

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 de gestion et d'achats de livres en ligne.

Dans un premier temps, les **employés** du magasin devront encoder les livres reçus en base de données. Pour cela, ils disposeront d'une application dédiée à cet effet.

Les **clients** du magasin pourront se rendre physiquement sur place pour acheter des livres mais pourront également, après un premier achat sur place, commander et acheter des livres en ligne. Pour l'achat en magasin, un **vendeur** utilisera une application « desktop » tandis que pour l'achat ligne, un **client** utilisera une application « desktop sécurisée ». Ces deux applications sont de type « **client lourd** ».



En outre, un **responsable** du magasin pourra mettre à jour la base de données du magasin à l'aide d'une application web (de type « **client léger** ») en utilisant un browser internet sur son PC ou sa tablette.

Consignes :

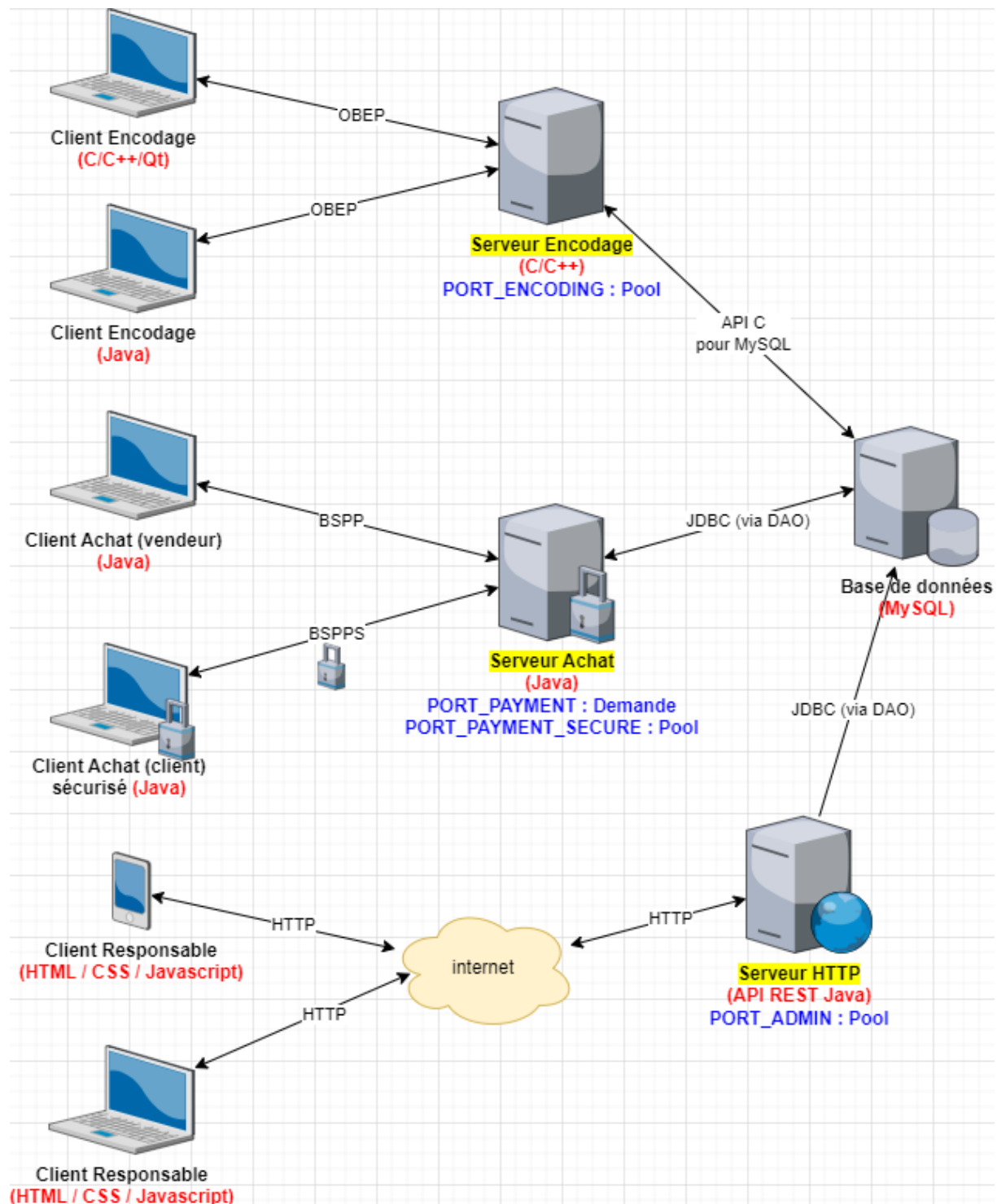
- Ce projet doit être réalisé par une équipe de 2 étudiants.
- Il y aura 3 évaluations de laboratoire :
 - 2 évaluations (continues) orales pendant le Q1 comptant chacune pour 30% de la cote de laboratoire, et
 - 1 évaluation orale pendant la session de janvier comptant pour 40% de la cote de laboratoire.
- 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	14/10/2024- 18/10/2024
Evaluation 2 (continue, 30%)	Client Java pour le serveur C, DAO d'accès à la BD, serveur et client « Achat » Java	11/11/2024- 15/11/2024
Evaluation 3 (examen, 40%)	Serveur et client « Achat » Java sécurisé (crypto), API web, backend Java du serveur web, frontend en HTML/CSS/Javascript	Date de l'examen de laboratoire de janvier 2025

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.

Consigne importante concernant ChatGPT :

« **ChatGPT** (ou toute aide « IA » à la production automatique de code) est ton ami mais... Pour qu'il le reste, tu devras être capable d'expliquer tout le code généré par **ChatGPT**. Tout code généré par **ChatGPT** que tu seras incapable d'expliquer en détails débouchera sur une cote nulle concernant la question sur ce code (**ET** pour la partie concernée du projet), même si celui-ci est correct et fonctionnel. »

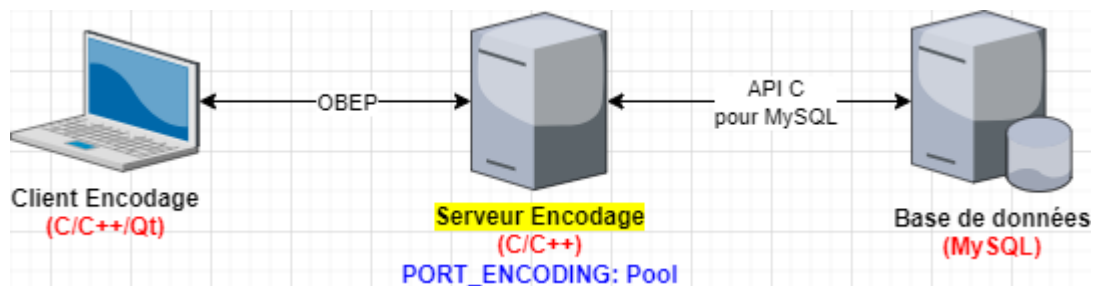
Partie / Evaluation 1

Date de remise : Semaine du 14/10/2024 au 18/10/2024

Cette partie comporte les éléments suivants :

- la construction d'une librairie de sockets en C/C++
- le serveur « Encodage » 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 Encodage »

C'est grâce à l'application « Client Encodage » (tournant sur une **machine Linux**) qu'un employé du magasin pourra encoder un nouveau livre, un nouvel auteur ou un nouveau sujet de livre. Tout ceci se fera bien sûr via un dialogue avec le serveur Encodage.

Pour faciliter le développement de l'application client, l'interface graphique Qt vous sera fournie (lien GitHub : https://github.com/hepl-rti/LaboRTI2024_sources)

The screenshot shows a web application window titled 'Application Encodage'. It has a 'Session' section with input fields for 'Titre', 'ISBN', 'Pages', 'Année du publication', 'Prix', and 'Stock'. There are also dropdown menus for 'Auteur' (set to 'Bernard Werber') and 'Sujet' (set to 'Roman'). Buttons for 'Ajouter' are next to the author and subject fields, and a larger 'Ajouter nouveau livre' button is at the bottom right. An orange button labeled 'Vider les champs' is also present. Below the input fields is a section titled 'Liste des livres encodés lors de cette session :'. It contains a table with 9 columns: Id, Titre, Auteur, Sujet, ISBN, Pages, Année, Prix, and Stock. The table lists three books: 'Les Thanatonautes' by Bernard Werber, 'Dune' by Frank Herbert, and 'Le silence des agneaux' by Thomas Harris.

Id	Titre	Auteur	Sujet	ISBN	Pages	Année	Prix	Stock
1	Les Thanatonautes	Bernard Werber	Science-Fiction	978-2253139225	505	1999	9.7	3
6	Dune	Frank Herbert	Science-Fiction	978-2266320481	929	2021	11.95	13
13	Le silence des agneaux	Thomas Harris	Thriller	978-2266208949	377	2015	7.7	17

Quelques explications :

- Dans le menu « Session » se trouve un item « Login ». Un clic sur cet item permet, après encodage par l'utilisateur, d'envoyer une requête de login contenant le login et le mot de passe de l'employé. Une fois loggé,
 - l'employé aura accès aux 4 boutons de l'interface graphique
 - deux requêtes seront envoyées automatiquement au serveur afin de récupérer la liste des auteurs et des sujets déjà existants. Ceci permettra de remplir les deux comboboxes de l'interface graphique.
- Dans le menu « Session » se trouve un item « Logout ». Un clic sur cet item permet d'envoyer une requête de logout au serveur.
- les boutons « Ajouter » permettent, après encodage des données nécessaires, d'envoyer une requête au serveur afin d'ajouter un nouvel auteur ou un nouveau sujet de livre dans la base de données. Les comboboxes seront alors mis à jour si l'ajout a réussi.
- le bouton « Ajouter nouveau livre » permet, après encodage des données nécessaires et récupération des **id** de l'auteur et du sujet, d'envoyer une requête permettant d'ajouter un nouveau livre dans la base de données. Si l'ajout est réussi, le livre créé apparaît dans la table de l'interface graphique qui ne contient que les livres encodés par l'employé lors de cette session.

L'application « Client Encodage » permet donc de créer un nouveau livre / auteur / sujet mais ne permet pas de les modifier, ni de les supprimer.

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 les tables books, authors et subjects et les pré-remplira. Les autres tables sont à votre charge. Voici un bref descriptif de ces tables :

Nom Table	Champs
books	id (clé primaire), author_id (clé étrangère), subject_id (clé étrangère), title, isbn, page_count, stock_quantity, price, publish_year
authors	id (clé primaire), last_name, first_name, birth_date
subjects	id (clé primaire), name
employees	id , login, password

clients	id , nom, prénom, numéro de client, adresse, ... → à vous de voir
caddies	id , clientId , date, amount, payed
caddy_items	id , caddyId , bookId , quantity
...	...

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 de « books » ou de « authors » 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 librairie 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 Encodage.

Le serveur Encodage

Le « Serveur Encodage » devra être un serveur

- **multi-threads écrit en C (threads POSIX)**,
- implémentant le modèle « **pool de threads** »
- attendant sur le port **PORT_ENCODING**
- 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_ENCODING** pourront être lus dans un **fichier (texte) de configuration** du serveur.

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

Serveur Encodage -- Protocole OBEP – PORT_ENCODING			
Commande	Requête	Réponse	Actions / Explications
« LOGIN »	Login, password de	Oui ou non	Vérification de l'existence

	l'employé		et du mot de passe de l'employé
« LOGOUT »
« GET_AUTHORS »		Liste des auteurs (id, last_name, first_name)	Récupération des auteurs en BD
« GET_SUBJECTS »		Liste des sujets (id, name)	Récupération des sujets en BD
« ADD_AUTHOR »	lastName, firstName	Id de l'auteur ajouté ou -1	Ajoute un nouvel auteur en BD. Retour -1 si auteur déjà existant.
« ADD_SUBJECT »	name	Id du sujet ajouté ou -1	Ajoute un nouveau sujet en BD. Retour -1 si sujet déjà existant.
« ADD_BOOK »	authorId, subjectId, title, isbn, pageCount, stockQuantity, price, publishYear	Id du livre ajouté ou -1	Ajoute un nouveau livre en BD. Retour -1 si auteur et/ou sujet inexistant.

Partie / Evaluation 2

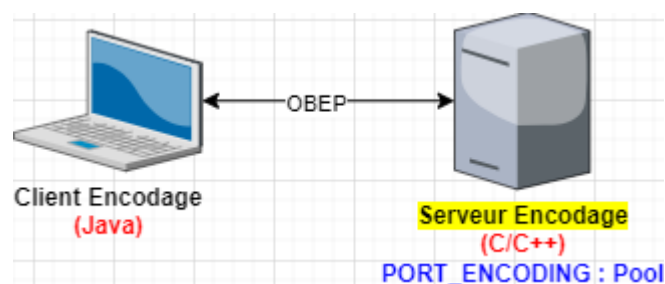
Date de remise : **Semaine du 11/11/2024 au 15/11/2024**

Cette partie comporte les éléments suivants :

- le client Java pour le serveur Encodage
- les DAO d'accès à la base de données construits en utilisant JDBC
- le serveur « Achat » multi-threads Java, utilisant les DAO développés
- le client Java « Achat » capable de communiquer avec ce serveur

L'application « Client Encodage » 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 Encodage » 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 employés du magasin ne disposant que d'une machine Windows avec Java installé.

Les DAO d'accès à la base de données

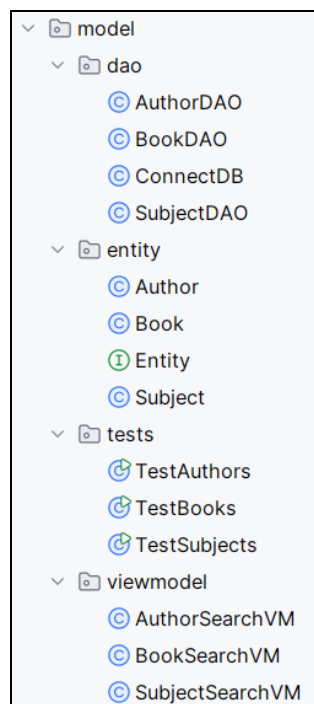
L'accès à la base de données ne devra pas se faire avec les primitives **JDBC** utilisées telles quelles, mais plutôt au moyen d'objets permettant d'accéder aux données, des **DAO** (« **Data Access Object** ») spécifiques à chaque entité.

On demande donc de construire un ensemble de classes et de packages :

- les classes « **entity** », c'est-à-dire les objets « métiers » : **Book**, **Author**, **Subject**, ... collant (« **object mapping** ») à la structure de la base de données
- Les classes « **DAO** » encapsulant la connexion à la base de données et les méthodes classiques dites « **CRUD** »
- Les classes « **xxxSearchVM** » permettant de réaliser des sélections d'entités selon certains critères. En particulier, on pourra réaliser des filtres de recherche sur
 - Les « **subject** » selon l'id et le name (exact)
 - Les « **author** » selon l'id, le lastName (« like ») et le firstName (« like »)
 - Les « **book** » selon l'id, le title (« like »), le lastName (« like ») et le firstName (« like ») de l'auteur, le name du sujet (« like ») et un prix maximum.
 - ... selon les besoins que vous aurez

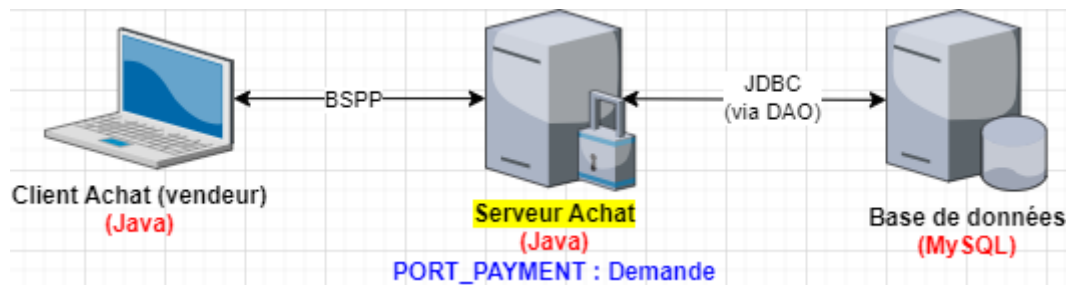
Attention qu'une seule instance de la connexion à la base de données devra être utilisée par tous les objets DAO instanciés (la connexion à la BD sera donc unique). Chaque type d'objet **xxxDAO** ne sera instancié qu'une seule fois et partagé entre les threads du serveur Achat. Ce serveur est un serveur multi-threads : attention donc aux accès concurrents !

Enfin, on veillera à une structure propre du projet. Par exemple, en se limitant aux entités « Book », « Author » et « Subject », on pourrait avoir :



Le serveur Achat et son application cliente

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



L'application Java « Client Achat » est utilisée par un vendeur qui accueille physiquement les clients venant au magasin acheter des livres en présentiel. Deux possibilités :

- Le client est nouveau : il donne alors son nom et prénom au vendeur. Celui-ci envoie une requête au serveur qui lui retournera un numéro de client créé par le serveur.
- Le client est déjà encodé dans la base de données : le vendeur encode alors le nom et le prénom du client et récupère via le serveur le numéro de client existant.

L'application est alors configurée pour s'occuper du client devant le vendeur. Celle-ci comportera

- Une zone où le vendeur peut réaliser des recherches de livres selon différents critères : titre du livre, auteur, sujet, prix inférieur à un certain montant (confer bookSearchVM mis en place) → cette zone comportera donc une table de livres (semblable à celle de l'application « client encodage »). Le vendeur pourra alors sélectionner un livre et encoder une quantité (le client peut acheter plusieurs exemplaires du même livre) → il pourra alors ajouter ce « **caddy_item** » au panier du client
- Une zone où l'on voit une table contenant le panier du client. Chaque ligne de cette table indiquera le livre et la quantité choisie.
- Plusieurs boutons permettront donc d'ajouter un « **caddy_item** » au panier, d'en supprimer un, de vider complètement le panier, et finalement de payer le panier. A chaque clic sur un bouton, une requête sera envoyée au serveur (voir ci-dessous).
- Pour sortir de session :
 - soit le vendeur devra valider la paiement (on supposera que le paiement a été réalisé en liquide ou par carte ; non géré ici) → la panier sera alors « flaggé payé » en base de données.
 - soit le vendeur cliquera sur un bouton « Annuler ».
 - dans les deux cas, l'application est prête à accueillir un nouveau client.

Etant donné, que l'application Java « Client Achat » et le « serveur Achat » 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 Achat et la base de données se fera en utilisant les DAO développés à la section précédente.

Le « Serveur Achat » devra être un serveur

- **multi-threads écrit en Java (utilisant le package java.net),**
- implémentant le modèle « **à la demande** »
- attendant sur le port PORT_PAYMENT

Le numéro de port PORT_PAYMENT, les paramètres de la BD seront lus dans un fichier (properties) de configuration du serveur.

L'application Java cliente sera développée en **Swing** et permettra au vendeur de communiquer avec le serveur Achat 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 « **Books Shopping Payment Protocol** » (BSPP). Les différentes commandes sont décrites ci-dessous :

Serveur Achat -- Protocole BSPP – PORT_PAYMENT			
Commande	Requête	Réponse	Actions / Explications
« CLIENT »	Nom, prénom du client, nouveau ou pas (booléen)	Numéro de client ou erreur	Vérification du nom/prénom du client. Si client existant, retourne son numéro, nouveau numéro sinon.
« SELECT_BOOK »	Critères de recherche de livre	Liste des livres répondant aux critères	On récupère les livres dans la BD à l'aide des DAO et searchVM
« ADD_CADDY_ITEM »	Id du livre et quantité	Oui ou non (si stock insuffisant)	Vérification du stock du livre en question. Si ok, mise à jour du stock et ajout d'un tuple « caddy_item » en BD. Si premier « caddy_item », création d'un tuple « caddy » également.
« DEL_CADDY_ITEM »	Id du livre et quantité	Oui ou non	Mise à jour ou suppression du « caddy_item » en BD
« CANCEL_CADDY »	Id du client	Oui ou non	Suppression du caddy et des caddy_item correspondants
« PAY_CADDY »	Id du client	Oui ou non	Le caddy est « flaggé » payé

Partie / Evaluation 3

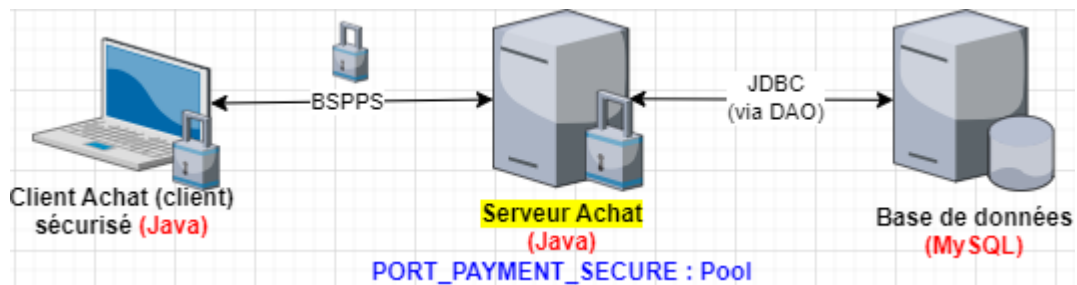
Date de remise : jour de l'examen de laboratoire en janvier 2025

Cette partie comporte les éléments suivants :

- la partie sécurisée du « Serveur Achat » utilisant les outils cryptographiques, et l'application « Client Achat » correspondante
- une API REST Web (Java) permettant les opérations CRUD sur les livres, les auteurs et les sujets de livre
- le backend Java du serveur web et son frontend en HTML/CSS/Javascript

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

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



Cette fonctionnalité de l'application permet aux clients déjà encodés dans la base de données (et disposant donc d'un numéro de client) d'acheter des livres en ligne et de se les faire livrer à domicile.

Ces clients vont utiliser l'application « Client Achat sécurisé » qui va dialoguer avec le « Serveur Achat » mais sur un port dédié aux communications sécurisées (PORT_PAYMENT_SECURE). Le protocole mis en place (BSPPS) sera pratiquement identique à BSPP 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 Achat pourra être une application à part entière ou alors on pourra simplement ajouter un nouveau port d'écoute au serveur Achat 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 Achat sécurisé » devra être un serveur

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

Le nombre de threads du pool ainsi que le PORT_PAYMENT_SECURE seront 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 Achat selon le protocole décrit ci-dessous. Cette application sera fort similaire à l'application utilisée par le vendeur en magasin à la différence près que le client devra encoder son nom, son prénom et son numéro de client (qui fera office de mot de passe) pour « entrer en session » sur le serveur.

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

Serveur Achat sécurisé -- Protocole BSPPS – PORT_PAYMENT_SECURE			
Commande	Requête	Réponse	Actions / Explications
« CLIENT »	Nom, prénom du client, ... avec digest salé (voir ci-dessous) → handshake pour l'envoi d'une clé de session	Oui ou non → réception d'une clé de session	Le numéro du client ne peut pas transiter en clair sur le réseau → digest salé Vérification du nom, prénom et du numéro dans la table des clients
« SELECT_BOOK »	Critères de recherche de livre	Liste des livres répondant aux critères	On récupère les livres dans la BD à l'aide des DAO et searchVM
« ADD_CADDY_ITEM »	Id du livre et quantité cryptés symétriquement	Oui ou non (si stock insuffisant)	Vérification du stock du livre en question. Si ok, mise à jour du stock et ajout d'un tuple « caddy_item » en BD. Si premier « caddy_item », création d'un tuple « caddy » également.
« DEL_CADDY_ITEM »	Id du livre et quantité cryptés symétriquement	Oui ou non	Mise à jour ou suppression du « caddy_item » en BD
« CANCEL_CADDY »	Id du client + signature du client	Oui ou non + HMAC de la réponse	Suppression du caddy et des caddy_item correspondants
« PAY_CADDY »	Id du client + numéro et nom carte VISA cryptés symétriquement + signature du client	Oui ou non + HMAC de la réponse	Le caddy est « flaggé » payé

Pour la commande LOGIN :

1. le client envoie son nom et son prénom,
2. le serveur vérifie l'existence du client. Si celui-ci existe, le serveur génère le « sel » et l'envoie au client,
3. le client génère un digest à partir de son numéro de client et du sel reçu du serveur, digest qu'il envoie au serveur,
4. le serveur vérifie le digest reçu sur base du sel et du numéro de client présent en BD.

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 préalable 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

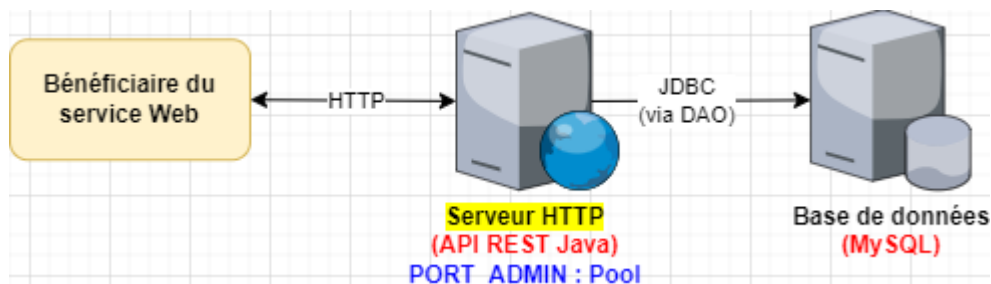
- lors de la phase de login, le client se génère un couple de clés privée/publique et envoie au serveur sa clé publique (mieux : son certificat)

BONUS : dans les notes de cours, le cryptage symétrique donné en exemple est DES, ce qui n'est plus tout à fait au goût du jour → remplacer **DES** par « **Triple-DES** »

L'API REST de gestion des livres, des auteurs et des sujets

Il s'agit ici de développer un service Web de type API REST qui permettra à toute personne (ou « bénéficiaire du service Web ») disposant de l'URL de réaliser les opérations CRUD sur les livres, les auteurs et les sujets présents dans la base de données.

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



Le protocole de l'API REST à développer est le suivant :

Routes / URL	Requête	Réponse
/api/subjects /api/subjects?id=3 /api/subjects?name=Sciences	Méthode : GET Query : critères de recherche Body : /	Liste de tous les sujets au format JSON (application/json)
/api/subjects	Méthode : POST Query : / Body : name=XXX	Ajoute un nouveau sujet en BD et retourne une chaîne de caractères informant du statut de la requête (text/plain)
/api/subjects?id=3	Méthode : PUT Query : id du sujet à modifier Body : name=XXX	Modifie le sujet d'id fourni en BD et retourne une chaîne de caractères informant du statut de la requête (text/plain)
/api/subjects?id=3	Méthode : DELETE Query : id du sujet à supprimer Body = /	Supprime le sujet d'id fourni et retourne une chaîne de caractères informant du statut de la requête (text/plain)
/api/authors /api/authors?id=2 /api/authors?lastName=Brown &firstName=Dan	Méthode : GET Query : critères de recherche Body : /	Liste de tous les auteurs au format JSON (application/json)

/api/authors	Méthode : POST Query : / Body : lastName=XXX &firstName=YYY &birthDate=yyyy-mm-dd	Ajoute un nouvel auteur en BD et retourne une chaîne de caractères informant du statut de la requête (text/plain)
/api/authors?id=3	Méthode : PUT Query : id de l'auteur à modifier Body : lastName=XXX &firstName=YYY &birthDate=yyyy-mm-dd	Modifie l'auteur d'id fourni en BD et retourne une chaîne de caractères informant du statut de la requête (text/plain)
/api/authors?id=3	Méthode : DELETE Query : id de l'auteur à supprimer Body : /	Supprime l'auteur d'id fourni et retourne une chaîne de caractères informant du statut de la requête (text/plain)
/api/books /api/books?id=2 /api/books?title=Dune &subjectName=Sciences&...	Méthode : GET Query : critères de recherche Body : /	Liste de tous les livres au format JSON (application/json)
/api/books	Méthode : POST Query : / Body : title=XXX &authorID=YY&subjectId=ZZ &isbn=WWW&...	Ajoute un nouveau livre en BD et retourne une chaîne de caractères informant du statut de la requête (text/plain)
/api/books?id=3	Méthode : PUT Query : id du livre à modifier Body : title=XXX &authorID=YY&subjectId=ZZ &isbn=WWW&...	Modifie le livre d'id fourni en BD et retourne une chaîne de caractères informant du statut de la requête (text/plain)
/api/books?id=3	Méthode : DELETE Query : id du livre à supprimer Body : /	Supprime le livre d'id fourni et retourne une chaîne de caractères informant du statut de la requête (text/plain)

Remarquez que

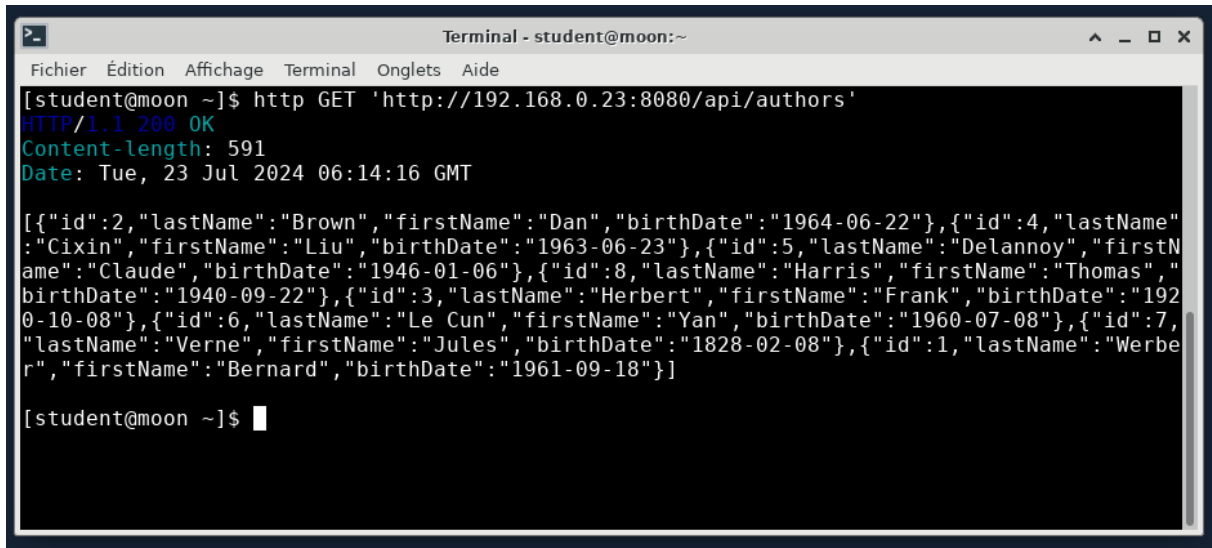
- lors d'un ajout (méthode **POST**), tous les champs (sauf l'id) sont nécessaires dans le body sous peine de renvoyer une erreur
- lors d'une sélection (méthode **GET**), les critères de recherche contenus dans la query ne doivent pas comporter tous les champs mais un sous-ensemble de ceux pour lesquels les xxxSearchVM ont été configurés
- lors d'une modification (méthode **PUT**), tous les champs ne sont pas nécessaires dans le body, uniquement ceux qui doivent être modifiés

Consignes d'implémentation

L'**API REST** devra être développée 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 API REST ainsi créée accèdera à la base de données via les **DAO** déjà développés.

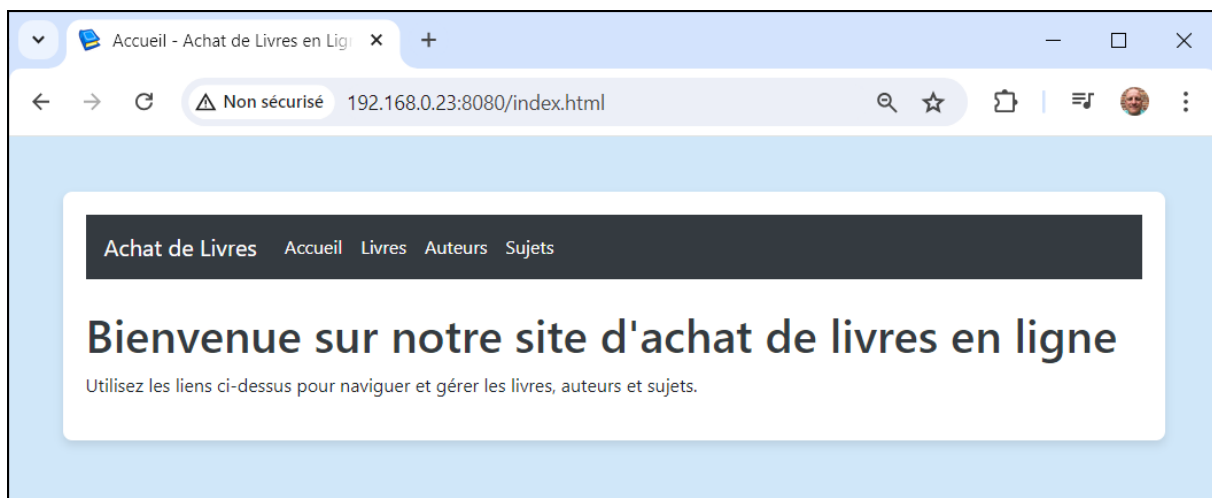
Les tests de votre API seront réalisés avec le logiciel **Postman** (<https://www.postman.com/>) et/ou la commande **http** de la machine Oracle Linux fournie :



```
Terminal - student@moon:~  
Fichier Édition Affichage Terminal Onglets Aide  
[student@moon ~]$ http GET 'http://192.168.0.23:8080/api/authors'  
HTTP/1.1 200 OK  
Content-length: 591  
Date: Tue, 23 Jul 2024 06:14:16 GMT  
  
[{"id":2,"lastName":"Brown","firstName":"Dan","birthDate":"1964-06-22"}, {"id":4,"lastName":  
:"Cixin","firstName":"Liu","birthDate":"1963-06-23"}, {"id":5,"lastName":"Delannoy","firstN  
ame":"Claude","birthDate":"1946-01-06"}, {"id":8,"lastName":"Harris","firstName":"Thomas",  
birthDate":"1940-09-22"}, {"id":3,"lastName":"Herbert","firstName":"Frank","birthDate":"192  
0-10-08"}, {"id":6,"lastName":"Le Cun","firstName":"Yan","birthDate":"1960-07-08"}, {"id":7,  
"lastName":"Verne","firstName":"Jules","birthDate":"1828-02-08"}, {"id":1,"lastName":"Werbe  
r","firstName":"Bernard","birthDate":"1961-09-18"}]  
  
[student@moon ~]$
```

Le frontend de l'application Web

L'application Web à développer ici sera utilisée par un responsable voulant mettre à jour (CRUD) la base de données des livres, auteurs et sujets. Visuellement, elle pourrait ressembler à ceci :



Gestion des Livres - Achat de Li x +

Non sécurisé 192.168.0.23:8080/books.html

Achat de Livres Accueil Livres Auteurs Sujets

Gestion des Livres

Liste des Livres

ID	Titre	Auteur	Sujet	ISBN	Nombre de Pages	Quantité en Stock	Prix	Année de Publication
5	Anges et demons	Dan Brown	Thriller	978-2253093008	720	2	10.4	2015
4	Da Vinci Code	Dan Brown	Thriller	978-2709624930	574	6	22.9	2004
6	Dune	Frank Herbert	Science-Fiction	978-2266320481	928	13	11.95	2021

Gestion des Sujets de Livre - Ac x +

Non sécurisé 192.168.0.23:8080/subjects.html

Achat de Livres Accueil Livres Auteurs Sujets

Gestion des Sujets de Livre

Liste des Sujets

ID	Nom
3	Informatique
1	Science-Fiction
2	Thriller

Gestion des Sujets

ID

Nom

Ajouter Modifier Supprimer Rechercher Vider les champs

Le site web comportera une page web d'accueil [index.html](#) permettant d'être redirigé vers une page HTML propre à chaque entité, ici [books.html](#), [authors.html](#) et [subjects.html](#).

Chaque page HTML « entité »

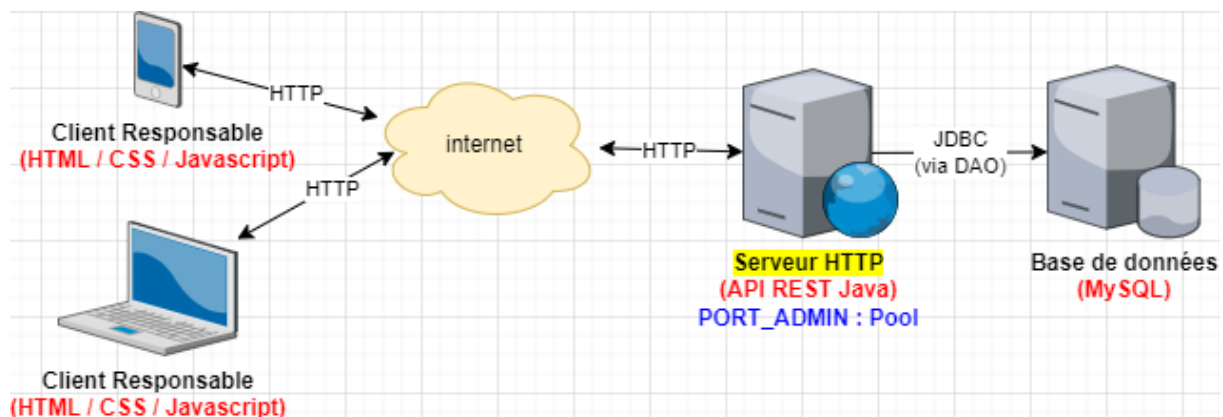
- comportera une table remplie avec les entités (toutes ou celles résultant d'une sélection)
- Un clic sur une ligne de la table remplira les champs du dessous avec les champs de l'entité sélectionnée
- Les clics sur les différents boutons permettront d'interagir avec l'API développée et de mettre à jour la page web en conséquence

Notez que vous pouvez tester votre application sur un dispositif mobile :



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 Javascript et CSS.

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

- une page HTML, propre à chaque entité, contenant un script Javascript
- Ce script **Javascript** utilisera l'API développée ci-dessus afin de mettre à jour la base de données (via les DAO de l'API), la table de la page HTML et de répondre aux clics sur les boutons de la page HTML. Pour cela, il utilisera la technologie **AJAX** afin d'interroger l'API (utilisation de **XMLHttpRequest**)
- On vous demande de créer également un **fichier CSS** utilisé pour votre application web afin d'en personnaliser l'affichage. Remarque : les pages HTML dont les captures sont données ci-dessous ont été personnalisées à l'aide de la librairie « **Bootstrap** ». Libre à vous de l'utiliser si vous le souhaitez.

BONUS 1 : Dans l'état actuel des choses, l'API REST et le site web (HTML, JS et CSS) sont accessibles sur le même port, ce qui n'est pas idéal → spécifier des ports différents pour l'API REST et le site web.

BONUS 2 : Dans l'état actuel des choses, n'importe quelle personne disposant de l'URL de l'API peut l'utiliser → Rechercher, proposer et implémenter un moyen de ne donner l'accès à cette API qu'à certaines personnes bien précises.

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

Bon travail 😊 !