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 et des Examens

Examen National du Brevet de Technicien Supérieur

Session de Mai 2021

- Sujet -

Filière :	**Développement des Systèmes d'Information – DSI –**	**Durée :**	**4 heures**
Épreuve :	**Développement des Applications Informatiques – DAI –**	**Coefficient :**	**45**

CONSIGNES

- ✓ **Le sujet comporte 3 dossiers ;**
- ✓ **Chaque dossier doit être traité dans une feuille séparée.**

Barème de notation

Dossier 1 : Service pédagogique		**22 points**
Sous-dossier 1-1 : Structure de données	14 points	
Sous-dossier 1-2 : Architecture client/serveur	08 points	
Dossier 2 : Gestion des inscriptions		**10 points**
Dossier 3 : Gestion des publications		**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 : Plateforme d'enseignement à distance

L'enseignement à distance est une approche d'apprentissage exploitée dans divers secteurs et utilisée dans les formations de certification. Cette approche est récemment, de plus en plus, adoptée dans les formations académiques.

L'enseignement à distance ou E-Learning est un moyen d'assurer la tâche d'enseignement non présentiel. La nécessité d'utiliser des plateformes assurant un tel enseignement est devenu primordial.

Dans les classes BTS (***B**revet de **T**echnicien **S**upérieur*), on souhaite mettre en place un environnement numérique d'enseignement à distance. Les intervenants sur cette plateforme sont :

- **Les responsables pédagogiques** : peuvent consulter la liste des étudiants par équipe.
- **Les enseignants :** peuvent gérer les supports électroniques ainsi que les étudiants par équipes.
- **Les étudiant :** peuvent consulter ou créer des publications.

La plateforme doit être composée de différents services, à savoir :

- **Service pédagogique** : Application Client/serveur en JAVA ;
- **Service d'inscription** : Application Desktop en VB.NET ;
- **Service de gestion des publications** : Application Web.

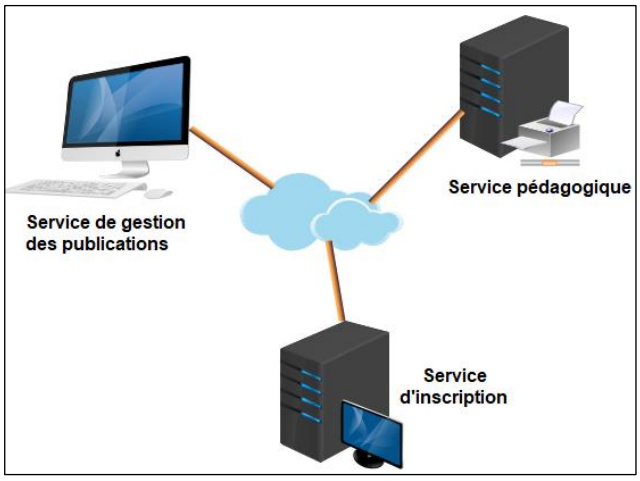


Figure 1 : Schéma structurel des services de l'enseignement à distance

DOSSIER I : SERVICE PÉDAGOGIQUE *(22 pts)*

❖ SOUS-DOSSIER 1-1 : STRUCTURE DE DONNÉES *(14 pts)*

La structure de données, utilisée dans ce service, est illustrée par le diagramme de classes suivant :

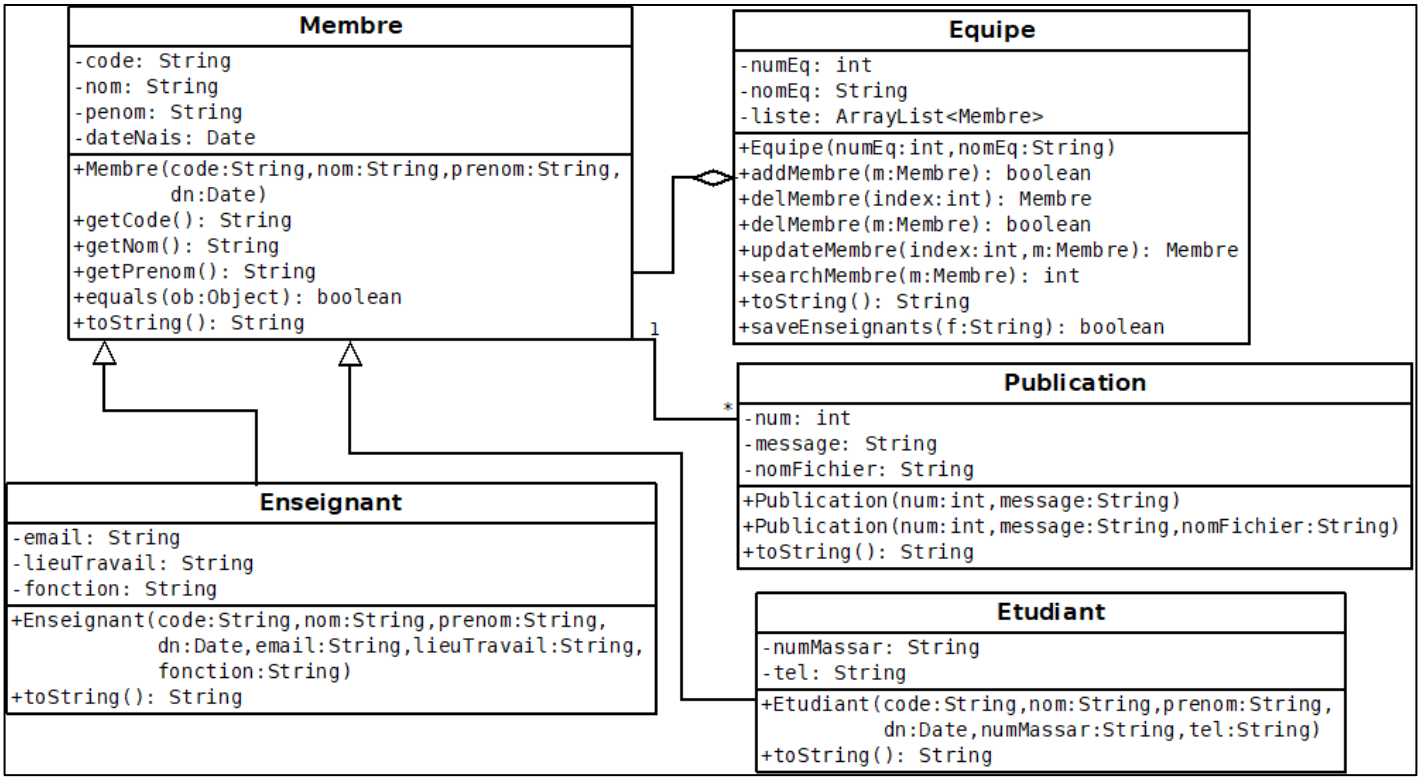


Figure 2 : Diagramme de classes

1. Implémentation de la classe « **Membre** » :

```
public class Membre implements Serializable{
   private String code;
   private String nom, prenom;
   private Date dateNais;

   public Membre(...) {...}
   //-------------------------------------------------------------
   public String getCode() {...}
   //-------------------------------------------------------------
   public String getNom() {...}
   //-------------------------------------------------------------
   public String getPrenom() {...}
   //-------------------------------------------------------------
   @Override
   public boolean equals(Object ob) { ... }
   //-------------------------------------------------------------
   @Override
   public String toString() { ... }
}
```

a) Définir un constructeur avec **4** arguments pour initialiser les attributs de cette classe. *(1 pt)*

b) Donner le code des **3** accesseurs *(1 pt)*

c) Redéfinir la méthode « **equals** » qui compare deux membres selon leurs codes *(La comparaison est insensible à la casse*). *(1 pt)*

d) Redéfinir la méthode « **toString** », afin de retourner une chaine porteuse d'informations sur un membre sous la forme : *(1 pt)*

```
Code : xxxx, Nom Complet : xxxx, Date naissance : jj/mm/aaaa
```

2. Implémentation de la classe « **Enseignant** » :

```
public class Enseignant extends Membre {
   private String email, lieuTravail, fonction;
   //-----------------------------------------------------------
   public Enseignant(...) { ... }
   //-----------------------------------------------------------
   @Override
   public String toString() { ... }
}
```

a) Définir un constructeur de **7** arguments pour initialiser tous les attributs de cette classe. *(1 pt)*

b) Donner la définition de la méthode « **toString** », afin de retourner une chaine porteuse d'informations sur un enseignant sous-forme : *(1 pt)*

```
Code : xxxx, Nom Complet : xxxx, Date naissance :  jj/mm/aaaa
Enseignant [ Email : xxxx, lieu de travail : xxxx , Fonction : xxxx ]
```

3. Implémentation de la classe « **Equipe** » :

```
public class Equipe {
   private int numEq;
   private String nomEq;
   private ArrayList<Membre> liste;

   public  Equipe(...) { ... }
      //-----------------------------------------------------
   public boolean addMembre(Membre m){... }
      //-----------------------------------------------------
   public Membre delMembre(int index){ ... }
      //-----------------------------------------------------
   public boolean delMembre(Membre m){ ... }
      //-----------------------------------------------------
   public Membre updateMembre(int index, Membre m) { ... }
      //-----------------------------------------------------
   public int searchMembre(Membre m) { ... }
      //-----------------------------------------------------
   @Override
   public String toString(){ ... }
     //-----------------------------------------------------
   public boolean saveEnseignants(String file){ ... }

}
```

Donner le code des méthodes suivantes :

a) Un constructeur avec **2** arguments qui initialise les **2** premiers attributs de la classe « **Equipe** » et instancie la collection « **liste** » ; *(1 pt)*

b) **addMembre(...)** : permet d'ajouter à la collection un nouveau membre et retourne l'état de l'opération ; *(1 pt)*

c) **delMembre(int…)** : permet de supprimer un membre de la collection en se basant sur un index et retourne le membre qui vient d'être supprimé. La vérification de la validité de l'index est indispensable ; *(1 pt)*

d) **delMembre(Membre ..)** : permet de supprimer un membre de la collection en se basant sur un objet « **Membre** » et retourne l'état de la suppression ; *(1 pt)*

e) **updateMembre(…)** : permet de modifier un membre déjà existant, la vérification de la validité de l'index est indispensable, elle retourne le membre avant sa modification ; *(1 pt)*

f) **toString ( ) :** retourne une chaine de caractères porteuse de toutes les informations d'une équipe sous-forme : *(1,5 pt)*

```
Equipe [numéro : xxxx, Nom Equipe : xxxx ] contient les membres suivants:
1-  Code : xxxx, Nom Complet : xxxx, ……
2-  ……
```

g) **saveEnseignants (String f):** permet de sauvegarder tous les enseignants dans un fichier d'objet et retourne l'état de l'opération ; *(1,5 pt)*

❖ SOUS-DOSSIER 1-2 : ARCHITECTURE CLIENT/SERVEUR *(8 pts)*

Afin qu'un responsable pédagogique de la plateforme puisse consulter la liste des étudiants d'une équipe, une architecture client/serveur est mise en place. Son schéma est donné par la figure suivante :

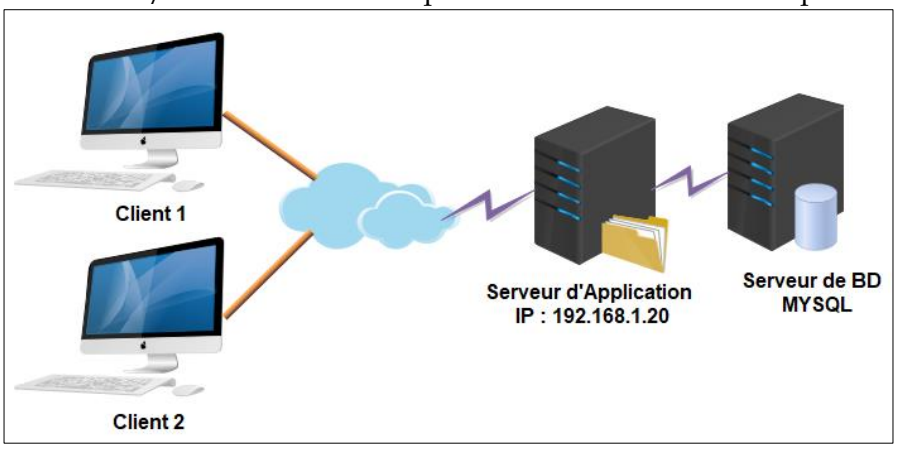


Figure 3 : Architecture client/serveur

1. Quel est le rôle d'un serveur d'application ? *(0,5 pt)*
2. Donner la classe d'adressage correspondante à l'adresse IP du serveur d'application : **192.168.1.20** *(0,5 pt)*

La communication entre le client et le serveur d'application peut se faire en utilisant les sockets TCP ou les objets partagés RMI.

⚠	- **Vous devez résoudre ce sous-dossier avec l'une des deux méthodes suivantes (*pas les deux à la fois*).** - **Veuillez préciser la méthode utilisée.**

3. MÉTHODE 1 : Socket TCP

a) Classe : Client TCP

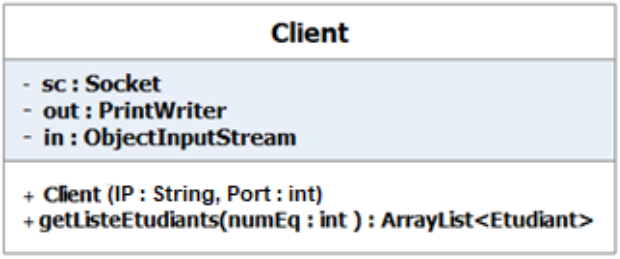


Figure 4 : Classe Client TCP

Implémenter la classe **« Client »** qui contient : *(4 pts)*

- Les **3** attributs ;
- Un constructeur avec **2** arguments qui crée un socket et initialise les objets d'entrée/sortie **in** et **out** ;
- Une méthode **«getListeEtudiants »** qui envoie au serveur **TCP** le numéro d'une équipe et qui reçoit et retourne une collection contenant les étudiants de cette équipe.

```
public class Client {
   private Socket sc=null;
   private ObjectInputStream in = null;
   private PrintWriter out=null;
   public Client(String IP, int Port){...}
   public ArrayList<Etudiant> getListeEtudiants(int numEq)
   { ... }
}
```

b) Classe : Serveur TCP

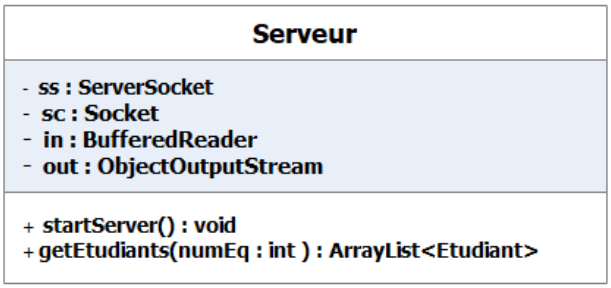


Figure 5 : Classe Serveur TCP

```
public class Serveur {
   private ServerSocket ss=null;
   private Socket sc=null;
   private BufferedReader in = null;
   private ObjectOutputStream out= null;
   public void startServer() {
      try{
         ... [1]
      while(true) {
         ... [2]
            While(true) {
             ... [3]
            }
       }
       }catch(Exception ex) {  }
   }
   public ArrayList<Etudiant> getEtudiants(int numEq) {
       // Le code source de la méthode getEtudiants est non demandé
   }
}
```

Compléter la méthode **startServer** de la classe Serveur de telle manière : *(3 pts)*

- Dans la partie **[1]** :
  - Instancier l'objet **ServerSocket** avec un port d'écoute : **3000 ;**
- Dans la partie **[2]** :
  - Accepter une demande de connexion du client **TCP** ;
  - Préparer les objets d'entrée/sortie : **in** et **out**.
- Dans la partie **[3]** :
  - Lire le code de l'équipe envoyé par le client **TCP** ;
  - Si le code reçu est strictement négatif, on sort de la boucle de la partie [3] ;
  - Sinon, on fait appel à la méthode **getEtudiants** pour récupérer un tableau **ArrayList** contenant les étudiants de cette équipe et on envoie cet objet **ArrayList** au client **TCP**.

3. MÉTHODE 2 : Objets partagés RMI

a) Classe : Serveur RMI

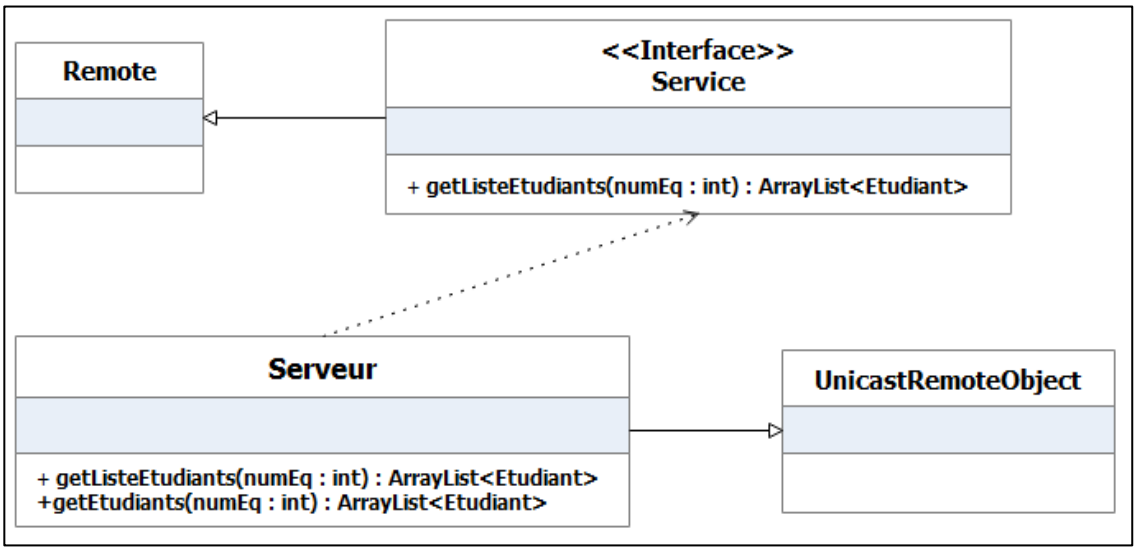


Figure 6 : Classe Serveur RMI

```
public Interface Service extends ... {
  ...
}
```

```
public class Serveur extends UnicastRemoteObject implements Service {
   public Serveur() throws RemoteException {}
   @Override
   public ArrayList<Etudiant> getListeEtudiants(int numEq) throws
    RemoteException {...}
  //-------------------------------------------------------------
  public ArrayList<Etudiant> getEtudiants(int numEq)
  { // Code source de la méthode getEtudiants est non demandé }
  //-------------------------------------------------------------
  public static void main(String [] args) throws Exception
  {...}
}
```

TRAVAIL DEMANDÉ

- Compléter l'interface **Service**. *(1 pt)*
- Donner les codes des méthodes suivantes : *(3 pts)*
  - **getListeEtudiants** : fait appel à la méthode **getEtudiants** pour récupérer un tableau **ArrayList** contenant les étudiants de cette équipe puis retourne cet objet **ArrayList**.
  - **main :** permet de
    - ✓ Démarrer le service de nom (**rmiregistry**) avec le numéro de port **1099** ;
    - ✓ Instancier un nouvel objet de la classe **Serveur** que l'on nomme : **obj** ;
    - ✓ Publier la référence de cet objet dans l'annuaire avec l'url : « **rmi://192.168.1.20/OBJ** ».

b) Classe : Client RMI

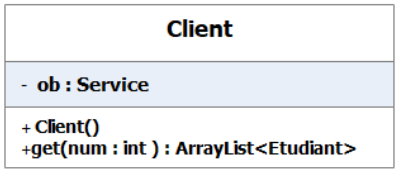


Figure 7 : Classe Client RMI

Implémenter la classe **client** qui contient : *(3 pts)*

- Un attribut de type **Service** ;
- Un constructeur sans arguments qui récupère la référence partagée par le serveur et initialise l'attribut **ob** ;
- Une méthode **get** qui récupère et retourne la liste des étudiants en appelant la méthode partagée par le serveur.

```
public class Client {
   private Service ob;
   public Client() throws Exception {...}
   public ArrayList<Etudiant> get(int numEq) throws RemoteException
   { ... }
}
```

DOSSIER II : GESTION DES INSCRIPTIONS *(10 pts)*

Afin de gérer les inscriptions des étudiants dans une équipe, on utilise une application Desktop connectée à une base de données SQL-Server. Les caractéristiques de ce serveur sont :

- IP du serveur : **192.168.1.200** ;
- Nom de la base de données : **db_Inscription** ;
- Authentification de Windows.

La structure de cette base de données est la suivante :

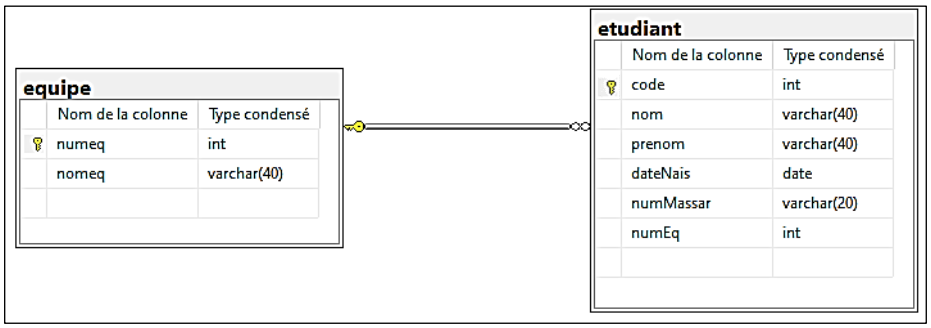


Figure 8 : Structure de la base de données

L'IHM de cette application est illustrée par la figure suivante :

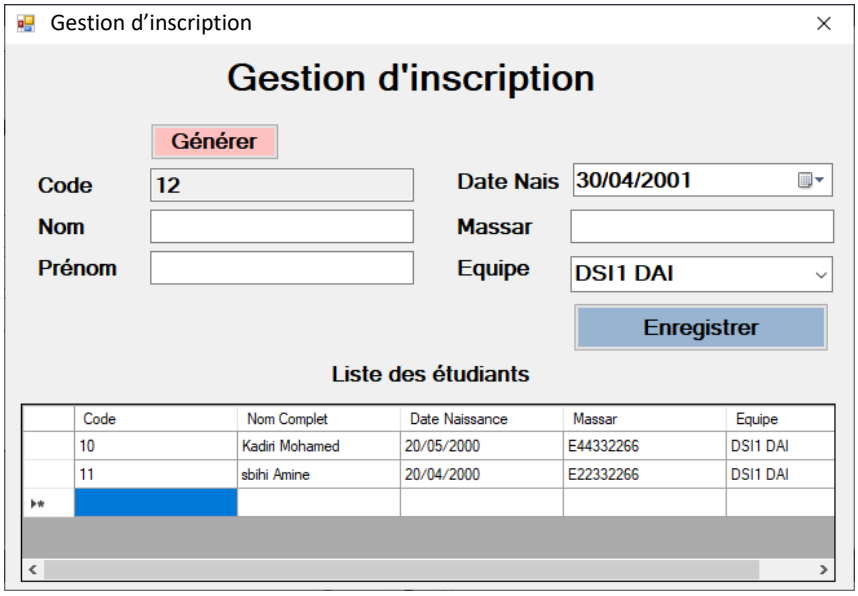


Figure 9 : IHM d'inscription des étudiants

Objet	Name	Texte
Boutons de commandes	CmdEnregistrer	Enregistrer
	CmdGenerer	Générer

Objet	Name
DataGridView	DGVListe
DateTimePicker	dtpDateNais
ComboBox	cmbEquipe
TextBox	txtCode
	txtNom
	txtPrenom
	txtMassar

TRAVAIL DEMANDÉ

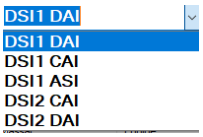
> **-Vous devez résoudre ce dossier avec l'un des deux modes : connecté ou non connecté (*pas les deux à la fois*).**
> **-Veuillez préciser le mode utilisé.**

1) Dans un module, déclarer les objets de connexion et créer la fonction **« Connexion»** qui établit une connexion avec le serveur de base de données (*Il faut gérer les exceptions*). *(2 pts)*

Signature de la procédure

```
Public Function Connexion() As Boolean
    ...
End Function
```

2) Dans la classe de l'IHM (Form), Écrire le code de la procédure **« charger_Equipes»** qui charge le comboBox **« cmbEquipe »** par les noms de toutes les équipes. Voici un aperçu de l'objet **« cmbEquipe »** : *(2 pts)*



Signature de la procédure

```
Private Sub charger_Equipes()
    ...
End Sub
```

3) Écrire le code de la procédure **«affiche_Etudiants »** qui liste tous les étudiants dans l'objet **DataGridView** nommé **« DGVListe ».** *(2 pts)*

Voici un aperçu de l'objet **DataGridView** (*La colonne **Equipe** contient les noms des équipes*).

Code	Nom Complet	Date Naissance	Massar	Equipe

Signature de la procédure

```
Private Sub affiche_Etudiants()
   ...
End Sub
```

4) Écrire le code de la fonction **« genererCode »** qui retourne un code valide d'un nouvel étudiant. Ce code doit correspondre au maximum des codes existants incrémenté de 1. *(2 pts)*

Signature de la procédure

```
Private Function genererCode() As Integer
   ...
End Function
```

5) Un clic sur le bouton **« CmdGenerer »** fait appel à la fonction **« genererCode »** pour affecter à la zone de texte **« txtCode »** un nouveau code.

```
Private Sub CmdGenerer_Click(……) Handles CmdGenerer.Click
     txtCode.Text = genererCode().ToString
End Sub
```

On suppose que le formulaire est bien rempli, donner le code de la procédure événementielle du bouton « **CmdEnregistrer** » permettant d'ajouter un nouvel étudiant. *(2 pts)*

Signature de la procédure

```
Private Sub CmdEnregistrer_Click(…) Handles CmdEnregistrer.Click
   ...
End Sub
```

DOSSIER III : GESTION DES PUBLICATIONS. *(8 pts)*

On souhaite créer une application Web, permettant aux membres de consulter et modifier leurs publications. Sous MySQL, on implémente une base de données **« db_Publication »** dont la structure est :

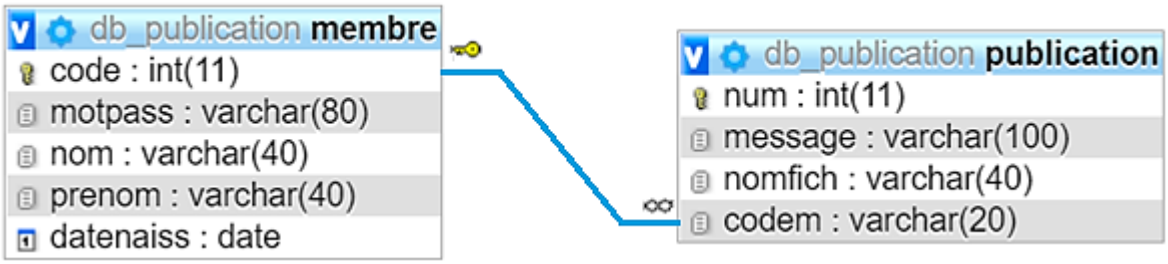


Figure 10 : Structure de la base de données « db_Publication »

TRAVAIL DEMANDÉ

1) Donner le code du script **« connecter.php »** permettant la connexion à la base de données **« db_Publication »** sachant que : *(2 pts)*

- Nom du serveur : **Server ;**
- Nom utilisateur : **membre ;**
- Mot de passe : **membre123.**

2) La page d'accueil **« index.php »** permet à un membre de s'authentifier. Le formulaire d'authentification est le suivant :

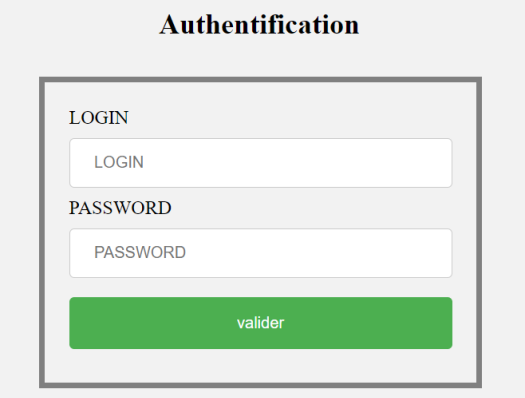


Figure 11 : Formulaire d'authentification

Le script **« index.php »** est le suivant :

```
<html>
  <head> </head>
  <body>
    <div>
        <h2> Authentification </h2>
        <form method='POST' action='validerAuth.php'>
            <label > CODE </label>
            <input type='text' placeholder="CODE" name='code' required> <br/>
            <label > PASSWORD </label>
            <input type='text' placeholder="PASSWORD" name='motpass' required>
            <input type='submit' value='valider' > </input>
        </form>
    </div>
  </body>
</html>
```

Donner le script du fichier **« validerAuth.php »** permettant de valider l'authentification d'un membre selon les étapes suivantes : *(3 pts)*

- Lire les données du formulaire ;
- Vérifier l'existence du code (login) et du mot de passe dans la table **« Membre »** de la base de données ;
- Dans le cas d'une authentification réussie, on mémorise le code, le nom et le prénom du membre dans une nouvelle session et on effectue une redirection vers le fichier **« lstpublications.php »** ;
- Dans le cas d'échec, on retourne au fichier **« index.php »**.

3) La validation du formulaire d'authentification permet de lister les publications du membre connecté.

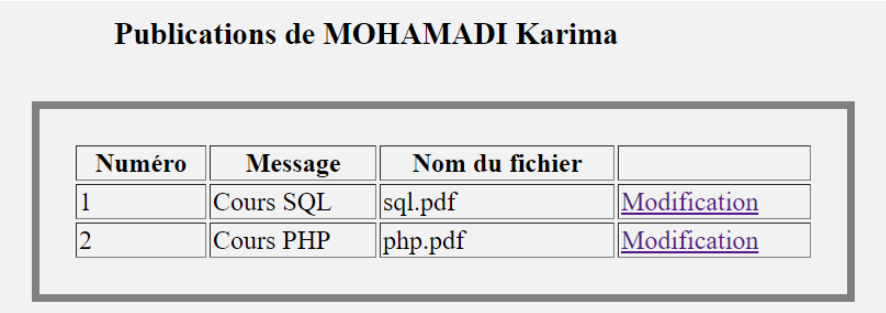


Figure 12 : Page de Listing des publications d'un membre

Le script **« lstpublications.php»** est le suivant :

```
<html>
  <head> </head>
  <body>
    <?php
      session_start() ;
      require 'connecter.php'; ?>
      <h3> Publications de <?php ... [1]      ?>
    <table>
       <thead>
        <tr>
           <th>Numéro</th> <th>Message</th> <th>Nom du fichier</th> <th> </th>
         </tr>
       </thead>
       <tbody>
           <?php  ...  [2]  ?>
       </tbody>
     </table>
   </body>
</html>
```

Reprendre et compléter les parties PHP du script ci-dessus tel que : *(3 pts)*

- **Dans la partie [1]** : Récupérer et afficher les informations du membre connecté (*nom et prénom*) ;
- **Dans la partie [2]** : Lister toutes les publications de ce membre, le lien hypertexte «**Modification** » redirige vers la page **« modifpubl.php »** en envoyant, via URL, le numéro de publication à modifier.

% Bon courage %