*****
*****  RÉSULTAT DE LA RECHERCHE  *****
*****
*****      Numéro du train : 17525
*****      Ville de départ : Grenoble
*****      Ville d'arrivée : Valence
*****      Horaire de départ : 16:55
*****      Horaire d'arrivée : 17:55
*****      Prix : 17.60
*****  -----
Souhaitez-vous consulter les trains? (Oui = 1/ Non = 0) : 

```

Image 15: Exemple d'une requête par horaire de départ

On peut voir ici que dans le cas d'un horaire de départ précis, un seul trajet est donné, celui qui se rapproche le plus de l'horaire indiqué (ici 16:50 et le trajet part à 16:55). On constate également qu'il n'est pas proposé de trier la liste, étant donné qu'un seul trajet est donné.

Conclusion

L'application est fonctionnelle. Le serveur est capable de recevoir et traiter des requêtes de plusieurs clients différents. Les clients reçoivent leurs informations avec les critères qu'ils ont renseignés et peuvent effectuer plusieurs requêtes d'affilée.

Pour améliorer l'application, nous pourrions faire en sorte que le serveur puisse comparer les valeurs entrées par le client et le prévienne si elles ne concordent avec aucune donnée existante. Lui mettre plus d'options à disposition. De plus, nous pourrions faire en sorte que le client compare différents résultats de serveurs et en dégage celui répondant le plus à ses critères. Le serveur pourrait également proposer des fonctionnalités spéciales telles que la récolte de statistique sur les consultations.

Ce projet nous a apporté une nouvelle expérience de travail en équipe et a affûté nos compétences en langage C et en réseau ainsi que l'utilisation du logiciel de version GitHub.