Royaume du Maroc
Ministère de l'Education Nationale
de la Formation Professionnelle
de l'Enseignement Supérieur et de la Recherche Scientifique
Centre National de l'Évaluation, des Examens et de l'Orientation

Examen National du Brevet de Technicien Supérieur
Session Mai 2019

– Sujet –

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

CONSIGNES

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

Dossier 1 : Système d'information de recyclage	12 points
Dossier 2 : Consultation du mapping des adresses IP par pays	10 points
Dossier 3 : Inventaire des logiciels installés	10 points
Dossier 4 : Validation de la demande de ramassage des déchets électroniques	8 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 : GESTION DE RECYCLAGE DES COMPOSANTS ÉLECTRONIQUES

NAT-AMI est une société installée à Rabat et spécialisée dans la collecte, le traitement et le recyclage de **d**échets d'**é**quipements **é**lectriques et **é**lectroniques (***DEEE***).

Une récente directive nationale impose désormais aux sociétés industrielles, le traitement et la valorisation des **DEEE**. Elle constitue une nouvelle perspective de croissance de l'activité de la **NAT-AMI**. La société regroupe, outre les services administratifs et de direction, une unité de recherche et développement intégrant des laboratoires pilotes chargés de tester de nouveaux procédés de recyclage et une usine de retraitement des composants électroniques de micro-ordinateurs.

Le parc informatique de la **NAT-AMI** a pris de l'ampleur au fil des années de manière assez désordonnée. Il est nécessaire de le faire évoluer pour en rationaliser la gestion et accompagner la croissance de la **NAT-AMI**.

Vous travaillez en tant que technicien(*ne*) chez la société et vous êtes chargé de gérer et faire évoluer l'ensemble des ressources informatiques. Ces tâches confiées sont organisées sous-forme de dossiers :

- Dossier 1 : Système d'information de recyclage.
- Dossier 2 : Consultation du mapping des adresses IP par pays.
- Dossier 3 : Inventaire des logiciels installés.
- Dossier 4 : Demande de ramassage des déchets électroniques.

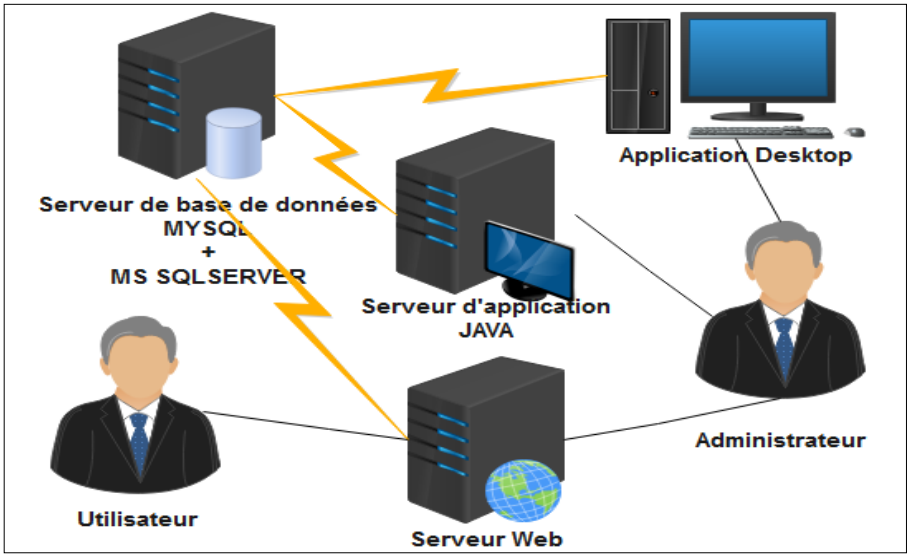

Figure 1 : Architecture de LA NAT-AMI

DOSSIER 1 : SYSTEME D'INFORMATION DE RECYCLAGE DE LA NAT-AMI *(12 pts)*

On désire mettre en place un système d'information de recyclage de la NAT-AMI, une partie de ce système est modélisée par le diagramme de classes suivant :

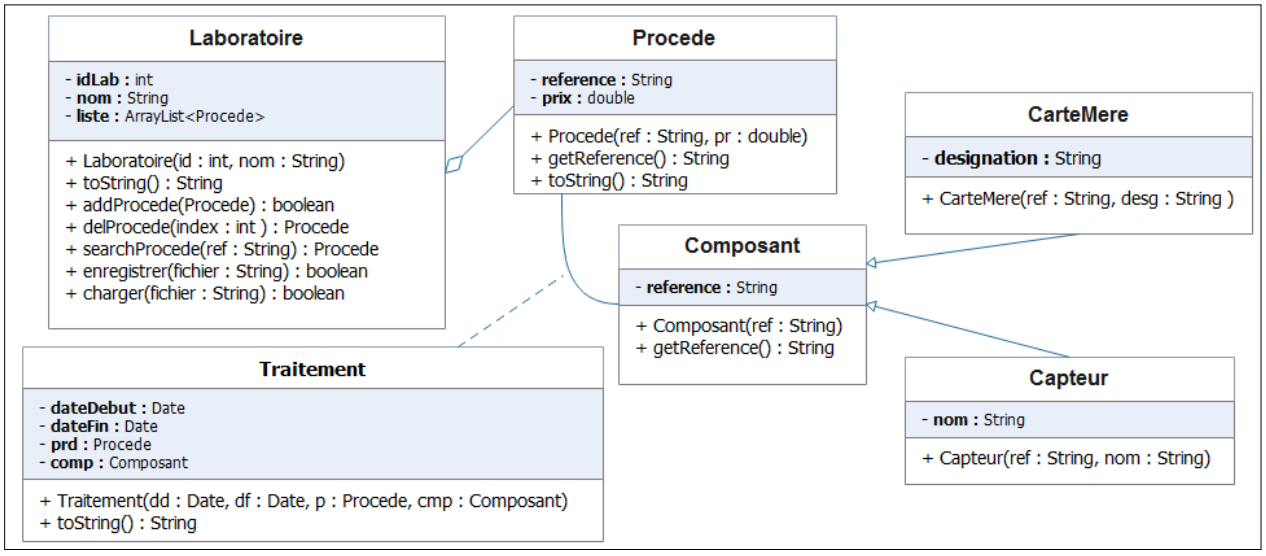

Figure 2 : Diagramme de classes de l'application

Remarque : Tous les attributs sont privés et toutes les méthodes sont publiques.

1. Classe « **Composant** ».

1.1. Implémenter cette classe. *(0,5 pt)*

1.2. Définir un constructeur avec un seul argument. *(0,5 pt)*

2. Classe « **CarteMere** ».

2.1 Implémenter cette classe. *(0,5 pt)*

2.2 Définir un constructeur avec deux paramètres. *(0,5 pt)*

3. Classe « **Procede** ».

3.1 Implémenter cette classe dont les objets sont « **Serializable** ». *(0,5 pt)*

3.2 Créer la classe d'exception « **ErreurPrix** » qui permet de récupérer un message d'erreur sous la forme "*Erreur prix : -xxx*" où (-xxx) est un prix négatif. *(0,5 pt)*

3.3 Proposer un constructeur avec deux paramètres permettant d'initialiser tous les attributs. Le constructeur lève l'exception « **ErreurPrix** » si le prix est négatif. *(1 pt)*

3.4 Définir l'accesseur « **getReference() »**. *(0,5 pt)*

La méthode « **toString() »** retourne une chaîne porteuse d'informations sur un procedé sous la forme suivante : (*Code non demandé*)

```
Référence : XXXX, Prix : XXXX
```

4. Classe « **Traitement** ».

4.1 Implémenter cette classe. *(0,5 pt)*

4.2 Proposer un constructeur avec quatre paramètres permettant d'initialiser tous les attributs. *(0,5 pt)*

4.3 Redéfinir la méthode « **toString() »** permettant de retourner une chaîne porteuse d'informations sur un traitement sous la forme suivante : *(0,5 pt)*

```
Date début : jj/mm/aaaa, date fin : jj/mm/aaaa, procédé : [XXXX], Ref composant : XXXX
```

5. Classe « **Laboratoire** ».

5.1 Implémenter cette classe. *(0,5 pt)*

5.2 Un constructeur avec deux paramètres permettant d'initialiser tous les attributs (*la liste des procédés doit être instanciée*). *(0,5 pt)*

5.3 Redéfinir la méthode « **toString()** » permettant de retourner une chaîne porteuse d'informations sous la forme suivante : *(0,5 pt)*

```
Id Labo : XXXX, Nom : XXXX
Liste des procédés :
Référence : XXXX, Prix : XXXX
……
Référence : XXXX, Prix : XXXX
```

5.4 Donner le code des méthodes suivantes :

a. **addProcede(Procede)** : permet d'ajouter un procédé et retourne l'état de l'opération. *(0,5 pt)*

b. **delProcede(int)** : permet de supprimer un procédé de la collection en se basant sur un index et retourne le procédé qui vient d'être supprimé. La vérification de la validité de l'index est indispensable. *(0,5 pt)*

c. **searchProcede(String)** : permet de rechercher un procédé donné par sa référence dans la collection et retourne un objet « **Procede** ». *(1,5 pt)*

d. **enregistrer(String)** : permet de sauvegarder la collection des procédés d'un laboratoire dans un fichier d'objet. *(1 pt)*

e. **charger(String)** : permet de charger la collection des procédés d'un laboratoire depuis un fichier d'objet dans notre collection. *(1 pt)*

DOSSIER 2 : CONSULTATION DU MAPPING DES ADRESSES IP PAR PAYS *(10 pts)*

La **NAT-AMI** cherche à diversifier sa clientèle dans le monde entier en vendant son expertise dans le domaine de la conception des produits et des procédés de recyclage. À cet effet, le site *web* de la société qui existe déjà en trois langues (*français, allemand et anglais*) constitue un média privilégié pour faire connaître son activité. On souhaite mettre en place une application Client/serveur permettant d'assurer le suivi des accès à ce site web.

Sous MySQL, on dispose d'une base de données nommée « **Suivi_acces** » dont le MLDR est le suivant :

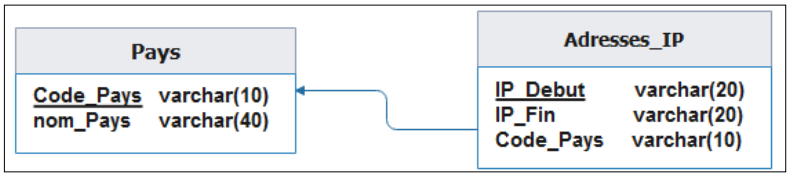

Figure 3 : MLDR de base de données de suivi des accès au site web

L'architecture Client/serveur est illustrée par la figure ci-dessous :

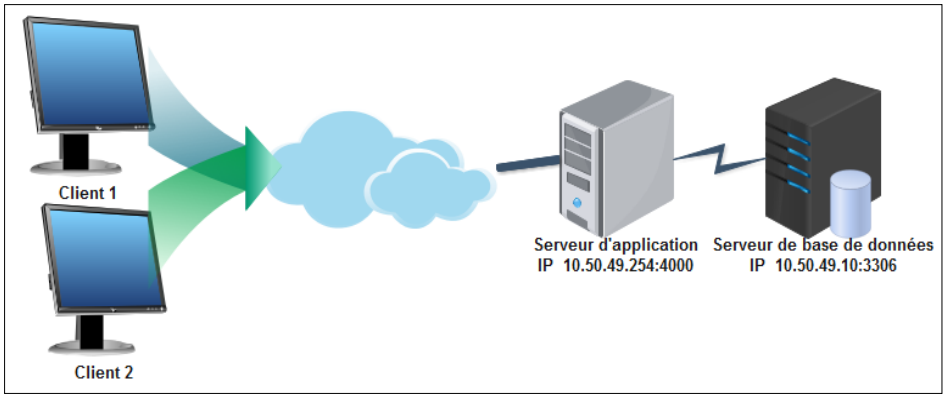

Figure 4 : Architecture Client/serveur

Le serveur d'application permet de fournir la liste des couples d'adresses : **IP_Debut** et **IP_Fin**, sous forme de String : **Adr IP de début : Adr IP de fin**, correspondante au **Code_Pays** envoyé par le client.

➢ Architecture :

1. S'agit-il d'une architecture 2-tiers ou 3-tiers ? Justifier votre réponse. *(1 pt)*
2. Proposer le modèle de Gartner correspondant à cette architecture. *(1 pt)*

➢ Côté Serveur :

3. Classe de connexion à la base de données :

```
public class SingletonConnection {
  private Connection conn=null;
  private static SingletonConnection  objConn=null;
  private SingletonConnection()     {.....................}
  public static SingletonConnection getConn()
   {  if(objConn==null) objConn=new SingletonConnection();
      return objConn; }
  public ResultSet  lire(String req)  {.....................}
  public int  ecrire(String req) {.....................}
}
```

3.1. Définir le constructeur de cette classe qui permet d'initialiser l'attribut « **conn** » en établissant une nouvelle connexion avec le serveur de base de données, on donne : *(1 pt)*

- URL : **jdbc:mysql://10.50.49.10:3306/Suivi_acces**
- Login : **admin**
- Password : **123456**

3.2. Définir la méthode « **lire(String)** » permettant d'exécuter une requête de lecture et retourner un objet « **ResultSet** ». En cas d'échec, elle doit retourner « **null** ». *(1 pt)*

3.3. Définir la méthode « **ecrire(String)** » permettant d'exécuter une requête de mise à jour et de retourner le nombre d'enregistrements touchés par cette requête. En cas d'échec, elle doit retourner **-1**. *(1 pt)*

4. Classe du serveur :

Dans cette partie, on peut envoyer les requêtes à notre base de données en utilisant la classe « **SingletonConnection** » de la manière suivante :

- ✓ **SingletonConnection.getConn().lire(**"requête de lecture"**)**
- ✓ **SingletonConnection.getConn().ecrire(**"requête de mise à jour"**)**

```
public class Serveur {
  private ServerSocket  serv ;
  private Socket ss ;
  ObjectOutputStream  out ;
  BufferedReader in ;
  public Serveur() {.....}
  public void start() {.....}
}
```

4.1. Donner le code du constructeur permettant d'initialiser l'objet « **ServerSocket** » sachant que le port d'écoute est : **4000.** *(1 pt)*

4.2. Donner le code de la méthode « **start()** » permettant de : *(2 pts)*

- ➢ attendre une demande de connexion de la part d'un client ;
- ➢ préparer les objets « **out** » et « **in** » ;
- ➢ répéter indéfiniment :
  - ✓ Lecture du code d'un pays envoyé par le client ;
  - ✓ Créer un objet « **ArrayList<String>** » nommé « **T** » ;
  - ✓ Lire depuis la base de données les adresses « **IP_Debut** » et « **IP_Fin** » correspondantes au « **Code_Pays** » reçu et les ajouter dans la liste « **T** » sous forme de « **String** » ;
  - ✓ Envoyer cette liste au client.

➢ **Côté client :**

Cette application permet aux différents responsables des sites de consulter les adresses IP (*début* et *fin*) correspondantes aux différents pays en se connectant au serveur d'application.

La classe « **Client** » se présente comme suit :

```
public  class Client {
    private String IP;
    private int  Port;
    private   Socket s=null;
    private   PrintWriter   pw=null;
    private  ObjectInputStream  ois=null;
    public  Client   (String IP, intPort){
            this.IP=IP;
            this.Port=Port;
    }
    private  boolean   setconnexion()  {     ..... }
    private ArrayList<String>  demandeAdressesIP(String code){
            .....
    }
}
```

5. Classe Client :

5.1. Écrire une méthode « **setconnexion() »** qui permet d'établir la connexion TCP/IP avec le serveur d'application et d'initialiser les objets « **pw »** et « **ois »**. La méthode retourne « **true »** en cas de succès et « **false** » dans le cas échéant. *(1 pt)*

5.2. Écrire la méthode « **demandeAdressesIP(String) »** qui envoie au serveur le code d'un pays et retourne la liste des adresses IP. *(1 pt)*

DOSSIER 3 : INVENTAIRE DES LOGICIELS INSTALLES *(10 pts)*

La société **NAT-AMI** souhaite créer, sous Visual Basic DOTNET, une application Desktop permettant d'enregistrer les logiciels installés ou mis à niveau en indiquant la date d'installation, le type d'installation, la version et en cas de besoin une remarque.

Toutes ces données sont modélisées par le modèle relationnel suivant :

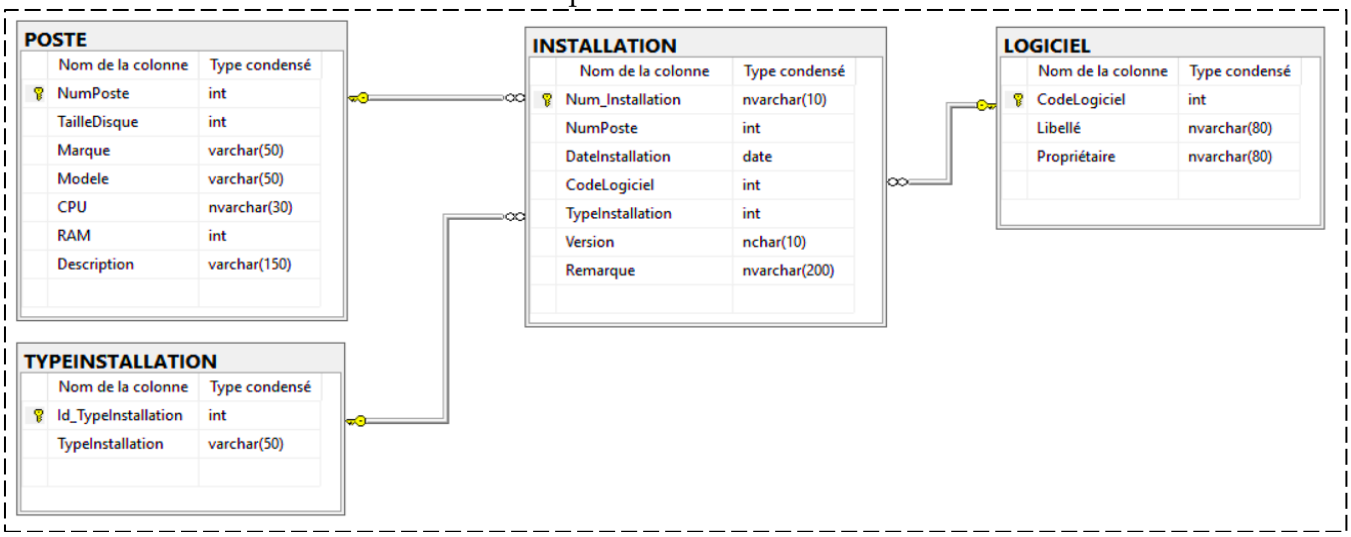

Figure 5 : Partie du MLDR de la base de données « BD_Installation».

La signification des attributs de la table « **INSTALLATION** » est précisée dans le tableau ci-dessous :

ATTRIBUT	SIGNIFICATION
Num_Installation	Numéro d'installation généré automatiquement (*0415/2018, 0012/2019, …*)
NumPoste	Numéro du poste sur lequel est installé le logiciel (*501, …*)
DateInstallation	Date d'installation ou de mise à niveau du logiciel (*01/02/2019, …*)
TypeInstallation	Type d'installation : Nouvelle installation ou mise à jour
CodeLogiciel	Identifiant du logiciel installé (*vs2017*, *w1032*, …)
Version	La version du logiciel installé (*12.0.2, 6.5.1, …*)
Remarque	Toute observation concernant l'installation (*Erreur déclenchée*, …)

NB.

- Vous avez le choix entre le mode **connecté** ou **non connecté** pour accéder à la base de données.
- La base de données « BD_Installation » est implantée sous Microsoft SQL Server.

1. Écrire, dans un module, le code de la fonction « **Connexion** » pour se connecter à la base de données. La fonction retourne « **True** » en cas de succès et « **False** » dans le cas échéant. *(1,5 pt)*

(Nom du serveur : **Serv1_Nat**, nom de la base de données : **BD_Installation**)

2. Écrire le code de la procédure « **Afficher_Liste** » permettant d'afficher la liste des installations, effectuées entre deux dates passées en paramètre, dans un objet **DataGridView** nommé « **DGVListe**». *(2 pts)*

Voici un aperçu de l'objet « **DGVListe** » :

	N° Installation	Date Installation	N° Poste	Marque	Modèle	Libellé Logiciel	Proprioétaire	Version	Remarque
▶*									

Signature de la procédure

```
Private Sub  Afficher_Liste(ByVal Du As Date,ByVal Au As Date)
   ……
End  Sub
```

3. Écrire le code de la procédure « **Supprimer_Installation** » permettant de supprimer une installation donnée. La procédure prend en argument le N° d'installation à supprimer. *(1,5 pt)*

Signature de la procédure

```
Private Sub Supprimer_Installation(ByVal Num As String)
   ……
End Sub
```

4. Écrire le code de la fonction « **Generer_Num_Installation** » permettant d'incrémenter le numéro d'installation. Ce numéro doit avoir la forme suivante : **xxxx/yyyy** (*exemple : 0015/2019*).
xxxx : représente un entier qui s'initialise au début de chaque année,
yyyy : représente l'année système. *(2 pts)*

Signature de la procédure

```
Private Function Generer_Num_Installation() As String
   ……
End  Function
```

NB : Informations complémentaires :

- **DateTime.Now** : retourne la date système.
- **MyDate.Year** : retourne l'année de la date MyDate.
- **MyDate.Month** : retourne le mois de la date MyDate.
- **MyDate.Day** : retourne le jour de la date MyDate.
- **MyDate.toString** (*FormatDésiré*) : retourne la date MyDate sous le format *FormatDésiré*.
  Exemple : **dim MyDate as DateTime=#15/03/2018#**
  MyDate.ToString("**yyyy**") retourne la chaine de caractère 2018.
- **Format** (*MaDonnee*, *FromatSouhaité*)
  Exemple : **Format** (*1.5* , "*000.00*") : retourne la chaine de caractère **001.50**

5. Écrire le code de la procédure « **Lister_Logiciel** » permettant de remplir l'objet **ComboBox** nommé «**CmbLogiciel** » par les libellés des logiciels à partir de la table « **LOGICIEL** ». *(1 pt)*

6. Écrire le code de la procédure « **Statistique** » permettant d'afficher, dans un objet DataGridView nommé « **DGV_Statistiques**», le nombre des installations effectuées pour chaque Logiciel.

Voici un aperçu de l'objet « **DGV_Statistiques** » : *(2 pts)*

Libellé Logiciel	Nombre d'Installations
Visual Studio 2015	10
Windows 10 Professionnel	50

DOSSIER 4 : VALIDATION DES DEMANDES DE RAMASSAGE DES DECHETS *(8 pts)*

L'objectif est de fournir à l'administrateur une interface web pour pouvoir valider les demandes de ramassage des déchets électroniques des clients.

La base de données est implémentée sous MySQL :

- Adresse IP du serveur web : **10.50.49.10**
- L'administrateur de la base de données : **admin**
- Le mot de passe de l'administrateur : **123456**
- Le nom de la base de données MySQL : **DB_Dechets**

La figure suivante montre un extrait de la base de données :

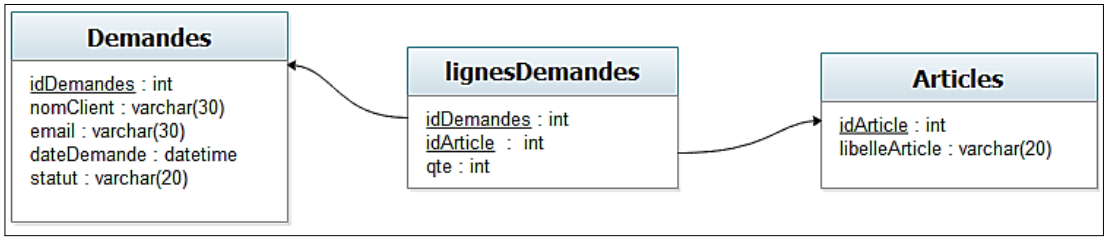

Figure 6 : Partie du MLDR de la base de données « **DB_Dechets**»

- La table « **Demandes** » contient la liste des demandes.
  - Le champ « **statut** » peut contenir deux valeurs : « **valide** » ou « **invalide** ».
- La table « **lignesDemandes** » contient la liste des articles d'une demande.
- La table « **Articles** » contient les libellés des articles (*Téléphone, Téléviseur, Ordinateur...*).

Le site Web exploite un ensemble de fonctions regroupées dans un fichier nommé « **mesFonctions.php** » dont la structure est donnée ci-dessous :

```
function getConnexion(){…}
function getInValidDemandes(){ …}
function getdetailsDemandes($idDemande){…}
function validerDemande($id){…}
```

La page d'accueil du site web, nommée « **index.php** », est la suivante :

Demandes invalides

Nom Client	Date de la demande	Détails	Statut
Adil Benani	2018-07-26 14:00:00	détails	invalide
Salma Tazi	2017-06-26 09:32:08	détails	invalide
Faysal Ouad	2018-02-26 10:32:08	détails	invalide
radi amine	2018-03-03 09:33:32	détails	invalide

Figure 7 : Page d'accueil du site web

Le code de cette page est donné ci-dessous :

```
<meta charset="utf8">
<link rel="stylesheet" type="text/css" href="style.css">
<?php
  include('fonctions.php');
  echo "<h1>Demandes invalides</h1>";
  echo "<table >";
  echo "<tr><th> Nom Client </th>  <th> Date de la   demande </th>
        <th> D&eacutetails </th> <th> Statut </th> </tr>";
   getInValidDemandes();
   echo "</table>";
?>
```

1. Donner le code de la fonction « **getConnexion ()** » qui permet d'établir une connexion avec le serveur de base de données et de retourner l'identifiant de la connexion.
   En cas d'échec, afficher un message d'erreur. *(1,5 pt)*
2. Donner le code de la fonction « **getInValidDemandes()** » qui affiche la liste des demandes invalides contenant les champs suivants : *(2 pts)*
   - Le nom du client ;
   - La date de la demande ;
   - Un lien hypertexte vers le détail de la demande « **detailsDemandes.php** » avec un argument nommé « **id** » qui contient « **idDemande** » de la table Demandes ;
   - Le statut de la demande.

Suite à un clic sur le lien « **détails** », la page « **detailsDemandes.php** » s'affiche :

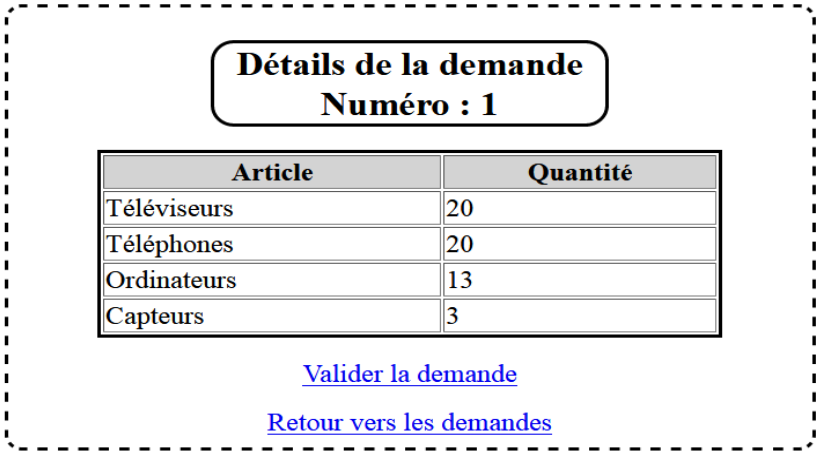

Figure 8 : Détails d'une demande

Le code de cette page web est le suivant :

```
<meta charset="utf8">
<link rel="stylesheet" type="text/css" href="style.css">
<?php
  include('fonctions.php');
  $id=$_GET['id'];  //ID de la demande récupéré depuis la page d'accueil
  echo "<h1>Détails de la demande Numéro : $id</h1>";
  echo "<table>";
  echo "<tr><th> Article </th><th> Quantité </th></tr>";
  getdetailsDemandes($id);
  echo "</table>";
  echo "<a href='traitement.php?id=$id'> Valider la demande </a>";
  echo "<br><a href='demandes.php'> Retour vers les demande </a>";
?>
```

3. Donner le code de la fonction **getdetailsDemandes($id)** permettant de lister le détail de la demande (*libellé d'article* "***libelleArticle***" *et la quantité* "***qte***") dont l'« **id** » est passé en paramètre. *(2 pts)*

Le clic sur le lien « **Valider la demande** » permet d'ouvrir la page **« traitement.php »** dont le script est le suivant :

```
1-   <meta charset="utf8">
2-   <?php
3-     include('fonctions.php');
4-     $id=$_GET['id'];
5-     validerDemande($id);
6-     echo "<script> .....</script>";   // à compléter
7-    echo "<script>document.location.href='index.php';</script>";
8-   ?>
```

4. Donner le code de la fonction « **validerDemande($id)** » qui permet d'attribuer la valeur « **valide** » au champ **« statut »** de la demande. L'identifiant de la demande est passé en argument. *(1,5 pt)*

5. Compléter **la ligne (6)** pour créer une alerte affichant le message suivant : **"Demande validée avec succès"**. *(1 pt)*

%Bon Courage%