**Mission 2 : Inscription des participants**

**Tâche 1 : Amélioration des inscriptions**

*(Voir le fichier suivant : Utilitaire.cs)*

Choix : Fonction dans utilitaires qui effectue une boucle foreach sur les différents éléments et prends le groupbox les contenant en paramètre. Nous avons fait ce choix car on fait appel à cette fonction quand on enregistre un bénévole, ou un intervenant ou un licencié.

public static void ViderGroupBox(Control UnControl, Boolean pVisible)

{

foreach (Control c in UnControl.Controls)

{

if (c is TextBox)

{

((TextBox)c).Clear();

}

else

{

if (c is MaskedTextBox)

{

((MaskedTextBox)c).Clear();

}

else

{

if (c is RadioButton)

{

((RadioButton)c).Checked = false;

}

else

{

if (c is CheckBox)

{

((CheckBox)c).Checked = false;

}

}

}

}

}

if (pVisible)

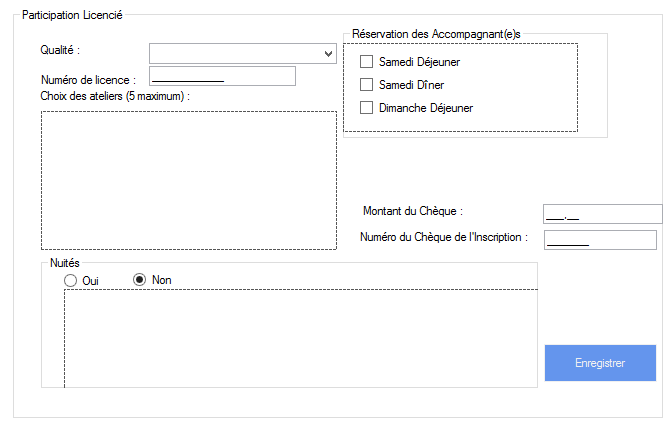
{

UnControl.Visible = false;

}

}

**Tâche 2 : Inscription des licenciés***(Voir les fichiers suivants : FrmPrincipale.cs, FrmPrincipale.csx, Bdd.cs)* Pour l’inscription des licenciés, nous avons commencé par coder l’interface, en voici un aperçu :

  
  
Ensuite, nous avons en nous appuyant sur le code de l’inscription des bénévoles et des intervenants écrit une fonction InscrireLicencie() surchargée (avec ou sans nuitées) après plusieurs contrôles dont celle avec nuitée est :

public void InscrireLicencie(String pNom, String pPrenom, String pAdresse1, String pAdresse2, String pCp, String pVille, String pTel, String pMail, Int16 pIdQualite, Int64 pNumeroLicence, Collection<Int16> pLesAteliers, Int64 pNumCheque, Int64 pMontantCheque, Collection<Int16> pLesAccompagnants, String pInscription)

{

String MessageErreur = "";

try

{

// pckparticipant.nouvelintervenant est une procédure surchargée

UneOracleCommand = new OracleCommand("pckparticipant.nouveaulicencie", CnOracle);

UneOracleCommand.CommandType = CommandType.StoredProcedure;

// début de la transaction Oracle : il vaut mieyx gérer les transactions dans l'applicatif que dans la bd.

UneOracleTransaction = this.CnOracle.BeginTransaction();

this.ParamCommunsNouveauxParticipants(UneOracleCommand, pNom, pPrenom, pAdresse1, pAdresse2, pCp, pVille, pTel, pMail);

this.ParamsSpecifiquesLicencie(UneOracleCommand, pIdQualite, pNumeroLicence, pNumCheque, pMontantCheque, pInscription);

// si aucun atelier, on rempli la collection d'un atelier à 0 afin de ne rien faire dans la base

if (pLesAteliers.Count == 0)

{

pLesAteliers.Add(0);

}

// Paramètre qui stocke les ateliers sélectionnées

OracleParameter pOraLesAteliers = new OracleParameter();

pOraLesAteliers.ParameterName = "pLesAteliers";

pOraLesAteliers.OracleDbType = OracleDbType.Int16;

pOraLesAteliers.CollectionType = OracleCollectionType.PLSQLAssociativeArray;

pOraLesAteliers.Value = pLesAteliers.ToArray();

pOraLesAteliers.Size = pLesAteliers.Count;

UneOracleCommand.Parameters.Add(pOraLesAteliers);

// si aucun accompagnant, on rempli la collection d'un accompagnant à 0 afin de ne rien faire dans la base

if (pLesAccompagnants.Count == 0)

{

pLesAccompagnants.Add(0);

}

// Paramètres qui stocke les accompagnants sélectionnées

OracleParameter pOraLesAccompagnants = new OracleParameter();

pOraLesAccompagnants.ParameterName = "plesaccompagnants";

pOraLesAccompagnants.OracleDbType = OracleDbType.Int16;

pOraLesAccompagnants.CollectionType = OracleCollectionType.PLSQLAssociativeArray;

pOraLesAccompagnants.Value = pLesAccompagnants.ToArray();

pOraLesAccompagnants.Size = pLesAccompagnants.Count;

UneOracleCommand.Parameters.Add(pOraLesAccompagnants);

//execution

UneOracleCommand.ExecuteNonQuery();

// fin de la transaction. Si on arrive à ce point, c'est qu'aucune exception n'a été levée

UneOracleTransaction.Commit();

}

catch (OracleException Oex)

{

MessageErreur = "Erreur Oracle \n" + this.GetMessageOracle(Oex.Message);

}

catch (Exception ex)

{

MessageErreur = "Autre Erreur, les informations n'ont pas été correctement saisies";

}

finally

{

if (MessageErreur.Length > 0)

{

// annulation de la transaction

UneOracleTransaction.Rollback();

// Déclenchement de l'exception

throw new Exception(MessageErreur);

}

}

}

Nous avons également modifié et créer les fonctions supplémentaires et complémentaires à nos fonctions (telles que rdbStatutLicencie\_StateChanged(), VerifBtnEnregistrerLicencie() etc).

**Tâche 3 : Confirmation d’inscription par mail**

*(Voir le fichier/dossier : Utilitaire.cs)*

Comme expliqué dans la description de la tâche, l’envoi de mail peut se faire sous Oracle ou en C#. La première technique soit, l'envoi de mail via la base de données est fiable, et facilement pris en charge. Le contenu d'un mail peut être le résultat d'une requête ou même des fichiers de n'importe quelle ressource du réseau. De plus, oracle pourra informer l'administrateur si un traitement ne s'est pas déroulé correctement. Cependant, la seconde solution est aussi fiable et permet une multitude de contrôles avec la génération d’exceptions comme sous oracle. Dans l’application nous n’avons pas choisi d’envoyer par mail le résultat d’une requête.   
 Notre choix final se porte donc sur la solution C# étant dans un premier temps plus facile à mettre en œuvre mais permettant également plusieurs contrôles tels que les Regex ou les exceptions qui sont immédiatement affichées dans un message box. Nous avons par ailleurs créé 3 fonctions surchargées pour correspondre aux contraintes imposées par le cahier des charges, voici celle concernant la confirmation d’une inscription :

public static void EnvoyerMail(String pDestinataire, String pNom, String pPrenom)

{

try

{

if (Regex.IsMatch(pDestinataire, @"^([0-9a-zA-Z]([-\.\w]\*[0-9a-zA-Z])\*@([0-9a-zA-Z][-\w]\*[0-9a-zA-Z]\.)+[a-zA-Z]{2,9})$"))

{

MailMessage mail = new MailMessage();

using (SmtpClient smtp = new SmtpClient(ConfigurationManager.AppSettings["SmtpServeur"]))

{

mail.From = new MailAddress(ConfigurationManager.AppSettings["SmtpFrom"], ConfigurationManager.AppSettings["TitreApplication"]);

mail.To.Add(pDestinataire);

mail.Subject = ConfigurationManager.AppSettings["SmtpSubject"];

mail.Body = " Votre inscription à l'atelier " + pAtelier + " au nom de " + pNom + " " + pPrenom + " n'a pas pu être prise en compte ce jour (" + DateTime.Now + ") car celui-ci était complet.\n\nCordialement, l'équipe de maison des ligues.";

smtp.Port = 587;

smtp.EnableSsl = true;

smtp.UseDefaultCredentials = false;

smtp.DeliveryMethod = SmtpDeliveryMethod.Network;

smtp.Credentials = new System.Net.NetworkCredential(ConfigurationManager.AppSettings["SmtpFrom"], ConfigurationManager.AppSettings["SmtpPwd"]);

smtp.Send(mail);

MessageBox.Show("Un mail vous a été envoyé pour vous informer que l'inscription a un atelier s'est mal déroulée.");

}

}

else

{

throw new Exception("L'adresse entrée est incorrecte, le mail de confirmation n'a pas pu être envoyé.");

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

**Tâche 4 : Mise en place des contrôles**

Dans un premier temps, afin de faire correspondre le montant du ou des chèques au total des prestations, nous proposons de faire un contrôle lors de l’appui sur le bouton Enregistrer qui va calculer le montant total dû en fonction des nuitées, des hôtels ect.

Ensuite, pour contrôler la possibilité d’avoir un ou deux paiements (chèques) par inscription, nous pensons qu’il est préférable de tout gérer côté C#. Ainsi, lors de l’appuie sur le bouton d’enregistrement d’un licencié, nous contrôlons si il existe un ou deux chèques, puis nous calculons le montant total que doivent faire chacun des deux chèques. Enfin, on vérifie que le montant des deux chèques (ou d’un seul chèque) est bien égal au montant total. Si tout se passe bien, on enregistre le licencié.

Le fait de limiter un hébergement à une seule chambre pour une même nuit peut se faire au niveau de l’interface en interdisant la sélection de deux chambres.  
  
  
 Enfin, pour inscrire d’office un participant dans le 6ème atelier si celui-ci n’est pas complet lors de son inscription dans un atelier qu’il l’est, nous admettons que si le 6ème atelier est complet un mail est envoyé et l’inscription est impossible et sinon si les contrôles sont passés avec succès le participant est inscrit dans le 6ème atelier. Nous avons eu l’idée de créer une procédure dans le PCKPARTICIPANT Body pour envoyer un mail.

CREATE OR REPLACE PROCEDURE send\_mail (pTo IN VARCHAR2,

pNom IN VARCHAR2,

pPrenom IN VARCHAR2,

pAtelier IN VARCHAR2)

AS

MailCnx UTL\_SMTP.connection;

BEGIN

MailCnx := UTL\_SMTP.open\_connection(‘smtp.gmail.com’, 527);

UTL\_SMTP.helo(MailCnx, smtp.gmail.com);

UTL\_SMTP.mail(MailCnx, ‘ppe2\_maisondesligues@gmail.com’);

UTL\_SMTP.rcpt(MailCnx, pTo);

UTL\_SMTP.data(MailCnx, « Votre inscription à l'atelier " || pAtelier || " au nom de " || pNom || " " || pPrenom || " n'a pas pu être prise en compte ce jour (" || DateTime.Now || ") car celui-ci était complet.\n\nCordialement, l'équipe de maison des ligues.»);

UTL\_SMTP.quit(MailCnx);

END;

**Tâche 5 : Contraintes du MCD**

Selon le schéma conceptuel des données, il existe trois contraintes. La première concernant les relations Intervenir et Animer et ayant pour pivot les entités Atelier et Intervenant, est une contrainte d’exclusion totale et elle s’explique par le fait qu’un intervenant ne peut pas animer et intervenir en même temps sur un atelier mais qu’il doit faire au moins l’un des deux. La seconde, elle, s’établit entre les relations Participer et Inscrire et a comme pivot les entités Atelier et Licencié. On peut justifier cette inclusion par le fait que tout licencié qui s’inscrit à un atelier doit y participer. Pour finir, la dernière contrainte prend en compte les relations Inclure et EtrePresent et a comme pivot la table Participant. Cette exclusion est présente car un participant étant bénévole ne peut pas être hébergé.  
 Deux des trois contraintes ont été gérées différemment au niveau du MLD. Celle qui concerne les intervenants a été gérée avec l’ajout d’une entité « Statut » permettant de savoir si l’intervenant intervient ou participe sur un atelier. Et celle qui s’établit sur l’inscription des licenciés à un atelier est régie par une contrainte entre les deux associations transformées en tables qui indique que l’idParticipant doit être égale à l’idLicencie et que l’idAtelier doit être le même.  
 Afin d’assurer la gestion de bénévole et de leur non-hébergement, nous avons opter pour la solution de deux triggers (un pour chaque table) qui avant une insertion vérifie respectivement si l’id du participant n’est pas présent dans l’autre table. Exemple : Si on inscrit un bénévole il faut vérifier que l’idBenevole soit différent de tous les idParticipant de la table contenuhebergement.

Code des deux triggers :

create or replace

trigger trgbiu\_benevole before insert or update on benevole

for each row

declare

nb integer;

nb2 integer :=0;

heberge exception;

begin

select 1 into nb from dual where not exists(select numerolicence from benevole

where numerolicence= :new.numerolicence) ;

select count(\*) into nb2 from contenuhebergement

where idparticipant=:new.idbenevole;

if nb2 >0 then

raise heberge;

end if;

exception

when no\_data\_found then

raise\_application\_error(-20110, 'Bénévole déjà inscrit, \n vous devez faire une modification de bénévole');

when heberge then

raise\_application\_error(-20111 ,'Le participant est hébergé, il ne peut pas être bénévole.');

when others then

raise\_application\_error(-20002, 'Erreur à l''enregistrement');

end;

create or replace

trigger trgbiu\_contenuhebergement before insert or update on contenuhebergement

FOR EACH ROW

declare

nb integer:=0;

dejabenevole exception;

begin

select count(\*) into nb from benevole

where idbenevole=:new.idparticipant;

if nb >0 then

raise dejabenevole;

end if;

exception

when others then

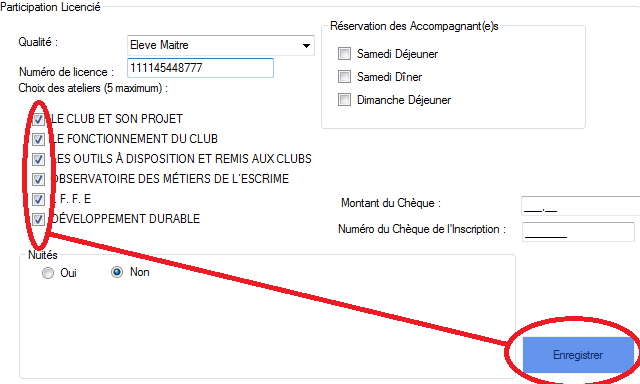
raise\_application\_error(-20112 ,'le participant est bénévole, il ne peut être hébergé');

end;

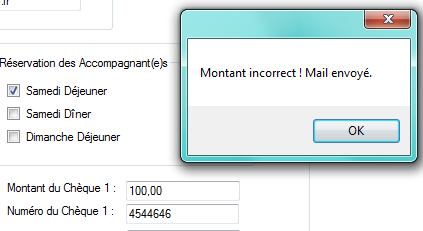
**Tâche 6 : Rapport de tests**

Pour contrôler le code ajouté dans les tâches précédentes, nous avons dû créer plusieurs tests :

* Le premier concerne l’interface, c’est un test permettant de voir si les contrôles sont efficaces, nous attendions comme résultat que le bouton Enregistrer soit grisé lorsqu’il manque des informations ou que le licencié sélectionne 6 ateliers. Celui que nous avons obtenu est le bon (voir les screens). Pour ce dernier nous pouvons en conclure que les contrôles effectués sur l’interface sont efficaces et que le licencié ne peut pas faire une fausse inscription.

****

* Le second s’établit sur l’enregistrement d’un licencié, c’est un test permettant de savoir si les mails s’envoient correctement et si les erreurs concernant les montants sont bien détectées. Nous attendions comme résultat des messagesbox affichant les exceptions ou informant l’utilisateur du bon déroulement de son inscription. Celui que nous avons obtenu est le bon puisque tous les contrôles sont encore une fois efficaces et aucunes erreurs ne peuvent être commises par l’utilisateur. En conclusion, on peut dire que ce test est réussi.



**Tâche 7 : Documentation technique**