Université du Québec à Chicoutimi
Département d'informatique et de mathématique

8INF349 – Technologies Web Avancées

TP2

Professeur : Hamid Mcheick	Groupe : **deux étudiant(e)s au maximum**
Trimestre : ÉTÉ 2024	Date de distribution : 30 Mai 2024
Pondération : 15 points	Date de remise : 13 Juin 2024 avant 9h00

Vous devez réaliser une **seule** question par groupe.

Avant de remettre votre tp, essayez de le faire fonctionner sur ta machine ou sur une machine Linux/Unix. SVP, mettez tous les fichiers (zip) dans le même répertoire pour faciliter la correction. Indiquez le fichier d'exécution sur dans le document remis. Aucun tp sera accepté par e-mail. Téléversez ce TP sur la page web du cours Moodle.

Question 1

But et description : Nous souhaitons de développer une application répartie (Web) permettant de gérer des comptes bancaires. Un serveur gérera tous les comptes bancaires et permettra à des clients de se connecter et d'effectuer les opérations suivantes : o Créer un compte en banque. o Consulter la position d'un compte o Ajouter une somme sur un compte o Retirer une somme d'un compte. Voici la déclaration des méthodes distantes :

void creer_compte(String id, double somme_initiale);

void ajouter(String id, double somme);

void retirer(String id, double somme);

Position position(String id); Où id est une chaîne permettant d'identifier un compte et Position est la classe suivante :

```
public class Position {
        public double solde;
        public Date derniereOperation;
        public Position(double solde) {
```

```
            this.solde = solde; this.derniereOperation = new Date();
        }
}
```

Dans cette application distribuée, vous avez :

1. Une interface Banque dérivant de Remote qui déclare les méthodes distantes.
2. La classe Compte qui permet de consulter la position d'un compte, d'ajouter et de retirer une somme à un compte.
3. Une classe BanqueImpl qui gère la partie serveur de notre application répartie. Les comptes seront stockés dans une Hashtable ou Hashmap qui permettra de retrouver un compte à partir de son identification.
4. Une classe BanqueClient qui gère la partie client de notre application répartie. L'application présentera un petit menu (sous forme textuelle) permettant d'accéder aux diverses méthodes.
5. Toutes les classes, les stubs générés (rmic banque.BanqueImpl), le serveur à lancer.
6. Plusieurs clients qui peuvent communiquer avec ce serveur.

Quoi faire :

a) Redévelopper cette application distribuée en Web en utilisant une technologie donnée : CGI Shell, CGI C/C++, PHP, Perl, Python, JavaScript, HTML, Java, etc.

b) Vous pouvez utiliser un Framework Web : Bootstrap, NodeJS, Angular, Symfony, Catalyst, etc.

c) Coté client : navigateur Web, formulaires, Javascript, Bootstrap, HTML, CSS, …

d) Coté serveur : implémentation des comptes bancaires, serveur web, PHP, JS, NodeJS, Perl, Java EE, …

e) Base de données : Relationnelle (MySQL, …), Objet ou NoSQL (Fichier, …)

Livrables :

Les livrables pour ce travail sont:

- Les codes documentés,
- La document (word, …) qui contient un
    - Noms des étudiants et codes permanents

o Manuel d'utilisation et

o Une trace d'exécution de dialogue.

Mode d'évaluation :

- Fonctionnement : 85%
- Documentation du code: 5%
- Trace d'exécution : 5%
- Manuel d'utilisation et de description (maximum 3 pages) : 5%

Voici l'application distribuée

```
import java.util.*;

/*
* Gère la position d'un compte.
* Note : il faut implanter l'interface Serializable pour qu'une instance
* de la classe Position puisse être envoyée par le serveur au client.
*/
public class Position implements java.io.Serializable {
      public double solde;    public Date
dernièreOperation;

      // constructeur   public
Position(double solde) {
this.solde = solde;
                  this.dernièreOperation = new Date();
      }
}
```

```
// interface Banque
// Ne pas oublier les "throws java.rmi.RemoteException" public
interface Banque extends java.rmi.Remote {   public void
creer_compte (String id, double somme) throws
java.rmi.RemoteException;

  public void ajouter(String id, double somme)throws java.rmi.RemoteException;

  public void retirer(String id, double somme)throws java.rmi.RemoteException;

  public Position position(String id) throws java.rmi.RemoteException;

}
```

```
import java.util.*;
```

```
/**
* Cette classe gère un seul compte
*/
public class Compte {

// La position courante du compte private
Position position;

// Constructeur
Compte(double solde_initial) {
  // on crée la position courante du compte   position =
new Position(solde_initial);
}

// ajoute une somme à la position courante // et
met à jour la date de dernière opération void
ajouter(double somme) {    position.solde += somme;
  position.derniereOperation = new Date(); }

// retire une somme à la position courante // et
met à jour la date de dernière opération void
retirer(double somme) {    position.solde -= somme;
  position.derniereOperation = new Date(); }

  // renvoie la position courante
  Position position() { return position; }
}
```

```
import bank.Banque;
import java.util.*;
import java.io.File;
import java.rmi.*;
import java.rmi.registry.LocateRegistry; import
java.rmi.server.*;

/**
* Implantation de la banque.
* La banque a une Hashtable qui contient tous les comptes des clients
*/
public class BanqueImpl extends java.rmi.server.UnicastRemoteObject implements
Banque {
// La Hashtable qui contient tous les comptes des clients
Hashtable clients;

// constructeur qui crée la Hashtable
public BanqueImpl() throws java.rmi.RemoteException {
  // ne pas oublier d'appeler le constructeur de la classe ancêtre
(UnicastRemoteObject)   super();
  clients = new Hashtable();
}
```

```
public void creer_compte(String id, double somme_initiale) throws
java.rmi.RemoteException {
  // on crèe un nouveau compte et on le rajoute dans la Hashtable
  Compte nouveau = new Compte(somme_initiale);   clients.put(id,
nouveau);
}
public void ajouter(String id, double somme) throws java.rmi.RemoteException {
  // on cherche le compte dans la Hashtable
  Compte c = (Compte)clients.get(id);
  // s'il existe on ajoute la somme   if
(c!=null)
      c.ajouter(somme);
}
public void retirer(String id, double somme) throws java.rmi.RemoteException {
  // on cherche le compte dans la Hashtable
  Compte c = (Compte)clients.get(id);
  // s'il existe on retire la somme   if
(c!=null)
      c.retirer(somme);
}
public Position position(String id) throws java.rmi.RemoteException {
  // on cherche le compte dans la Hashtable
  Compte c = (Compte)clients.get(id);
  // s'il existe on renvoie sa position   if
(c!=null)         return c.position();
  else // sinon on renvoie null     return
null;
  // Note : l'instance de la classe Position est envoyée au client
  // à travers le réseau
}  public static void main(String[] args) {
  // on positionne le gestionnaire de sécurité
  /// System.setSecurityManager(new RMISecurityManager());

  try {

        File f1= new File ("./bin");
        String
codeBase=f1.getAbsoluteFile().toURI().toURL().toString();
System.setProperty("java.rmi.server.codebase", codeBase);

        LocateRegistry.createRegistry(8989);

    // on crèe la banque et on l'enregistre grâce au Naming
    BanqueImpl banque = new BanqueImpl();     System.out.println("start
server");
    java.rmi.Naming.rebind("rmi://localhost:8989/MaBanque", banque);

 //java.rmi.Naming.rebind("rmi://arabica.info.uqam.ca:31337/MaBanq ue",
banque);
    System.out.println("L'objet serveur RMI 'MaBanque' est
enregistre");   }
  catch(Exception e) {
```

```
    e.printStackTrace();
  }
}
}
```

```
  import bank.Banque;
import bank.Position;
//import banque.BanqueImpl;

import java.io.*;

/**
* Le client. */
public class BanqueClient {


public static void main(String[] args) {   try {


    java.io.DataInputStream in = new
java.io.DataInputStream(System.in);     System.out.println("Client
start :");
    // On demande le proxy gérant l'objet serveur MaBanque
    // Le proxy implante l'interface Banque donc on peut le caster en
Banque
    Banque b =
(Banque)java.rmi.Naming.lookup("rmi://localhost:8989/MaBanque");
      // On appelle les méthodes de l'interface
        System.out.println("1 - Creer un compte");
        System.out.println("2 - Ajouter de l'argent");
        System.out.println("3 - Retirer de l'argent");
        System.out.println("4 - Position d'un compte");
        System.out.println("5 - Quitter");

        int choice = Integer.parseInt(in.readLine());
  String Nom;       float montant;
        while(choice != 5)
        {

              switch (choice) {
            case 1:  System.out.println("Entrez le nom du compte");
                              Nom = in.readLine();
                             System.out.println("Entrez un montant à
ajouter");
                              montant =
Float.valueOf(in.readLine()).floatValue();
```

```
                                b.creer_compte(Nom, montant);
break;
            case 2:  System.out.println("Entrez le nom du compte");
                                Nom = in.readLine();
                                System.out.println("Entrez un montant à
ajouter");
                                montant =
Float.valueOf(in.readLine()).floatValue();
                                b.ajouter(Nom, montant);
break;
            case 3:  System.out.println("Entrez le nom du compte");
                                Nom = in.readLine();
                                System.out.println("Entrez un montant à
retirer");
                                montant =
Float.valueOf(in.readLine()).floatValue();
                                b.retirer(Nom, montant);
break;
            case 4:  System.out.println("Entrez le nom du compte");
                                Nom = in.readLine();
                                Position p = b.position(Nom);
                                System.out.println("Position au
"+p.dernieresOperation+":"+ p.solde);
                           break;
             }
                 System.out.println("1 - Creer un compte");
                 System.out.println("2 - Ajouter de l'argent");
                 System.out.println("3 - Retirer de l'argent");
                 System.out.println("4 - Position d'un compte");
                 System.out.println("5 - Quitter");

                 choice = Integer.parseInt(in.readLine());
           }
  }
  catch(Exception e) {
    e.printStackTrace();
  }
}

}
```

Question 2

Afin de contribuer à la protection de l'environnement, une entreprise **TravelExpress** désire développer une application web de covoiturage. L'application permettra aux gens qui partagent les mêmes parcours de se connaître et de s'organiser pour faire du covoiturage. Ce projet aide les gens à s'organiser entre eux afin de partager leur intérêt pour l'environnement. Les utilisateurs (conducteurs) auront l'opportunité d'afficher leurs déplacements et d'offrir la possibilité aux gens de voyager avec eux. Les utilisateurs (passagers) intéressés à voyager n'auront qu'à réserver leur place et d'imprimer leur billet si nécessaire ou le télécharger sur leurs appareils. Une fois au point de rencontre du voyage, ils présenteront ce billet afin de certifier qu'une place leur est réservée.

Les fonctions principales de l'application de l'entreprise **TravelExpress** sont :

- ***Inscription des usagers*** : Cette fonction permet aux usagers de s'inscrire sur le site de covoiturage. L'usager introduit ses informations personnelles ainsi qu'un nom d'usager, une adresse électronique, un numéro de téléphone et un mot de passe qui lui permettront d'ouvrir une session.
- ***Publication d'un voyage*** : L'usager dont la session est ouverte doit être en mesure de publier les itinéraires de ses voyages. Chaque itinéraire indiquera aussi l'horaire, les dates et les fréquences de chacun de ses voyages.
- ***Préférences*** : L'usager (conducteur) indique dans son profil ses préférences en indiquant ce qui tolère aux passagers et ce qui sera interdit. Si l'usager est comblé, une indication dans son profil doit permettre de savoir.
- ***Recherche*** : Cette fonction permettra aux usagers de faire des recherches selon certains critères, de visualiser les voyages disponibles, et d'obtenir les coordonnées des personnes intéressées.
- ***Annulation d'un voyage*** : L'administrateur du système peut annuler un voyage après la demande de conducteur et il doit informer les passagers de cette annulation. Un passager peut aussi annuler un voyage (une réservation) et aviser le conducteur et l'administrateur de l'entreprise. Un conducteur peut remplir un voyage Complet s'il ne veut pas de passagers. Mais un conducteur ne peut pas annuler un voyage s'il existe un passager dans sa liste de réservation sans communication avec l'administrateur de l'entreprise TravelExpress. Après trois annulations, un passager et un conducteur pourront être exclus temporairement (3 mois) des services offerts par l'entreprise. Cependant, un administrateur peut changer cette situation d'un passager.

- ***Inscription de conducteur*** : les conducteurs s'inscrivent comme membres pour annoncer un départ (voyage). Ils peuvent être passagers et conducteurs en même temps.
- ***Réservation une ou plusieurs places par passager*** : Un passager peut réserver une ou plusieurs places (au maximum le nombre de places disponibles du véhicule en question). L'usager doit payer au conducteur le montant spécifié de la page web par le conducteur selon la distance.

Travail à faire :

1. (80%) Implémenter les fonctionnalités de l'application et donner le code source bien documenté :
    a. Des pages HTML, CSS XML, Javascript, Ajax, Bootstrap, React, ...
    b. Du code serveur : NodeJS, Symfony, Spring, JSP, Servlets, JSF, REST, SOAP, RUBY, Shell, C/C++, PHP, PERL, Python, C#, ...
    c. Base de données (relationnelle, Objet ou fichier)
    d. D'autres classes si nécessaire
    e. etc.
2. (20%) Produire une documentation du système incluant :
    a. Introduction et description du système
    b. Schémas et diagrammes de conception (classes de l'application)
    c. Le guide de l'usager
    d. Le test (trace d'exécution)
    e. Références (ex. page web, livre, article, ... utilisés)

Conditions exigées:

- L'application Web pourrait être réalisée en utilisant NodeJS, Symfony, Spring, servlet-JSP, Java-JSF, Java EE, Ruby on rails, C++, PHP, C#, .Net, ...
- Notez le mapping entre les objets (Java, PHP, C#, C++, Ruby, PHP, ...) la base de données (Oracle, mysql, ...) pour la gestion de la persistance
- Ce projet doit être exécutable dans les divers environnements et l'application doit être accessible de l'extérieur par l'Internet. Il faut fournir l'URL pour accéder à votre application.
- Le rapport et les fichiers de votre application doivent être remis au démonstrateur. Chaque étudiant d'une équipe doit nous montrer sa contribution dans ce TP. L'exécution de ce TP sera faite en classe. Des pénalités de 10% par jour seront appliquées pour tout travail remis après cette date.

Question 3 :

Il s'agit de concevoir et de mettre en œuvre application web pour le cours 8INF349 : Programmation Web. Voici quelques exigences :

1. La conception du site doit être conforme à un seul décorateur de votre choix ;
2. Il est important de séparer la structure du formatage en utilisant les feuilles de style CSS ;
3. Les étudiants, le démonstrateur et le professeur peuvent se connecter avec leur code ;
4. Le démonstrateur et le professeur ont le droit d'ajouter des sections (e.g. Notes de cours, Ateliers, Examens, Travaux pratiques etc.) ;
5. Une fois connecté, le démonstrateur et le professeur auront accès à un lien permettant de visualiser la liste des étudiants inscrits;
6. Dans chaque section, on peut définir un titre, une description (facultative) et ajouter des ressources et des activités ;
7. Une ressource peut être un hyper lien html (vers un fichier, une URL, etc.) ;
8. On distingue deux types d'activités (i) ajouter un devoir et (ii) remettre un devoir ;
9. On peut définir une date de disponibilité dans le futur (facultative) et de remise pour chaque devoir ;
10. Les étudiants peuvent consulter les ressources et remettre les devoirs ;
11. Le démonstrateur et le professeur peuvent éditer (ajouter, modifier et supprimer) le contenu du site ;
12. Un utilisateur authentifié peut envoyer un courriel au groupe via un lien dans le décorateur ;
13. Le professeur peut attribuer le rôle d'étudiant ou de démonstrateur à un nouvel utilisateur;
14. Un utilisateur ne peut pas accéder à la page du cours sans être authentifié.
15. Une fois connecté, chaque utilisateur disposera d'un lien «déconnecter » qui permet de se déconnecter. Ce lien conduira vers une page annonçant à l'utilisateur qu'il est déconnecté.

L'objectif du second travail est d'implémenter les exigences fonctionnelles du site.
À la fin de ce livrable, il est important que le site que vous allez fournir soit fonctionnel.
Voici quelques contraintes de conception et de mise en œuvre :

1. Vous devez fournir un fichier README. Ce dernier doit spécifier le point d'entrée de votre site et toute autre information que vous jugez utile (e.g. les navigateurs supportés) ;
2. La conception du site doit être conforme à un seul décorateur de votre choix ;
3. Il est important de séparer la structure du formatage en utilisant les feuilles de style CSS ;
4. Tous les chemins vers les ressources (e.g. fichiers, images, etc.) doivent être relatifs ;
5. Un utilisateur ne peut pas accéder directement (e.g. via un raccourci) à une page du site, autre que la page d'authentification, à moins qu'il soit authentifié ;
6. Vos pages doivent être légères i.e. éviter les pages qui permettent de gérer plusieurs types d'opérations (e.g. affichage, ajout et modification) avec beaucoup de logiques Javascript ;
7. Votre décorateur (i.e. conception ou gabarit) ne doit inclure que la logique applicable à toutes les pages ;
8. Vous pouvez utiliser les technologies vues en classes ou autres technologies : C/C++, PHP, Python, HTML, Javacript, XML, React, Ajax, Bootstrap, Java, NodeJS, etc.

Évaluation de la mission :

- 80% Conception et implémentation de ces exigences, description de votre solution,
- 20% Introduction et description du système, guide d'utilisation et trace d'exécution

Question 4 :

Ce travail consiste à concevoir et développer une application web pour calculer l'indice de masse corporelle IMC d'une personne. Vous allez utiliser des services Cloud tels que le stockage (par exemple, Docker desktop, Docker Hub, Azure MS, Amazon Cloud, Google) et des systèmes de gestion de bases de données locales (MSSQL, MySQL, etc.) et des services d'hébergement (par exemple nginx, Glassfish, Tomcat, IIS, Apache), vous pouvez également implémenter et exécuter cette application localement avec Docker desktop. L'étudiant devra choisir un langage de programmation et un framework.

Évaluation de la mission :

- (80 %) Conception et Implémentation des fonctionnalités de l'application et fournir un code source bien documenté. Je vous laisse le choix de sélectionner les technologies (ou combinaison) pour développer cette application :
  - L'application peut être réalisée à l'aide d'ASP.NET, JAVA, Python, HTML, CSS, XML, Node.JS, Ruby, PHP, C++, C#, React, Angular JS, Xamarin, Android Studio, Flutter ou d'autres technologies.
  - La connexion entre le Code (Java, C#, C++, PHP, etc.) et la Base de données (Oracle, MSSQL, MySQL, etc…).
  - Le travail peut être exécuté sur plusieurs plateformes (Windows, Linux, Android, iOS, etc.) et l'application doit être accessible de l'extérieur via Internet ou Telnet ou en local (serveur web local sur la même machine). Vous devez fournir l'URL pour accéder à votre application.
- (20%) Présentation et documentation comprenant :
  - Introduction et description du système
  - Le guide de l'utilisateur
  - Le test (exécution de code)

Bon courage !