Filière :	Développement des Systèmes d'Information	Durée :	4 Heures
Épreuve :	Développement d'Applications Informatiques – DAI -	Coefficient :	45

CONSIGNES

- ✓ **Le sujet comporte 4 dossiers ;**
- ✓ **chaque dossier doit être traité dans une feuille séparée.**

Barème de notation

Dossier I : Gestion des filiales	14 points
Dossier II : Serveur d'application	08 points
Dossier III : Suivi des formations	10 points
Dossier IV : Site Web de la société	08 points
Total	**40 points**

- ✓ **Il sera pris en considération la qualité de la rédaction lors de la correction.**
- ✓ **Aucun document n'est autorisé.**

ÉTUDE DE CAS : SOCIÉTÉ DE SERVICES EN INGÉNIERIE INFORMATIQUE

MarocNET est une **S**ociété, de **S**ervices en **I**ngénierie **I**nformatique (*SSII*), qui s'est développée depuis une dizaine d'années. Elle dispose de différentes filiales spécialisées à travers le Maroc :

- ***CloudNET*** située à Casablanca : spécialisée dans l'hébergement de données et de systèmes d'informations dématérialisés. Elle dispose de différents sites hébergeant ses serveurs.
- ***FormaNET*** située à Fès : spécialisée dans la formation et proposant conseils et audits,
- ***WebNET*** située à Rabat : spécialisée dans l'identité des entreprises sur Internet et les réseaux sociaux,

L'informatique dématérialisé est un domaine encore complexe et trop récent pour les entreprises. C'est pourquoi **MarocNET**, à l'aide de sa filiale ***CloudNET***, propose différentes solutions d'hébergements de SI pour ses clients :

- ***Infrastructure*** : service qui offre un accès à un parc informatique virtualisé. Des machines virtuelles sur lesquelles le client peut installer un système d'exploitation et des applications.
- ***Plateforme*** : le système d'exploitation et les outils d'infrastructure sont sous la responsabilité de ***CloudNET***. Le client a le contrôle des applications et peut ajouter ses propres outils.
- ***Software :*** les applications sont mises à disposition des clients. Le client n'a pas à se soucier de la configuration, des mises à jour et d'assurer la disponibilité du service.

Vous avez été détaché(e) de l'équipe informatique de **MarocNET** pour répondre aux différents besoins des clients et intervenir sur tout le processus d'externalisation de leur SI.

DOSSIER I : GESTION DES FILIALES *(14 pts)*

Une partie du système d'information de **MarocNET** est modélisée par le diagramme de classes suivant :

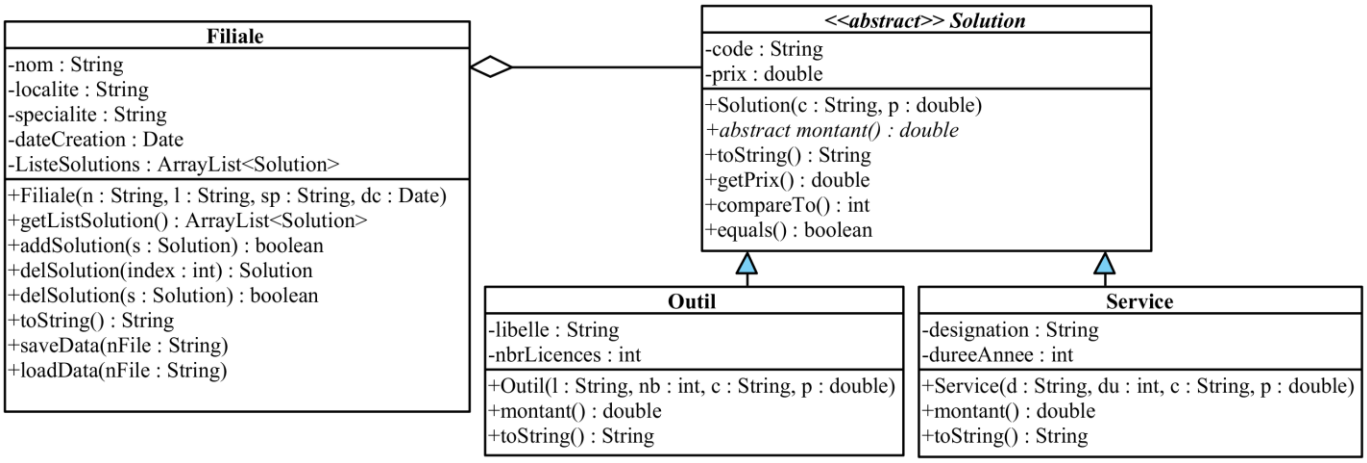

Figure 1 : Diagramme de classes

NB : Tous les attributs sont privés et les types sont indiqués dans le diagramme de classes.

1. Implémenter la classe abstraite **« Solution »**, sachant que cette classe doit :

- Implémenter les interfaces « ***Serializable*** » et « ***Comparable*** »
  Contenir la méthode abstraite **« `montant()` »** et retournant le montant total d'une solution. *(0,5 pt)*
- Avoir un constructeur avec arguments qui initialise tous les attributs de la classe. Une exception personnalisée **`ErreurPrix`** (*à créer*) est générée si le prix est négatif. On doit récupérer un message d'erreur sous la forme "*Erreur de prix de la solution* ". *(1 pt)*
- Définir l'accesseur **`getPrix()`**. *(0,5 pt)*
- Définir la méthode « **`toString`** », afin de retourner une chaîne porteuse d'informations sur une solution, la chaîne aura la forme suivante : *(0,5 pt)*

```
Code : xxxx, Prix : xxxx
```

- Redéfinir la méthode « **`compareTo`** » afin de comparer deux solutions selon leurs prix. *(0,5 pt)*
- Redéfinir la méthode « **`equals`** » afin de comparer deux solutions selon leur code. *(0,5 pt)*

2. Implémenter la classe **« `Service` »**, sachant qu'elle doit :

- Hériter de la classe **« `Solution` »**. *(0,5 pt)*
- Contenir un constructeur avec paramètres permettant d'initialiser tous les attributs. *(0,5 pt)*
- Contenir la méthode **`montant( )`** permettant de calculer le montant total : *(1 pt)*

```
le Montant=le prix * la durée en années
Si la durée en années >=3 alors il y a une réduction de 20% du montant
```

- Redéfinir la méthode « **`toString`** », afin de retourner une chaîne porteuse d'informations sur une solution, la chaîne aura la forme suivante : *(0,5 pt)*

```
Code : xxxx, Prix : xxxx, Désignation :xxxx, Durée (années) : xxxx
```

3. Implémenter la classe **« `Outil` »**, sachant qu'elle doit :

- Hériter de la classe **« `Solution` »**. *(0,5 pt)*
- Contenir un constructeur avec paramètres permettant d'initialiser tous les attributs. *(0,5 pt)*
- Contenir la méthode **`montant( )`** permettant de calculer le montant total : *(0,5 pt)*

```
le Montant=le prix * le nombre de licences
Si nombre de licences>=10 alors il y a une réduction de 10% du Montant
```

4. Implémenter la classe« **`Filiale`** » sachant qu'elle doit :

- Contenir un constructeur adéquat avec 4 paramètres permettant d'initialiser tous les attributs (*la liste* « ***`listeSolutions`*** » *sera initialisée à vide*). *(0,5 pt)*
- Définir l'accesseur **`getListeSolutions ().`** *(0,5 pt)*
- Contenir les méthodes suivantes :
  - **`addSolution(Solution)`** : permet d'ajouter une solution et retourne l'état de l'opération. *(1 pt)*
  - **`delSolution(int)`** : permet de supprimer une solution de la collection en se basant sur un index et retourne la solution qui vient d'être supprimée. La vérification de la validité de l'index est indispensable. *(0,75 pt)*
  - **`delSolution(Solution)`** : permet de supprimer une solution de la collection en se basant sur un objet « Solution » et retourne l'état de la suppression. *(0,75 pt)*
  - **`toString() :`** permet de retourner une chaîne porteuse d'informations sur une filiale, la chaîne aura la forme suivante : *(0,5 pt)*

```
nom : xxxx, Date Création :jj/mm/aaaa
La liste des solutions:
1- Code : xxxx, Prix : xxxx
2- Code : xxxx, Prix : xxxx
.............
```

  - **`saveData(String)`** : permet de sauvegarder, dans un fichier d'objets, les objets de notre collection dont le prix est supérieur à 300, le nom de fichier de stockage est reçu comme argument. *(1 pt)*
  - **`loadData(String) :`** permet de charger la collection des solutions d'une filiale depuis un fichier d'objets. *(1,5 pt)*

DOSSIER II : SERVEUR D'APPLICATION *(8 pts)*

Dans un serveur d'application, on veut créer une classe permettant la connexion à la base de données « **`SolutionsInfo`** » de la société **MarocNET** en utilisant l'API **`JDBC`**. La structure de cette classe est la suivante :

Connect
- cn : Connection -A : Connect

A. PROGRAMMATION SERVEUR

La partie serveur est constituée de 2 classes :

- **`ServeurApplication`** : qui répond aux demandes des clients. (**Le code de cette classe n'est pas demandé**)
- **`Connect`** : qui sert d'intermédiaire entre la classe précédente et la base de données.

La base de données « **`SolutionsInfo`** » se trouve dans un serveur MySQL ayant l'adresse IP : **`81.172.0.2`** et le port d'accès : **`3306`**.

On se limite à la gestion des solutions (Services). La structure de la table « **`Service`** » est la suivante :

	#	Colonne	Type	Interclassement	Attributs	Null
☐	1	CODE	varchar(10)	latin1_swedish_ci		Non
☐	2	PRIX	double			Non
☐	3	DESIGNATION	varchar(40)	latin1_swedish_ci		Non
☐	4	DUREEANNEE	int(11)			Non

Figure 2 : Structure de la table « Service »

```
public class Connect {
     private Connection cn;                  //objet de connexion
     private static Connect A=null;

     private  Connect() {
          ...................
     }
     public static Connect  getInstance(){
             if(A==null) A=new Connect();
              return A;
        }

     public  ResultSet  lireServices(double prix){
     ...................
      }
     public ArrayList<String> toArray(ResultSet rs){
      ...................
      }
public void addService(String code, double Prix, String Des, int duree)
     {
      ...................
     }
}
```

1. Donner le code du constructeur permettant d'établir une connexion avec notre base de données.
   On donne:
   ```
   URL = "jdbc:mysql://81.172.0.2: 3306/ SolutionsInfo"
   User="Serveur"
   Password="112233"
   ```
   En cas d'échec de connexion, un message d'erreur doit être affiché. *(1 pt)*
2. Une méthode « **`lireServices`**» permet de retourner un objet « ***ResultSet*** » contenant tous les services ayant le prix reçu comme argument. En cas d'échec, elle retournera « ***null*** ». *(1 pt)*
3. Une méthode « **`addService`**» permet d'ajouter un enregistrement à la table service. En cas d'échec, un message d'erreur sera affiché. *(1 pt)*
4. Une méthode « **`toArray`**» permet de retourner un **`ArrayList`** de String à partir d'un objet **ResultSet** reçu comme argument. Chaque String contient les valeurs des champs d'un enregistrement sous la forme :
   `Code: Désignation, Prix.` *(1 pt)*
5. On considère le code de test suivant: *(0.5 pt)*
   ```
   ResultSet rs=Connect.getInstance().lireServices(650);
   ArrayList<String> L= Connect.getInstance().toArray(rs);
   System.outprintln(L);
   ```
   Après l'exécution de ce code, combien de fois l'attribut **`cn`** de la classe **`Connect`** a été instancié ? Justifier votre réponse.

B. PROGRAMMATION CLIENT

Cette partie permet aux entreprises clientes de saisir un « **`prix`** » afin de recevoir la liste des services ayant un prix égale au prix proposé. La classe « **`Client`** » se présente comme suit :

Client
- IPServeur : String - PortServeur : int **- s :** Socket - pw : PrintWriter - ois : ObjectInputStream
+ Client(…) + connecter() :boolean + demandeListeServices(…) : ArrayList<String>

```
public class Client {
     private String IPServeur;
     private int PortServeur;
     private Socket s=null;
     private PrintWriter pw=null;
     private ObjectInputStream ois=null;
     public Client(String IP, int Port){
           this.IPServeur=IP;
           this.PortServeur=Port;
     }
     public boolean connecter(){
           ……
     }
     public ArrayList<String> demandeListeServices(double prix){
           ……
     }
}
```

1. Écrire la méthode « **connecter** » qui permet d'établir la connexion **TCP/IP** avec le serveur (**ServeurApplication**) et initialiser les objets **pw** et **ois**. La méthode retourne « **true** » en cas de succès et « **false** » dans le cas échéant. *(1 pt)*
2. Écrire la méthode « **demandeListeServices** » qui envoie au serveur un prix, reçoit depuis le serveur la liste des services ayant ce prix et retourne la collection reçue. *(1 pt)*
3. L'interface graphique suivante, permet d'envoyer au serveur un prix et afficher la collection reçue :

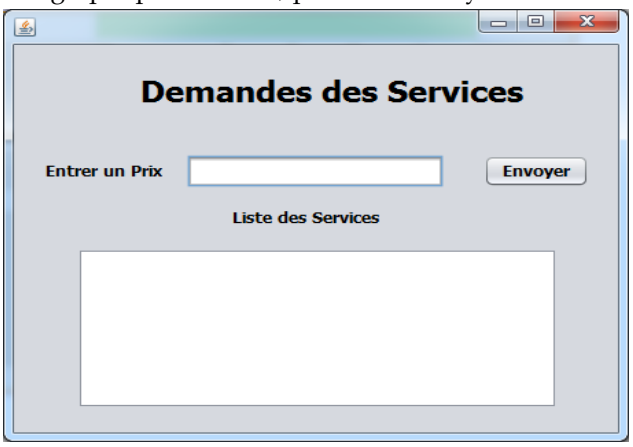

Figure 3 : Interface Graphique du client

Objet	Name	Texte
JTextField	txtPrix	
JList	liste	
JButton	cmdEnvoyer	Envoyer

Compléter le code de la méthode événementielle du bouton « **cmdEnvoyer** » qui permet de : *(1.5 pts)*

- Envoyer au serveur le prix saisi dans la zone de texte « **txtPrix** » ,
- Lire la collection des services reçue depuis le serveur,
- Afficher cette collection dans la liste « **liste** ».

```
public class ihmClient {
    private Client clt = new Client("123.43.12.22", 1111);

    private void formWindowOpened(java.awt.event.WindowEvent evt) {
        clt.connecter();
    }
    private void cmdEnvoyerActionPerformed(java.awt.event.ActionEvent evt) {
         //Le code à compléter.
        ....
    }
}
```

DOSSIER III : SUIVI DES FORMATIONS *(10 pts)*

Le responsable de la filiale ***FormaNET***, souhaite disposer d'une application informatique sous VB.Net, permettant le suivi des formations. Le client (société) peut choisir la formation convenable à ses employés et attendre l'accord ou le refus du responsable de formation.

Le responsable, suite à des critères bien déterminés, accepte ou refuse la demande d'employé.

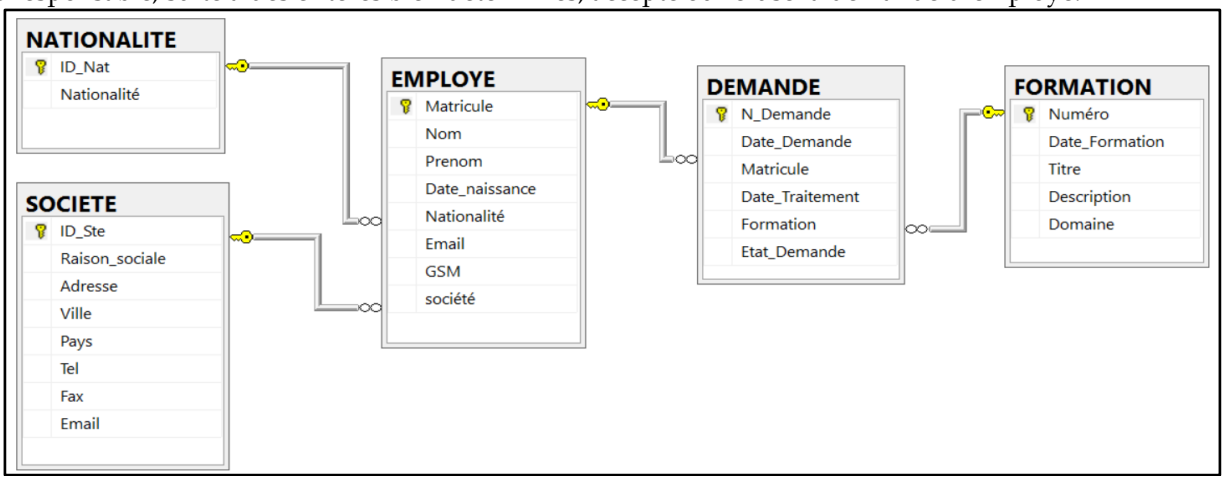

Figure 4 : Partie du MLDR de la base de données « **`BD_Formations`** ».

Remarque : Dans la table « **`DEMANDE`** »:

- Le champ « **`Etat_Demande`** » permet de connaître l'état de traitement de la demande. Ce champ prend l'une des valeurs suivantes :"*en cours*" (valeur par défaut), " N*on acceptée* ", "M*ise en attente*", "*Acceptée* ".
- Le champ « **`Date_Traitement`**» prend la valeur **`null`** tant que le champ « **`Etat_Demande`** » est « *en cours* ».

La feuille permettant de lister les demandes est la suivante :

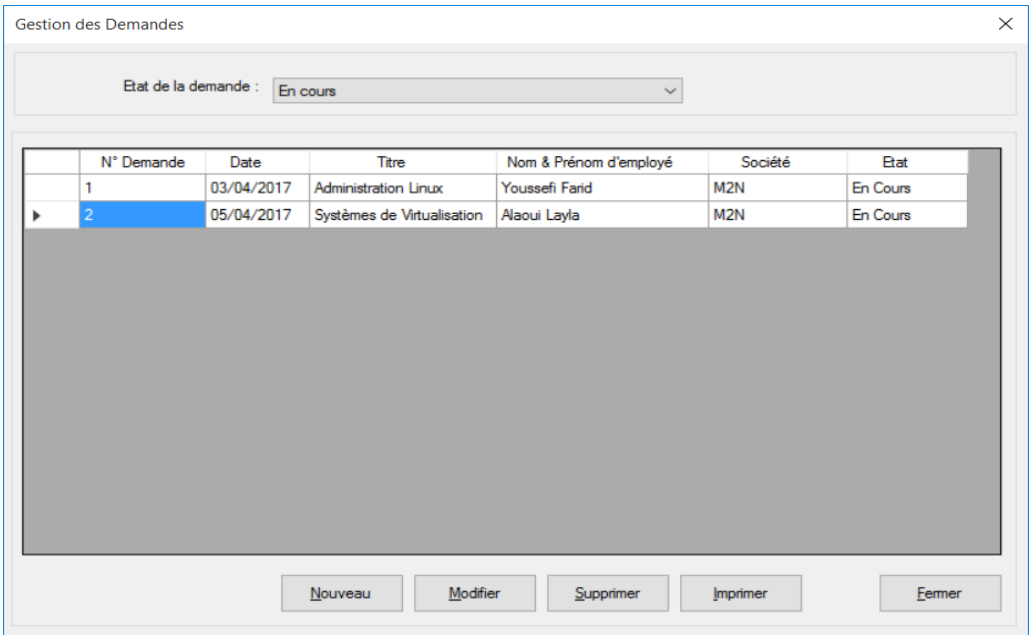

Figure 5 : Formulaire '***en mode exécution'*** affichant les demandes réalisées.

Objet	Name	Texte
Boutons de commandes	CmdNouveau	Nouveau
	CmdModifier	Modifier
	CmdSupprimer	Supprimer
	CmdImprimer	Imprimer
	CmdFermer	Fermer

Objet	Name
ComboBox	CmbEtat
DataGridView	DGListe

1. Écrire le code de la procédure « **Connexion** » déclarée dans un module pour se connecter à la de base de données nommée « **BD_Formations**» et logée dans le serveur « **Srv-FormatNet** ». Gérer les exceptions. *(1,5 pts)*
2. Écrire le code de la procédure « **Lister_demandes** », permettant d'afficher la liste des demandes dans l'objet « **DataGridView** » nommé « **DGListe** » selon l'état de la demande donnée en argument . *(2,5 pts)*

```
Public Sub Lister_demandes  (ByVal Etat As String)
      ....
End Sub
```

3. Écrire le code de la procédure « **Supprimer_demande** » permettant de supprimer une demande donnée en argument. *(1 pt)*

```
Public Sub Supprimer_Demande (ByVal N_Demande As Integer)
      …
End Sub
```

4. Pour ajouter une nouvelle demande on utilise le formulaire suivant :

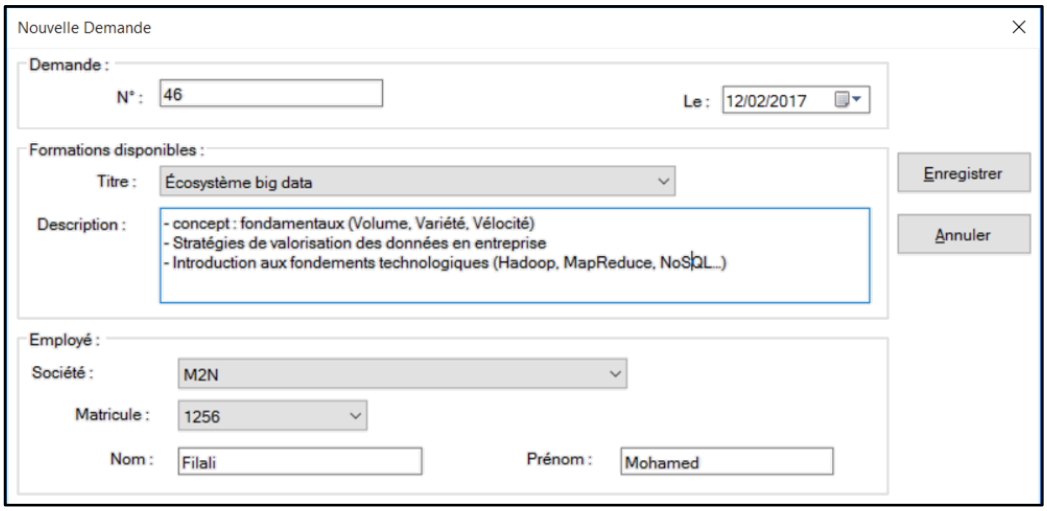

Figure 6 : Formulaire '***en mode exécution***' permettant d'ajouter une nouvelle demande.

Objet	Name
Étiquète (Label)	LblNum_Demande
	Lbl Description
	Lbl Nom
	LblPrenom
Boutons de Commande	CmdEnregistrer
	CmdAnnuler

Objet	Name	DropDownStyle
ComboBox	CmbSociete	DropDownList
	CmbTitre	DropDownList
	CmbMatricule	DropDownList

Objet	Name	Format
DateTimePicker	Date_Demande	Short

a) Écrire le code d'une fonction nommée « **Generer_Code** » permettant d'incrémenter le numéro de la demande pour pouvoir l'afficher dans l'objet « **LblNum_Demande** ». *(1 pt)*

```
Public Function Generer_Code() As Integer
      ……..
End Function
```

b) Donner le code permettant d'afficher, dans l'objet « **LblDescription** », la description de la formation dont on sélectionne son titre à l'aide de **Combobox** « **CmbTitre** ». On suppose que ce dernier est déjà rempli. *(1 pt)*

c) Écrire le code de la procédure « **Remlpir_Cmb_Societe** » permettant de remplir le **ComboBox** «**CmbSociete**» par les raisons sociales des sociétés à partir de la Table « **SOCIETE**». *(1 pt)*

d) Écrire le code correspond à l'événement **Clic** du bouton « **Enregistrer** » permettant d'ajouter une nouvelle demande. La vérification de validité des données du formulaire n'est pas demandée. *(2 pts)*

DOSSIER IV : SITE WEB DE LA SOCIÉTÉ *(8 pts)*

La Société **MarocNet** souhaite informatiser ses activités. Elle propose de mettre en place un site web dynamique. Ce site va permettre de gérer les entreprises ainsi que leurs filiales. Dans ce contexte, une base de données MySQL «**gssii**» contient des tables suivantes :

```
ENTREPRISE(ide,Raison_Sociale, capital)
FILIALE(idf,nom,localité, specialite, dateCreation,#ide)
```

1. Créer le fichier « **connexion.php**» qui permet la connexion à cette base de données en utilisant les paramètres par défaut. *(1 pt)*
2. Compléter le code de la page d'accueil « **index.php** » permettant de charger la liste déroulante par les raisons sociales des entreprises à partir de la table « **Entreprise** ». *(2 pts)*

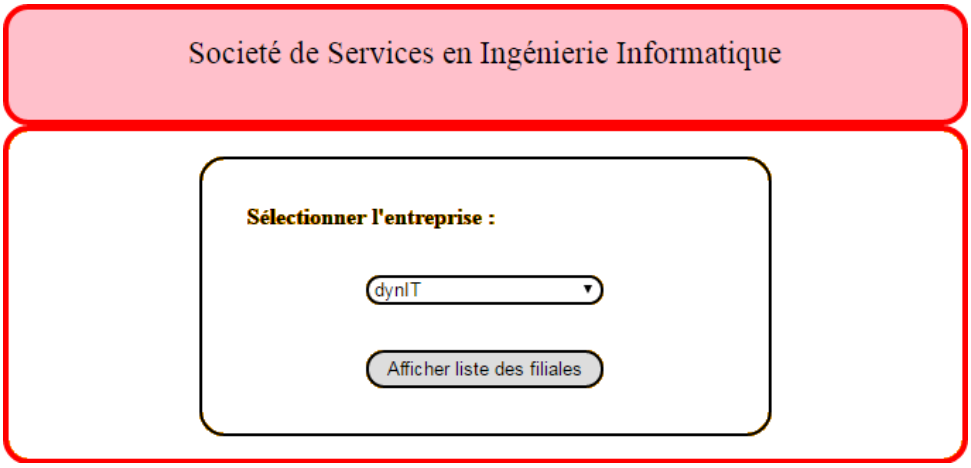

Figure 7 : Page « **index.php** »

```
<html>
  <head>
    <style type="text/css">
    /* Code non demandé */
    .........
    </style>
  </head>
  <body>
     <div id="P">
       <div id="titre">
       Société de Services en Ingénierie Informatique
       </div>
       <div id="data">
       <form id="f" name="f" method="post" action="filiale.php">
       <!-- Partie à compléter -->
         ..............................
     </form>
       </div>
     </div>
  </body>
</html>
```

3. La page web nommée «**`filiale.php`** » contient un tableau de données des filiales de l'entreprise sélectionnée dans la page «**`index.php`**» avec la possibilité de supprimer une filiale via un lien hypertexte. *(3 pts)*

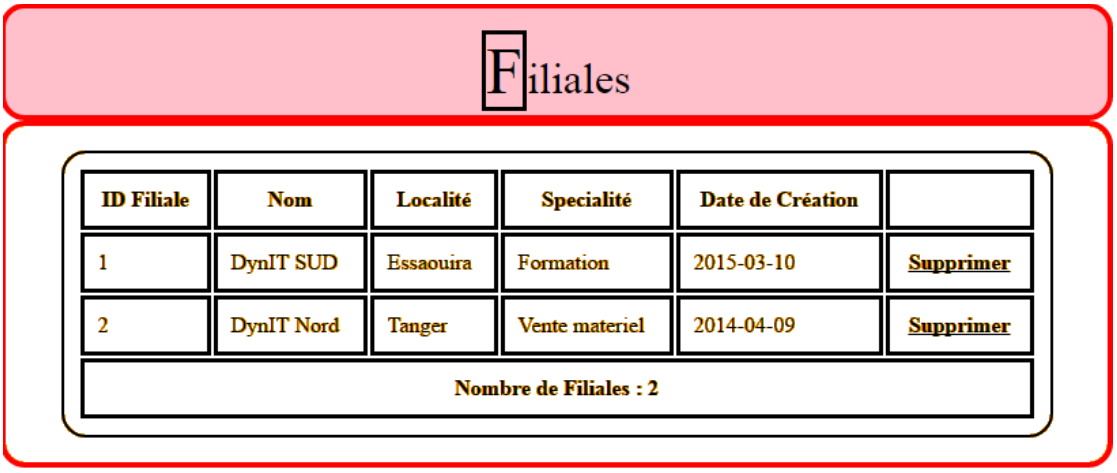

Figure 8 : Page « **`filiale.php`** »

Compléter le code de la page des filiales « **`filiale.php`** ». Le lien « supprimer » effectue une redirection vers le fichier « **`supprimer.php`** » qui s'occupera de la suppression.

```
<html>
  <head>
      <style type="text/css">  /* Code non demandé */  </style>
  </head>
  <body link="black">
     <div id="P">
          <div id="titre">
          Filiales
          </div>
          <div id="data">
          <table>
          <tr><th>ID
     Filiale</th><th>Nom</th><th>Localité</th><th>Specialité</th>
     <th>Date de Création</th><th></th></tr>
     <?php
     //Code PHP à compléter
     ..........
           echo "<tr><th colspan='6'>Nombre de Filiales : $n</th></tr>";
           ?>
          </table>
          </form>
          </div>
     </div>
  </body>
</html>
```

4. Écrire le code de la page « **`supprimer.php`** » qui reçoit l' « **`Id`** » de la page « **`filiale.php`** », supprime la ligne correspondante de la table et retourne vers la page « **`filiale.php`** ». *(2 pts)*