Documentation technique

Projet « Train Commander » Supinfo 2015 – 2016

Sommaire

[Choix du langage 2](#_Toc422153697)

[Choix de la plateforme et de l’API 2](#_Toc422153698)

[Transférer des données 3](#_Toc422153699)

[Connexion 3](#_Toc422153700)

[Transfert 3](#_Toc422153701)

[Rendu et utilisation 4](#_Toc422153702)

# Choix des technologies

En ce qui concerne le site web, nous avons choisi d’utiliser le framework Symfony2. Puissant et très répandu, Symfony est un choix logique si l’on considère notre objectif : créer un site web rapide et robuste, capable de renseigner des milliers d’utilisateurs chaque jour.

Pour ce qui est des applications mobiles (smartphones, tablettes), nous avons mis à contribution les compétences acquises au cours de l’année en les développant en langage natif. Nous avons ainsi obtenu des applications adaptées à leur support.

Enfin, la Web API (une de nos fonctionnalités bonus) a été écrite en PHP. Elle sert de liaison entre la base de données et le site web, ainsi qu’avec les applications mobiles.

Elle permet :

* Un meilleur contrôle de l’accès aux données. En effet, l’utilisateur ne demande jamais rien à la base de données directement. Il doit passer par la Web API, d’où une meilleure sécurité.
* Un accès uniformisé aux données. On évite par exemple de devoir installer une base de données locale sur le smartphone d’un utilisateur, ce qui prendrait de la place et offrirait des possibilités de piratage.
* De faire de l’Open Data. Nous suivons la tendance de la SNCF notamment. Nous permettons à n’importe qui d’accéder à nos données (de manière contrôlée, bien entendu) et à notre algorithme de recherche de trains. Nous encourageons ainsi les développeurs à créer des applications et à promouvoir nos services.

# Fonctionnalités du site

## Côté utilisateur

## Côté administration

# Explication de l’algorithme

## Principe général

L’algorithme de recherche doit répondre à la question suivante : parmi tous les trajets, comment trouver le meilleur en fonction des critères de l’utilisateur ?

Nous avons divisé cette question en deux sous-problèmes.

1. Parmi tous les trajets imaginables, comment obtenir uniquement ceux qui partent de la station demandée et arrivent à la bonne station de la manière la plus optimisée ?
2. Parmi les trajets ainsi obtenus, comment choisir ceux qui correspondent le mieux aux critères de l’utilisateur ?

## Recherche par temps

## Recherche par coût

# Optimisation de la base de données

## Application Windows

# Fonctionnalités bonus

## Application Windows

Avec le protocole TCP, il faut établir une connexion avant tout envoi de données. On définit un serveur, la machine qui envoie les données, et un client, la machine qui reçoit les données. L’un des deux doit « écouter », c’est-à-dire s’attendre à recevoir une connexion.

## Web API

Comme expliqué en première partie,

## CMS du site

# Choix de la plateforme et de l’API

Développer sous Linux est plus contraignant dans le sens où le projet peut être dépendant de sa distribution. Nous avons donc choisi de travailler sous Windows avec l’API Winsock2.

Winsock2 offre toutes les fonctionnalités nécessaires à la création d’une pile TCP/IP ainsi qu’une couche d’abstraction pour envoyer les paquets formés, via des sockets.

Utiliser Winsock2 prolonge notre choix du duo C/C++ : nous voulions des outils de développement stables et complets. De ce fait, notre bibliothèque n’est utilisable que dans un environnement Windows.

Si travailler sous Linux aurait permis une approche de développement plus libre, les contraintes de Windows sont compensées par ses fonctionnalités.

Nous avons travaillé sous des machines virtuelles Windows 8 Pro.

Les 4 programmes fournis avec la bibliothèque ont besoin d’un package de Visual C++ pour fonctionner si Visual Studio 2013 n’est pas installé.

<https://www.microsoft.com/en-us/download/details.aspx?id=40784>

# Transférer des données

## Connexion

Avec le protocole TCP, il faut établir une connexion avant tout envoi de données. On définit un serveur, la machine qui envoie les données, et un client, la machine qui reçoit les données. L’un des deux doit « écouter », c’est-à-dire s’attendre à recevoir une connexion.

Par exemple, nous faisons écouter un serveur sur le port 20000. Le client va envoyer un message au serveur sur ce port pour lui demander une connexion. Le serveur accepte la connexion et en informe le client. C’est un handshake de type SYN/ACK.

Une fonction de ping peut permettre de vérifier l’état de la connexion régulièrement. Tant que la machine distante répond, la connexion est maintenue.

La méthode de connexion étant une fonction bloquante, il convient de multi-threader le serveur s’il veut pouvoir servir plusieurs clients simultanément.

## Transfert

En ce qui concerne l’envoi de données, seuls les messages de tableaux de char peuvent être encodés et envoyés. En effet, comme on crée le message du paquet octet par octet, un tableau de char est plus indiqué pour les manipulations touchant à la mémoire. Ainsi, toute variable (int, string, contenus de fichiers…) sont convertis avec les fonctions appropriées.

Le choix du C se justifie ici, où l’on a un contrôle total ; une erreur à cette étape peut corrompre la frame toute entière. Une fois le message encodé, on l’encapsule en ajoutant les headers TCP/IPv4.

On récupère ensuite la taille de la frame nouvellement formée. Plutôt que d’envoyer une frame gigantesque sur le réseau (imaginez le transfert d’une vidéo), ce que Windows ne permet pas de toutes façons, la frame est découpée en plusieurs paquets qui sont envoyés individuellement. Windows s’occupe de l’encapsulation en frame Ethernet.

L’envoi se fait via les sockets fournis par Windows, qui reprennent les informations de la connexion établie pour forwarder la frame sur la carte réseau, puis vers la machine cliente.

Le client reçoit les paquets un par un, contrôle leur intégrité (RFC 793) et reconstitue la frame. Il n’y a plus qu’à récupérer le contenu du paquet et à le renvoyer à l’utilisateur.

# Rendu et utilisation

La bibliothèque finale se présente sous la forme d’un .dll et d’un .lib, qui contiennent toutes les fonctions de notre projet.

On y retrouve des fonctions ARP, qui permettent de retrouver, par exemple, une adresse MAC à partir d’une adresse IP du réseau, mais aussi des fonctions ICMP, qui assurent de la disponibilité des deux participants à un transfert, et, *last but not least*, des fonctions de transfert de données.

La fonction sendFile, par exemple, permet d’envoyer le contenu d’un fichier par le réseau, via la méthode décrite précédemment. Mais il peut également s’agir de n’importe quel type de fichiers : images, vidéos, etc… Puisque le contenu du fichier, peu importe son format, sera envoyé bit par bit.

Il suffit de le reconstituer à l’arrivée en donnant au contenu reçu l’extension appropriée ; cela n’étant pas géré automatiquement (comme le nom du fichier, par ailleurs), l’utilisateur doit spécifier le nom du fichier à créer.

Typiquement, le serveur utilise la fonction sendFile pour envoyer « image\_à\_envoyer.png », tandis que le client va utiliser la fonction ReceiveFile avec pour argument « image\_reçue.png ».

Il est à noter que le nom du fichier peut être précédé de son chemin sur le disque.

Le manuel explique comment intégrer notre bibliothèque à un projet et décrit le fonctionnement de chaque fonction.