Cow-Lock Project

Sommaire :

Introduction …

Cow-Lock est un logiciel de gestion de colocation ayant pour but :

- de créer une instance de colocation
- de centraliser les informations concernant cette colocation
- de rajouter des colocataires sur la colocation définie
- d'assurer le suivi des dépenses au quotidien
- d'avoir une vue globale de ces dépenses mensuellement
- de permettre la répartition de ces dépenses à tous les colocataires présents

Expression des besoins

Après étude auprès de colocataires et expérimentation personnelle, l'idée de développer une application sur ce thème exprime un réel besoin.

On distingue deux cas de figure dans les colocation actuellement :

1. Le cas ou chaque colocataire fait compte à part

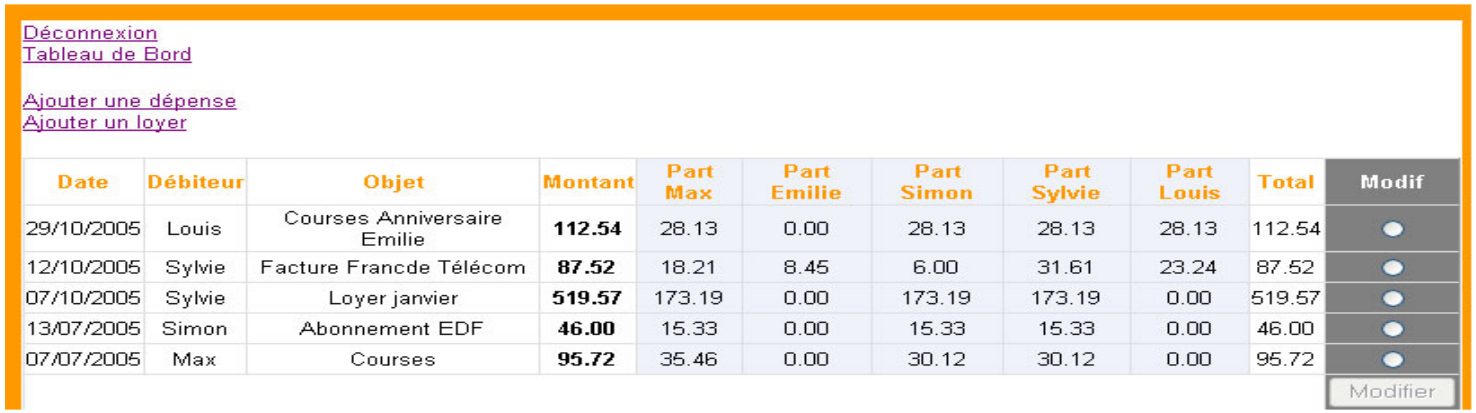
   - Dépenses séparées, gestion individuelle
   - Cette méthode est génératrice de conflits au sein de la colocation

2. Le cas ou les colocataires font compte commun
   - Du temps à passer pour effectuer la répartition à la fin du mois
   - Un calcul fastidieux
   - C'est ici que le logiciel « Cow-Lock » intervient

Positionnement :

Cow-Lock souhaite se placer sur le marché internet de gestion de colocations ou bon nombre de sites existent actuellement pour mettre en relation des colocataires.

Aucun projets de ce type à été répertorié internationalement à ce jour.



Déconnexion
Tableau de Bord

Ajouter une dépense
Ajouter un loyer

Date	Débiteur	Objet	Montant	Part Max	Part Emilie	Part Simon	Part Sylvie	Part Louis	Total	Modif
29/10/2005	Louis	Courses Anniversaire Emilie	**112.54**	28.13	0.00	28.13	28.13	28.13	112.54	
12/10/2005	Sylvie	Facture Francde Télécom	**87.52**	18.21	8.45	6.00	31.61	23.24	87.52	
07/10/2005	Sylvie	Loyer janvier	**519.57**	173.19	0.00	173.19	173.19	0.00	519.57	
13/07/2005	Simon	Abonnement EDF	**46.00**	15.33	0.00	15.33	15.33	0.00	46.00	
07/07/2005	Max	Courses	**95.72**	35.46	0.00	30.12	30.12	0.00	95.72	

Modifier

Un logiciel de gestion de colocation à été développé par le passé mais n'a pas été suivi faute de temps à y consacrer par son créateur.

Des logiciels financiers tel que « EBP » possède quelques fonctions pouvant assurer la gestion de colocation mais ne sont pas vraiment adaptés car très génériques.

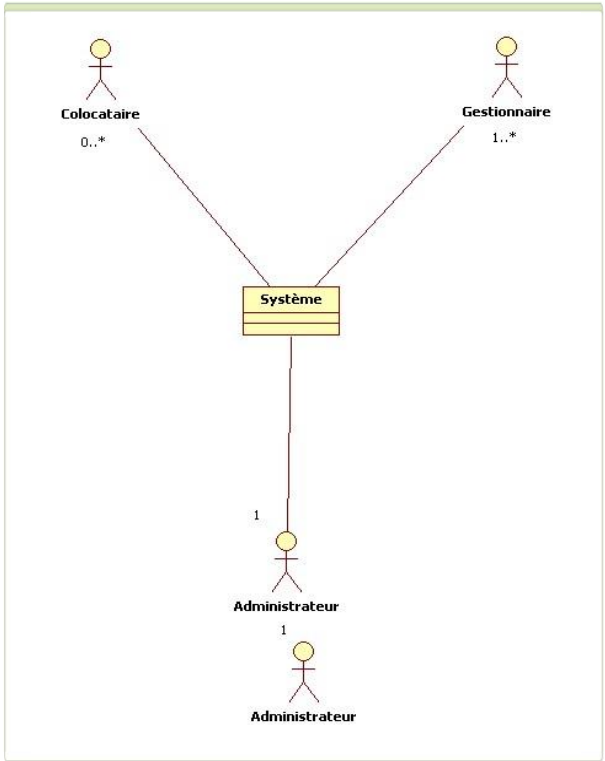
Choix Techniques

- Modélisation U.M.L piloté par le Processus 2TUP
- Architecture 2 Tiers
- SQL Server 2008 comme SGBDR
- Déploiement du client avec ASP. Net et ADO. Net
- Langage C# pour tout le codage
- AJAX pour certains contrôles

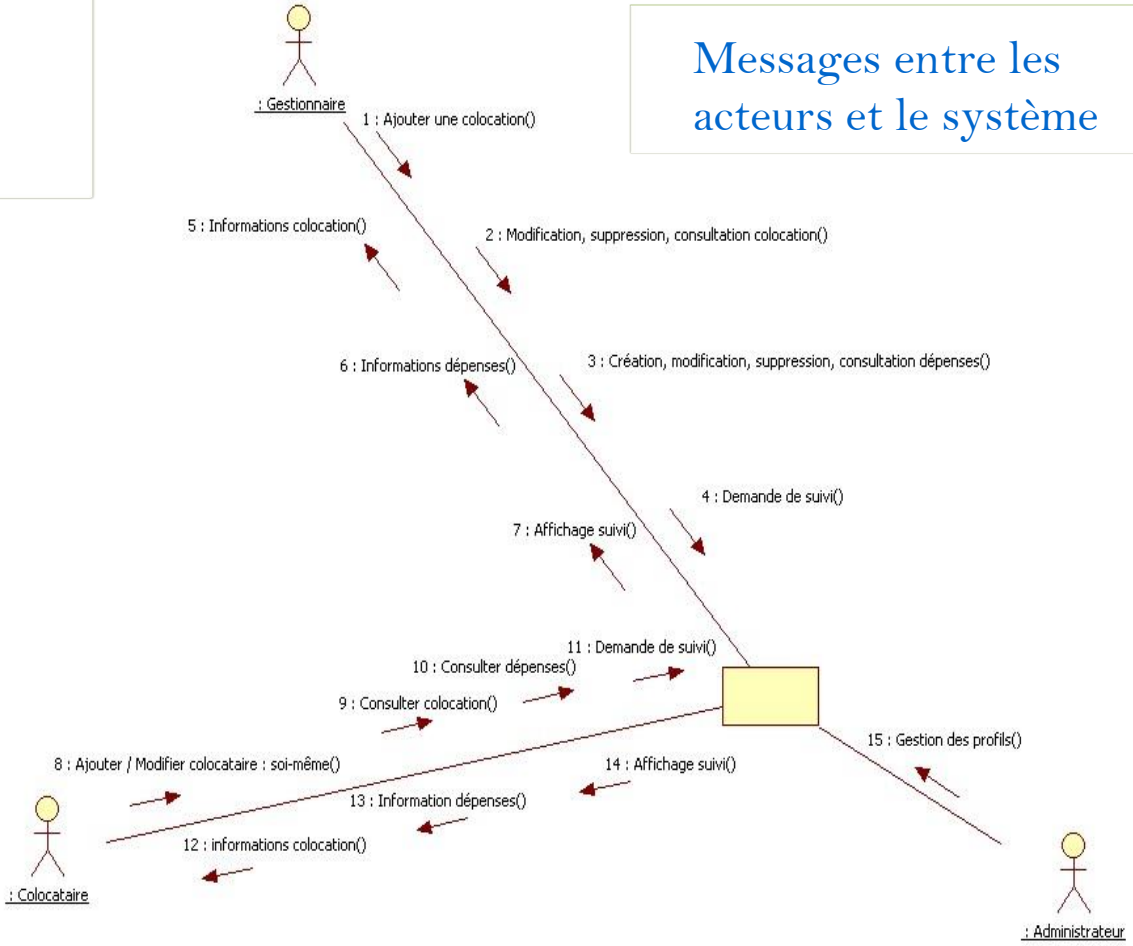
Acteurs et Messages

On distingue 3 niveaux d'accès :

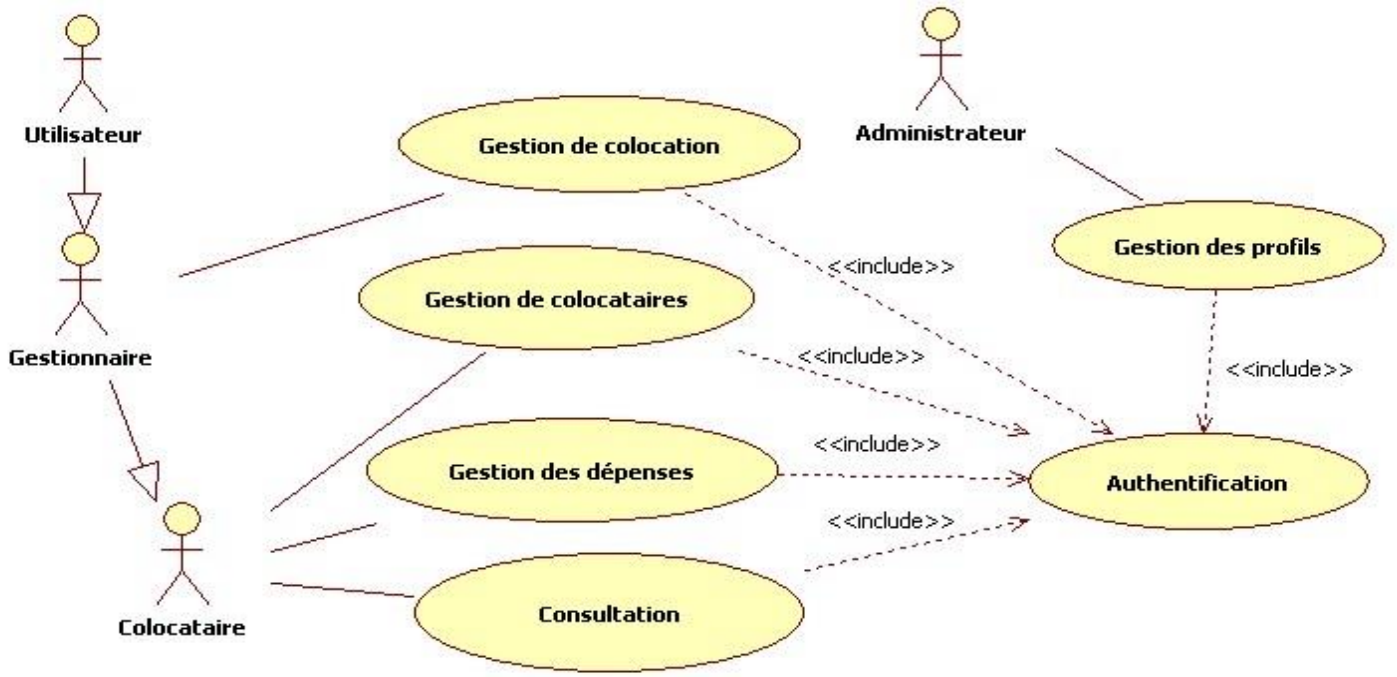
- 0. Utilisateur enregistré
- 1. Colocataire
- 2. Gestionnaire



Messages entre les acteurs et le système



Cas d'utilisations



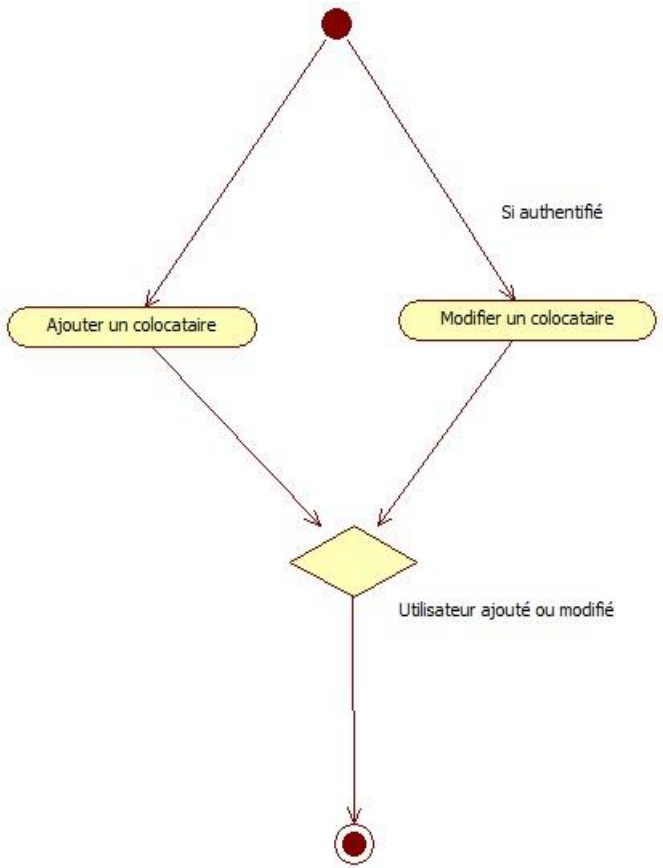
* Cas 1: Gestion des Colocataires
* Cas 2: Gestion de la Colocation
* Cas 3: Gestions des Dépenses
* Cas 4: Suivi des Dépenses

Cas d'utilisation 1 : Gestion des Colocataires

Descriptif des enchainements :

- 1. Ajouter un Colocataire
  - Saisie des données personnelles et d'identification.
  - Authentification de l'utilisateur sur le système.

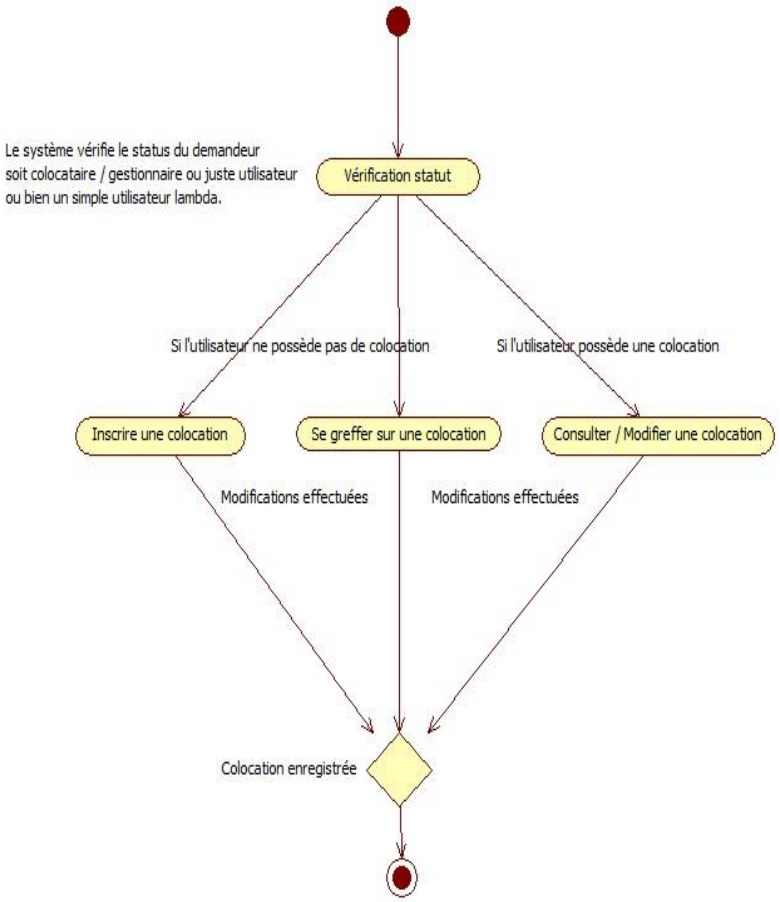
- 2. Modifier un colocataire
  - Affichage des informations de l'utilisateur.
  - Modifications apportées.
  - Enregistrement des modifications par le système.



Cas d'utilisation 2 : Gestion de la Colocation

Descriptif des enchainements :

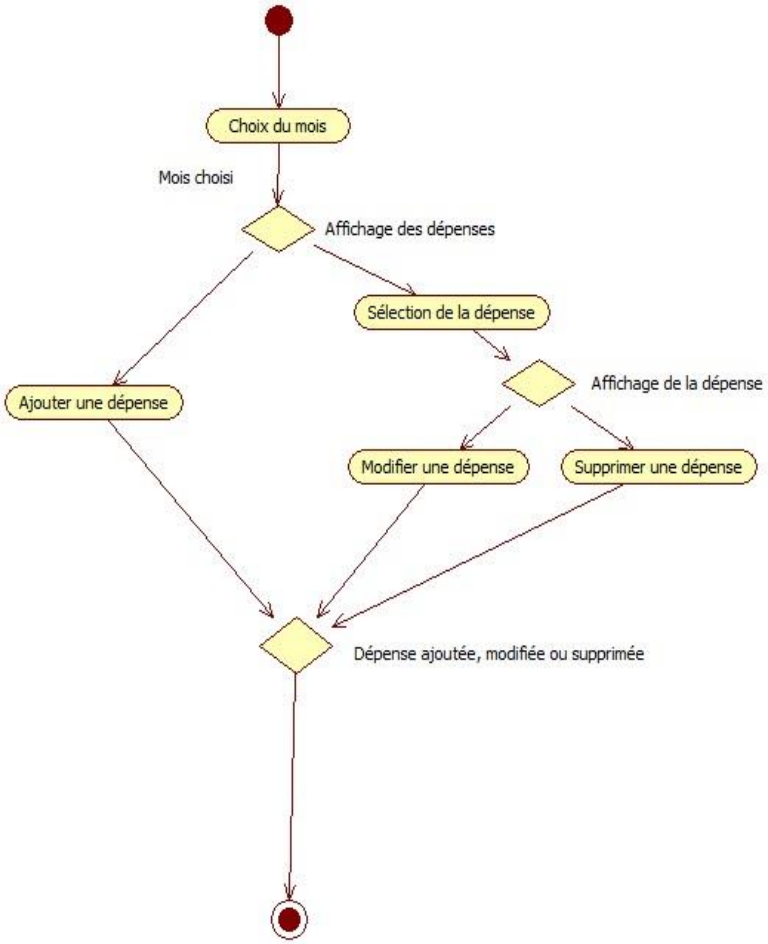
- 1. Vérification du statut
  - Utilisateur authentifié ?
  - Possède une colocation ?
- 2. Enregistrer une colocation
  - Saisie des informations
  - Association de l'utilisateur à la colocation
  - Enregistrement
- 3. Se greffer sur une colocation
  - Saisie du login/mdp de la colocation
  - Enregistrement du colocataire
- 4. Consulter / modifier une colocation
  - Affichage des informations colocataires & colocation
  - Saisie des modification si gestionnaire
  - Enregistrement des modifications



Cas d'utilisation 3 : Gestion des Dépenses

Descriptif des enchainements :

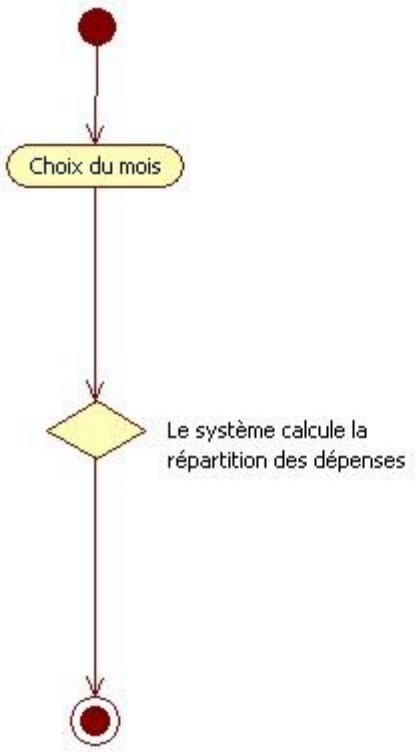
- 1. Consulter les dépenses
  - o L'utilisateur choisis le mois
  - o Affichage des dépenses
  - o Sélection de la dépense
  - o Affichage de la dépense
- 2. Ajouter une dépense
  - o Saisie des informations
  - o Enregistre la dépense
- 3. Modifier une dépense
  - o Saisie les modifications
  - o Enregistre les modifications
- 4. Supprimer une dépense
  - o Valide la demande
  - o Supprime



Cas d'utilisation 4 : Suivi des Dépenses

Descriptif des enchainements :

- 1. Suivi des dépenses
  - L'utilisateur choisis le mois
  - Affichage de la colocation
  - Affichage du montant du loyer et des charges
  - Affichage des informations sur les dépenses
  - Affichage du montant total et du nombre de dépenses
  - Affichage du montant que doit payer chaque colocataires
  - Affichage de la répartition de ces dépenses



Analyse: Classes participantes

Opérations Classiques:

- 1. Ajout
- 2. Suppression
- 3. Consultation
- 4. Modification

Autres Opérations:

- 1. Vérification statut
- 2. Assigner un colocataire
- 3. Dés assigner un colocataire
- 4. Modifier mdp. de colocation
- 5. Récupérer Identifiants

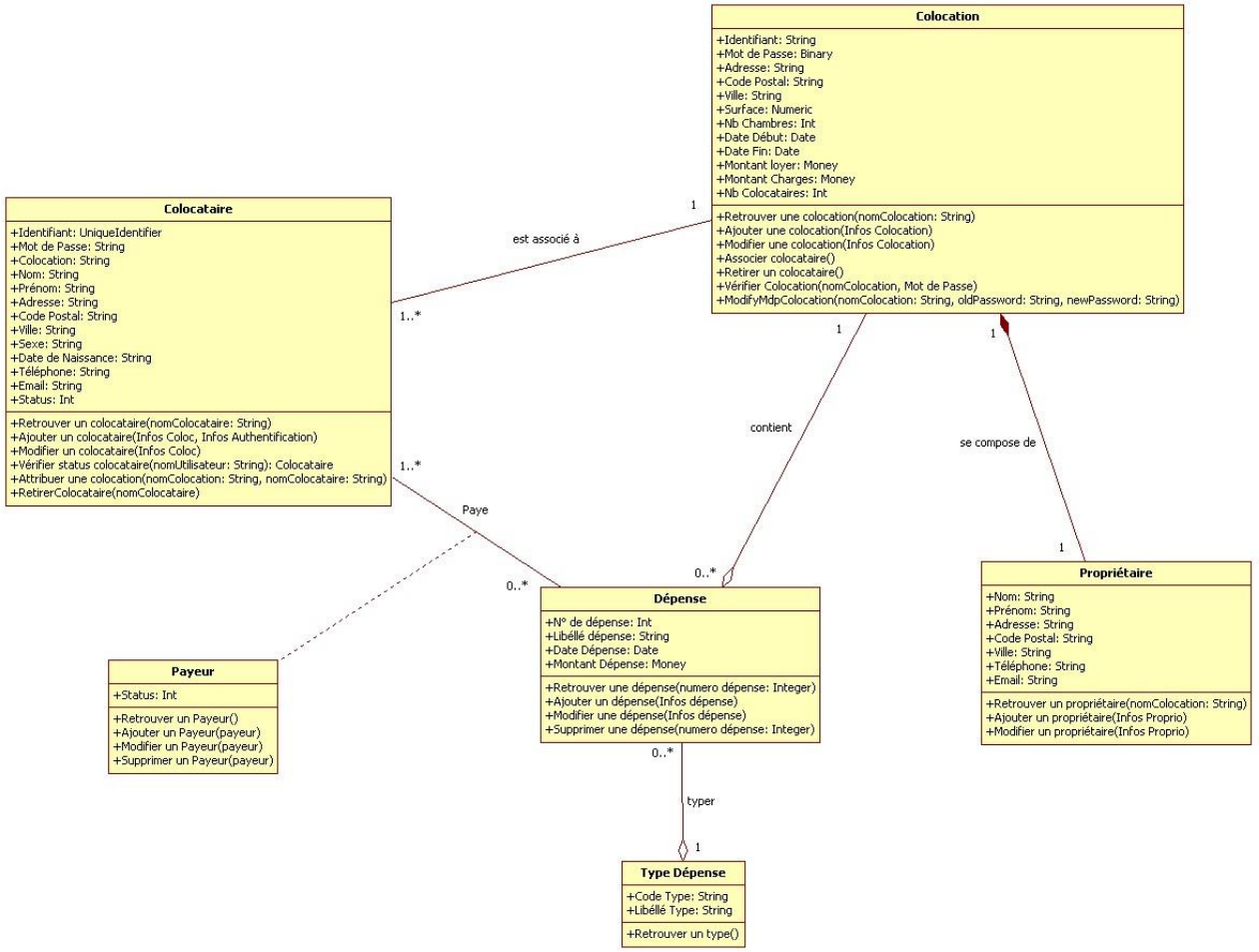


Diagramme de Séquence 1: Gestion des colocataires

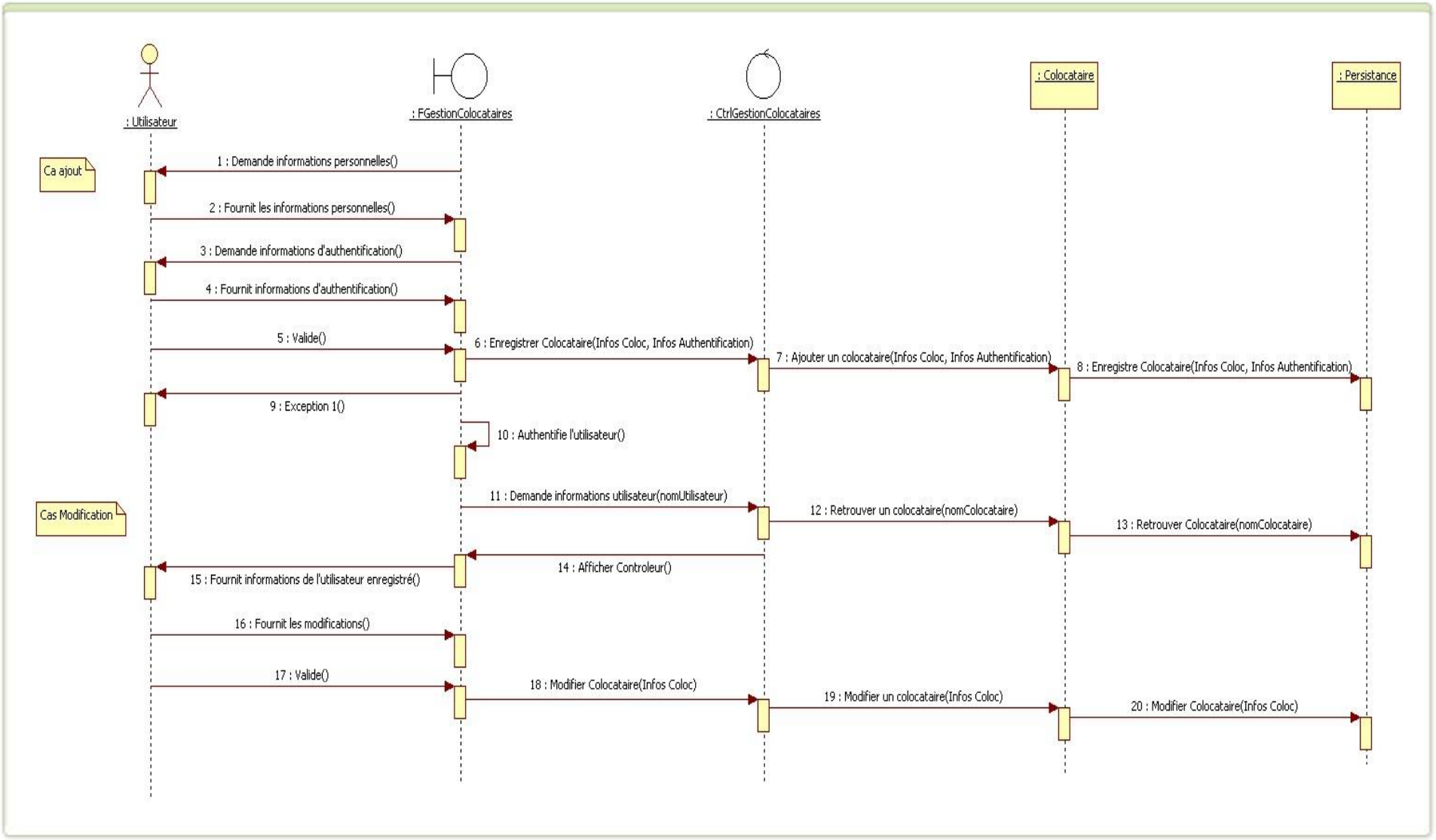


Diagramme de Séquence 2: Gestion de colocation

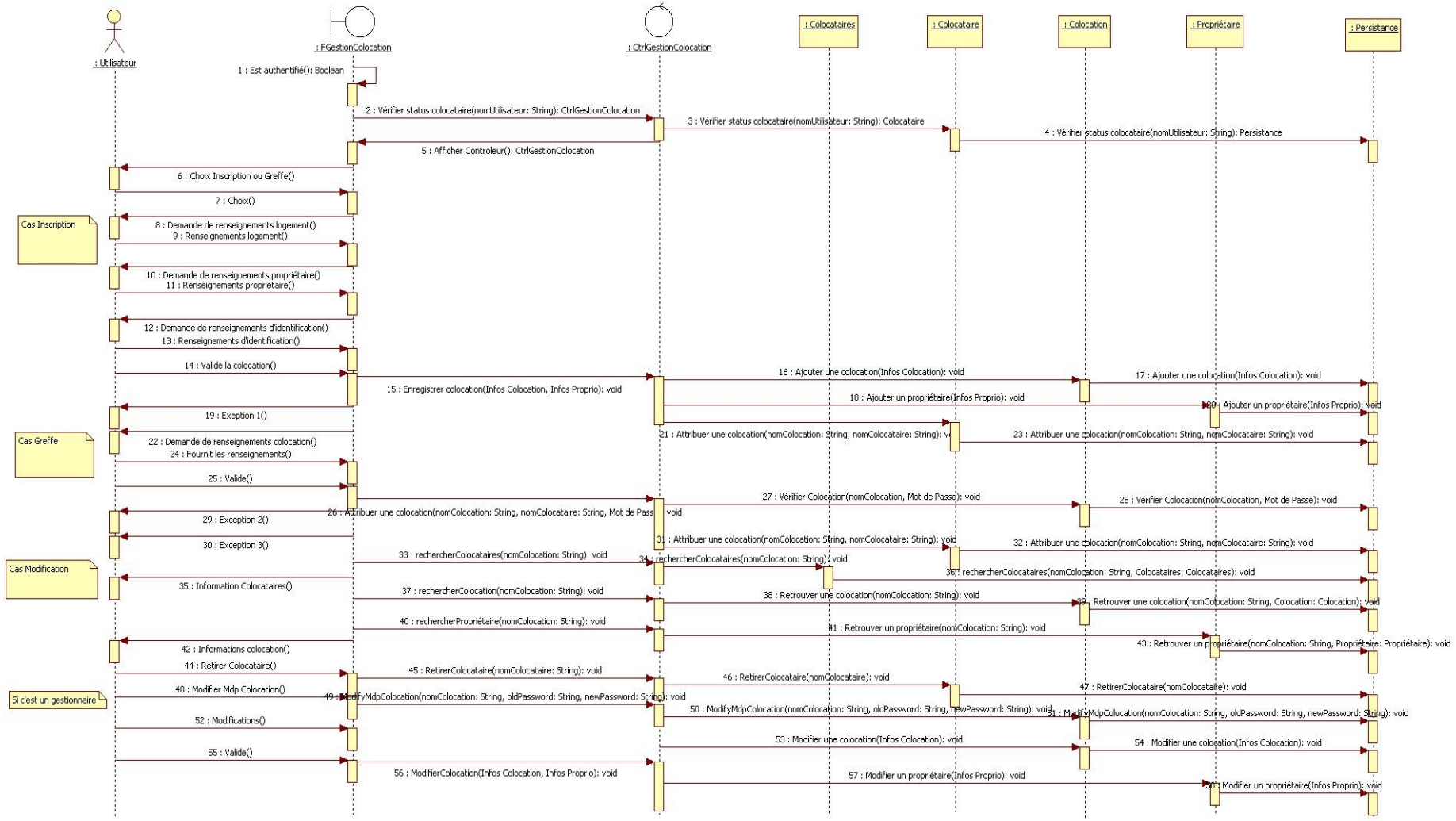


Diagramme de Séquence 3: Gestion des dépenses

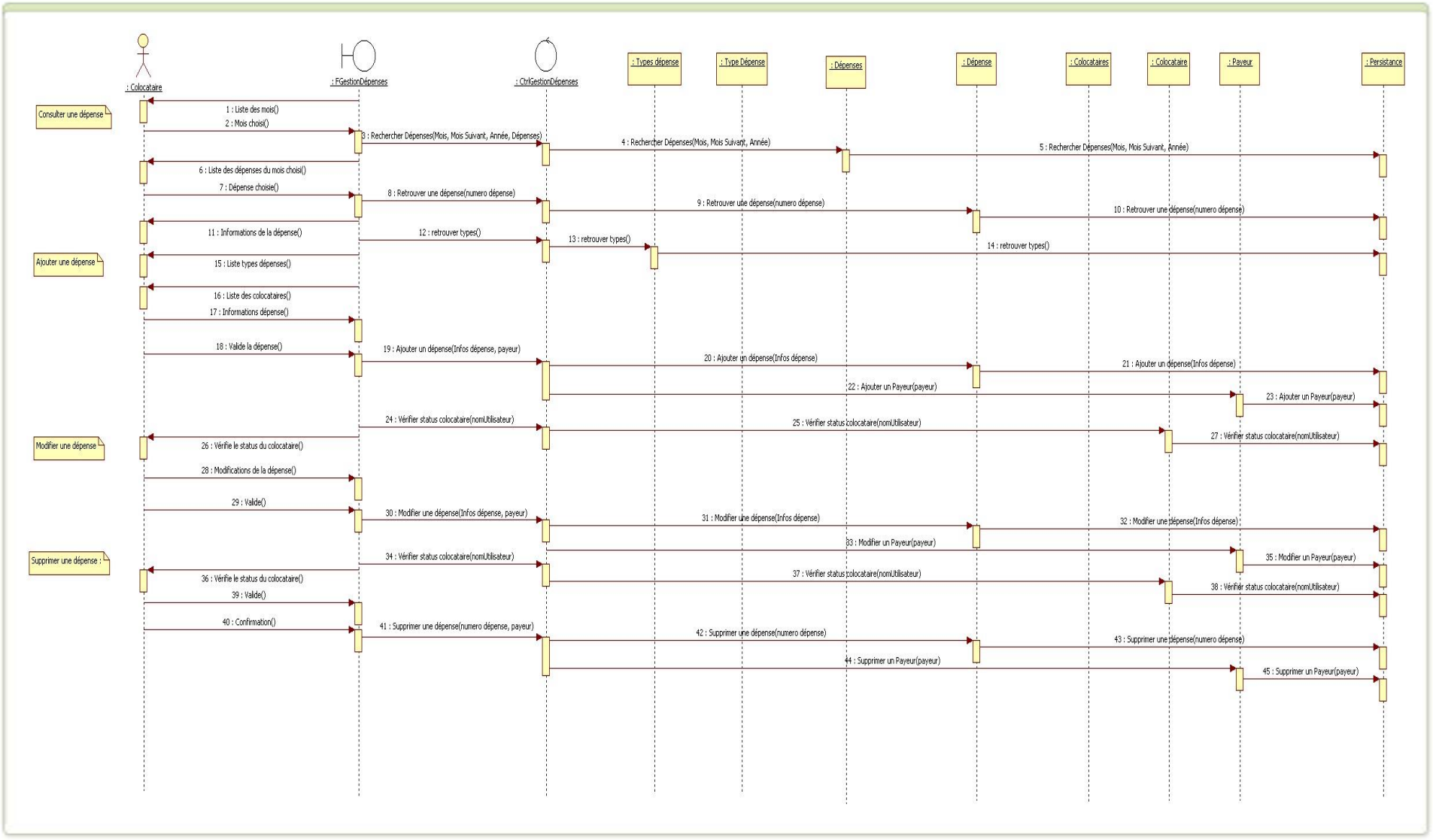
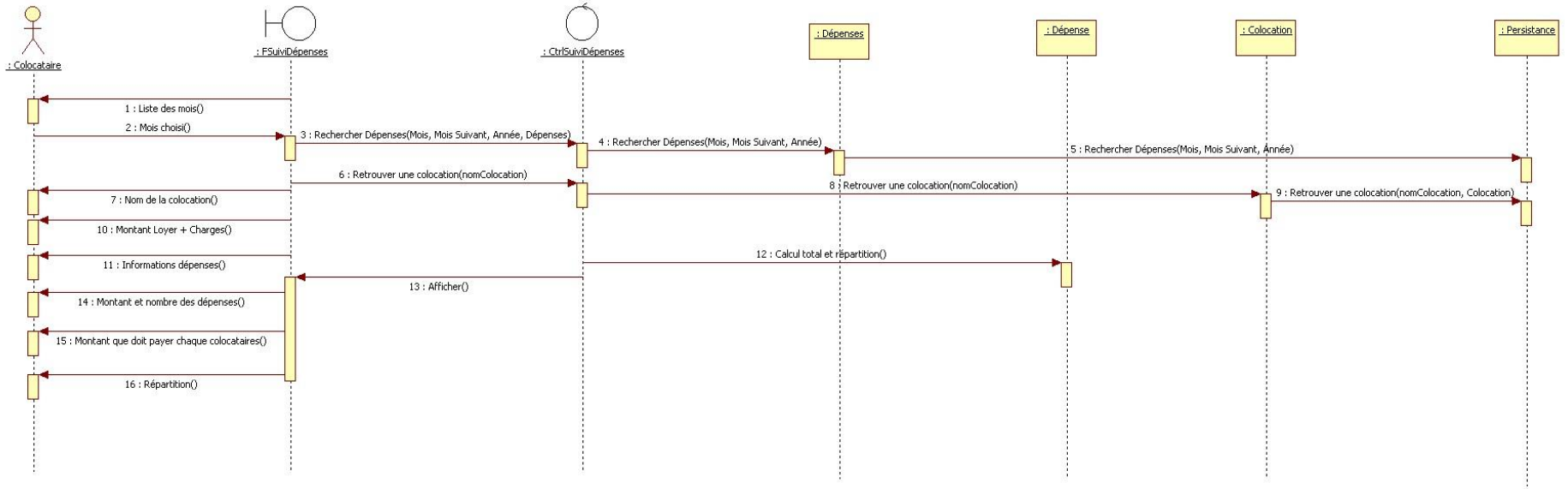


Diagramme de Séquence 4: Suivi des dépenses



Clef de voute du logiciel.

Ici s'effectue :

- Le calcul du montant total des dépenses.
- Le calcul du montant que chaque colocataire doit payer.
- Le calcul de la répartition de ces sommes en fonction de ce que chacun à payé.

IHM: CSS, AJAX & XHTML

Emplacement des cadres gérés par CSS…

Fichiers CSS différents pour Internet Explorer ou Firefox !

Gestion utilisateur et Login

Menu Dynamique

liens qui s'active en fonction du statut de l'utilisateur : membre, colocataire ou gestionnaire

Présentation des classes : La classe Colocataire

Son but ? … Créer, Assigner, Dés assigner, Rechercher & Modifier un colocataire !

```
public class Colocataire
{
    Persistance sBdd;
    private string sUserName;
    private string sNom;
    …
    { get … set }
    …
    public Colocataire(string userName)       // Constructeur
    {
        Bdd = new Persistance(userName);
        sBdd = Bdd;
    }
    public int rechercherColocataire(string userName)
    {
        try
        {
            // détail méthode
        }
        …
```

- Jointure avec le Membership Provider ASP
- Ajout de champs propres tel que Adresse…
- Premier système d'authentification

Présentation des classes : La classe Colocation

Les méthodes… Créer rechercher et modifier une colocation. Identification et vérification.

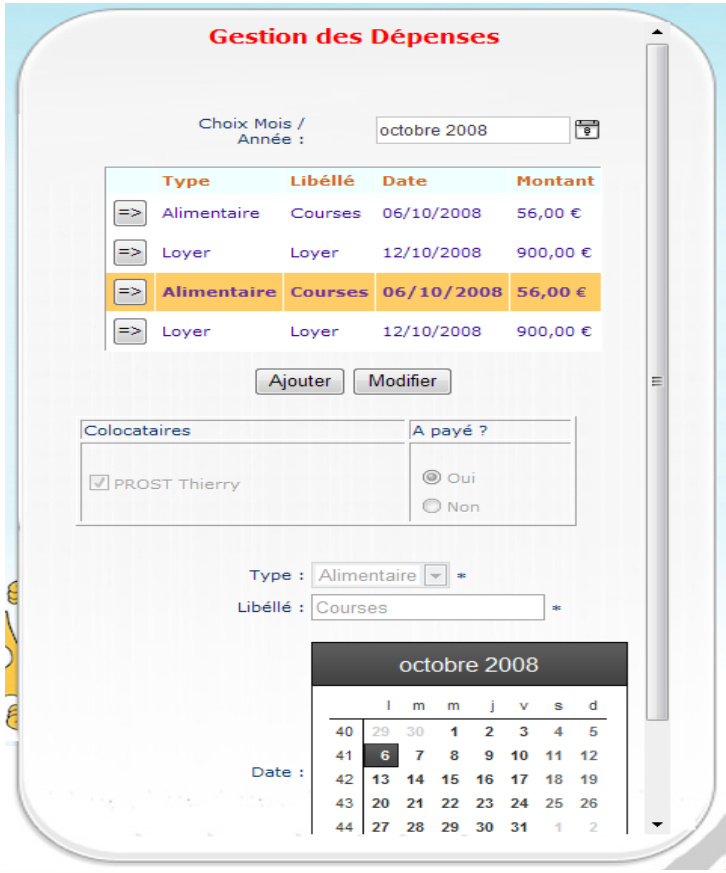
```
public class Colocation
{
   private string sNom;      //Membres
   …
   { get … set }
   …
   public Colocataire(string userName)      // Constructeur
   { }
   public Boolean VerifyColocation(string Colocation, string Pwd…
   {
      Persistance Bdd = new Persistance();

      Bdd.VeryfyColocation(Colocation, Password, activeUser);

      if (Bdd.Colocation == Colocation && Bdd.Mdp == Password)
      {
         return true;
      }
      return false;
   }
```

- Deuxième système d'authentification
- Activation des menus une fois la colocation créée

Présentation des classes : La classe Dépenses

Centralisation des dépenses : pour l'ajout, la modification et la suppression

```
public class Dépenses
{
   private string …
   …
   { get … set }
   …
   public int calculerDépenses()
   {
      foreach (Colocataire cl in Controleur.Colocation)
      {
         decimal TotalDu = 0;
         decimal TotalRegle = 0;
         foreach (Payeur py in pys)
         {
            Nom = py.Nom;
            Prenom = py.Prenom;
            TotalDu = TotalDu + py.MontantDu;
            TotalRegle = TotalRegle + py.MontantRegle;
            …
```



Les autres classes : Contrôleur, Propriétaire, Type & Payeurs

Types de dépense ou Propriétaire assigné à une colocation: Ajout, Modification …

```
public class Ctrl
{
   private string sNom;      //Membres
   …
   { get … set }
   public Ctrl(IQuote input)
                {
     try
     {
        if (input != null)
        {
           Colocataire Coloc = new Colocataire( input.userActive );
           sColocation = Coloc.Bdd.Colocation;
           sStatus = Coloc.Bdd.Status;
        }


     }
     catch (Exception e)
     …
```

La classe contrôleur connait les classes métiers.

C'est elle qui vient faire les appels de méthodes entre le formulaire et les classes métiers.

Assigner un propriétaire à la colocation.

Assigner des types aux dépenses.

Classe Payeur :

Une Classe d'Association qui s'interface entre les classes Dépenses et Colocataires.

Les classes collection et interface :

Types de dépense ou Propriétaire assigné à une colocation: Ajout, Modification …

```
public interface IQuote                          //Interface
{
   string userActive
   {
      get;
   }
}


public class Dépenses : System.Collections.CollectionBase
{
                  public Dépenses()          // Collection
                  {

                  }

   public int Add(Dépense value)
   {
      return (List.Add(value));
   }
...
```

Utilité de l'interface IQuote :

Cette interface se charge de transmettre des valeurs de variables entre les formulaire .aspx.cs et le répertoire App_Code afin de les utiliser dans des constructeurs de classe.

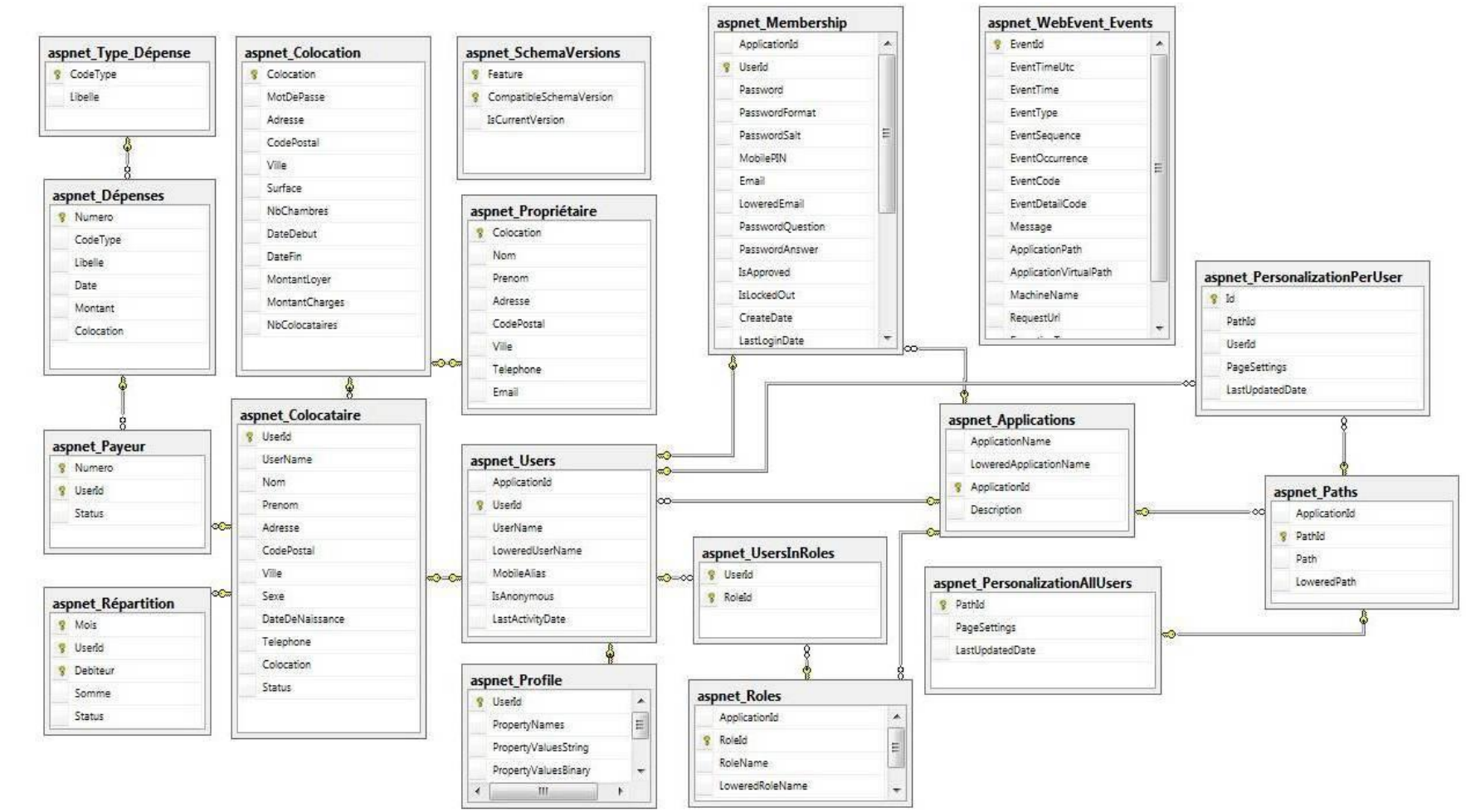
Les classes collections utilisées :

Elles peuvent créer une collection de dépenses ou bien de colocataires selon les besoin applicatifs.

L'itération, le classement, l'ajout ou la suppression sont gérés.

Schéma de la base de donnée :

Les tables de l'application sont reliées au Membership Provider ASP.Net



Les différentes procédures stockées :

Création de colocation, modification du mot de passe de colocation par cryptage, etc.…

Exemple de décryptage Sql avec ADO :

```
public void VeryfyColocation(string Colocation, string Password,
string activeUser)
   {
      string sSql = "SELECT Colocation FROM aspnet_Colocation
WHERE Colocation = '" + Colocation + "'";
      SqlCommand cd = new SqlCommand(sSql, cn);
      cn.Open();
      SqlDataReader dr = cd.ExecuteReader();

      for (int i = 0; dr.Read(); i++)
      {
         sColocation = dr["Colocation"].ToString().Trim();
      }

      dr.Close();

      cd.CommandType = CommandType.StoredProcedure;
      cd.Connection = cn;
      cd.CommandText = "dbo.aspnet_GetId";
      SqlParameter param1 = new SqlParameter("@Colocation", …
```

Utilisation de la classe Persistance pour tous les échanges de données

```
dbo.aspnet_GetId

      @Colocation varchar(50)

AS
      BEGIN

            SET NOCOUNT ON

                  DECLARE @PassPhrase nvarchar(50)
                  SET @PassPhrase = N'cow'

                  SELECT CONVERT(nvarchar, DecryptByPassphrase(@PassPhrase,
MotDePasse, NULL)) AS 'DecryptedPassword' FROM dbo.aspnet_colocation WHERE
Colocation = @Colocation

            RETURN

END
```

Compte-rendu de stage, Entreprise « Script'Games » :

Entreprise éditrice de jeux de type « Serious Games » basée à Cap Omega à Montpellier.

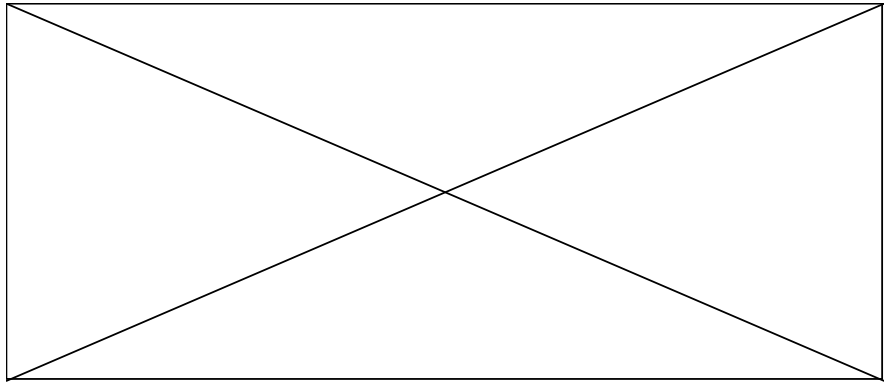
Les « Serious Games » ont pour vocation la simulation informatique utilisant les techniques de jeux vidéo.

On les retrouve pour des programmes d'entrainement en situation réelle mais aussi dans des logiciels d'éducation ou bien publicitaires.

Activité de l'entreprise :

Reposant sur l'évolution des NTIC (Nouvelles Technologies de l'Information et de la Communication), la société « Script'Games » réalise des logiciels de simulation dédiés à l'infanterie française et aux forces européennes de type « FPS ».

A ce jour, deux produits (Instinct et IPCA) ont été développé et un troisième projet est en cours (Bellum) sur lequel j'ai pu apporter ma contribution en terme de développement.



« Script'Games » est également en charge de la création d'exercices spécialisés, de réalisations techniques d'environnements tridimensionnelles adaptés à l'entraînement des forces et de modélisations plus personnalisées.

Techniques utilisées et descriptif du travail effectué

Au sujet de l'interface de développement :

- Moteur de jeu 3D Shiva*
- Langage C et Lua utilisés
- IDE adapté au développement et à l'animation des modèles 3D

Le travail à été pendant ces deux mois effectué en collaboration avec le chef de projet, un modéliseur 3D et un animateur.

Le travail réalisé porte principalement sur le rendu d'un terrain, l'animation de personnages et véhicules ainsi que sur des fonctions pilotant ces éléments.

Conclusion

Pour le logiciel Cow-Lock :

Il sera continué et supportera :

-La répartition au cas ou un colocataire s'en va
-La suppression d'une colocation
-L'assignation de gestionnaires

Mais aussi :

-Un forum
-Un système de petites annonces
-Et un chat pour les mettre en relation
-Une gestion utilisateur plus évoluée
-Menu d'aide et documents légaux
-Internationalisation

C'est FINI !

Merci de votre attention !