Laboratoire de Réseaux et technologie Internet (TCP/IP & HTTP en C/C++/Java/HTML/CSS/Javascript)

Contexte général :

L'application à réaliser est une application d'achats en ligne de fruits/légumes intitulée « Le Maraicher en ligne ».
Les **clients** du « Maraicher en ligne » doivent se logger à partir d'une application « desktop » avant de pouvoir naviguer dans le catalogue du magasin en ligne et faire leurs achats. Ces achats sont stockés dans un panier virtuel avant d'être validé par le client.
Afin d'obtenir sa commande, le client devra tout d'abord payer la facture générée lors de sa session d'achat :

- Une première possibilité pour lui est de se rendre physiquement au comptoir du magasin où il s'adressera à un employé. Il pourra alors s'acquitter de sa facture auprès de cet employé avant de repartir avec sa commande de fruits et légumes.
- Une seconde possibilité sera de réaliser le paiement en ligne en utilisant une autre application « desktop » permettant un traitement sécurisé de sa facture. Il pourra ensuite se rendre physiquement au magasin pour récupérer sa commande.

La gestion et le réapprovisionnement du stock de marchandises seront réalisés par un gérant qui utilisera soit son PC, soit une tablette via une application web dédiée.

Consignes :

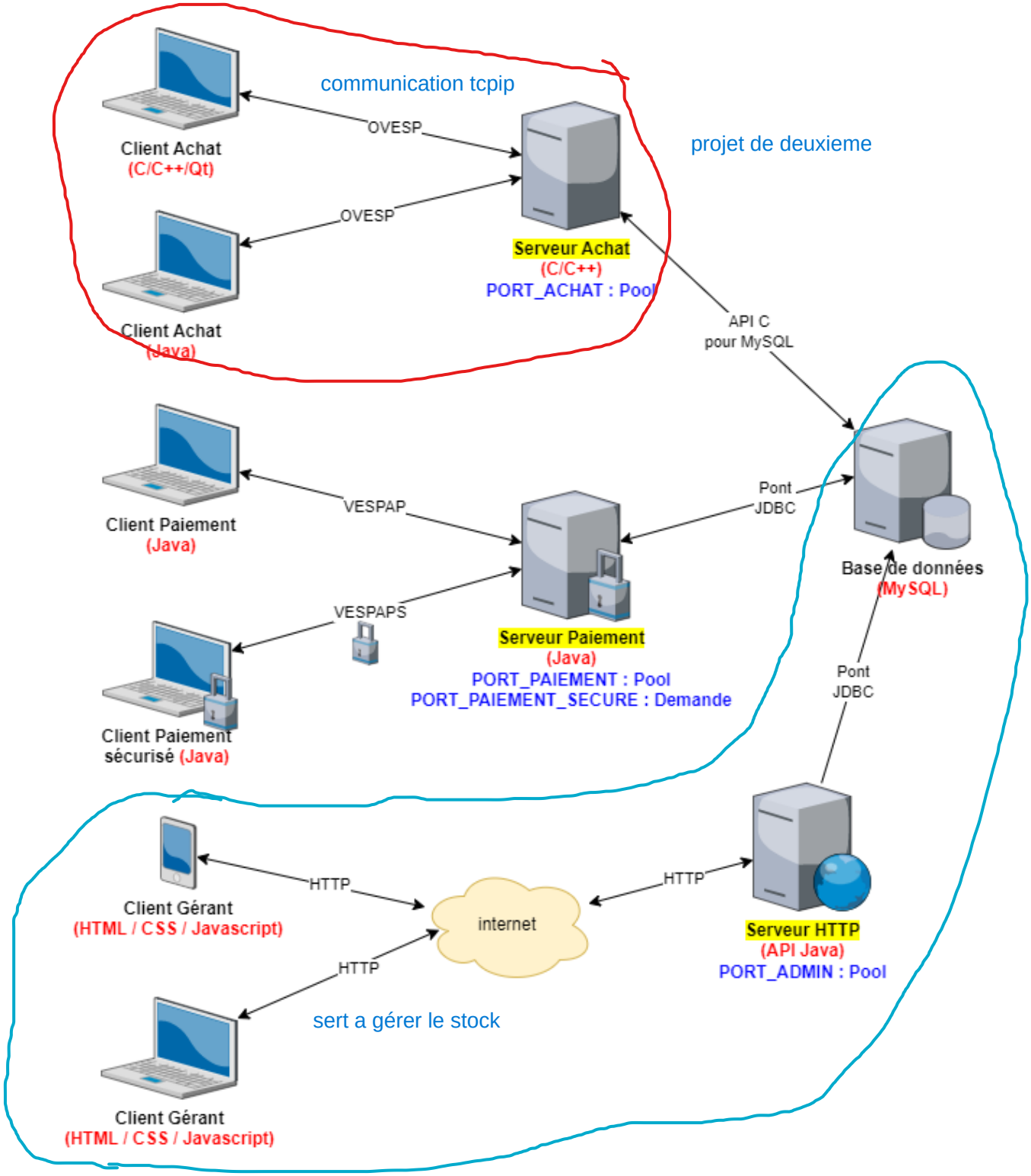
- Ce projet doit être réalisé par une équipe de 2 étudiants.
- Il y aura 3 évaluations de laboratoire :
  - 2 évaluations (continues) orales pendant le Q1 comptant chacune pour 30% de la cote de laboratoire, et
  - 1 évaluation orale pendant la session de janvier comptant pour 40% de la cote de laboratoire.
- Idéalement, toutes les démonstrations de vos clients / serveurs devront être réalisées en réseau (machines physiques différentes, VM vers VM, …)
- Construction de la cote globale de l'AA : **moyenne géométrique** entre la cote de l'examen de théorie de janvier (50%) et de la cote de laboratoire (50%)

Planning des évaluations :

Laboratoire	Contenus	Date
Evaluation 1 (continue, 30%)	Librairie de sockets C/C++, serveur multi-threads C/C++, client C (C++/Qt), base de données MySql	9/10/2023-13/10/2023
Evaluation 2 (continue, 30%)	Client Java pour le serveur C, JavaBean d'accès à la BD, serveur et client « Paiement » Java	13/11/2023-17/11/2023
Evaluation 3 (examen, 40%)	Serveur et client « Paiement » Java sécurisé (crypto), API web, backend Java du serveur web, frontend en HTML/CSS/Javascript	Date de l'examen de laboratoire de janvier 2024

Schéma global de l'application

L'application globale comporte plusieurs serveurs et clients écrits dans différents langages de programmation et paradigmes imposés. Voici le schéma global de l'application à réaliser :



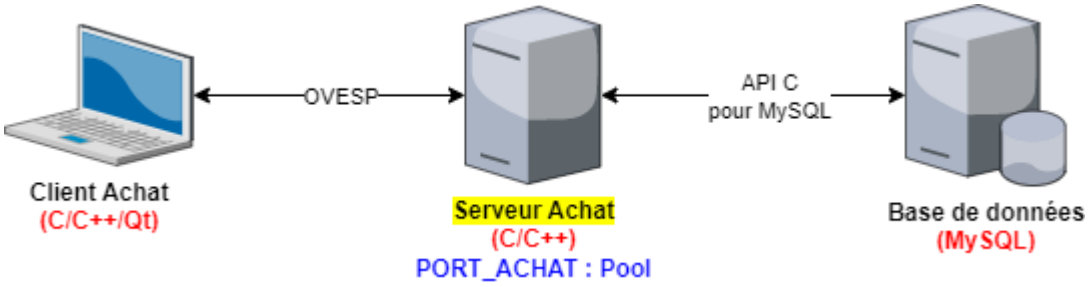
Le développement de ce projet sera découpé en 3 parties (correspondant aux 3 évaluations) décrites ci-dessous.

Partie / Evaluation 1
Date de remise : Semaine du 9/10/2023 au 13/10/2023

Cette partie comporte les éléments suivants :

- la construction d'une libraire de sockets en C/C++
- le serveur « Achat » multi-threads C/C++ tournant sur une machine Linux (et utilisant la librairie de sockets développée)
- le client C/C++/Qt capable de communiquer avec ce serveur
- la mise en place de la base de données MySQL

Sur le schéma global de l'application, cela correspond à :



L'application « Client Achat »

C'est grâce à l'application « Client Achat » (tournant sur une **machine Linux**) qu'un nouveau client (ou client existant) pourra parcourir les pages du magasin en ligne, remplir son panier et ensuite confirmer sa commande. Tout ceci se fera bien sûr via un dialogue avec le serveur Achat.

Pour faciliter le développement de l'application client, l'interface graphique Qt vous sera fournie (lien GitHub : https://github.com/hepl-rti/LaboRTI2023_sources)

Pour ceux qui ont réalisé le projet de développement en C/Linux de 2022-2023, cette interface doit vous être familière. Pour rappel,

- le bouton « Login » permet d'envoyer une requête de login contenant le nom et le mot de passe du client, ainsi qu'un booléen précisant s'il s'agit d'un nouveau client ou non. Une fois loggé, le client recevra du serveur son numéro de client.
- les boutons « >>> » et « <<< » permettent de parcourir le catalogue du magasin en envoyant à chaque clic une requête de consultation.
- le bouton « Acheter » permet d'envoyer une requête d'achat au serveur et de placer cet article (et la quantité) dans un caddie stocké au niveau du serveur.
- les boutons « Supprimer article » et « Vider le panier » permettent d'envoyer des requêtes au serveur afin de supprimer un ou tous les articles du panier.
- le bouton « Confirmer achat » permet d'envoyer une requête de confirmation au serveur. Cela a pour effet de créer une facture qui sera stockée dans la base de données avec le statut « impayé ».

L'application « Client Achat » permet donc de réaliser une commande et de la valider mais elle ne permet pas le paiement de celle-ci.

La base de données

Il s'agira d'une **base de données MySQL**. Vous pouvez la créer vous-même ou alors vous pouvez utiliser le programme **CreationBD** fourni. Ce programme (à utiliser sur la VM Oracle Linux fournie) créera la table articles et la pré-remplira. Les autres tables sont à votre charge. Voici un bref descriptif de ces tables :

Nom Table	Champs
articles	Id, intitulé, prix à unité, stock, image
clients	Id, login, password
factures	Id, idClient, date, montant, payé
ventes	idFacture, idArticle, quantité
…	…

Pour rappel, le champ image de la table articles correspond au fichier jpg correspondant à l'article. Les images jpg des fruits et légumes vont sont également fournies. La manipulation de la base de données se fera en utilisant l'**API C/MySQL** utilisée en BAC 2 et décrite dans le syllabus « Système d'exploitation Linux – Programmation avancée en C », annexe 2.

La librairie de sockets

Dans un premier temps, on vous demande de développer une petite **librairie C ou C++** de sockets (fournie sous forme de fichiers .cpp et .h) dont les caractéristiques sont :

- elle doit **générique** : on ne doit pas voir apparaître la notion d' « articles » ou de « clients » dans le prototype des fonctions. Elle doit pouvoir être réutilisée telle quelle dans une autre application
- elle doit **abstraite** : l'utilisation de votre librairie doit permettre d'éviter de voir apparaître les structures systèmes du genre « sockaddr_in » dans les programmes qui utilisent votre librairie

Conseils / propositions pour le développement :

- Une proposition de prototypes de fonctions pour votre libraire est fournie dans un des pdf de théorie (« Communications Réseaux en C »)
- Vous avez le droit de développer cette librairie en C++. Néanmoins, on ne vous l'impose/le conseille pas car le développement d'une librairie correcte en C++ vous prendra plus de temps qu'en C
- Des tests élémentaires de votre librairie sont plus que bienvenus avant de l'embarquer dans votre client Qt ou votre serveur Achat.

Le serveur Achat

Le « Serveur Achat » devra être un serveur

- **multi-threads écrit en C (threads POSIX)**,
- implémentant le modèle **« pool de threads »**
- attendant sur le port PORT_ACHAT
- tournant sur une **machine Linux** (pouvant être la VM Oracle Linux dont vous disposez déjà)

Le nombre de threads du pool ainsi que le PORT_ACHAT pourront être lus dans un fichier (texte) de configuration du serveur.

Pour la construction du protocole (trame, …), vous devez savoir que dans la second partie, un **client Java** similaire devra être développé. Ce protocole s'appellera **« Online VEgetables Shopping Protocol » (OVESP)**. Les différentes commandes sont décrites ci-dessous :

Serveur Achat -- Protocole OVESP – PORT_ACHAT			
Commande	**Requête**	**Réponse**	**Actions / Explications**
« Login »	Login, password, nouveau client ou pas	Oui ou non, message (+ idClient) ou raison	Vérification de l'existence et du mot de passe du client / Création d'un nouveau client dans la table clients
« Consult »	idArticle	idArticle ou -1, intitule, stock, prix, image	Consultation d'un article en BD → si article non trouvé, retour -1 au client
« Achat »	idArticle, quantité	idArticle ou -1, quantité ou 0, prix	Si article non trouvé, retour -1. Si trouvé mais que stock insuffisant, retour d'une quantité 0 → Si ok, le stock est mis à jour en BD et le contenu du caddie est mémorisé au niveau du serveur → actuellement aucune action sur tables factures et ventes
« Caddie »		Contenu du panier : (idArticle, intitulé, quantité, prix) × nombre d'articles du panier	Retourne l'entièreté du contenu du caddie au client
« Cancel »	idArticle	Oui ou non	Supprime un article du caddie et met à jour à la BD

« Cancel All »			Supprime tous les articles du caddie et met à jour la BD
« Confirmer »		Numéro de facture générée	Création d'une facture et BD et ajout des éléments du caddie dans la BD
« Logout »			Si Caddie en cours, vide le caddie et met à jour la BD

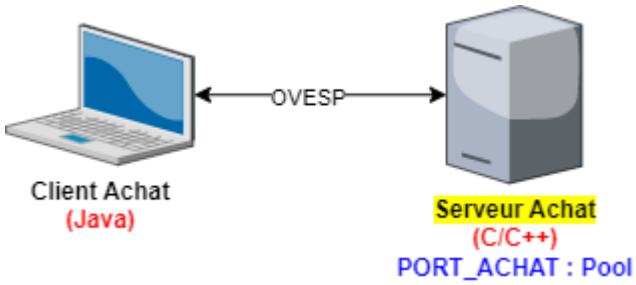
Partie / Evaluation 2
Date de remise : Semaine du 13/11/2023 au 17/11/2023

Cette partie comporte les éléments suivants :

- le client Java pour le serveur Achat
- le Java Bean d'accès à la base de données construit en utilisant JDBC
- le serveur « Paiement » multi-threads Java, utilisant le bean développé
- le client Java « Paiement » capable de communiquer avec ce serveur

L'application « Client Achat » Java

Sur le schéma global de l'application, cela correspond à :



Vous devez donc développer ici une application fenêtrée en **Java (Swing)** capable d'interagir avec le « Serveur Achat » déjà développé et similaire visuellement à celle fournie en Qt. Le serveur ne doit en aucune façon faire de différence entre le client C et le client Java, et ne doit se rendre compte de rien. Cette application sera utilisée par les clients du magasin ne disposant que d'une machine Windows avec Java installé.

Le Java Bean d'accès à la base de données

L'accès à la base de données ne devra pas se faire avec les primitives **JDBC** utilisées telles quelles, mais plutôt au moyen d'objets métiers encapsulant le travail (typiquement de type Java Beans, mais **sans mécanisme d'events**).

On demande donc de construire une petite librairie constituée d'un ensemble de telles classes permettant l'accès (c'est-à-dire à tout le moins la connexion et les opérations de base SELECT, INSERT, UPDATE) le plus simple possible. On pourrait imaginer

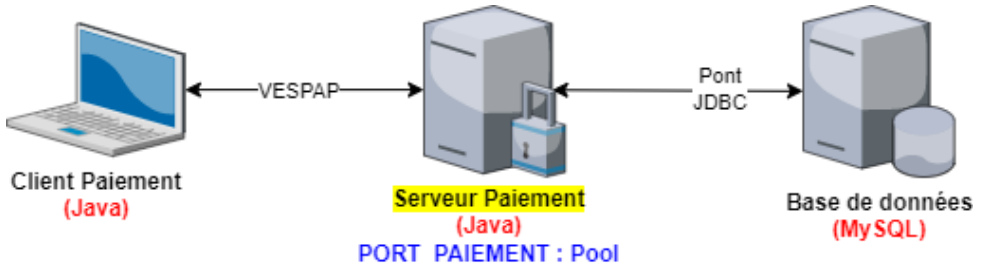
- un **bean générique** (indépendant de l'application) encapsulant la connexion à la BD et le traitement de requêtes SQL simples via les **primitives JDBC**

- un **bean « métier »** héritant (ou utilisant) le bean précédent mais dédié à la logique métier du serveur paiement. Ce bean pourrait alors s'utiliser sans aucune connaissance de JDBC ou même du langage SQL.

Attention qu'une seule instance de ce(s) bean(s) ne sera utilisée dans le serveur Paiement et que ce serveur sera un serveur multi-threads : attention donc aux accès concurrents !

Le serveur paiement et son application cliente

Sur le schéma global de l'application, cela correspond à :



L'application Java « Client Paiement » est utilisée par un employé qui accueille les clients venant payer une facture et récupérer la commande correspondante. Grâce au numéro du client donné de vive voix par le client, l'employé peut alors se connecter sur le serveur paiement et obtenir toutes les factures impayées par le client. Le client précise alors quelle facture (et donc quelle commande) il veut payer. Ensuite, il fournit à l'employé les références de sa carte VISA (nom + numéro de carte). L'employé envoie alors ces informations au serveur paiement qui se contentera de vérifier si le numéro de carte est valide.

A des fins de simplifications, si le numéro de carte est valide, on considérera que le paiement est réalisé sans problème. **NB** : Pour les bacheliers en informatique orientation « Réseaux et Télécommunications », le paiement se fera de manière sécurisée dans le laboratoire de « Complément Réseau » de Mr. Charlet.

Etant donné, que l'application Java « Client Paiement » et le « serveur Paiement » se situent tous les deux dans même réseau local, aucun mécanisme de sécurisation (cryptage, signature électronique, …) n'a été envisagé ici.

Consignes d'implémentation

Tout d'abord, le pont JDBC entre le serveur paiement et la base de données se fera en utilisant le bean d'accès développé à la section précédente.

Le « Serveur Paiement » devra être un serveur

- **multi-threads écrit en Java (utilisant le package java.net)**,
- implémentant le modèle **« pool de threads »**
- attendant sur le port PORT_PAIEMENT

Le nombre de threads du pool ainsi que le PORT_PAIEMENT pourront être lus dans un fichier (properties) de configuration du serveur.

L'application Java cliente sera développée en **Swing** et permettra au client de communiquer avec le serveur paiement selon le protocole décrit ci-dessous.

Pour la construction du protocole (trame, objets sérialisés, …), vous devez savoir que **tous les intervenants seront écrits en Java**. Ce protocole s'appellera **« Vegetables Shopping Payment Protocol » (VESPAP)**. Les différentes commandes sont décrites ci-dessous :

Serveur Paiement -- Protocole VESPAP – PORT_PAIEMENT			
Commande	**Requête**	**Réponse**	**Actions / Explications**
« Login »	Login, password (d'un employé)	Oui ou non	Vérification du login et du mot passe dans la table des employés
« Get Factures »	idClient (fournie par le client sur place)	Liste des factures (idFacture, date, montant, payé)	On récupère simplement les factures du client dans la table factures (sans le contenu détaillé de la commande donc)
« Pay Facture »	idFacture, nom et numéro de la carte VISA	Oui ou non (carte VISA invalide)	Le serveur se contente de vérifier la validité du numéro de carte → si ok, on considère que le paiement est réalisé
« Logout »			

En bonus, on pourrait compléter le protocole avec une commande « Get Facture » qui permettrait de récupérer l'ensemble des articles concernant une facture dont on fournirait l'id au serveur.

L'interface graphique de votre client devra permettre à l'utilisateur d'encoder les données nécessaires aux requêtes et d'en afficher les résultats, tout en étant le plus ergonomique possible.

Pour la validation du numéro de carte VISA, vous pouvez vous contenter d'un « choix aléatoire » du serveur. Pour les puristes qui voudraient faire mieux, l'algorithme de Luhn peut vous intéresser (voir par exemple https://ma-petite-encyclopedie.org/accueil?lex_id=1939)

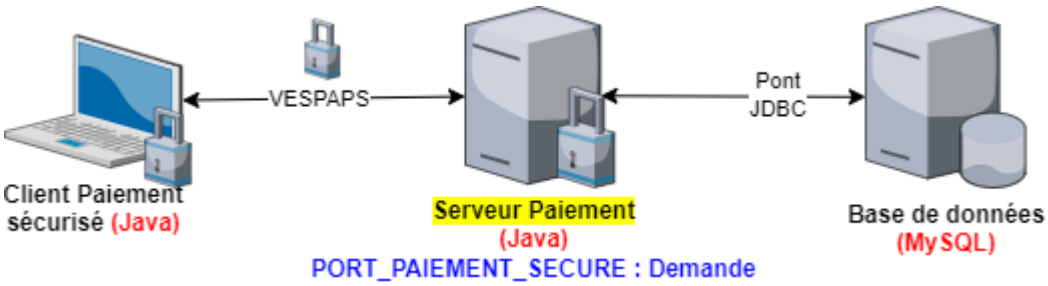
Partie / Evaluation 3
Date de remise : jour de l'examen de laboratoire en janvier 2024

Cette partie comporte les éléments suivants :

- la partie sécurisée du « Serveur Paiement » utilisant les outils cryptographiques, et l'application « Client Paiement » correspondante
- le service Web (Java) permettant récupérer et mettre à jour le stock du magasin
- le backend Java du serveur web et son frontend en HTML/CSS/Javascript

La partie sécurisée du « Serveur Paiement » et son application client

Sur le schéma global de l'application, cela correspond à :



Cette fonctionnalité de l'application permet aux clients du magasin de payer une facture de chez eux avant d'aller récupérer leur commande au magasin. Ces clients vont utiliser l'application « Client Paiement sécurisé » qui va dialoguer avec le « Serveur Paiement » mais sur un port dédié aux communications sécurisées (PORT_PAIEMENT_SECURE). Le protocole mis en place (VESPAPS) sera pratiquement identique à VESPAP déjà développé à la différence près qu'il va mettre en œuvre les outils cryptographiques classiques : cryptages symétrique et asymétrique, signature électronique, HMAC, …

La partie sécurisée du serveur paiement pourra être une application à part entière ou alors on pourra simplement ajouter un nouveau port d'écoute au serveur paiement déjà existant. Cette dernière solution est la plus simple et la plus propre étant donné que le protocole est pratiquement identique et que l'accès à la base de données est déjà mis en place.

Le « Serveur Paiement sécurisé » » devra être un serveur

- **multi-threads écrit en Java (utilisant le package java.net)**,
- implémentant le modèle **« à la demande »**
- utilisant les outils cryptographiques de la librairie Java **Bouncy Castle** dont le jar vous sera fourni
- attendant sur le port PORT_PAIEMENT_SECURE

Le nombre de threads du pool ainsi que le PORT_PAIEMENT_SECURE pourront être lus dans un fichier (properties) de configuration du serveur.

De nouveau, l'application Java cliente sera développée en **Swing** et permettra au client de communiquer avec le serveur paiement selon le protocole décrit ci-dessous.

Pour la construction du protocole (trame, objets sérialisés, …), vous devez savoir que **tous les intervenants seront écrits en Java**. Ce protocole s'appellera **« Vegetables Shopping Payment Protocol Secure » (VESPAPS)**. Les différentes commandes sont décrites ci-dessous :

Serveur Paiement sécurisé -- Protocole VESPAPS – PORT_PAIEMENT_SECURE			
Commande	**Requête**	**Réponse**	**Actions / Explications**
« Login »	Login, … avec **digest salé** → **hanshake** pour l'envoi d'une **clé de session**	Oui ou non → réception d'une **clé de session**	Le mot de passe du client ne peut pas transiter en clair sur le réseau → digest salé Vérification du login et du mot passe dans la table des clients
« Get Factures »	idClient + **signature du client**	Liste des factures **cryptée symétriquement**	On récupère simplement les factures du client dans la table factures (sans le contenu détaillé de la commande donc)
« Pay Facture »	idFacture, nom et numéro de la carte VISA, le tout **crypté symétriquement**	Oui ou non + **HMAC de la réponse**	Le serveur se contente de vérifier la validité du numéro de carte → si ok, on considère que le paiement est réalisé
« Logout »			

De nouveau, on pourrait compléter le protocole avec la commande « Get Facture » mais où la requête serait signée par le client et la réponse cryptée symétriquement.

L'échange (handshake) d'une clé de session et la signature électronique nécessite un couple de clés privée/publique. Dans un premier temps, vous pourrez vous contenter de fichiers sérialisés (au préalablement créés dans une application dédiée) et connus du client et du serveur. Néanmoins, dans un second temps, plusieurs solutions peuvent être envisagées :

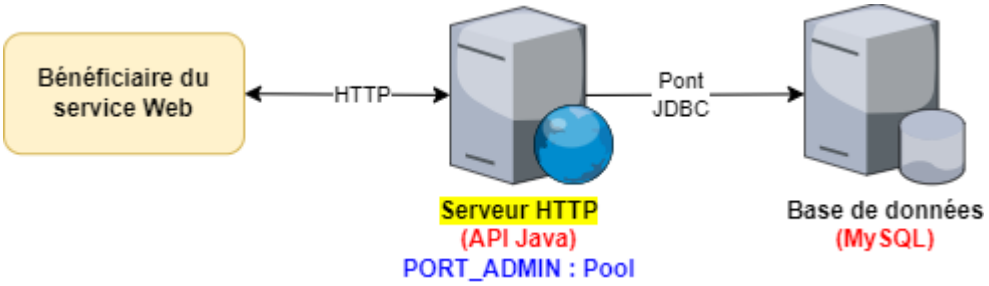
- le client et le serveur disposent chacun d'un keystore → Le client possède dans son keystore son couple de clés privée/publique tandis que le serveur dispose dans son keystore du certificat du client. A des fins de simplification du développement, on pourra se contenter d'un seul couple de clés privée/publique pour toutes les instances de l'application client
- lors de la phase de login, le client se génère un couple de clés privée/publique et envoie au serveur sa clé publique (mieux : son certificat)

Le service Web de gestion du stock du magasin

Il s'agit ici de développer un petit service Web qui permettra à toute personne (ou « bénéficiaire du service Web ») disposant de l'URL de

- récupérer le contenu complet du stock du magasin sous la forme d'un **fichier JSON**
- mettre à jour le stock et le prix d'un article du magasin

Sur le schéma global de l'application, cela correspond à :



L'API à développer ne comportera que 2 requêtes :

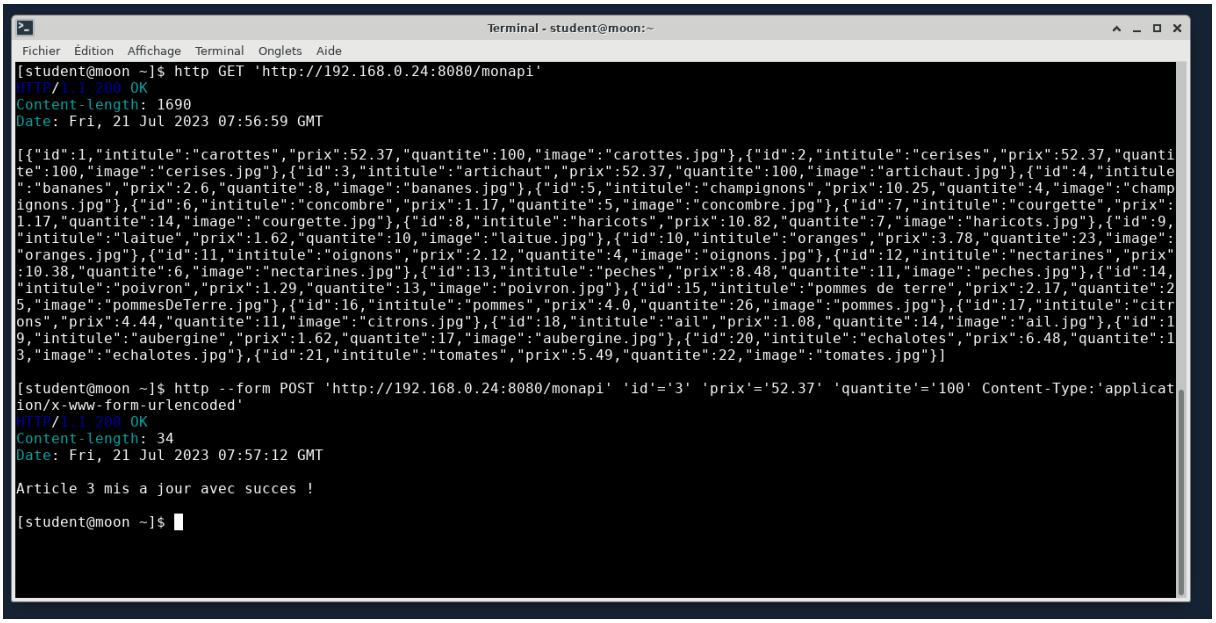
Méthode	Requête	Réponse
GET		Liste de tous les articles au format JSON
POST	idArticle, prix, stock	Oui ou non au format texte

Consignes d'implémentation

Le **service Web** devra être développé en **Java « from scratch »**, c'est-à-dire sans avoir recours à un serveur d'applications comme Apache Tomcat ou autre. On vous demande d'utiliser la classe **HttpServer** et l'interface **HttpHandler** du **package Java com.sun.net.httpserver**.

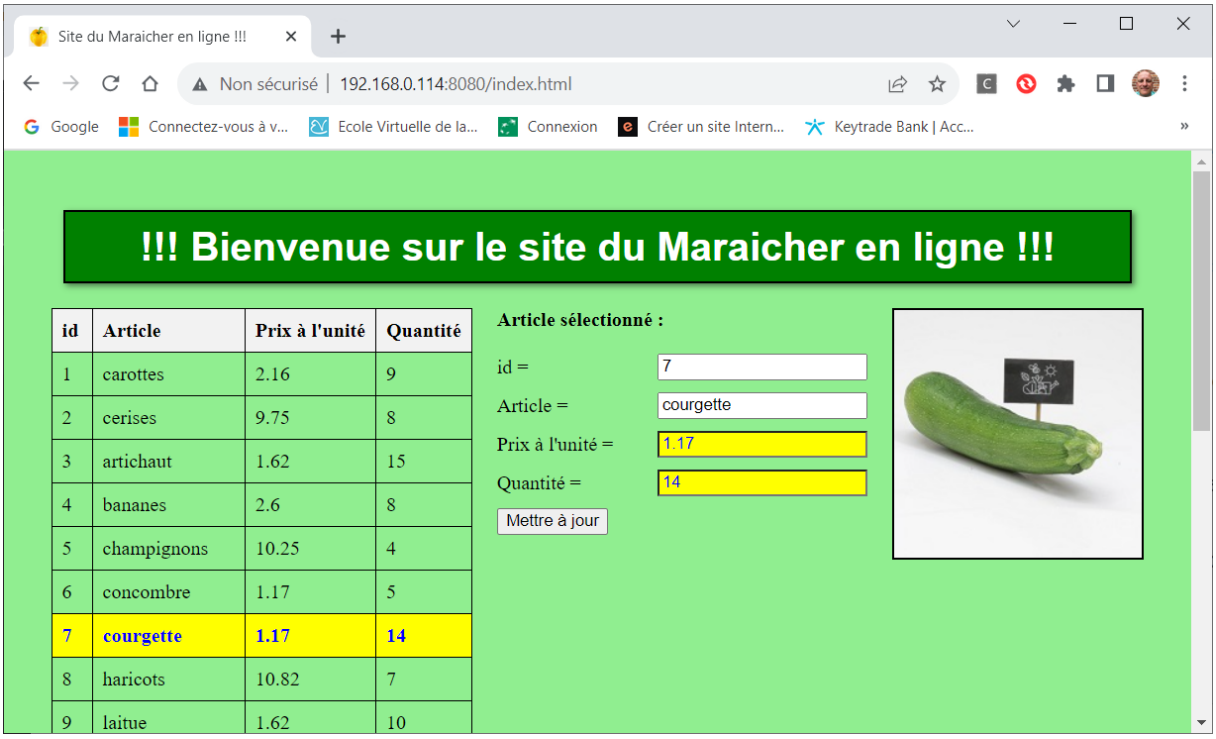
Votre serveur Web ainsi créé accèdera à la base de données via le Java Bean déjà développé.

Les tests de votre service Web seront réalisés avec le logiciel **Postman** (https://www.postman.com/) et/ou la commande **http** de la machine Oracle Linux fournie :



Le frontend de l'application Web

L'application Web à développer ici sera utilisée par un gérant voulant consulter et mettre à jour le stock du magasin. Visuellement, elle pourrait ressembler à ceci :



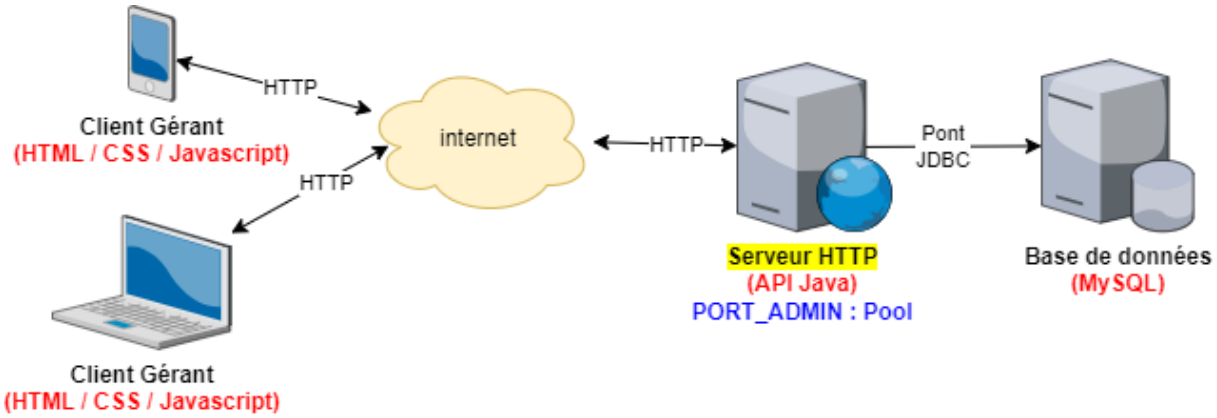
voir même

sur un dispositif mobile.

Le site web ajustera automatiquement le nombre de lignes de la table au nombre d'articles du magasin. Un clic sur une ligne de la table fera apparaître (dans la zone centrale ici) l'article sélectionné et son image (zone de droite ici), ce qui permettra de modifier son prix à l'unité et son stock (les 2 champs de texte jaune). Le clic sur le bouton « Mettre à jour » permettra de mettre à jour le stock dans la base de données en passant par le serveur Web.

Consignes d'implémentation

Sur le schéma global de l'application, cela correspond à :



Au niveau **backend**, votre **serveur Web Java** va devoir être complété afin qu'il réponde aux requêtes du browser en ce qui concerne les pages HTML, les fichiers images, les fichiers Javascript et CSS.

Au niveau **frontend**, on demande de créer une application Web du type **« Single Page Application » (SPA)** , c'est-à-dire

- une seule page HTML contenant un script Javascript
- Ce script **Javascript** utilisera le service Web développé ci-dessus afin de mettre à jour la table et de répondre au clic sur le bouton de la page HTML. Pour cela, il utilisera la technologie **AJAX** afin d'interroger le service Web (utilisation de **XMLHttpRequest**)
- On vous demande de créer également un **fichier CSS** utilisé pour votre application web afin d'en personnaliser l'affichage.

N'hésitez pas à demander conseil à vos professeurs de laboratoire pour tout choix d'implémentation.

Bon travail 😊 !