III. Sujet de Développement des applications informatiques (DAI) :

THÈME : GESTION DES ÉMISSIONS D'UNE CHAÎNE TV

Une chaîne de télévision gère ses émissions via une application informatique et un système d'information. L'organisation du service informatique de cette chaîne est représentée par la figure ci-dessous :

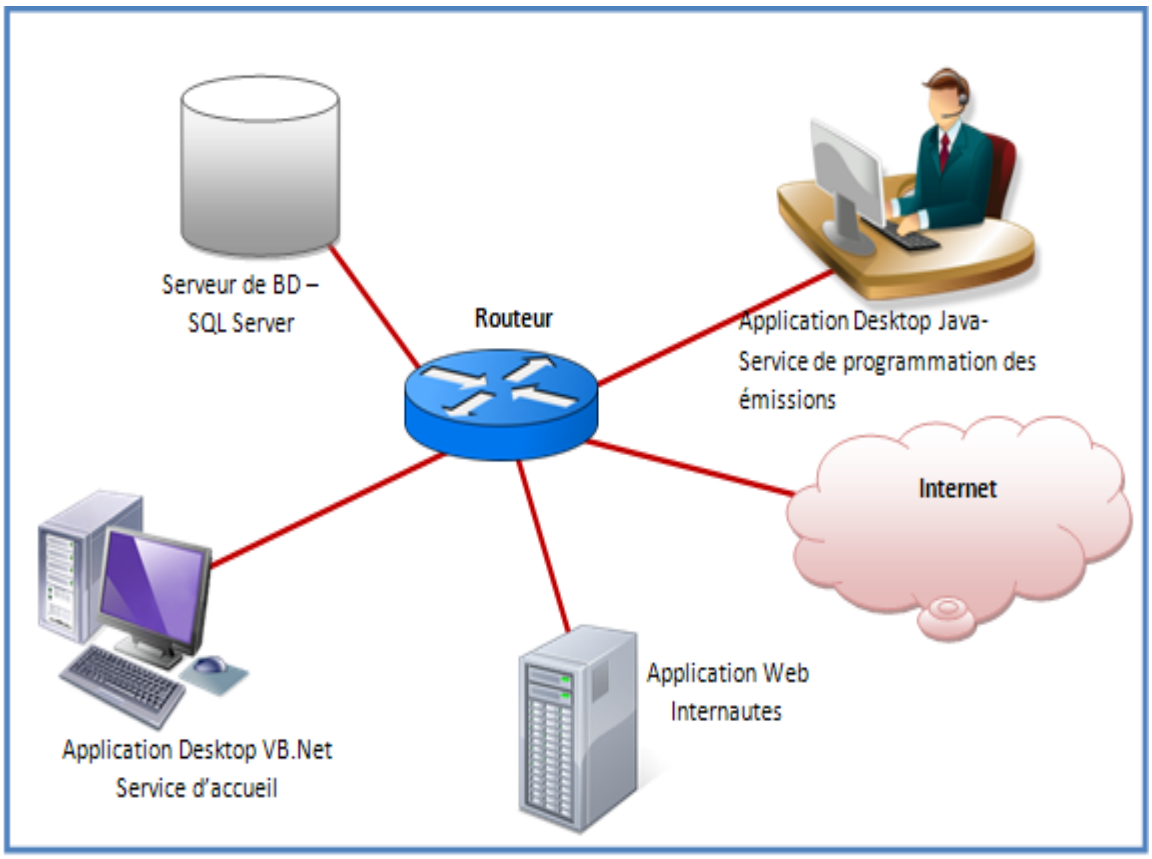

Figure 1 : *Organisation de l'entreprise*

Cette organisation est constituée principalement par :

- Un serveur de base de données sous Microsoft SQL Server
- Un service d'accueil dans lequel, on trouve une application Desktop sous VB.NET qui permet de consulter les émissions programmées et d'ajouter d'autres.
- Un service de programmation qui valide les émissions proposées par le service d'accueil.
- Une application Web facilite, aux internautes, la consultation en ligne des émissions programmées et de donner leurs avis.

DOSSIER I : SERVICE D'ACCUEIL *(10 pts)*

Dans le serveur de base de données, nommé « **dbServer** », on a implanté une base de données nommée « **dbGEm** » dont le modèle physique est le suivant :

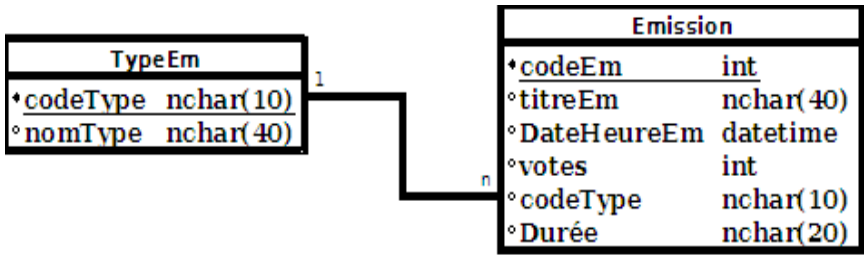

Figure 2 : *Modèle physique de données(MPD)*

Une description des champs de MPD est donnée dans les tableaux suivants :

Champ	Type	Détail
codeType	nchar(10)	Clé primaire
nomType	nchar(40)	

Champ	Type	Détail
CodeEm	int	Clé primaire auto incrémenté
titreEm	nchar(40)	
DateHeureEm	Datetime	
Votes	int	Valeur par défaut est zéro
codeType	nchar(10)	Clé étrangère
Durée	nchar(20)	

L'application, sous VB.Net, permet la consultation des émissions proposées selon leur titre et permet aussi la création, la modification et la suppression d'une émission. L'IHM de cette application est la suivante :

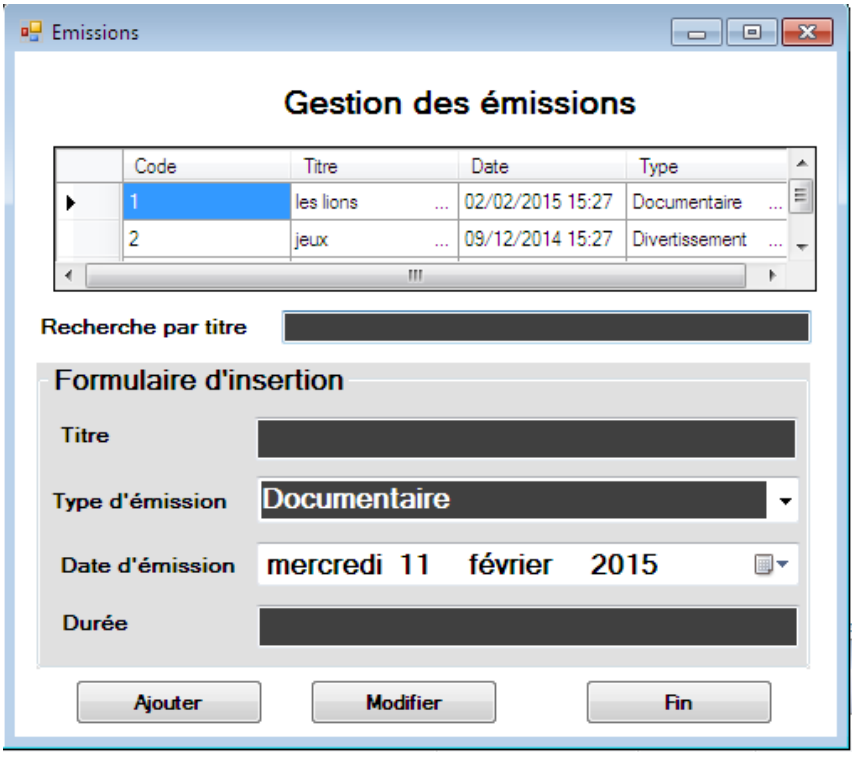

Figure 3 : *IHM de l'application de service d'accueil*

Les contrôles de notre IHM sont listés ci-après :

Objet	Name	Text
Zone de texte	TxtTitre	
	TxtDuree	
	TxtRech	
Boutons de Commande	CmdAjouter	Ajouter
	CmdModifier	Modifier
	CmdFin	Fin

Objet	Name
ComboBox	CmbType
DataGridView	DGV
DateTimePicker	DateEm

N.B : Vous pouvez utiliser soit le **mode connecté** soit le **mode non connecté**

Travail à Faire :

I-1. Selon le mode utilisé, citer les objets nécessaires pour accéder et manipuler cette base de données. *(0,5 pt)*

I-2. Donner le code d'une procédure nommée « **Connexion** » permettant la connexion à la base de données. Une gestion d'exception est obligatoire. *(1,5 pt)*

I-3. Donner le code de la procédure « **charger_Types »** permettant de remplir l'objet **ComboBox** nommé « **CmbType** » par les noms de différents types d'émissions. *(1,5 pt)*

I-4. Donner le code de la procédure « **Lister_Emissions »** permettant de remplir l'objet **DataGridView** nommé « **DGV** » par toutes les émissions (*Code, Titre, Date et le nom de type*). *(1,5 pt)*

I-5. Écrire le code des procédures évènementielles associées aux boutons :

- **CmdAjouter** : pour enregistrer une nouvelle émission dans la base de données. Tous les champs doivent êtres remplis. *(1,5 pt)*
- **CmdModifier** : permet d'enregistrer les modifications apportées aux données du formulaire d'insertion et actualiser, par la suite, l'affichage de l'objet **DataGridView**. *(1,5 pt)*
- **CmdFin** : permet de mettre fin à l'application après confirmation de l'utilisateur. *(0,5 pt)*

I-6. Donner le code de la procédure évènementielle de la zone texte « **TxtRech** ». *(1,5 pt)*

```
Private Sub txtRech_TextChanged(ByVal sender …., ByVal e As …) Handles txtRech.TextChanged
        ……
End Sub
```

Cette procédure permet de rechercher une émission dont le titre **commence** par les caractères saisis dans la zone de texte « **TxtRech** ». Le résultat de recherche s'affichera dans l'objet **DataGridView** (**DGV**).

DOSSIER II : SERVICE WEB *(8 pts)*

Le service Web dispose d'une copie de la base de données de Microsoft SQL Server. Cette copie est implantée dans un serveur MySQL dont les caractéristiques sont les suivantes :

- Nom de serveur : ***localhost***
- Login : ***user***
- Password : ***12345***

On veut créer une page Web dynamique permettant de :

- Lister toutes les émissions avec une case à cocher au niveau de chacune ;
- Valider le vote via le bouton « **Voter** ».

La page Web est donnée ci-dessous :

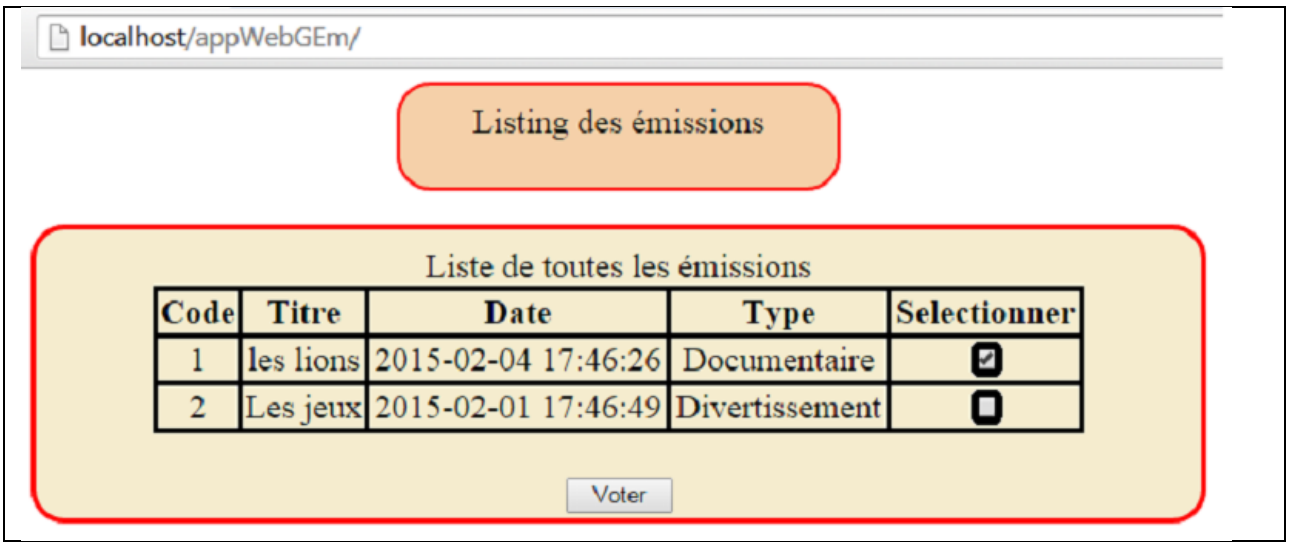

Figure 4 : *Page Web de consultation des émissions*

II-1. Donner le code PHP de la fonction « **connecter** » permettant d'établir une connexion avec la base de données dont le prototype est le suivant : *(2 pts)*

```
<?php
  function connecter () {
      ……
  }
?>
```

II-2. Le code HTML de notre page web est le suivant : *(3 pts)*

```
<html>
   <head>
     <?php
       function connecter () {……}
     ?>
   </head>
<body>
<h1 align=”center”>Listing des émissions</h1>
 <form method="POST" action="voter.php">
 <table align=”center” border=”2” width=”60%”>
   <caption>Liste de toutes les émissions</caption>
   <tr>
     <th>Code</th>
     <th>Titre</th>
     <th>Date</th>
     <th>Type</th>
     <th>Sélectionner</th>
   </tr>
    <?php
      connecter();
       ……
    ?>
</table>
</form>
</body></html>
```

Compléter le script PHP de la balise **<Body>** permettant de lister « ***Code, Titre, Date*** et ***Type*** » de toutes les émissions avec **une case à cocher** au niveau de chacune. (*Voir figure 4*)

II-3. Donner le code de fichier « **voter.php** » qui permet de :

- Récupérer les codes des émissions sélectionnées via les cases à cocher ; *(1 pt)*
- Incrémenter par **1** le champ « **votes** » de chaque émission sélectionnée ; *(1 pt)*
- Retourner à la page d'accueil « **index.php** ». *(1 pt)*

DOSSIER III : SERVICE DE PROGRAMMATION DES ÉMISSIONS *(14 pts)*

Le modèle abstrait des données est le suivant : ***(14 pts : 1pt par question)***

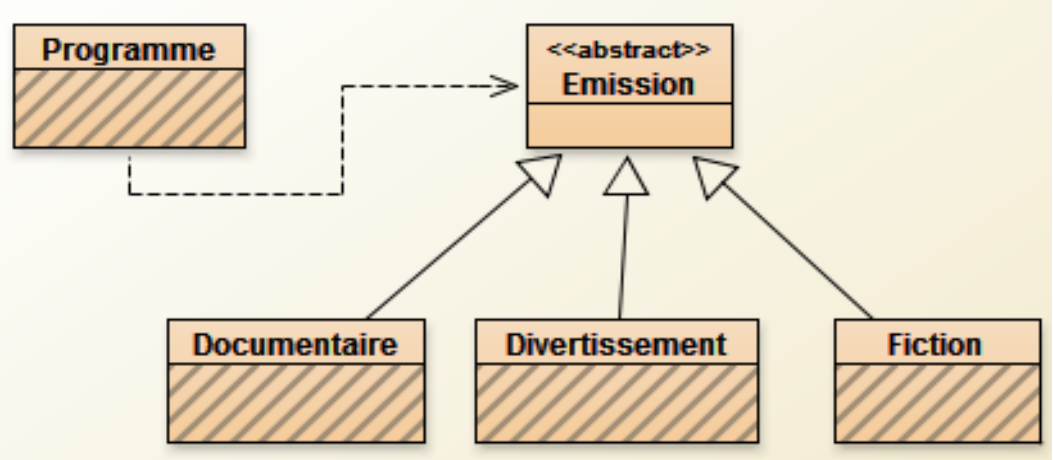

Figure 5 : *Diagramme de classe*

III-1. Le détail de la classe « **Emission** » est le suivant :

```
public abstract class Emission implements Serializable {
   private String TitreEm;
   private String duréeEm;
   private Date dateEm;
//-----------------------------------------------------------------
   public  Emission(String TitreEm, String duréeEm, Date dateEm) {
                              ……
   }
//-----------------------------------------------------------------
   public  Emission(String TitreEm, String duréeEm) {
                              ……
   }
//-----------------------------------------------------------------
   @Override
   public  boolean  equals(Object ob) {
                              ……
   }
//-----------------------------------------------------------------
    @Override
   public  String  toString() {
                              ……
   }
//-----------------------------------------------------------------
   public  abstract  String getInfos();
}
```

a) Définir les deux constructeurs pour initialiser les attributs de cette classe. Le constructeur avec deux arguments initialise la date d'émission par celle du système.

b) Définir la méthode « **equals** » qui compare deux émissions selon leurs titres.

c) Donner la définition de la méthode « **toString** », afin de retourner une chaine porteuse d'informations sur une émission, la chaine aura la forme suivante :

Titre : xxxx , la durée : xxxxx , Date d'émission : jour/mois/année.

III-2. Le détail de la classe « **Documentaire** » est le suivant :

```
public class Documentaire extends Emission {
   private String theme;
   public  Documentatire(……) {
                              ……
   }
//-------------------------------------------------------------
   public  String  getInfos(){
                              ……
   }
}
```

a) Proposer un constructeur adéquat permettant d'initialiser tous les attributs ;

b) Donner le code de la méthode **getInfos( )** permettant de retourner une chaine sous la forme suivante :

Titre : xxxx , la durée : xxxxx , Date d'émission : jour/mois/année.

Thème : xxxxxxxx

III-3. Le détail de la classe « **Fiction** » est le suivant :

```
public class Fiction extends Emission {
   private String réalisateur, producteur;
   public  Fiction(……) {
                              ……
   }
//-------------------------------------------------------
   public  String  getInfos() {
                              ……
   }
}
```

a) Proposer un constructeur adéquat permettant d'initialiser tous les attributs ;

b) Donner le code de la méthode **getInfos( )** permettant de retourner une chaine sous la forme suivante :

Titre : xxxx , la durée : xxxxx , Date d'émission : jour/mois/année.

Réalisateur : xxxxxxxx

Producteur : xxxxxxxx

III-4. Le détail de la classe **« Programme »** est le suivant :

```
public class Programme {
   private ArrayList<Emission> Liste=new ArrayList<Emission>() ;
//-----------------------------------------------------------------
   public  boolean  addEmission(Emission x) {
                              ……
   }
//-----------------------------------------------------------------
   public  Emission  delEmission(int index){
                              ……
   }
//-----------------------------------------------------------------
   public  boolean  delEmission(Emission p){
                              ……
   }
//-----------------------------------------------------------------
   public  Emission  setEmission(int index,Emission x){
                              ……
   }
//-----------------------------------------------------------------
   public  void  affiche( ){
                              ……
   }
//-----------------------------------------------------------------
   public  boolean  saveAll(String f){
                              ……
   }
//-----------------------------------------------------------------
   public  boolean  loadAll(String f) {
                              ……
   }
}
```

Donner le code des méthodes suivantes :

a- `addEmission(…)` : permet d'ajouter à la collection une nouvelle émission et retourne l'état de l'opération ;

b- `delEmission(int)` : permet de supprimer une émission de la collection en se basant sur un index et retourne l'émission qui vient d'être supprimée. La vérification de la validité de l'index est indispensable ;

c- `delEmission(Emission ..)` : permet de supprimer une émission de la collection en se basant sur un objet « Emission » et retourne l'état de la suppression ;

d- `setEmission(...)` : permet de modifier une émission déjà existante, La vérification de la validité de l'index est indispensable ;

e- `afficher()` : affiche toutes les émissions de la collection ;

f- **`saveAll`** (**`String`**): permet de sauvegarder toutes les émissions de la collection dans un fichier d'objet ;

g- **`loadAll`**(`String`): permet de charger toutes les émissions depuis un fichier d'objet dans notre collection.

DOSSIER IV : ADMINISTRATION DISTANTE DE L'APPLICATION *(8 pts)*

❖ SERVEUR D'APPLICATION

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

Connexion
- cn : Connection - URL : String - User : String - Password : String
+ …()

Figure 6 : *Classe « Connexion »*

```
public class Connexion{
   private String URL,User, Password ;
   private Connection cn;
//-----------------------------------------------------------------
   public  Connexion(String URL, String User, String Password) {
     this.URL=URL;
     this.User=User;
     this.Password=Password;   }
//-----------------------------------------------------------------
   public  boolean  connecter( ){
                              ……
   }
//-----------------------------------------------------------------
   public  ResulatSet  lire(String requete){
                            ……     }
//-----------------------------------------------------------------
   public  Vector<Object[]>  toVector(String     requete, int n)
{
                            ……     }
//-----------------------------------------------------------------
   public  boolean  miseAjour(String  requete){
                              ……
   }
//-----------------------------------------------------------------
   public  boolean  fermer( ){
    try{
       cn.close();
       return true;
      }Catch(SQLException ex){
       return false;
      } }
```

Travail à Faire :

Définir les méthodes suivantes de la classe « **Connexion** » :

IV-1. Une méthode « **connecter** » pour se connecter à la base de données en se basant sur les attributs déjà initialisés par le constructeur. Cette méthode retourne « ***true*** » en cas de succès ou « ***false*** » en cas d'échec. *(1 pt)*

IV-2. Une méthode « **lire** » permet d'exécuter une requête de sélection et retourner le résultat dans un objet « ***ResultSet*** ». En cas d'échec, elle retournera « ***null*** ». *(1 pt)*

IV-3. Une méthode « **miseAjour** » permettant d'exécuter une requête de mise à jour. Cette méthode retourne « ***true*** » en cas de succès ou « ***false*** » en cas d'échec. *(1 pt)*

IV-4. Une méthode « **toVector** » qui reçoit une ***requête*** et un nombre de champs puis retourne un ***Vector de tableaux d'objets.*** Chaque tableau contient les ***n*** champs d'un enregistrement. *(1 pt)*

❖ CLIENT

Dans cette partie, on veut implémenter un client qui envoie des requêtes au serveur d'application en utilisant les sockets en mode connecté.

Le modèle Orienté Objet du client est donné dans la figure ci-contre :

Client
- **IP : String** - **Port : int**
+Client(…) **+demande(requete : String) : Object**

Figure 7 : *Classe Client*

```
public class Client {
   private String IP;
   private int Port;
   public Client (String IP, int Port){
                              ……
   }
   public Object demande(String requete){
                              ……
   } }
```

Travail à Faire :

IV-5. Définir un Socket ***(0,5 pt)***;

IV-6. Comparer les deux modes **TCP** et **UDP** ***(0,5 pt)***;

IV-7. Définir le constructeur de la classe « **Client** » ***(0,5 pt)***;

IV-8. Définir la méthode « **demande** » qui permet de :

- Créer un Socket client ***(0,5 pt)***;
- Créer les objets de communication ***(0,5 pt)***;
- Envoyer une requête au serveur ***(0,5 pt)*** ;
- Lire la réponse provenant du serveur ***(0,5 pt)***;
- Retourner cette réponse ***(0,5 pt)***.