c. Sujet de développement des applications informatiques (DAI) :

Etude de cas : GESTION DES PROJETS

La gestion des projets est très importante dans des domaines divers, par exemple, dans le domaine de la formation post-bac. Comme cette gestion est primordiale, on a besoin de connaitre à tout moment l'état d'avancement des étudiants pour réaliser des diverses tâches associées à un projet donné.

Une vue abstraite de cette gestion nous a conduit au diagramme de classes suivant :

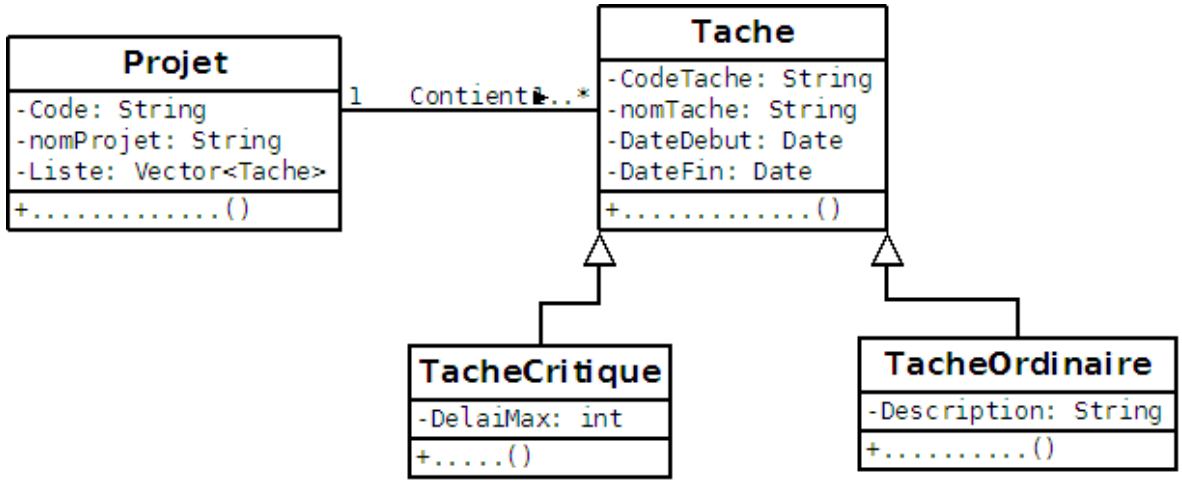

Figure 1 : Diagramme de classe

Partie A : Réalisation des classes : (14 pts)

1. Le prototype de la classe **Tache** est défini comme suit :

```
public class Tache {
private String CodeTache ;
private String nomTache ;
private Date DateDebut ;
private Date DateFin ;

public Tache(String CodeTache, String nomTache, Date DateDebut, Date DateFin)
{ ……}

public void setDateDebut(Date d)
{……}

public void setDateFin(Date d)
{……}

@Override
public String toString( )
{……}

@Override
public boolean equals(Object Obj )  {……}
}
```

a- Définir le constructeur d'initialisation d'une tâche. **(0,5 pt)**

b- Définir les deux mutateurs pour modifier les dates de début et de fin d'une tâche. **(1 pt)**

c- Donner la définition de la méthode **toString** afin de retourner une chaine porteuse d'informations sur une tâche, la chaine aura la forme suivante : **(1 pt)**

`Code:xxxx, la tâche intitulée:xxxxx, Date de début:xxxxx, Date de fin:xxxxxx`

d- Redéfinir la méthode **equals** qui compare deux tâches en se basant sur leurs codes. **(1 pt)**

2- Réalisation des classes dérivées de la classe **Tache** :

a- Créer la classe **TacheOrdinaire** qui contient : **(1 pt)**

- Un constructeur avec arguments ;
- Une méthode **toString** qui retourne une chaine décrite comme suit :

`Code:xxx, la tâche intitulée:xxx, Date de début:xxx, Date de fin:xxx,Description:xxx`

b- Créer la classe **TacheCritique** qui contient : **(1 pt)**

- Un constructeur avec arguments ;
- Une méthode **toString** qui retourne une chaine décrite comme suit :

`Code:xxx, la tâche intitulée:xxx, Date de début:xxx, Date de fin:xxx, DelaiMax:xxx`

3- Le prototype de la classe **Projet** est défini comme suit :

```
public class Projet {
private String Code ;
private String nomProjet ;
private Vector<Tache>Liste=new Vector<Tache>( ) ;

public Projet(String Code, String nomProjet)
{ ……}

public void ajouterTache(Tache d)
{……}

public void supprimerTache(int index)
{……}

 @Override
public String toString( )
{……}

@Override
public boolean equals(Object  Obj )
{……}
}
```

a- Pourquoi cette classe contient un attribut nommé **Liste** de type collection ? **(0,5 pt)**

b- Définir le constructeur d'initialisation. **(0,5 pt)**

c- Définir la méthode **ajouterTache(Tache d)** qui ajoute une nouvelle tâche à la collection en vérifiant s'elle n'existe pas déjà. **(1 pt)**

d- Définir la méthode **supprimerTache(int index)** qui supprime une tâche dont l'index est reçu comme argument. **(1 pt)**

e- Donner la définition de la méthode **toString** afin de retourner une chaîne porteuse d'informations sur un projet. La chaîne aura la forme suivante : **(2 pts)**

```
Code : xxxx , Nom projet : xxxxx
La liste des tâches de ce projet :
xxxxxxxxx
xxxxxxxxx
……
```

f- Redéfinir la méthode **equals** qui compare deux projets en se basant sur leurs codes. **(0,5pt)**

4- Test des classes : **(3 pts)**

Dans un programme de test, on demande de :

- Créer un objet **P** de type **Projet** ;
- Créer un tableau de Tache **T** de taille 2 ;
- Instancier le premier élément de **T** sur la classe **TacheOrdinaire** et le deuxième sur la classe **TacheCritique** ;
- Lister les éléments de ce tableau ;
- Ajouter ces éléments au projet **P ;**
- Afficher ce projet.

Partie B : Persistance des données : (10 pts)

Afin de rendre nos informations persistantes, on a eu recours à une modélisation relationnelle dont le modèle logique de données est illustré ci-après. Le système de gestion de base de données relationnel que l'on utilise est SQL Server.

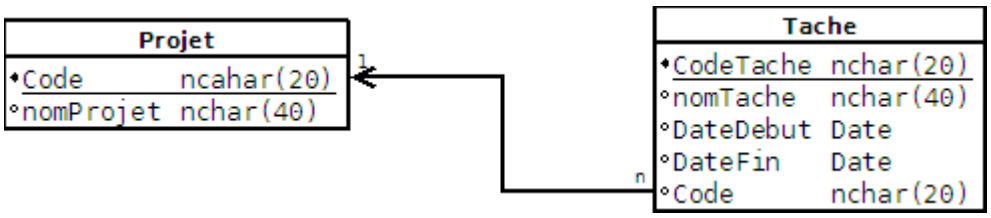

Figure 2 : Modèle logique de données

L'utilisateur dispose d'une interface sous VB.NET permettant de gérer les tâches des projets. Cette gestion consiste à :

- Ajouter une tâche à un projet ;
- Supprimer une tâche d'un projet.

L'IHM est illustrée par la figure suivante :

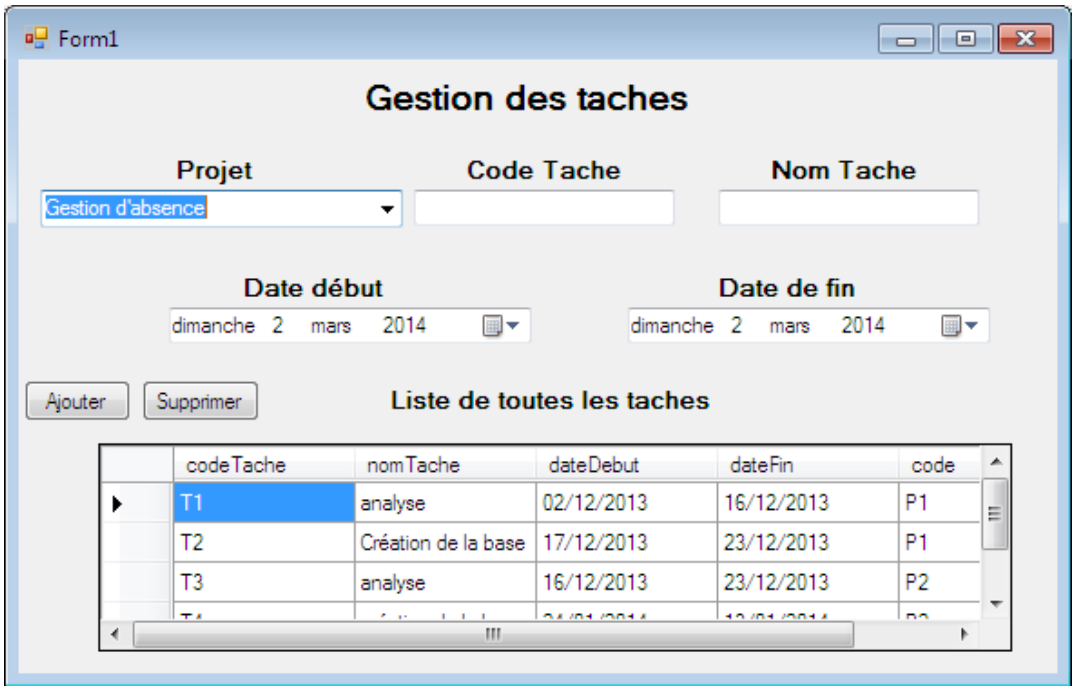

Figure 3 : IHM de gestion des tâches des projets

Objet graphique	Propriété Name
TextBox	txtCodeTache
TextBox	txtNomTache
ComboBox	cmbProjets
DataGridView	dgListeTaches
DateTimePicker	dtDebut
DateTimePicker	dtFin
Button	btnAjouter
Button	btnSupprimer

Le prototype de la classe Form est le suivant :

```
Imports System.Data.SqlClient
Public class Form1
……..    // Déclaration des objets globaux
Sub connecter()
Try
……..
Catch ex As Exception
                ……..
End Try
End Sub

Sub chargerDataSet()
……..
End Sub
Sub listerProjets()
          ……..
End Sub
```

```
Sub listerTaches()
……
End Sub


Function codeTacheExiste(ByVal Code As String) As Boolean
……
End Function

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) HandlesMyBase.Load
        connecter()
        chargerDataSet()
        listerProjets()
        listerTaches()
End Sub

Private Sub btnAjouter_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnAjouter.Click
……
End Sub

Private Sub btnSupprimer_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnSupprimer.Click
……
End Sub
End Class
```

1- Déclarer les objets nécessaires à la connexion, en mode **non connecté**, à la base de données. **(0,5 pt)**

2- Créer la procédure **connecter** qui permet de se connecter à la base de données. On donne la chaine de connexion à utiliser : **(1 pt)**

"Data Source=SERVER\SQLEXPRESS;Integrated Security=true;Initial Catalog=Base"

En cas d'erreur, on affiche un message d'erreur.

3- Donner le code de la procédure **chargerDataSet** permettant de charger le DataSet par les deux tables **Projet** et **Tache**. **(1,5 pts)**

4- Donner le code de la procédure **listerProjet** permettant de remplir le comboBox nommé **cmbProjets** par les noms des projets. **(1 pt)**

5- Donner le code de la procédure **listerTaches** permettant de lister toutes les tâches dans la grille **dgListeTaches**. **(1 pt)**

6- Donner le code de la fonction **codeTacheExiste** retournant **True** si le code reçu en argument existe dans la table **Tache** et **False** dans le cas échéant. **(1,5 pts)**

7- Donner le code du bouton **btnAjouter** qui permet l'ajout d'une nouvelle tâche à un projet. Lors de l'ajout, on peut suivre les étapes suivantes : **(2 pts)**

- Vérifier si les **textBox** et le **comboBox** ne sont pas vides ;
- Vérifier que la date début est antérieure à la date fin ;
- Vérifier si le code de la nouvelle tâche est unique ;
- Ajouter une nouvelle tâche ;
- Mettre à jour la base de données.

8- Donner le code du bouton **btnSupprimer** permettant la suppression d'une tâche sélectionnée dans la grille. Lors de la suppression, on peut suivre les étapes suivantes : **(1,5 pts)**

- Récupérer, depuis la grille, la position de la tâche sélectionnée ;
- Supprimer cette tâche ;
- Mettre à jour la base de données.

Partie C : Consultation en ligne : (8 pts)

On souhaite créer une page Web, selon le modèle donné ci-dessous, permettant à un utilisateur de consulter toutes les tâches associées à un projet choisi à partir d'une liste déroulante. Ce listing aura lieu après un clic sur le bouton dont l'étiquette est : "**Afficher details".**

Sur le tableau d'affichage des tâches, on peut supprimer une tâche via un lien hypertexte **« sup »**.

Figure 4 : Portail de consultation en ligne

On donne les notations à utiliser :

Elément	Nom
Liste déroulante	ListeProjets
Bouton de validation	Afficher

Donner le code (html + PHP) permettant :

1- d'afficher le formulaire. Les noms des projets proviennent de la table **Projet** de notre base de données Mysql nommée **Base** image de la base de données SQL Server. **(3 pts)**

2- de lister toutes les tâches associées au projet sélectionné dans la liste déroulante. **(3 pts)**

3- de supprimer la tâche choisie à partir du lien hypertexte **sup**. (2 pts)

Partie D : Communication Intranet : (8 pts)

Au cours de la réalisation des projets, les étudiants doivent consulter, à distance, les mises à jour effectuées par leurs encadrants sur la planification des tâches relatives aux différents projets.

Un étudiant peut accéder à partir de son poste au serveur d'application et de lui envoyer le code de son projet. Le serveur, disposant de la liste des projets sous forme d'une collection **ArrayList< Projet>** nommée **PFE**, recherche le projet demandé et renvoie sa description (renvoie le message **« projet inexistant »** si le projet est introuvable).

Le schéma de notre réseau Intranet est le suivant:

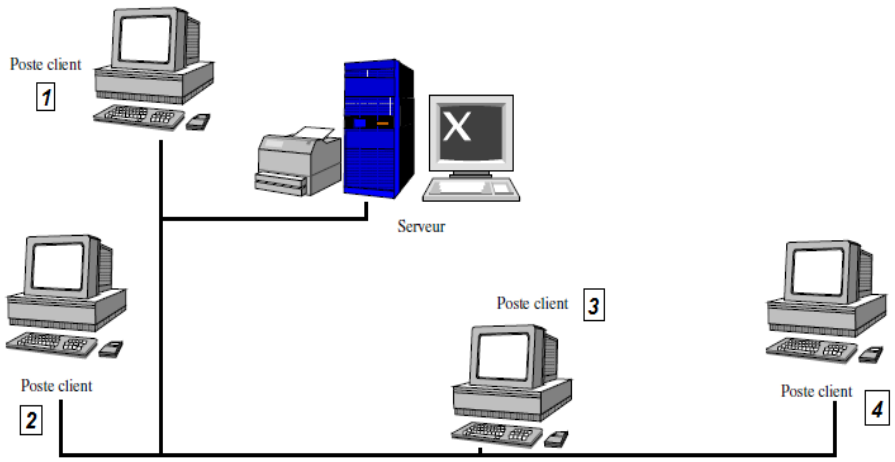

Figure 5 : Schéma du réseau Intranet

1- Questions de cours :

a- L'architecture utilisée est qualifiée de **2-tiers**, donner le modèle de **Gartner Group** correspondant. **(0,5 pt)**

b- Dans notre application, si on suppose que le serveur réalise la grande partie du traitement, de quel type de serveur s'agit-il ? **(0,5 pt)**

c- Si notre serveur veut collecter des données depuis un SGBDR via JDBC, préciser les deux objets nécessaires pour effectuer cette tâche et donner le rôle de chacun. **(1 pt)**

2- Programme Client :

a- Ecrire une méthode **socketConnexion** qui permet d'établir la connexion TCP/IP avec le serveur de sockets et initialise les objets **in** et **out**. La méthode retourne un flag booléen qui contient **True** en cas de succès ou **False** le cas échéant. L'adresse IP et le port de serveur des sockets sont reçus comme arguments. **(1 pt)**

b- Ecrire la méthode **demandeTaches** qui envoie au serveur le code de projet et retourne l'objet reçu à partir du serveur. **(1 pt)**

```
public class Client {
private Socket sc=null;
```

```
private DataInputStream  in = null;
private PrintStream  out=null;
public boolean socketConnexion (String IP, int Port )
{……}
public Object demandeTaches(String codeProjet)
{……}
}
```

3- Programme Serveur :

a- Ecrire une méthode **acceptConnexion** qui permet d'accepter une demande de connexion TCP/IP (le numéro de port sera reçu comme argument) et retourne **True** en cas de succès ou **False** en cas d'échec. **(1 pt)**

b- Ecrire une méthode **envoiTaches** qui permet d'envoyer une chaîne descriptive du projet dont le code est reçu de la part du client. **(1,5 pts)**

```
public class Serveur {
private ServerSocket socketserver
private Socket sc=null;

private DataInputStream  in = null;

private PrintStream  out= null;
public boolean acceptConnexion (int Port)
{……}

public void envoiTaches()
{……}
}
```

c- Notre programme serveur traite un client à la fois. Proposer une solution permettant une connexion multi-clients (code demandé). **(1,5 pts)**