2017-2018
Spécialité SAGI

Ken'go Software

Rapport de projet EI4

Aloïs Bretaudeau
Mickaël Meneux
Julien Raillard

Sous la direction de M. Cottenceau Bertrand

IstiA
ÉCOLE D'INGÉNIEURS

université angers

REMERCIEMENTS

Nous tenons à remercier toutes les personnes qui ont contribué à la réalisation de ce projet.

Tout d'abord, nous remercions monsieur Marc Saunot, le président du Ken'go Judo Ju-jitsu, de nous avoir soutenus dans notre démarche de numérisation de son processus de gestion de ses licenciés.

Nous tenons ensuite à remercier notre responsable de promotion monsieur Mehdi Lhommeau d'avoir accepté notre proposition de projet.

Nous remercions également notre tuteur de projet, monsieur Bertrand Cottenceau de ses conseils avisés autour de la conception et de la réalisation de ce logiciel.

Pour finir nous tenions à remercier toutes les personnes qui nous ont aidé de près ou de loin dans toutes les étapes de notre projet.

ENGAGEMENT

DE NON PLAGIAT

Nous, soussignés Bretaudeau Aloïs, Meneux Mickaël, Raillard Julien
déclarons être pleinement conscients que le plagiat de documents ou d'une
partie d'un document publiée sur toutes formes de support, y compris l'internet,
constitue une violation des droits d'auteur ainsi qu'une fraude caractérisée.
En conséquence, nous nous engageons à citer toutes les sources que nous avons utilisées
pour écrire ce rapport.

signé par les étudiants le …. / …. / ……..

Table des matières

Introduction

Nous sommes Aloïs Bretaudeau, Mickaël Meneux et Julien Raillard, trois étudiants en 4ème année à l'ISTIA dans la spécialité systèmes automatisés et génie informatique. Notre programme inclue la réalisation d'un projet tutoré de 80h en accord avec notre formation. Ce rapport en fait l'explication.

Julien Raillard fait depuis 11 ans parti d'un club de judo. Ken'go software est un logiciel de gestion à l'attention de ce dernier, le Ken'go Judo Ju-jitsu. Depuis sa création le club a toujours géré l'inscription de ses licenciés via des formulaires papiers. Ces informations étaient ensuite retranscrites sur un fichier Excel, un par saison. Les membres du bureau pouvaient alors faire leurs statistiques sur ces fichiers. Le problème étant que ces fichiers sont complexes à analyser et que les dossiers papiers prennent de plus en plus de place.

Au cours de l'année 2017 nous sommes entrés en contact avec Marc Saunot, président du club, afin de lui proposer une solution numérique à son processus d'inscription et de gestion de ses licenciés. D'abord dubitatif à l'idée de ce changement nous avons su lui montrer les avantages qu'apporterait un logiciel de gestion : facilité de recherche, d'inscription, de stockage des données ou encore de retranscription de recherches sous différents formats. Ken'go Software permettrait également de faire d'autres actions jusqu'ici considérées comme « annexes » telles que la création de listes de contacts personnalisées, l'envoi de mail et plus tard la gestion d'inscription à une compétition et de classement par groupe de niveau pour les trophées de fin d'année.

Une première version développée en PHP a été réalisée par nos soins en fin d'année scolaire 2016-2017. Il s'agissait d'une simple page web pouvant récupérer un fichier Excel préconçu avec les informations de tous les licenciés d'une année et les enregistrer dans une base de données. L'action réciproque était également possible. Ken'go Software est donc la deuxième version de notre solution numérique, une application Windows Form développée en C#.

Ce rapport présente notre phase de conception du projet depuis septembre dernier, la réalisation du projet au travers des différentes couches implémentées dans notre logiciel et enfin un rapport des éléments restant à ajouter au logiciel ainsi que des axes d'améliorations.

1. Conception du projet

Ce projet n'étant pas proposé par l'école, il nous a fallu concevoir l'ensemble du logiciel, puis le présenter au responsable projet. Dans cette conception, nous avons inclus une fiche de spécifications, les différentes technologies utilisées ainsi que la planification et le partage des tâches.

1.1. Spécificités & Technos

Pour réaliser la fiche de spécification nécessaire à la conception du projet, nous avons été en contact constant avec Marc Saunot, le dirigeant du club.

Dans un premier temps, il nous a envoyer une liste détaillée des points importants du logiciels. Voici cette liste :

Spécifications

EX_01
L'application doit permettre l'import massif de données depuis un fichier Excel ou csv.

EX_02
L'application doit permettre l'export de la base de données sous la forme d'un fichier Excel ou csv

EX_03
L'application doit permettre d'ajouter unitairement un nouveau licencié par l'intermédiaire d'une IHM.

- Offrir une interface pour saisir tous les champs associés à un licencié
- Vérifier que les valeurs saisies sont correctes [Optionnel]

EX_04
L'application doit permettre de modifier tous les champs d'un licencié par l'intermédiaire d'une IHM

EX_05
Un licencié ne peut avoir qu'un nom, prénom, date de naissance, adresse postale, adresse mail.

EX_06
Un licencié peut avoir plusieurs numéros de téléphone, être inscrit à plusieurs

EX_07
Seuls les licenciés âgés de plus 14 peuvent être inscrit au Jujitsu ou Ne Waza [Optionnel]

EX_08
Pour les licenciés de moins 13 ans, l'application doit demander le jour des cours.

EX_09
L'application permettre de conserver l'historique des inscriptions année par année

EX_10
L'application doit un menu de réévaluation des catégories d'âge.
L'application n'effectue aucune vérification de date, cette vérification est de la responsabilité de l'opérateur.
L'application doit demander confirmation avant d'exécuter cette action.

EX_11
L'architecture de la base de données doit être évolutive.
Elle doit permettre de gérer tous les cas classiques de l'association

EX_12
Pour accéder à l'application, l'opérateur doit saisir un login et un mot de passe

EX_13
L'architecture de l'application permet une connexion à distance entre la base de données et l'IHM [Optionnel]

EX_14
L'application gère 2 types de compte: Administrateur et Utilisateur [Optionnel]

EX_15
Il n'existe qu'un seul compte Administrateur [Optionnel]

EX_16
L'opérateur peut ajouter un nouveau compte Utilisateur [Optionnel]

EX_17
L'application doit permettre de modifier le mot de passe associé à un login [Optionnel]

EX_18
Les informations dans la base de données sont cryptés [Optionnel]

Figure 1 : Liste des spécifications

Une fois cette liste étudiée, nous avons commencé la rédaction de la fiche de spécifications. Pour cela nous avons effectué des échanges avec le directeur du club jusqu'à avoir une fiche de spécification qui lui convenait (disponible en annexe).

En parallèle de la rédaction de cette fiche, nous avons réfléchie à un schéma de base de données. Pour commencer nous avons posé dans des tables toutes les informations que l'application nécessitaient puis nous avons créé les différents liens entres elles. Pour finir nous avons contraint les champs qui devait être non nul, le type de variable à stocker et les différentes cardinalités. Le script SQL de création de la base est disponible en annexe.

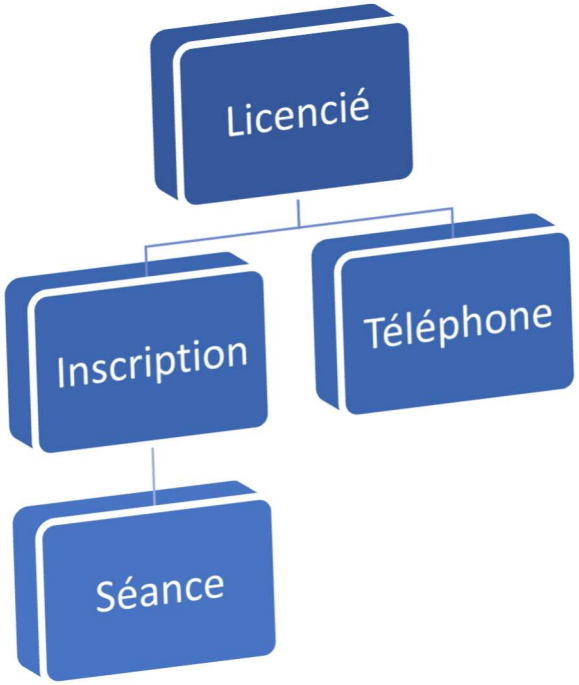


Figure 2 : Schéma de la base de données

Suite à cela, nous avons commencé la création de l'application en elle-même. Nous nous sommes organisés par onglets. Pour chaque onglet de l'application nous avons réalisé un schéma. Il contenait tout l'interface utilisateur du logiciel. De cette conception graphique, nous avons déterminé les fonctions que nous voulions utiliser avec leurs prototypes et leurs entrées-sorties. Nous avons cependant fait évoluer ces prototypes lors du développement par manque d'expérience dans ce domaine.

Lors de cette phase de réflexion, nous avons aussi abordé le côté technique. Tous les membres du groupe étant intéressé par le .Net et C# et ces technologies étant plus adapté au développement de logiciel, nous avons décidé d'utiliser celles-ci. Pour la réalisation de l'interface, ayant déjà utiliser les classes Windows Form nous avons décidé de les reprendre pour notre application.

Pour ce qu'il en est de la base de données nous avons sélectionné Microsoft Azure pour plusieurs raisons. La première est la possibilité de se connecter a plusieurs en même temps et à partir d'endroits différents. Cela à grandement simplifier la phase de développement à plusieurs car nous avons pu travailler sans peupler de base de données sur nos ordinateurs personnels. La deuxième raison est la compatibilité avec C#. En effet les serveurs Azure utilisent Microsoft SQL Server comme système de gestion de base de données.

Pour finir, nous avons décidé d'utiliser Entity Framework en tant qu'objet-relation manager(ORM). Il permet de créer des objets automatiquement en correspondance avec la base de données relationnel que nous avons conçu auparavant.

1.2. Architecture du logiciel

Afin d'apporter une vue claire sur l'architecture du projet dans son modèle actuel, nous avons établi différents diagrammes UML dont un diagramme de classe et cinq diagrammes d'activité. Pour commencer, nous avons choisi pour notre application une architecture trois couches UI – Métier – Dao avec en parallèle un package Entities contenant les structures de données que nous allions utiliser au fil du projet. Selon une vue de type diagramme de classe l'application suit donc ce modèle (une vue détaillée de chaque couche est présente en annexe) :

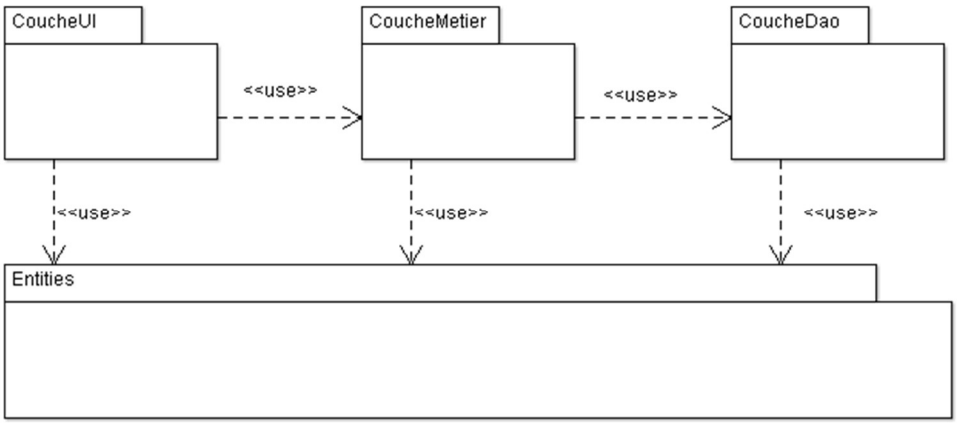


Figure 3 : Diagramme de classe de Ken'go Software

En terme répartition du travail, chaque couche a été confiée à un développeur différent, avec Aloïs en charge de la couche UI, Mickaël en charge de la couche Métier et Julien en charge de la couche Dao. Les entités ont été développées au fur et à mesure du temps. Deux de ces entités dérivent de l'interface IDisposable afin d'en hériter la méthode Dispose(), qui permet des libérer la mémoire qui leur est affecté sans attendre. Une autre entité sur laquelle on peut s'arrêter un instant est la classe DGVPrinter,

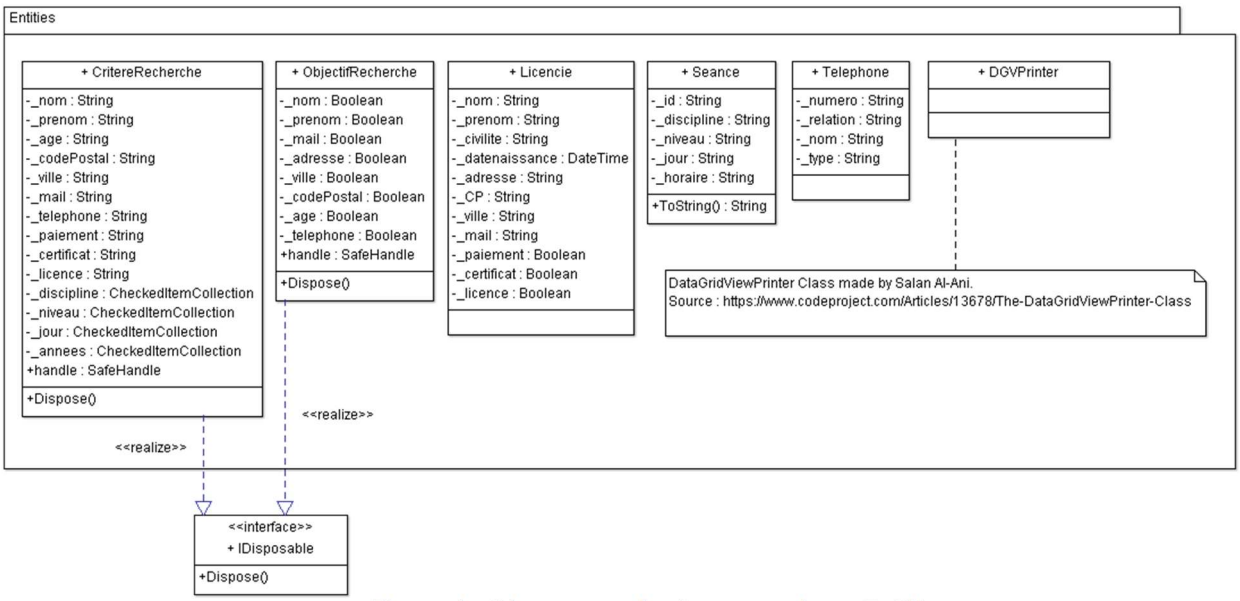


Figure 4 : Diagramme de classe : package Entities

développée indépendamment par Salan Al-Ani et permettant d'utiliser de nombreuses fonctions de sortie de type impression pour les DataGridView que nous exploitions.

Dans la couche UI, on peut remarquer une interface IUi qui ne contient qu'une méthode, SetMétier(). Cette méthode a une utilité très importante pour l'application, elle permet de conserver la même instance de la couche Métier (et par conséquent de la couche Dao car celle-ci est instanciée par la couche Métier) à travers les trois types de formulaire de la couche UI. Sans cela, chaque formulaire instancierait sa propre couche Métier puis Dao. Cette manière de fonctionnement était inenvisageable puisque réduire la consommation de la mémoire est un point critique en C#. Lors de la création des formulaires 'secondaires' on appellera donc cette méthode en passant en paramètre l'instance actuelle de la couche métier.

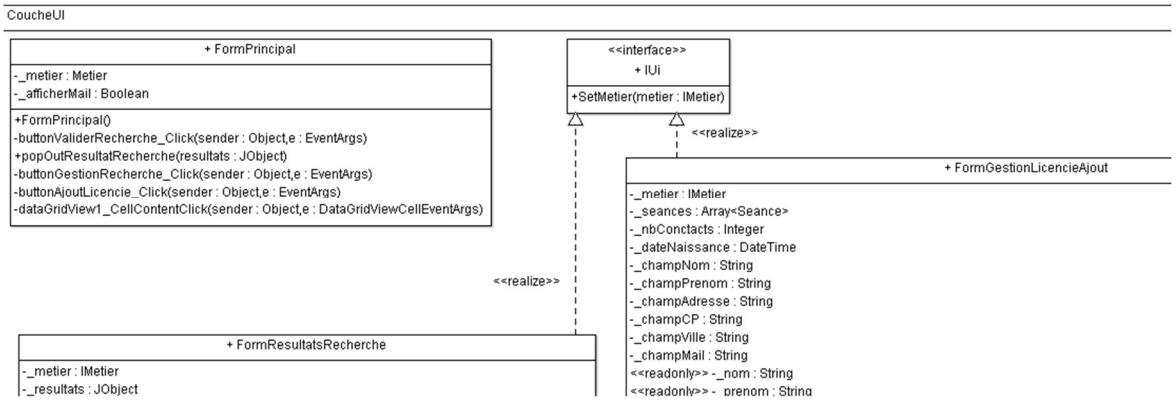


Figure 5 : Diagramme de classe : couche UI (interface IUi)

Nous avons inclus en annexe cinq diagrammes d'activité qui présentent le flux d'activité des 5 grands cas d'utilisation de notre application dans son état actuel :

- Rechercher des licenciés
- Ajouter des licenciés
- Supprimer des licenciés
- Éditer les informations des licenciés
- Réinscrire les licenciés

Ces deux derniers cas d'utilisation sont rassemblés en un seul diagramme d'activité car dans l'essence leurs fonctionnements sont identiques. Il existe cependant quelques différences dans leurs utilisations :

La fonction d'édition et la fonction de réinscription ont beau sembler identiques, elles ont été développées pour deux buts différents : on utilise l'édition pour remplacer les inscriptions ou les informations de l'année d'enseignement courante et la réinscription pour ajouter à un licencié une ou plusieurs inscriptions concernant l'année en cours ou l'année suivante, car Ken'go propose à ses licenciés de se réinscrire dès les trois derniers mois d'enseignement de l'année courante si ils le désirent.

Par conséquent, les fonctions ont une légère différence d'exécution sur la base de données. Les deux fonctions agissent de la même manière sur les tables Licencié et Téléphone, respectivement en remplaçant les données de la première et en supprimant les anciennes données de la seconde avant d'y rajouter les nouvelles données. En revanche, elles n'interagissent pas de la même manière avec la table Inscription. L'édition d'un licencié supprime toutes les inscriptions pour l'année courante existantes précédemment avant d'ajouter à la table les inscriptions sélectionnées dans le formulaire ayant lancé l'exécution de la fonction d'édition. En revanche, la fonction de réinscription se contente d'ajouter à la table les inscriptions sélectionnées, en utilisant comme 'clé' l'année d'inscription choisie dans le formulaire.

Autre fait à noter concernant les diagrammes d'activité, il ne faut pas confondre la fonction de recherche, que nous avons créé pour l'onglet Recherche avec la fonction de recherche par nom et prénom, qui est en fait un traitement préliminaire obligatoire pour la fonction d'ajout de licencié et remplit en même temps une grille de résultats sur laquelle on peut ensuite lancer les fonctions d'édition, de réinscription ou de suppression pour un licencié donné.

1.3. Planification

1.3.1. Planning prévisionnel

Lors de notre de phase de conception, il nous a fallu faire la planification de notre projet (voir Gantt ci-dessous). Ce dernier représente toutes les étapes permettant l'achèvement total du logiciel Ken'go Software. Le développement de ce projet a montré que les délais étaient fortement sous-estimés (voir diagramme de la répartition du temps de travail ci-dessous). En effet nous n'avons pu réaliser que deux des sept phases, ces deux dernières servant de canevas pour les cinq autres.

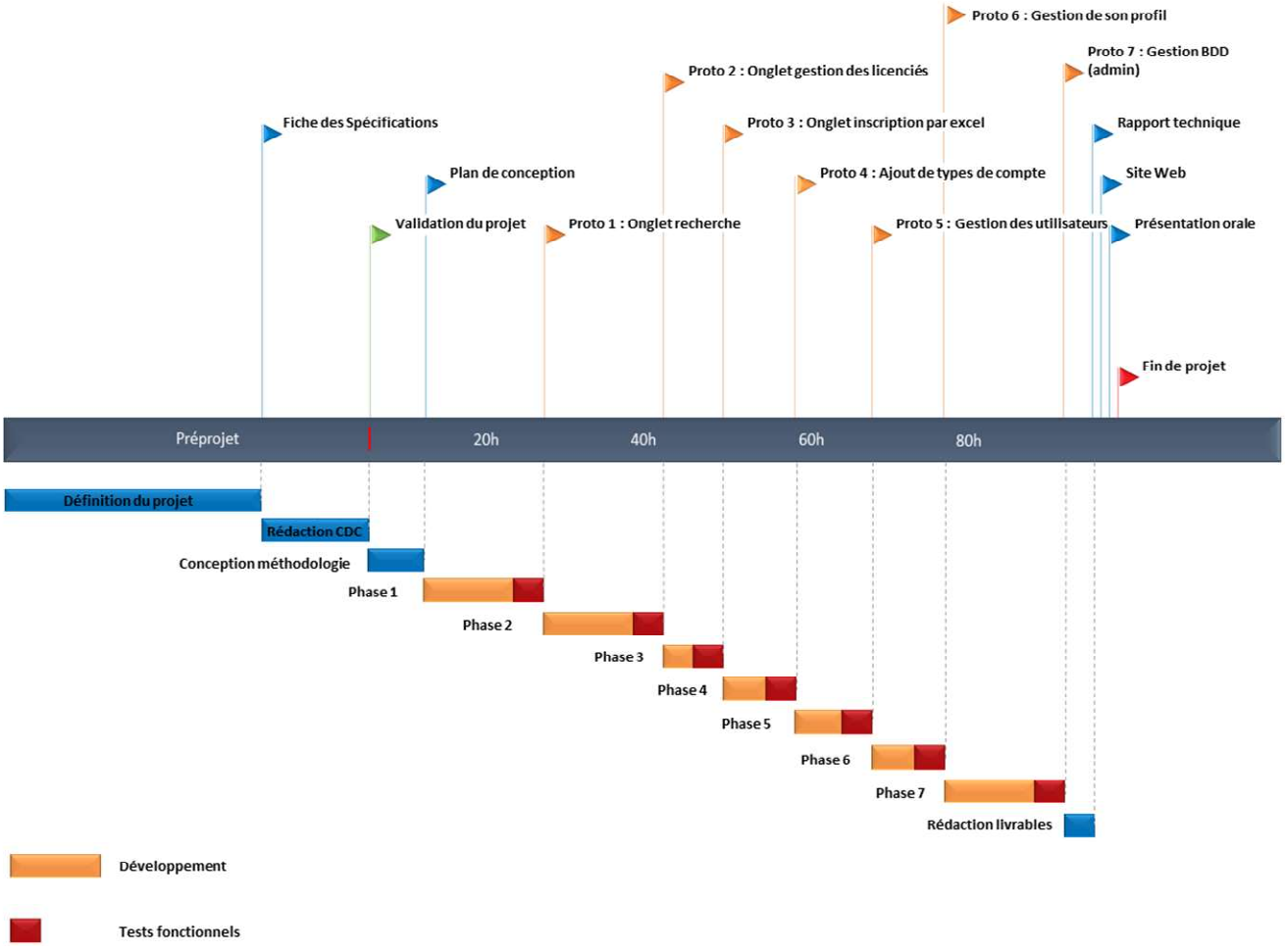


Figure 6 - Gantt Planning Prévisionnel

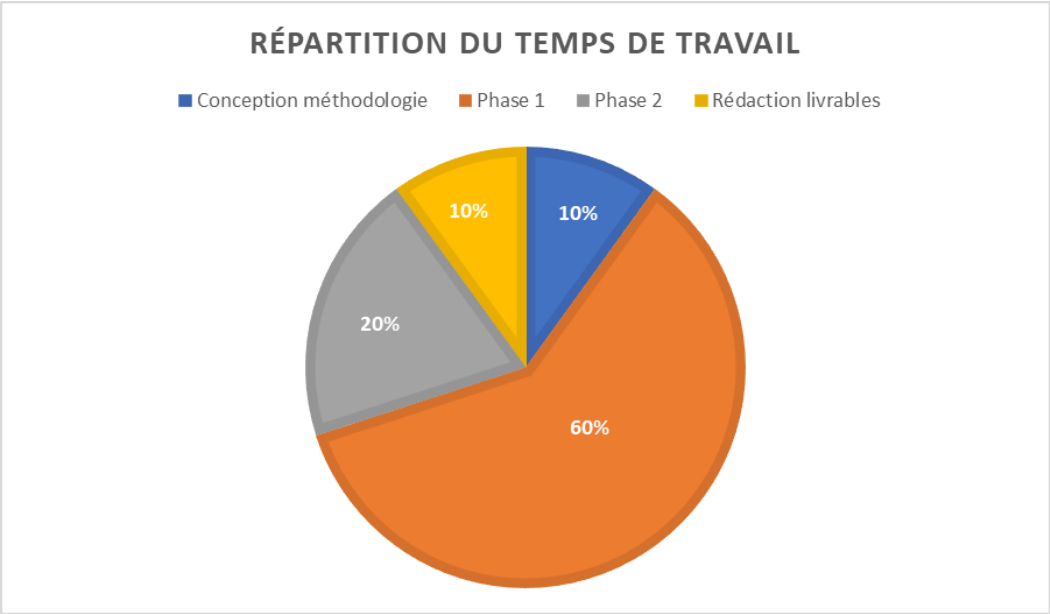


Figure 7- Diagramme de répartition du temps de travail

1.3.2. Méthodologie

Travaillant sur une architecture en trois couches (ui, métier et dao), chaque membre s'est vu affecté sa propre couche afin de permettre un travail simultané.

Dans ce sens, le logiciel de gestion de version Git nous a permis de partager un même projet Windows Form. Notre Git était géré de la façon suivante : tout le monde pouvait commit sur la branche unique, master, à n'importe quel instant du moment que l'ajout était fonctionnel (sous-entendu déjà soumis à des tests unitaires et fonctionnels). Ce management nous a parfois posé problème, notamment en période de rush où les tests pouvaient avoir tendance à être bâclés. Après plusieurs recherches vers la fin du projet nous sommes tombés sur une meilleure méthode de management de dépôt Git. Le principe étant d'avoir trois branches master, release et dev. Chaque personne commit ses changements sur une branche créée à partir de dev et commit cette branche sur dev une fois testée. Une fois cela fait, à diverses étapes du projet une autre batterie de tests (souvent automatisés) est effectuée sur la branche dev afin d'aboutir à une version commitable sur la branche release. Sur cette branche sont enfin effectués les tests de performances et d'optimisations (CPU, mémoire, temps, etc.) afin de ramener le résultat sur la branche master.

En termes d'organisation de notre temps, ayant pu travailler par journées entières nous avons convenu du déroulé suivant pour chacune d'entre elles : briefing rapide sur ce que chacun doit faire dans les prochaines heures, des rapports informels à l'oral à chaque pause (toutes les 1h20-2h40) et enfin un débriefing en fin de journée pour faire le point sur l'avancée de chacun et le plan pour la prochaine séance de travail. Une « to do list » était mise à jour à chaque étape majeure (2-3semaines) afin de situer l'avancée de chacun par rapport à la globalité du projet et se rendre compte de la nécessité ou non d'un rush pour tenir les délais.

2. Réalisation du projet

2.1. UI

Dès la création de la fiche de spécifications, nous avons choisi de développer une application de type Windows Form en langage C#/.NET. Le développement de l'IHM est donc passé par quelques semaines de conception de visuels afin de créer différents formulaires et d'avoir un agencement basique des informations. Dans la partie suivante nous allons donc aborder différents points importants de la création de l'interface utilisateur.

2.1.1. Les formulaires

Les formulaires sont les structures de base constituant l'IHM de notre application. Dans son état actuel, celle-ci en comprend trois types : un formulaire principal permettant l'accès aux fonctionnalités principales (Recherche et gestion des licenciés), un formulaire de résultat pour la fonction de recherche principale donnant accès au traitement de ces résultats et un formulaire de gestion des licenciés servant à ajouter ou modifier les informations enregistrées dans la base de données.

a) Le FormPrincipal

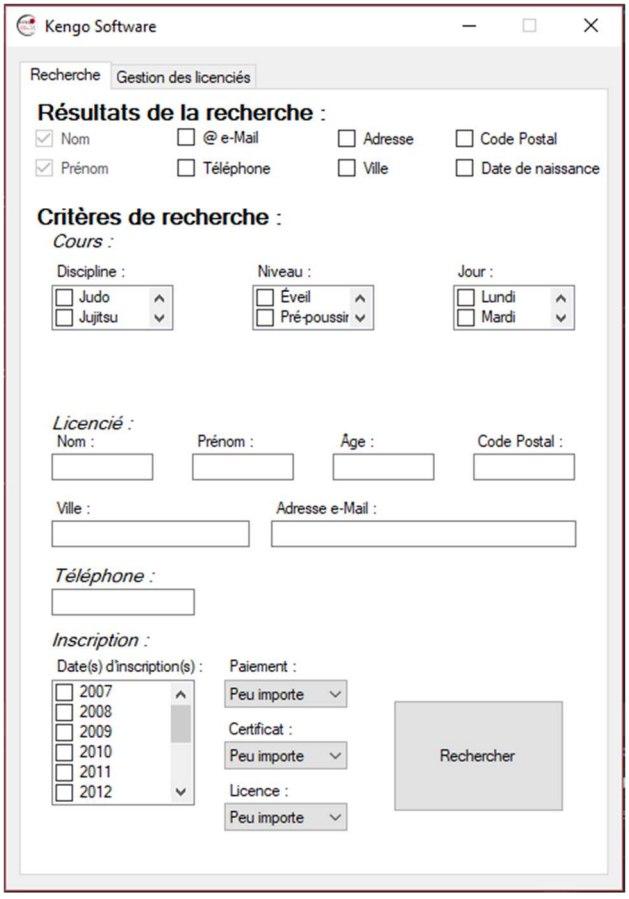


Figure 8 : Formulaire principal (onglet recherche)

La page d'accueil actuelle du logiciel est celle à partir de laquelle on peut lancer une recherche de licenciés. En plus de cet onglet, le formulaire contient un onglet destiné à la gestion des licenciés déjà présents dans la base de données.

L'onglet de recherche est divisé en deux parties : la sélection des champs que l'utilisateur cherche à obtenir de la base de données et le choix des critères auxquels les résultats devront être conformes.

Chacune de ces parties seront analysées puis donneront place à la création d'une entité (de classes respectives ObjectifRecherche et CritereRecherche) lors d'une demande de recherche. Ces entités seront ensuite envoyées à la couche Métier pour être retransformées en JObjects plus propices au fonctionnement de la couche Dao.

Si le mode de fonctionnement des cases à cocher pour les objectifs de la recherche sont assez explicites (Si la case est cochée, le champ sera

recherché et affiché, sinon on ne monopolisera pas de mémoire pour cela), le principe d'utilisation des champs de critères diverge suivant le type des champs (CheckBoxList,TextBox ou ComboBox).

Deux choses sont cependant à noter concernant les Objectifs d'une recherche : Le nom et le prénom seront recherchés dans tous les cas (pour des raisons évidentes de clarté des résultats) et c'est également le cas pour l'adresse e-mail. Dans ce dernier cas cela dit, l'utilisateur a la possibilité de l'afficher ou non à sa convenance, mais il sera tout de même gardé en mémoire pour permettre l'utilisation de deux des fonctionnalités du formulaire de résultat (décrites ci-dessous).

Pour ce qui est des champs de critère, le principe est comme suit :

- Un critère laissé dans son état par défaut (vide) ne sera pas considéré lors de la création de la requête. Cela implique que si tous les champs sont laissés à l'état par défaut, la recherche retournera les informations de tout les licenciés contenus dans la base.
- La sélection de plusieurs items dans une CheckBoxList (discipline, niveau, jour et date d'inscription) retournera les informations de tous les licenciés correspondant à au moins un des critères choisis pour chaque liste ainsi utilisée.
- Les TextBox comme le nom ou le prénom retourneront tous les licenciés dont le champ correspondant dans la base de données commence de la même manière que le contenu de la TextBox. On peut donc rechercher tous les utilisateurs dont le nom commence par 'Ra' par exemple.

Les ComboBox (paiement, licence et certificat) proposent trois options : 'attendu', 'réglé' et 'peu importe'. Le choix de l'option 'peu importe' retire le critère correspondant de la recherche.

Le second onglet du formulaire principal permet la gestion des licenciés. Cela inclut des fonctionnalités telles que la suppression ou l'ajout d'un licencié à la base de données ou encore l'édition de ses inscriptions courantes ou sa réinscription pour l'année suivante ou en cours.

Afin d'accéder à ces fonctionnalités, il est nécessaire de vérifier l'existence du licencié désiré dans la BDD auparavant. Dans le cas de faire un ajout à la base, nous avons fait ce choix de bloquer ainsi l'utilisateur afin d'éviter la création de doublons.

Ainsi il suffit de chercher une personne dans la base et de cliquer sur le bouton correspondant (en bas de page ou dans la

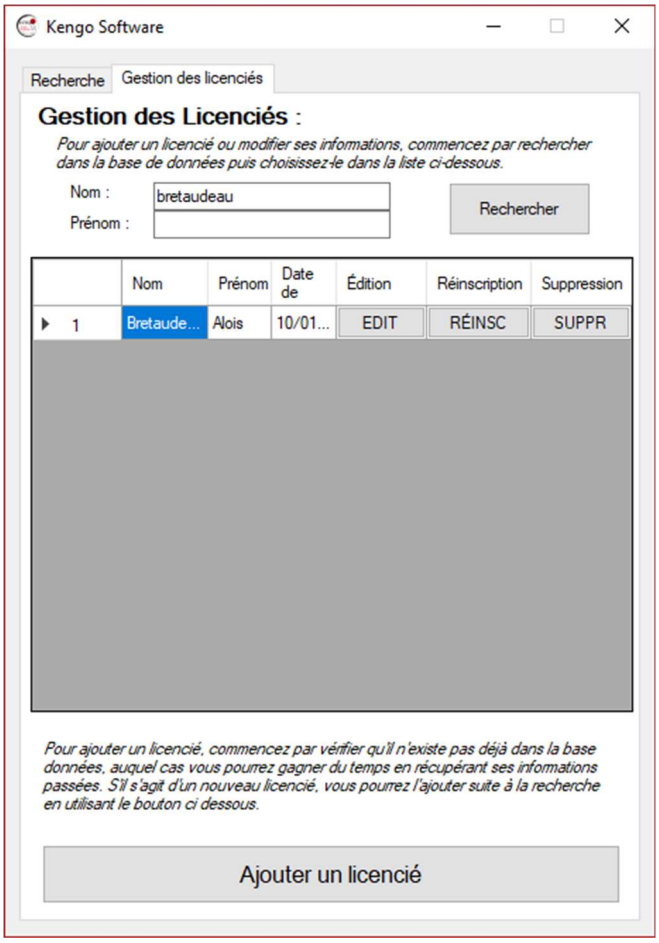


Figure 9 : Formulaire principal (onglet Gestion)

DataGridView) pour ouvrir le formulaire correspondant (un FormGestionLicencieAjout dans le cas de l'ajout, l'édition ou la réinscription ou une boite de dialogue de confirmation dans le cas d'une suppression).

b) Le FormResultatsRecherche

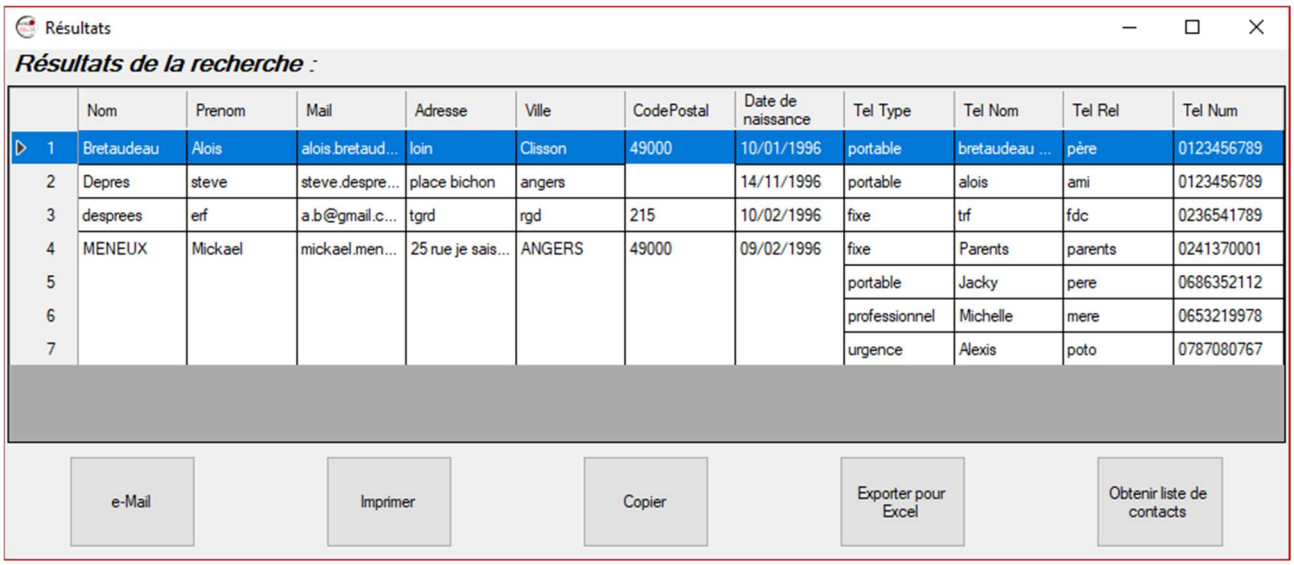


Figure 10 : Formulaire de résultats de la recherche

Ce formulaire fut créé en parallèle du formulaire principal. Son but est d'afficher pour l'utilisateur les résultats correspondants à sa recherche et lui permettre de les utiliser de cinq manières différentes. En raison de la quantité d'informations souvent importante devant être affichée, ce formulaire a été conçu via un TableLayoutPanel, un composant permettant de gérer la taille et la position des contrôles qu'il englobe lors d'un redimensionnement du formulaire. Cela permet un affichage fluide aussi bien en mode fenêtré que plein écran.

Sur cette table des licenciés correspondants à la recherche, il est possible de sélectionner une ou plusieurs entrée(s) afin d'exécuter un traitement géré par la couche Métier. On peut ainsi envoyer des e-mails, imprimer, copier ou exporter au format Excel la sélection ou encore créer une liste de contact Outlook pour envoyer ensuite des mails groupés.

c) Le FormGestionLicencieAjout

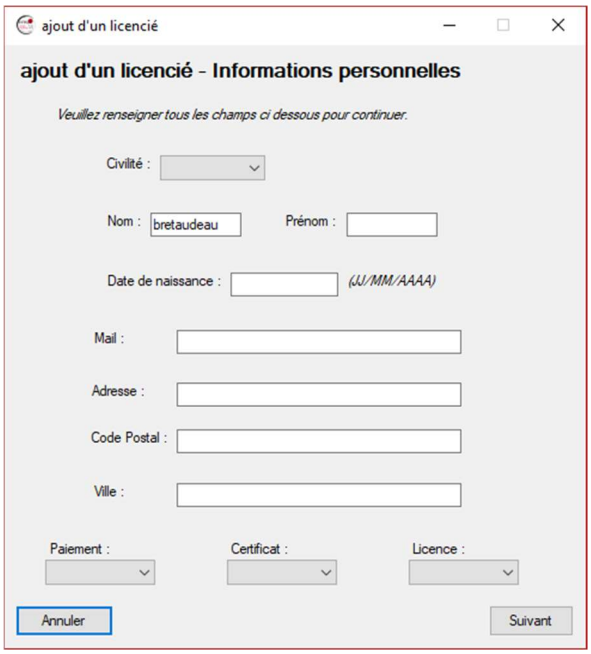


Figure 11 : Formulaire de gestion, première page

Ce formulaire est à usages multiples. Il possède notamment un constructeur servant en cas d'ajout à la base de données, auquel cas le formulaire est vide à la création, et un autre servant lors d'une édition ou réinscription, auquel cas le formulaire pourra être prérempli avec les informations actuelles du licencié.

Une des caractéristiques de ce formulaire est son agencement divisé en trois pages. Cette division, créée en rassemblant chaque 'page' de contrôles dans un panel, nous sert de deux manières différentes et les boutons 'précédent' et 'suivant' de chaque page font que rendre visible ou non chaque panel de contrôle.

Premièrement, afin de déterminer la catégorie d'âge à laquelle le licencié appartient (via l'analyse de sa date de naissance) et donc les séances auxquelles le licencié est autorisé à s'inscrire. Nous avons besoin de déterminer ces séances avant de donner à l'utilisateur l'accès à cette liste de séance, et donc utilisons l'évènement présent sur le bouton suivant de la page 1 afin de déterminer ces séances pour remplir la liste d'inscription de la page 2.

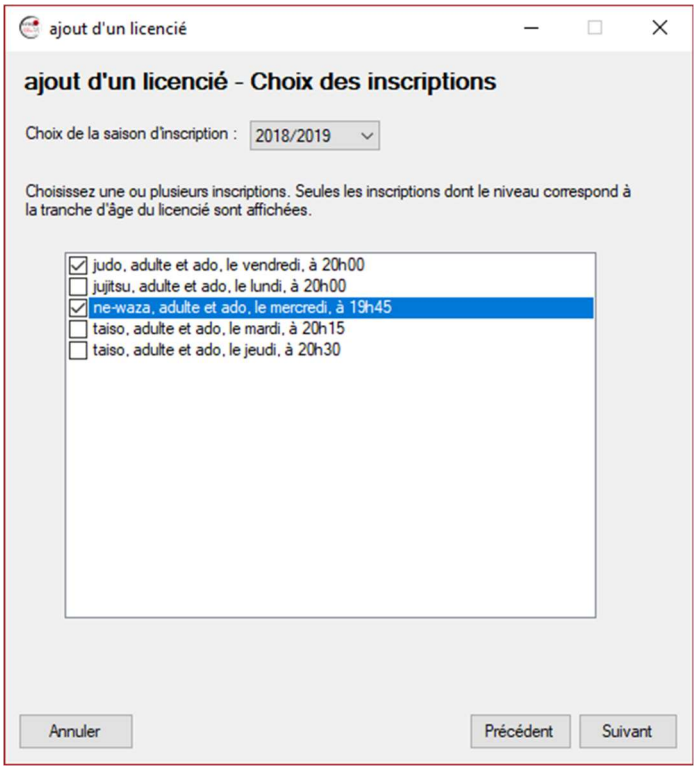


Figure 12 : Formulaire de gestion, deuxième page

Également, diviser ce formulaire en trois est un choix UX (User Experience) que nous avons fait afin de ne pas surcharger l'utilisateur avec trop de demandes de renseignements d'un coup, étant donné que chaque champ est obligatoire afin de réaliser la communication avec la BDD au vu du script SQL de cette dernière (présent en annexe) et lui permettre de procéder par étape dans la complétion de ce formulaire.

Afin de passer à la troisième page, il est nécessaire de sélectionner au moins une inscription dans la liste, sélection qui sera effacée à chaque changement de la date de naissance sur la page 1 car un

changement de cette date entraine une réévaluation de la catégorie d'âge et donc une actualisation de la liste des inscriptions autorisées.

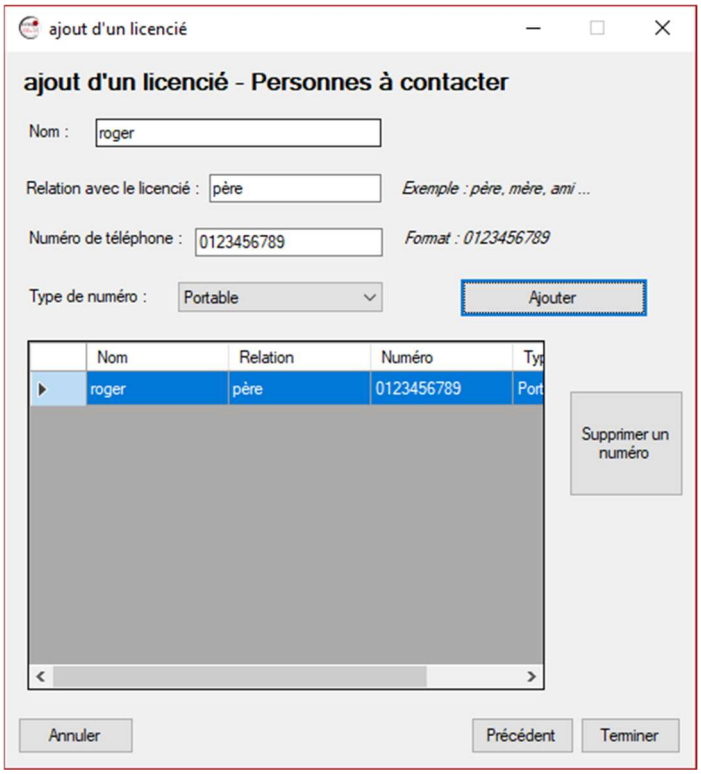


Figure 13 : Formulaire de gestion, troisième page

Sur la troisième page, l'utilisateur peut ajouter ou supprimer des contacts avant de lancer l'opération d'ajout, de réinscription ou d'édition en cliquant sur le bouton terminer.

Si l'opération liée à ce formulaire est menée à bien, alors la DataGridView de l'onglet de gestion dans le formulaire principal sera actualisée avec les nouvelles informations.

Il sera à noter que pour éviter les conflits de gestion et contrairement au type précédent, ce formulaire est appelé en tant que boite de dialogue, ce qui a deux atouts : n'autoriser qu'une seule instance de ce formulaire en empêchant de perdre le focus sur la fenêtre (à l'instar des demandes de confirmation que l'on peut rencontrer dans les logiciels Windows ; et donner la capacité d'indiquer le résultat de l'opération.

2.1.2. La sécurité informatique

Dans l'utilisation d'une base de données, la sécurité informatique est toujours importante pour la protection des données personnelles, et c'est particulièrement le cas à deux moments : lors du passage par la couche Dao et bien entendu lorsqu'un utilisateur a accès à un moyen d'entrer des données a destination de la base. Afin d'éviter notamment les injections SQL nous avons mis en place dans la couche UI une vérification de tous les champs avec lesquels un utilisateur interagit pour l'empêcher de valider des données qui ne correspondraient pas à ce qu'elles sont supposées être. Nous avons pour cela utilisé des expressions régulières.

Le procédé de sécurité est relativement identique pour le formulaire principal et celui de gestion, et fonctionne de manière simple. Le code suivant est encapsulé dans un bloc **try**{}. Tout d'abord il faut déclarer les expressions régulières que nous allons utiliser.

```
Regex _regexNomPrenomVille = new Regex("^[-a-zÀ-ž ']*$", RegexOptions.IgnoreCase);
Regex _regexAgeCP = new Regex("^[0-9]*$");
Regex _regexMail = new Regex("^\\w+([-+.']\\w+)*@\\w+([-.]\\w+)*\\.\\w+([-.]\\w+)*$", RegexOptions.IgnoreCase);
Regex _regexTel = new Regex("^0[1-9][0-9]{8}$");
```

Figure 14 : Expressions régulières du formulaire principal

Ici, nous en avons choisi quatre :

- Un nom, un prénom ou une ville est composé de lettres qui peuvent être accentuées (on enlèvera ces accents dans la couche métier de toute manière) et peut contenir un tiret, une apostrophe ou un espace.
- L'âge et le code postal sont une suite de chiffre.
- L'adresse e-mail suit un pattern complexe créé spécialement à cet effet.
- Un numéro de téléphone est une suite de dix chiffres commençant par un 0 suivi par un chiffre non nul.

Ensuite on récupère les informations des champs entrés par l'utilisateur, et on supprime les espaces inutiles possiblement laissés par inattention.

```
string _champNom = textBoxNom.Text.Trim();
string _champPrenom = textBoxPrenom.Text.Replace(" ", string.Empty);
string _champAge = textBoxAge.Text.Replace(" ", string.Empty);
string _champVille = textBoxVille.Text.Trim();
string _champCP = textBoxCP.Text.Replace(" ", string.Empty);
string _champMail = textBoxMail.Text.Replace(" ", string.Empty);
string _champTel = textBoxTelephone.Text.Replace(" ", string.Empty);
```

Figure 15 : Récupération des champs

Enfin, on vérifie que l'expression régulière correspondante à chaque champ examiné reconnait bien le contenu du champ. Parfois, des conditions supplémentaires viennent s'ajouter à la vérification, comme par exemple pour l'âge (il est interdit de s'inscrire pour les enfants de moins de trois ans).

```
MatchCollection matches;
matches = _regexNomPrenomVille.Matches(_champNom);
if (matches.Count != 1)
    throw new Exception("Le nom n'est pas correct.");
matches = _regexNomPrenomVille.Matches(_champPrenom);
if (matches.Count != 1)
    throw new Exception("Le prénom n'est pas correct.");
matches = _regexAgeCP.Matches(_champAge);
if (matches.Count != 1 || matches.Count == 1 && _champAge != string.Empty && int.Parse(_champAge) < 3)
    throw new Exception("L'âge n'est pas correct.");
```

Figure 16 : Vérification des champs

Dans ce cas précis, il s'agit des expressions régulières utilisées pour vérifier les critères de recherche du formulaire principal, et qui peuvent donc être laissés vides, d'où le fait que lorsqu'aucun match n'est trouvé mais que le champ est vide, on ne renvoie pas une exception.

```
matches = _regexMail.Matches(_champMail);
if (matches.Count != 1 && _champMail != string.Empty)
    throw new Exception("L'adress e-mail n'est pas correcte.");
matches = _regexTel.Matches(_champTel);
if (matches.Count != 1 && _champTel != string.Empty)
    throw new Exception("Le numéro de téléphone n'est pas correct.");
```

Figure 17 : La deuxième condition (string.Empty) n'est pas présente dans le formulaire de gestion

Enfin, l'exception éventuelle est attrapée dans le bloc **catch**{} et est affichée à l'utilisateur par le biais d'un popup. Dès qu'une exception est trouvée, la fonction en cours est stoppée.

Dans le formulaire principal, la sécurité informatique est appelée lors d'un clic sur le bouton 'Rechercher'. En revanche, dans le formulaire de gestion on appelle la sécurité à chaque changement de page vers l'avant (boutons 'suivant' et 'terminer'). Nous avons fait ce choix dans un soucis d'améliorer l'UX en localisant facilement et rapidement les erreurs.

```
catch (Exception ex) { MessageBox.Show(ex.Message); }
```

Figure 18 : Affichage de l'erreur

2.2. Métier

2.2.1. La construction du JObject

Lors de la phase de développement, la moitié du temps a été consacré à la fonction Recherche. Cette fonction est la composante principale de l'onglet recherche. Elle permet à l'utilisateur de rechercher des licenciés en fonction de plusieurs critères comme le nom ou l'âge. Le résultat est affiché dans une autre fenêtre en fonction des champs cochés sur l'onglet principale.

La majeure difficulté de cette fonction est son aspect modulable. En effet en fonction de ce que l'utilisateur veux rechercher, la requête ne vas pas aller chercher dans les mêmes tables de la base de données. Par exemple, une recherche par nom et prénom ne demande que la table « Licencié » de la base alors qu'une recherche par téléphone demande en plus de « Licencié », la table « Téléphone » (cf. figure-2 schéma de la base de données).

Cette difficulté se reflète dans la couche métier par la construction des informations à transmettre à la couche Dao. Lorsque l'utilisateur coche les champs qu'il veut voir afficher dans la partie « Résultats de la recherche », la couche métier reçoit un objet qu'il convertit en « JObject » puis pour chaque pair de clé-valeur, détermine s'il doit être envoyer à la couche Dao pour la partie « Select » de la requête SQL. Le résultat est un « JArray » contenant le nom des champs voulue, un « JArray » étant une suite de valeur, ici des strings, séparer par une virgule. Ce « JArray » sera intégré à un « JObject » plus tard dans la fonction.

```
//parcours recherche et création
foreach (KeyValuePair<String, JToken> j in rechercheObjectif)
{

    if (j.Value.ToString() == TestTrue.ToString())
    {
        tabRecherche.Add(j.Key);
    }

}
```

Figure 19 : Code de la création du JArray

La seconde partie du code de cette fonction est consacré à la création des quatre « JObject » pour la partie WHERE de la requête. Pour faciliter le travail de la partie Dao, ces quatre « JObject » sont liés aux quatre tables de la base de données. A chaque recherche, ces objets sont remplis en fonction des valeurs des champs de l'interface. Si le champ est vide alors le JObject correspondant sera vide.

```
//parcours critere et remplissage
foreach (KeyValuePair<String, JToken> j in critereObjectif)
{

    if (j.Value.ToString() != "" && j.Value.ToString() != "[]")
    {
        switch (j.Key)
        {
            case "Nom":
                tabCriterelcn.Add(j.Key, j.Value);
                break;
            case "Prenom":
                tabCriterelcn.Add(j.Key, j.Value);
                break;
            case "Age":
                tabCriterelcn.Add(j.Key, (DateTime.Now.Year - (int)j.Value).ToString());
                break;
            case "CodePostal":
                tabCriterelcn.Add(j.Key, j.Value);
                break;
```

Figure 20 : Remplissage des JObjects

Ces quatre objets sont finalement rassemblés avec le premier « JArray » pour former un dernier « JObject » qui sera envoyer à la couche DAO pour la requête.

```
ToSendDao = new JObject { new JProperty("Recherche", tabRecherche), new JProperty("Critere", addlicencie, addtelephone, addinscription, addseance) };
```

Figure 21 : Création du JObject final

2.2.2. Les fonctionnalités externes

En plus de cette fonction recherche, l'autre partie du code de la partie métier est consacrée aux différentes fonctionnalités. Ces options sont disponibles sur la page des résultats de l'onglet recherche qui se présente comme ci-joint.

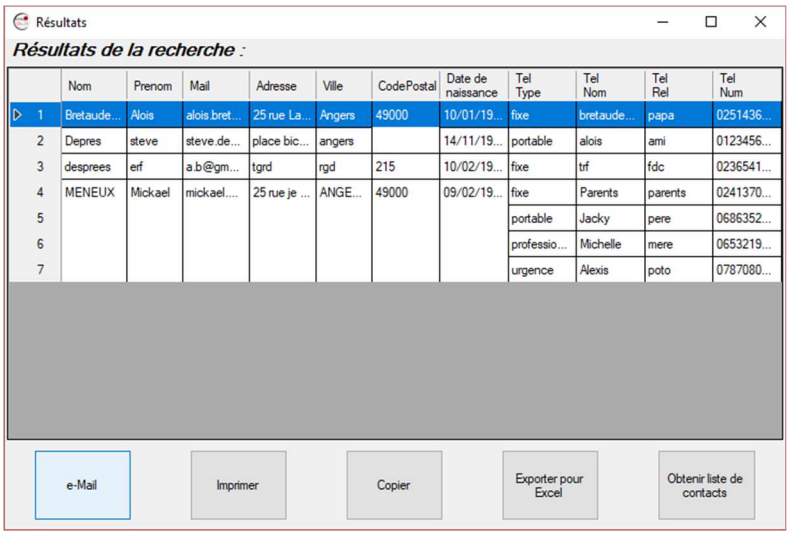


Figure 22 : Résultat d'une Recherche

La première, et celle qui à demander le plus de recherche, est la création d'une liste de distribution Outlook. Cette liste a pour vocation l'envoi de mails groupés pour chaque groupe de licencié. Le cas pratique

le plus probable est la création d'une liste de mails contenant toutes les adresses des judokas d'une séance. Cela permettra à l'utilisateur dans le futur d'envoyer un mail à toutes les personnes de cette liste sans faire de recherches. Pour réaliser cette fonction, ainsi que les fonctions e-Mail et export Excel, le package Microsoft Office a été utilisé. Pour la partie e-Mail le logiciel fait appel à Outlook, comme demandé dans la fiche de spécification, pour envoyer un mail au licencié sélectionné dans la liste de résultats. L'export pour Excel utilise lui aussi le package Microsoft Office. Son code simple et efficace est constitué d'un double for pour parcourir les données du « DataGridView » ainsi que le fichier Excel et de faire la copie de celle-ci.

Le bouton copier fais appel à l'objet clipboard qui récupère les données sélectionnées du tableau grâce à la méthode « GetClipboardContent() » de l'objet « DataGridView ». La fonction imprimer, elle, est la raison de la présence de la classe DGVPrinter dans les « Entities ». En effet après quelques recherches sur un moyen pratique d'imprimer un « DataGridView » les seules solutions étaient un ensemble de trois fonctions plutôt complexes ou bien la classe « DGVPrinter ». L'avantage de cette solution est sa simplicité d'utilisation car elle possède une fonction « PrintDataGridView() » qui prend en entré un « DataGridView » et l'imprime selon les paramètres configurés auparavant.

Le reste du code de la couche métier est un ensemble de fonctions de transfert de données entre la couche Ui et Dao et de mise en forme des données comme la fonction « RemoveDiacritics() » qui enlève les accents ou « CalculateAge() ».

2.3. Dao

2.3.1. Fonction de recherche : utilisation des JObject

La méthode « Recherche() » permet, dans la couche Dao, de faire la requête SQL appropriée à partir du formulaire de l'onglet « Recherche » de l'application. La complexité étant la modularité de la requête ; premièrement l'utilisateur ne va pas constamment tout cocher/remplir et deuxièmement on ne va pas mettre des éléments vides dans la requête (dans la clause WHERE). Il nous fallait donc un moyen d'envoyer des objets modulables ou de penser à une architecture de classes afin de nous faciliter le travail. Nous avons fini par choisir de construire une trame Json (voir ci-après).

```
json =
{
    {
        "Recherche": ["Nom", "Prenom", "Mail"],  //attributs selectionnés ou par défau
        "Critere":
        [
            {       "licencie": {"Nom"="test"}    },  ///attributs selectionnés avec le
            {       "telephone": {}    },
            {       "inscription": {}    },
            {       "seance": {}    }
        ]
```

Figure 23 : Trame Json pour l'onglet Recherche

Ce Json est donc séparé en deux parties : ce que l'utilisateur recherche, et ses critères. La taille de chaque sous-ensemble est modulable pour les raisons citées au-dessus. Cette trame étant relativement complexe à traiter, nous avons choisi d'utiliser la classe JObject pour effectuer son parcours. Un JObject est constitué d'une clé à laquelle est associée une valeur de type « JToken » (semblable aux dictionnaires). Cette valeur peut prendre plusieurs formes : un tableau (typé JArray) ou une propriété clé-valeur (typé JProperty).

Ses informations en poche, le parcours se fait tout naturellement (voir pseudo-code ci-joint). Toute l'astuce sera ensuite de constituer la requête de type « Select » au fur et à mesure du parcours de la trame et de son contenu.

```
foreach (KeyValuePair<string, JToken> kvp in jObject
{
    if(kvp.Key == "Recherche")
    {
        /* parcours des informations à afficher */
    }

    if(kvp.Key == "Critere")
    {
        /* parcours critères de recherches */
```

Figure 24 : Pseudo-code parcours d'un JObject

2.3.2. Requêtes SQL : SqlCommand et SqlDataReader

EntityFramework (EF) était censé nous servir à faire toutes nos requêtes. Le problème est que nous n'avons trouvé aucun moyen algorithmiquement ou même conceptuellement de réaliser la méthode « Recherche() » avec cette technologie. Nous nous sommes donc tournés vers la solution native proposée par .Net, à savoir les classes « SqlCommand » et « SqlDataReader ». Par soucis d'homogénéité dans le code et d'économie des ressources, nous les avons également utilisées dans les autres méthodes qui elles en revanche, étaient relativement simple à réaliser à l'aide d'EF.

La classe SqlCommand permet, entre autres, de gérer la connexion à la base de données ainsi que la création et l'exécution d'une requête SQL. La classe SqlDataReader permet quant à elle de récupérer et lire le résultat d'une requête. Le déroulé de chaque méthode de la classe « Dao » suit ainsi le pseudo-code ci-après. À noter que la déclaration des variables « cmd » (SqlCommand) et « reader » (SqlDataReader) ainsi que le paramétrage de la connexion se fait dans le constructeur de la classe « Dao ». Une particularité pour la fermeture de la connexion qui sera expliquée dans la prochaine partie.

```
try
{
    cmd.CommandText = "SELECT lcn_mail FROM donnee.licencie WHERE lcn_nom LIKE @nom"; // requête
    cmd.Parameters.Add("@nom", SqlDbType.VarChar).Value = nom; //ajout de paramètre et de sa valeur
    cmd.Connection.Open(); //ouverture de la connexion

    reader = cmd.ExecuteReader(); //execution de la requete
    if (reader.HasRows) //si la requête a renvoyé des données
    {
        /* récupération des donnée */
    }
    reader.Close(); //fermer le contexte de lecture
    /* return ... ; */
}
catch(Exception e) // traitement des exceptions
{
    throw new Exception("Erreur Dao.cs/NomFonction() commentaires :\r\n"+e.GetMessage(), e);
}finally
{
    /* libération des ressources */
}
```

Figure 25: Pseudo-code d'une requête SQL

3) Transactions et libérations de ressources

Les fonctionnalités d'édition, de suppression et de réinscription d'un licencié sont un peu particulières. Elles nécessitent de combiner à la fois modification, création et enfin suppression d'éléments de la base de données. On comprend aisément que si l'une d'entre elle génère une quelconque erreur, le résultat en base de données sera totalement erroné ; d'où l'utilisation de transactions. Les transactions permettent également d'avoir une trace écrite en base de données de l'exécution de requêtes. C'est la classe « SqlCommand » qui permet de les gérer via les instructions suivantes :

```
cmd.Transaction = cmd.Connection.BeginTransaction("NomTransaction"); // créer la transaction
cmd.Transaction.Commit() //commit la transaction
cmd.Transaction.Rollback(); //rollback si on a catch une exception
```

Figure 26 : Utilisation des transactions avec SqlCommand

Pour finir s'est posé le problème de la libération des ressources. Il se décline en plusieurs points :

- on ne peut pas fermer une connexion déjà fermée (génère une erreur) ;
- on ne peut créer deux paramètres de requêtes identiques ;
- le résultat d'une requête pouvant être lourd, le contexte de lecture de ce résultat doit passer dans le Garbage Collector (gérant la mémoire allouée au programme) au plus vite.

C'est ce problème qui nous a orienté vers l'architecture « try-catch-finally » afin de répondre proprement à chacun de ces points. La partie « finally » étant un passage obligatoire avant tout fin de fonction, elle est toute disposée à contenir le code propre à la libération des ressources.

3. Pour aller plus loin

Comme indiqué dans le planning nous n'avons pas pu terminer l'application dans le temps de projet. Cependant nous ne nous attendions pas à finir seulement deux des sept phases. Cette erreur de prévision vient de notre manque d'expérience dans le domaine du développement de logiciel complexe.

3.1. Fonctionnalités à rajouter

3.1.1. L'import de fichier Excel

La plus grosse partie que nous aurions développée, si nous avions eu le temps, est l'inscription par Excel. En effet cette fonctionnalité ne nécessite que peu de temps de développement pour la partie traitement des données, une interface simpliste et aucune fonction à écrire pour la couche Dao.

Le processus comme nous l'avons pensé se serait déroulé de la façon suivante, un onglet de l'application contenant un bouton pour parcourir des fichiers dans l'arborescence Windows. Si besoin un fichier type sera disponible via un bouton de l'onglet. Une fois le fichier importé, son nom s'affiche ainsi qu'un bouton pour valider le fichier et lancer l'import.

Ce fichier sera envoyé à la couche Métier qui, grâce au package Microsoft Office Excel le parcourra ligne par ligne, chaque ligne regroupant les informations nécessaires pour l'inscription d'un licencié. Une fois ces informations stockées dans les objets prévus, la couche Métier appelle des fonctions de la couche Dao. Premièrement une recherche par nom, prénom et mail pour vérifier s'il existe, si oui on modifiera ses données, sinon on effectuera un ajout de licencié.

Une fois l'ajout de chaque licencié contenu dans le fichier, et si tout s'est bien déroulé, un pop-up indiquant la réussite de l'import sera affiché à l'écran.

3.1.2. Gestion des comptes

La seconde partie qui était prévue dans le temps de projet et que nous aurions aimé développer est la gestion des différents comptes utilisateurs. Dans la fiche de spécifications conçue auparavant trois comptes existent : un compte administrateur, rédacteur, et lecteur avec chacun des droits différents.

Pour mettre en place cela, il faut avant tout mettre en place un nouveau schéma dans la base de données pour les différents comptes. Suite à cela, on ajoutera un formulaire de connexion lors du lancement de l'application qui permettra la création de variables globales. Ces variables serviront à afficher les différents onglets ou non en fonction des droits du compte.

Le compte lecteur n'a accès qu'aux onglets Recherche et à la gestion de son profil où il peut changer ses informations personnelles. Le compte rédacteur a accès à tous les onglets concernant l'inscription des licenciés en plus des onglets du lecteur. Pour finir l'administrateur peut accéder à tous les onglets dont la gestion des profils en eux-mêmes. Il peut donc modifier les droits des autres utilisateurs.

3.1.3. Administration de la base de données

La dernière partie à implémenter est l'administration de la base de données. Par cela nous entendons la sauvegarde de la base qui sera effectuée en routine tous les mois. Cette tâche sera validée par l'administrateur via un pop-up où il devra ou non accepter la sauvegarde. Une autre routine à mettre en place est la réévaluation des catégories d'âge pour les licenciés. Cette tache devra être effectuée tous les ans un peu avant le moment des réinscriptions pour ne pas avoir à rééditer tous les licenciés.

3.2. Management autour de la base de données

3.2.1. Procédures stockées

Actuellement toutes les requêtes SQL sont écrites dans la couche Dao et donc exécutées côté client. L'efficacité des requêtes (temps de réponse, mémoire allouée, etc.) dépendra donc de l'ordinateur de l'utilisateur. Il pourrait être intéressant de stocker toutes les requêtes nécessaires à l'application en tant que procédures dans la base de données. La couche Dao aurait alors simplement besoin d'appeler ces procédures en passant les paramètres spécifiques.

Dans ce cas précis ce serait la configuration de la base de données qui régirait l'efficacité de ces requêtes. Elle serait pondérée par le nombre d'utilisateur qui va alors utiliser ces procédures. Dans notre cas, seuls les membres du bureau du club y auraient accès et une petite dizaine de bénévoles, on n'aurait donc aucun soucis de performances. Les requêtes étant déjà rédigées dans la couche Dao et la documentation du langage Transact-SQL sur les procédures stockées étant assez riche, ce changement de conception devrait être facilement mis en place (voir exemple ci-dessous).

```
GO  /* début envoi instructions Transact-SQL */
CREATE PROCEDURE Licencie.RechercherParNomPrenom  /* NomTable.NomProcedure (convention) */
    @LastName nvarchar(50),   /* variables d'entrée de la procédure */
    @FirstName nvarchar(50)
AS
    /* début code de procédure */
    SET NOCOUNT ON; /* empêche envoi des messages "DONE_IN_PROC" très lourds, peu utiles sur des petites requêtes *
    SELECT lcn_nom AS Nom, lcn_prenom AS Prenom, lcn_datenaiss AS DateDeNaissance  /* éléments recherchés */
    FROM donnee.licencie /* nom de ma table (dans le schéma "donnee") */
    WHERE lcn_nom = @FirstName AND lcn_prenom = @LastName /* critères de recherches */
GO /* fin envoi instructions Transact-SQL */
```

Figure 27 : Exemple de procédure stockée pour la méthode RechercheNomPrenom() de la couche Dao

3.2.2. Indexation

En complément de cela, il s'agirait de mettre en place des indexeurs permettant de rendre plus rapide les temps de réponse des requêtes. On ne peut pas simplement en mettre sur chaque attribut de chaque table car cela alourdirait fortement le temps nécessaire au serveur pour générer un backup (fonctionnalité demandée par le client). Il conviendra donc de faire une étude sur les indexeurs à mettre en fonction des requêtes possibles et des attributs les plus souvent utilisés.

Le langage Transact-SQL propose un grand nombre de possibilités en termes d'indexeurs. Les deux principaux sont les indexeurs clusters et non clusters. Un indexeur cluster stocke et trie les données selon un ordre précis sur les valeurs d'une table. Aussi il ne peut y en avoir qu'un par table. Un indexeur non-cluster à l'inverse, par une structure bien spécifique, permet d'ajouter plusieurs indexeurs à une même table.

3.2.3. Sécurisation des entrées dans la base de données

Les données envoyées en base de données doivent toujours être filtrées pour éviter quelques mauvaises surprises (type injection SQL ou entrées totalement erronées). La partie sécurisation des entrées est gérée par la couche UI qui interagit directement avec l'utilisateur. Il est en général préconisé de faire ces contrôles également sur la couche Dao voire sur la base de données afin d'éviter tout conflit dans le cas ou quelqu'un reprendrait le code de l'application en changeant de couche UI.

Dans l'optique où les procédures stockées seraient de vigueur en base de données, la couche Dao allégée de ses requêtes, ces tests de sécurité pourraient être ajoutés sans alourdir le code. Dans notre application, de simples expressions régulières suffisent car nous n'avons à gérer que des noms, prénoms, mails, numéros de téléphone, dates et adresses.

3.3. Intégration

Un aspect du projet intéressant mais que nous n'avons pas pu aborder par manque de temps est le déploiement de l'application. Tous les sujets proposés à l'ISTIA ne permettent pas forcément de s'essayer à cette partie des métiers de l'ingénieur et c'est pourquoi nous pensons que cette facette du projet devrait être exploitée dans le cadre d'une continuation du projet l'an prochain. Ce déploiement pourrait être séparé en deux ou trois parties suivant le temps disponible.

Premièrement, l'installation du logiciel chez le client. Un membre de l'équipe pourra se rendre dans les bureaux du Ken'go Judo Ju-jitsu afin d'installer l'application sur les ordinateurs du personnel du club, à Bécon-les-Granits en périphérie d'Angers. Cette installation passera notamment par la vérification de la bonne marche de l'application sur ces ordinateurs et l'adaptation éventuelle des DLL utilisées. Afin de vérifier le fonctionnement de toutes les fonctionnalités, une checklist de test pourra être établie au préalable.

À la suite de cette installation, il faudra prévoir une formation pour le personnel du club qui utilisera le logiciel afin de lui apprendre les capacités de l'application et les manières de bien l'utiliser. Cela afin de permettre une prise en main rapide et efficace pour les membres du bureau du club mais également dans un but de permettre au personnel de former correctement d'éventuels nouveaux arrivants dans les bureaux du club.

Dans un deuxième temps, il sera intéressant de rédiger un manuel de l'utilisateur permettant une complémentarité avec la formation décrite ci-dessus. Ce manuel devra permettre à un utilisateur de répondre lui-même à ses questions et mettra à disposition un moyen de contacter les développeurs en cas de bug de l'application. Le manuel devra examiner les points clés abordés dans la formation mais aussi les origines possibles de défauts de fonctionnement.

Enfin, afin d'apporter un argument long terme au projet, on pourra établir une enquête de satisfaction à faire remplir par le club après un an d'utilisation afin d'étudier des perspectives d'amélioration jusqu'alors inconnues ou ignorées.

Conclusion

Durant ce projet, nous avons pu découvrir de nouvelles technologies et méthodes de travail comme les JObjects ou l'utilisation d'une base de données Azure mais aussi renforcer notre maîtrise de certaines que nous connaissions déjà, notamment le langage C# et l'utilisation de Windows Forms ou de l'architecture trois couches. Nous avons pu faire l'expérience de l'élaboration d'une fiche de spécifications et pour certains nous découvrir un intérêt pour l'UX design ainsi que pour le maquettage et la conception de vues. Nous avons pu mettre à profit nous connaissances de cours afin de réaliser différents diagrammes (UML, Gantt…) ou manager une base de données. Nous avons pu utiliser GitHub et Git Bash pour du versioning et nous sommes essayés à satisfaire une demande client en respectant des délais.

Cependant, comme nous nous en doutions lors de la conception de la fiche de spécifications, nous n'avons pas pu achever le logiciel dans les temps. Nous avons dû faire le choix de nous arrêter à la fin de la deuxième des sept phases prévues afin de terminer le rapport alors que nous visions la complétion de quatre d'entre elles.

Dans l'ensemble, ce projet s'est avéré très enrichissant pour nous trois et au vu de l'avancement du logiciel (Nous estimons avoir pu développer 45% du projet.), nous proposons sa reconduction l'an prochain en EI4-SAGI. Nous nous tiendrons à la disposition des élèves souhaitant le reprendre afin de répondre à leurs questions et de les aiguiller dans le projet.

Bibliographie :

Documentation sur les JObject :
https://www.newtonsoft.com/json/help/html/T_Newtonsoft_Json_Linq_JObject.htm

Documentation sur la classe SqlCommand :

https://msdn.microsoft.com/fr-fr/library/system.data.sqlclient.sqlcommand(v=vs.110).aspx

Documentation de la classe SqlDataReader :

https://msdn.microsoft.com/fr-fr/library/system.data.sqlclient.sqldatareader(v=vs.110).aspx

Documentation sur les requêtes en langage Transact-SQL :

https://docs.microsoft.com/fr-fr/sql/t-sql/queries/queries

Documentation sur les transactions :

https://docs.microsoft.com/fr-fr/sql/t-sql/language-elements/transactions-transact-sql

Documentation sur les indexeurs :

https://docs.microsoft.com/fr-fr/sql/t-sql/statements/create-index-transact-sql

Documentation sur la libération de ressources :

https://docs.microsoft.com/fr-fr/dotnet/standard/garbage-collection/unmanaged

Documentation Entity Framework :

http://www.entityframeworktutorial.net/entityframework6/introduction.aspx

https://docs.microsoft.com/fr-fr/dotnet/framework/data/adonet/ef/overview

https://msdn.microsoft.com/en-us/library/jj682076(v=vs.113).aspx

Documentation de la classe DGVPrinter :

https://www.codeproject.com/Articles/18042/Another-DataGridView-Printer?msg=3124321

Annexes

1. Fiche de spécifications

Projet EI4 : Logiciel de gestion pour un club de judo

- Fiche de spécifications -

Historique de versions :

- Version 1 : Création du document
- Version 2 : Validation client

Rédacteurs :

- Julien Raillard (Chef de projet)
- Aloïs Bretaudeau
- Mickaël Meneux

Client :

- Marc Saunot (Kengo Judo)

Responsable Pédagogique :

- Medhi Lhommeau

EX_001 :

L'application doit permettre de se connecter via trois types de comptes : **Admin, Rédacteur, et Lecteur**.

EX_002 :

Le **Lecteur** doit avoir accès à 2 onglets : "**Gestion de Profil**" et "**Recherche**".

Ex_003 :

Le **Rédacteur** doit avoir accès 4 à onglets : "**Gestion de Profil**", "**Recherche**", "**Gestion des licenciés**" et "**Inscription par Excel**".

Ex_004 :

L'**Admin** doit avoir accès 5 à onglets : "**Gestion de Profil**", "**Recherche**", "**Gestion des licenciés**", "**Inscription par Excel**", "**Gestion Base de Données**" et "**Gestion des utilisateurs**".

EX_005 :

L'onglet **Profil** doit permettre de modifier les informations du compte : mot de passe et adresse de redirection.

Exception : seul l'**Admin** peut modifier son identifiant.

EX_006 :

L'onglet **Recherche** doit :

EX_006.1 : permettre d'effectuer une recherche par les éléments suivants :

- Cours (discipline / niveau / créneau (=jour+heure));
- Licencié (nom / prénom / age / mail / ville);
- Numéro de téléphone (quel qu'il soit "Portable", "Fixe", "Professionnel", "Urgence");
- Inscription (année d'inscription(une ou plusieurs)/ paiement / certificat / licence).

EX_006.2 : permettre d'afficher un ou plusieurs champs parmi : -

Nom et prénom (toujours affiché);

- Adresse mail (format outlook, liste de distribution, etc.);
- Numéro de téléphone; - Adresse + CP + ville; - Date de naissance.

EX_006.3 : L'affichage, après validation de l'utilisateur, doit se faire de la manière suivante :

1. Une fenêtre se crée (pop up qui permettrait d'afficher plusieurs résultats en même temps);
2. Un tableau est affiché avec les différents champs souhaités; 3. 5 boutons doivent respectivement permettre de :

i. copier les données du tableaux;
ii. exporter ces données au format excel;
iii. imprimer le contenu de la fenêtre;
iv. envoyer un mail;
v. générer une liste de contacts.

EX_007:

L'onglet "**Gestion des licenciés**" doit permettre : EX_007.1 : L'ajout d'un licencié.

EX_007.2 : La modification des informations/inscriptions d'un licencié.

EX_007.3 : La suppression d'un licencié.

Cas d'erreurs :

1. Dans le cas où un licencié existe déjà, ses informations personnelles seront modifiées (si elles ont changées) mais ses précédentes inscriptions ne seront jamais effacées.
2. La seule exception est la suivante : dans le cas d'une erreur lors de l'inscription, ex: inscrit en minime au lieu de sénior, taiso au lieu de jujitsu, etc.. il doit être possible de modifier ou supprimer cette dernière si et seulement si elles ont eu lieu l'année courante.

EX_008 :

L'onglet "**Inscription par Excel**" doit :

EX_008.1 : permettre l'import d'un fichier au format Excel (fichier normalisé) comportant la liste des licenciés à inscrire pour **une** année donnée.

Cas d'erreurs :

1. Des licenciés sont déjà inscrit pour l'année spécifiée, l'import d'un fichier au format Excel écrasera les inscriptions des licenciés présent sur ce fichier et les informations personnelles seront mises à jour.
2. Le fichier est vide / le format de données est incorrect pour une ligne => les données ne seront pas enregistrées (abandon de la procédure).

EX_008.2 : permettre l'export d'un fichier normalisé vide.

EX_009 :

L'onglet "**Gestion Base de Données**" doit permettre :

EX_009.1 : de faire une sauvegarde des données de l'année courante (routine tous les mois proposée à l'**Admin** qui doit la confirmer).

EX_009.2 : la réévaluation des catégories d'âge soumise à validation par l'**Admin**.

EX_010 :

L'onglet "**Gestion des utilisateurs**" doit permettre :

EX_010.1 : Permettre d'ajouter un compte utilisateur.

EX_010.2 : Permettre de modifier les informations d'un compte utilisateur (id, mdp, adresse de redirection, statut = Rédacteur ou Lecteur).

EX_010.3 : Permettre de supprimer un compte utilisateur.

Pour aller plus loin :

EX_100 :

L'**Admin** et le **Rédacteur** ont accès à l'onglet "**Compétition**".

EX_101 :

L'onglet "**Compétition**" doit permettre de :

EX_101.1 : retranscrire l'inscription d'un licencié à une compétition.

EX_101.2 : retranscrire ses résultats.

EX_102 :

L'onglet "**Recherche**" doit également permettre de **rechercher par les compétitions** (niveau = département, région, etc. ; catégorie d'âge ; résultats) en vue de faciliter la remise de trophées en fin d'année.

2. Script SQL de la base de données

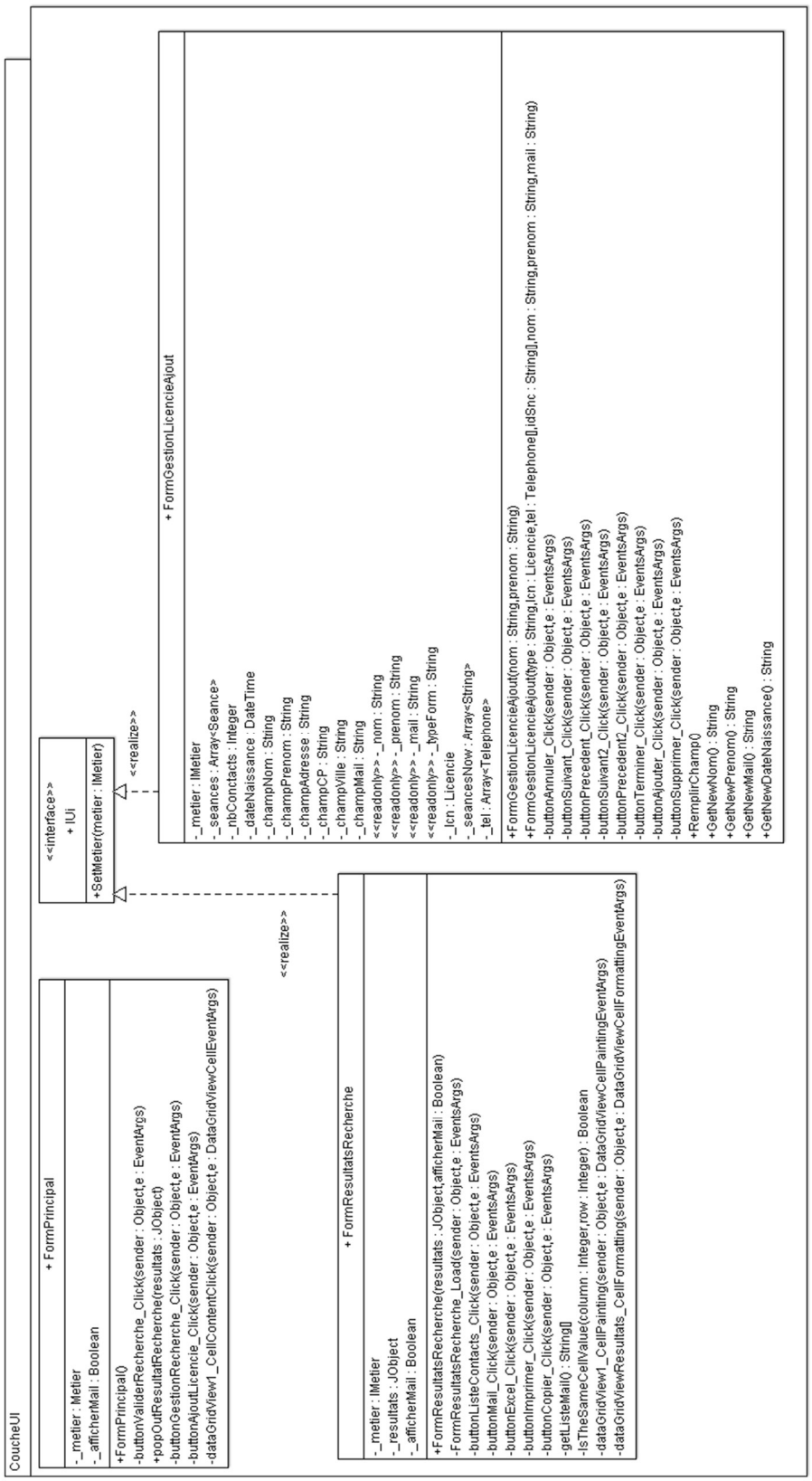
```
begin transaction @creerBDD_v1
/******************* Schéma pour les données ************************/
create schema donnee;

/* créer schema */
GO

/* table licencie */
create table donnee.licencie
(lcn_id int IDENTITY,
lcn_nom varchar(25) not null,
lcn_prenom varchar(25) not null,
lcn_datenaiss date not null,
lcn_genre varchar(25) not null,
lcn_adresse varchar(50) not null,
lcn_cp varchar(10) NOT NULL,
lcn_ville varchar(50) not null,
lcn_mail varchar(50) not null,
lcn_paye bit not null,
lcn_certif bit not null,
lcn_licence bit not null,
constraint pk_licencie primary key(lcn_id),
constraint chk_1 check(lcn_genre IN ('H','F')));

/* table telephone */
create table donnee.telephone
(tel_id int IDENTITY,
tel_lcn int not null,
tel_nom varchar(50) not null,
tel_cond nvarchar (25) not null,
tel_type nvarchar(25) not null,
tel_num nvarchar(15) not null,
constraint pk_telephone primary key(tel_id),
constraint fk_telephone foreign key(tel_lcn) references donnee.licencie(lcn_id),
constraint chk_2 check(tel_type IN ('fixe','portable','professionnel','urgence')));

/* table seance */
```

```
create table donnee.seance
(snc_id nchar(8) not null,
snc_discip nvarchar(25) not null,
snc_niveau nvarchar(25) not null,
snc_jour nvarchar(10) not null,
snc_horaire nvarchar(10) not null,
constraint seance_pri primary key(snc_id),
constraint chk_3 check(snc_discip IN ('judo','taiso','jujitsu','ne-waza')),
constraint chk_4 check(snc_niveau IN('eveil','pre-poussin','poussin','benjamin','minime','adulte et
    ado')),
constraint chk_5 check(snc_jour IN ('lundi','mardi','mercredi','jeudi','vendredi')));
INSERT INTO donnee.seance VALUES('S01', 'judo', 'eveil', 'mardi', '17h30');
INSERT INTO donnee.seance VALUES('S02', 'judo', 'eveil', 'mercredi', '17h00');
INSERT INTO donnee.seance VALUES('S03', 'judo', 'pre-poussin', 'lundi', '17h30');
INSERT INTO donnee.seance VALUES('S04', 'judo', 'pre-poussin', 'vendredi', '17h30');
INSERT INTO donnee.seance VALUES('S05', 'judo', 'poussin', 'mercredi', '18h00');
INSERT INTO donnee.seance VALUES('S06', 'judo', 'poussin', 'jeudi', '17h30');
INSERT INTO donnee.seance VALUES('S07', 'judo', 'benjamin', 'mardi', '18h30');
INSERT INTO donnee.seance VALUES('S08', 'judo', 'benjamin', 'vendredi', '18h30');
INSERT INTO donnee.seance VALUES('S09', 'judo', 'minime', 'lundi', '18h30');
INSERT INTO donnee.seance VALUES('S10', 'judo', 'minime', 'jeudi', '19h00');
INSERT INTO donnee.seance VALUES('S11', 'judo', 'adulte et ado', 'vendredi', '20h00');
INSERT INTO donnee.seance VALUES('S12', 'jujitsu', 'adulte et ado', 'lundi', '20h00');
INSERT INTO donnee.seance VALUES('S13', 'ne-waza', 'adulte et ado', 'mercredi', '19h45');
INSERT INTO donnee.seance VALUES('S14', 'taiso', 'adulte et ado', 'mardi', '20h15');
INSERT INTO donnee.seance VALUES('S15', 'taiso', 'adulte et ado', 'jeudi', '20h30');


/* table inscription */
create table donnee.inscription
(isc_id int IDENTITY,
isc_lcn int not null,
isc_snc nchar(8) not null,
isc_date date not null,
constraint pk_inscription primary key(isc_id),
constraint inscription_fk1 foreign key(isc_lcn) references
donnee.licencie(lcn_id),
constraint inscription_fk2 foreign key(isc_snc) references
```

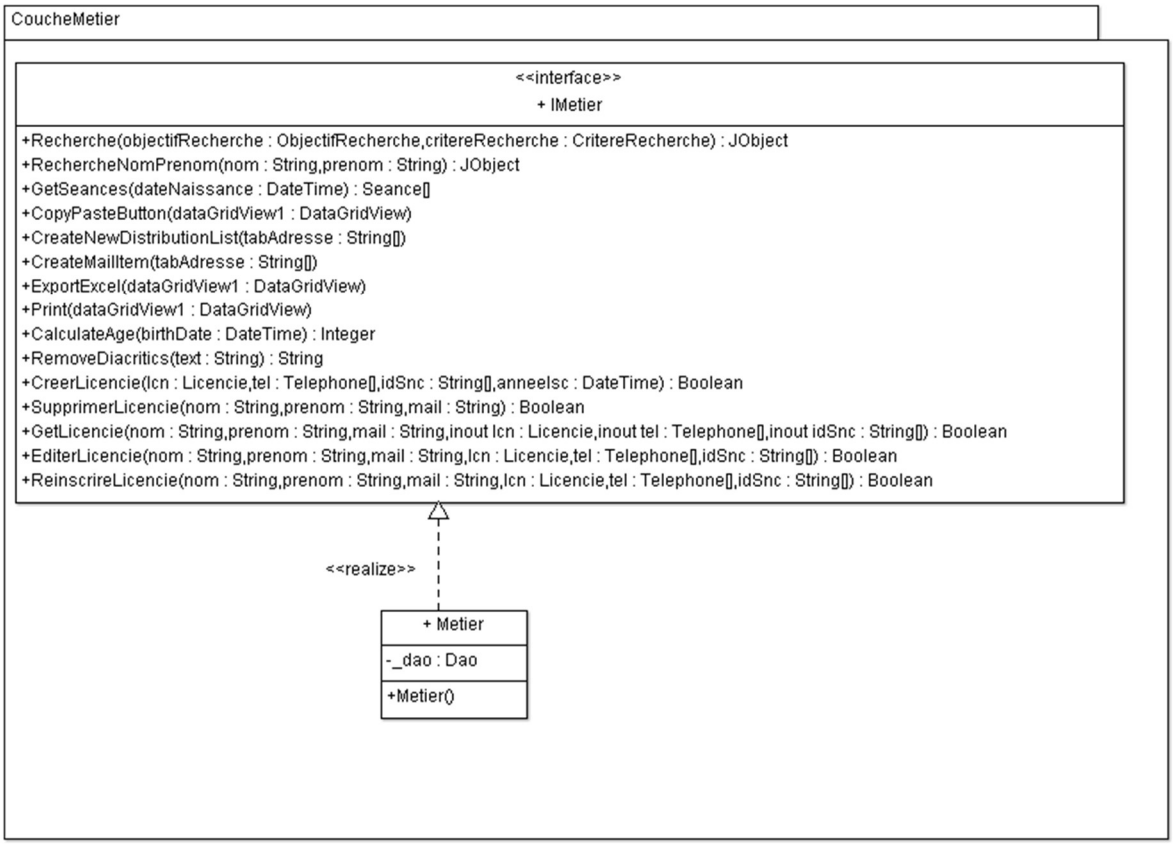
```
donnee.seance(snc_id));

/* si toutes les tables n'ont pas rencontré d'erreurs */
GO

/****************** Schéma pour les comptes de l'application**********************/
create schema compte;
GO
commit transaction @creerBDD_v1
```

3. Diagrammes de classe

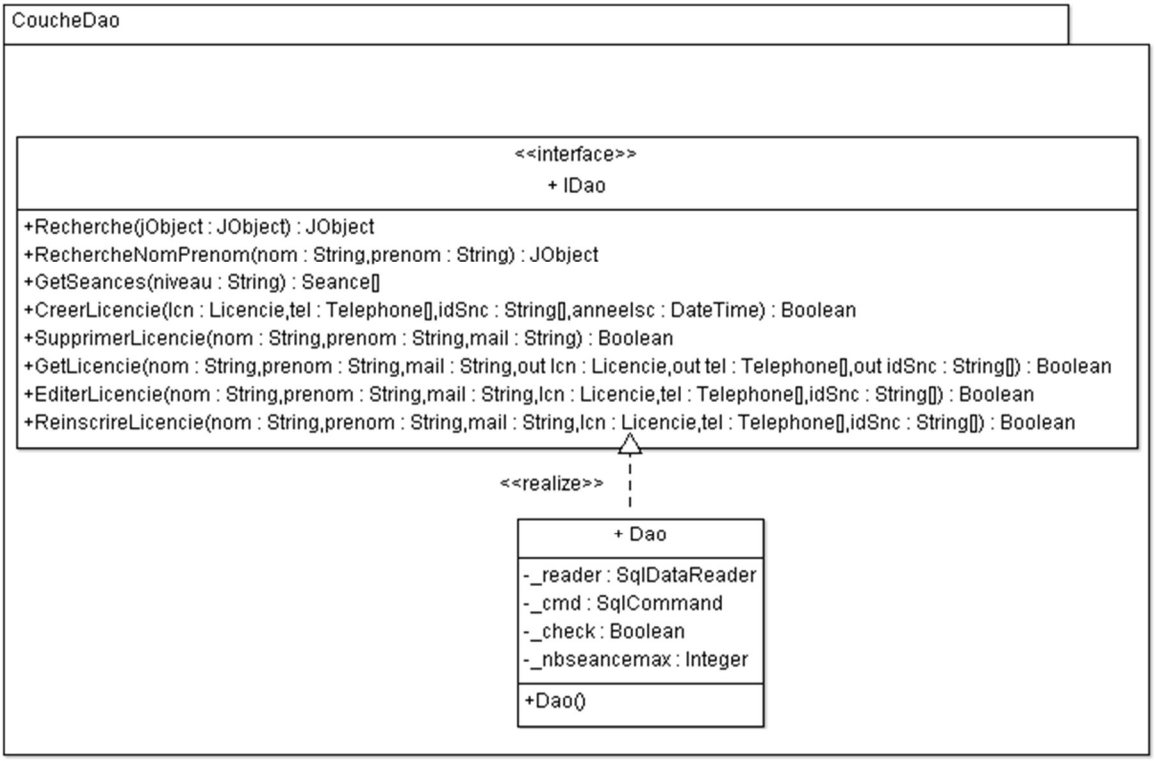
3.1. Couche UI



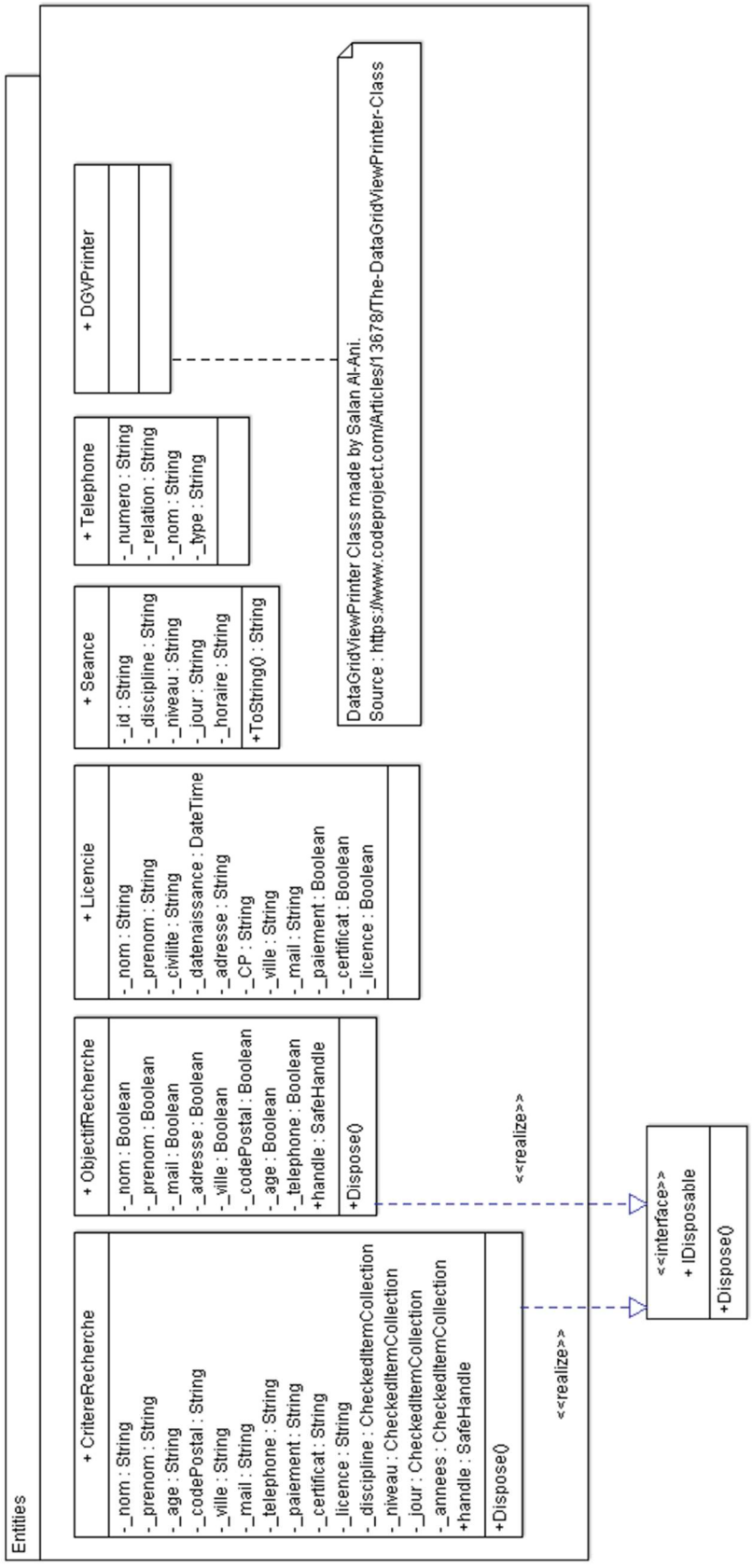
3.2. Couche Métier



3.3. Couche Dao



3.4. Package Entities



4. Diagrammes d'activité

Diagramme de **Recherche** (onglet Recherche)

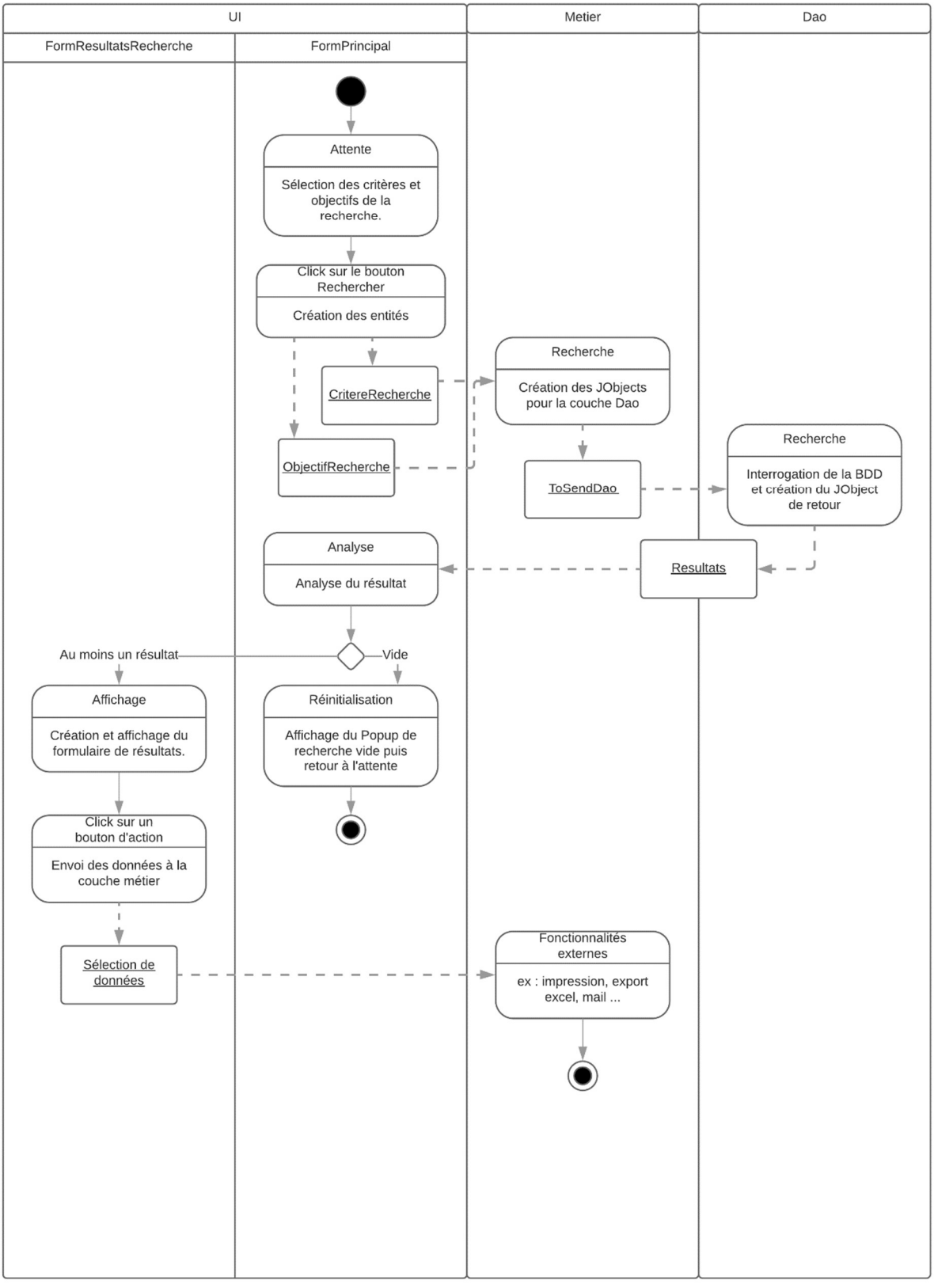


Diagramme de **Recherche par nom et/ou prénom** (onglet Gestion)

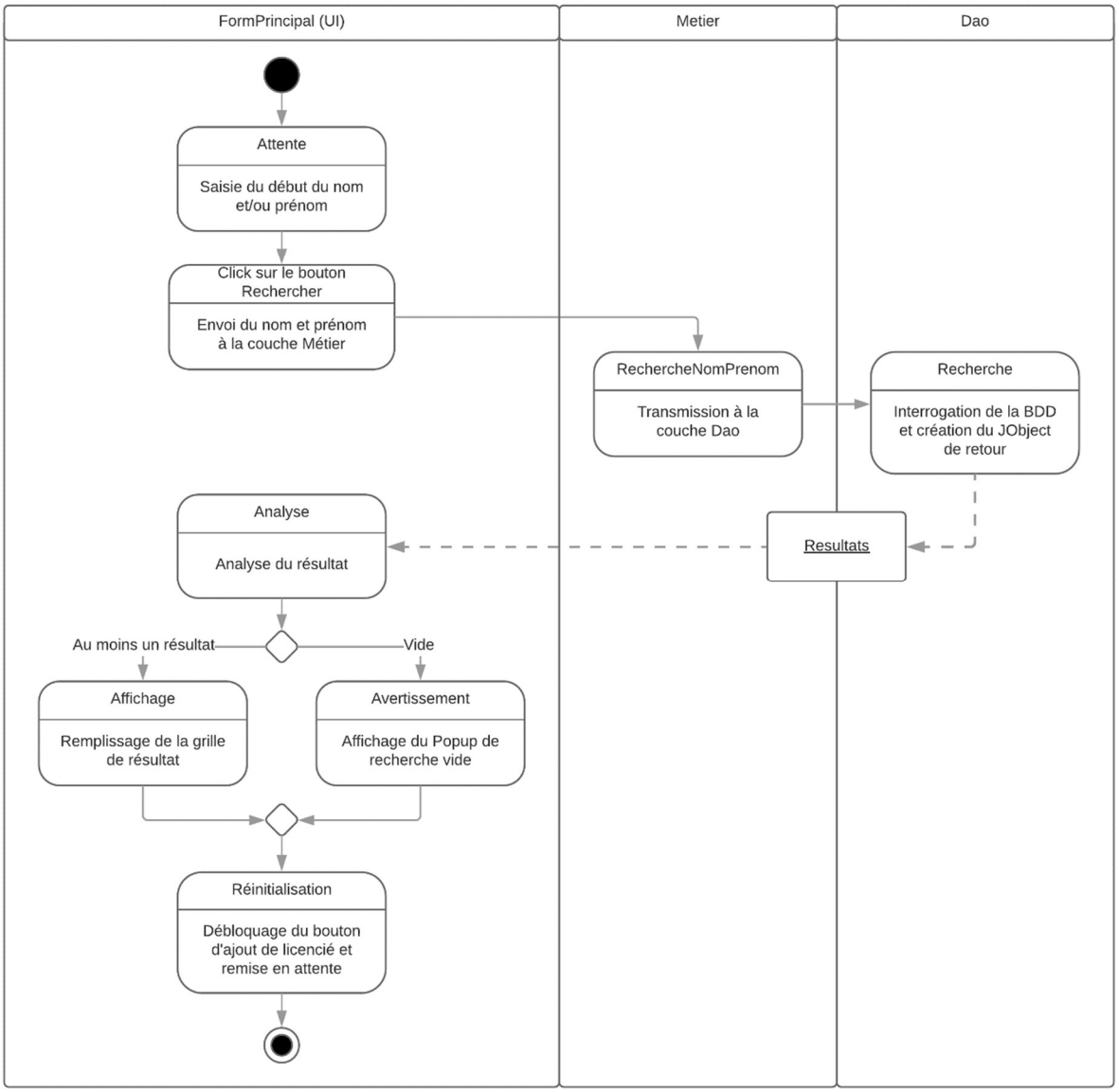


Diagramme d'Ajout de licencié (onglet Gestion)

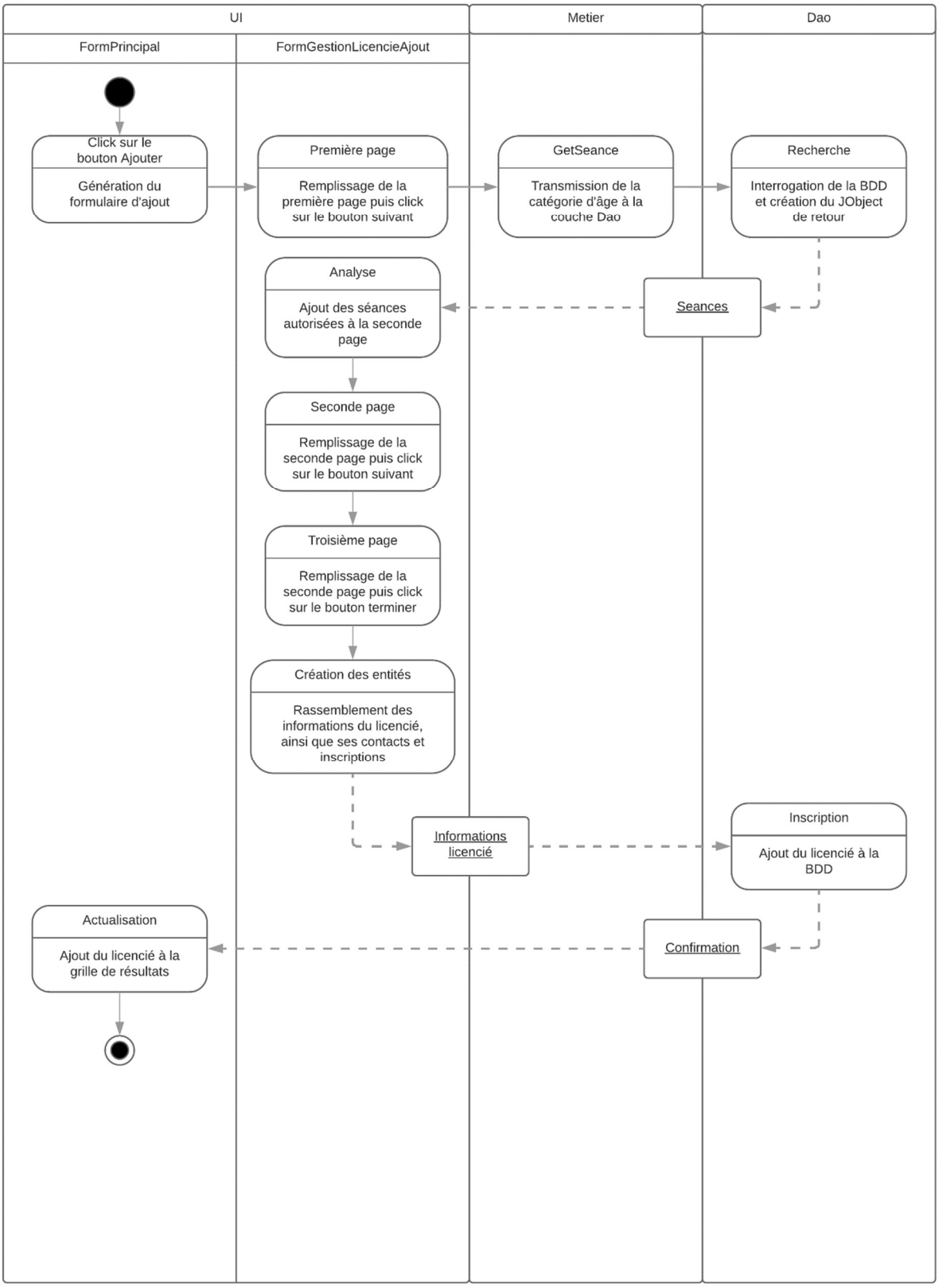


Diagramme de Réinscription (fonctionnement identique à l'Édition) de licencié (onglet Gestion)

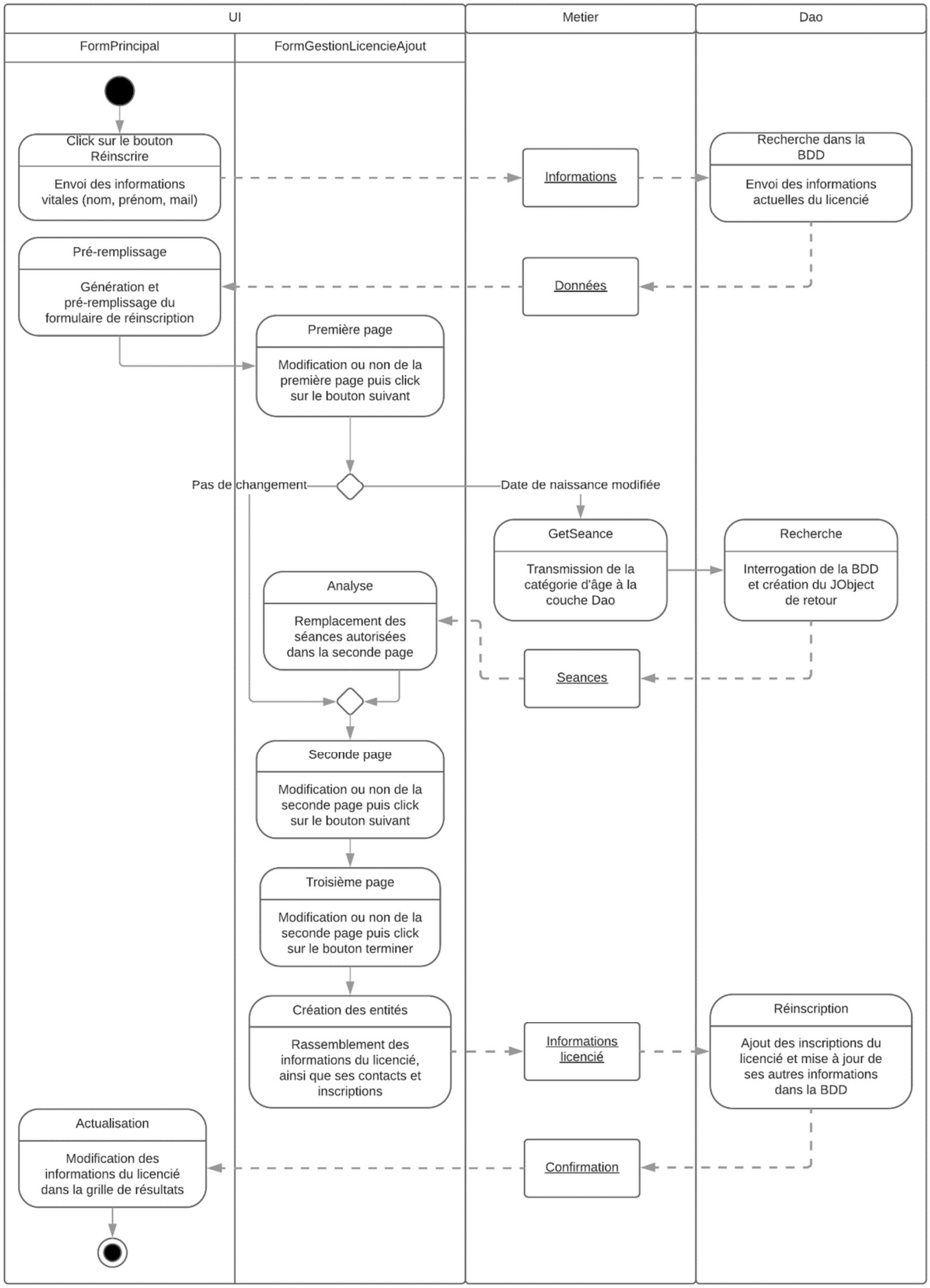
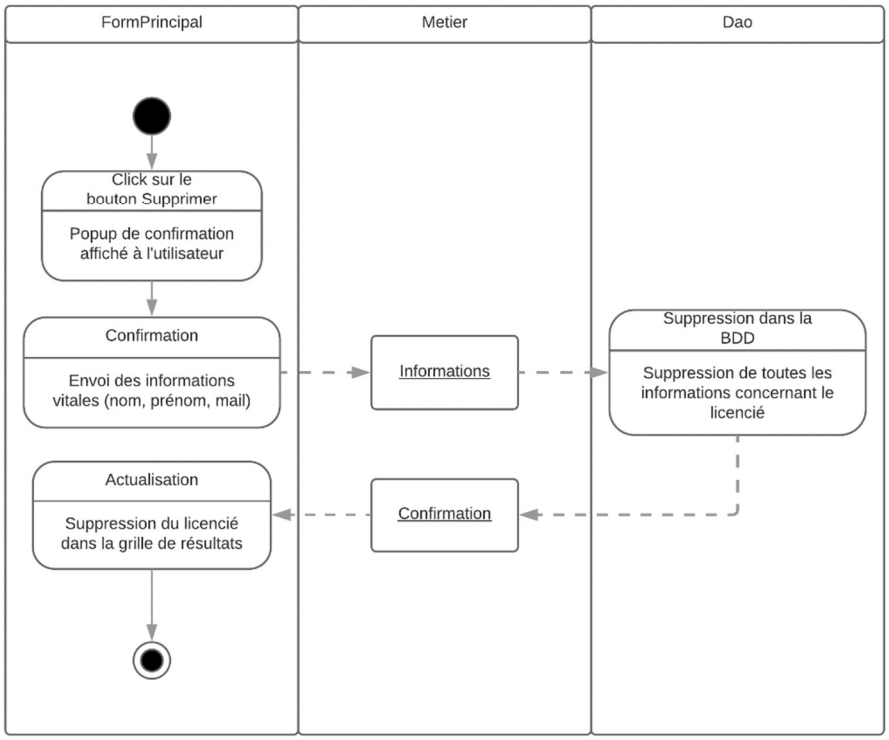


Diagramme de **Suppression** de licencié (onglet Gestion)



Résumé/Abstract

Ce projet de logiciel de gestion a été réalisé dans le cadre de la deuxième année d'ingénieur à l'Istia. Il a été conçu et développé par une équipe de trois élèves, Julien Raillard, Aloïs Bretaudeau et Mickael Meneux pour le club Ken'go Judo. Ils ont tout d'abord conçu le projet dans son ensemble, incluant une fiche de spécifications rédigée en lien avec le dirigeant du club Marc Saunot ainsi qu'un document répertoriant les technologies utilisées. Une fois ces documents présentés au responsable projet et acceptés, le développement a pu commencer. Au final le logiciel Ken'go Software contient deux onglets, un pour la recherche de licenciés et l'autre pour l'ajout, suppression, édition et réinscription de licencié dans la base de données. L'équipe n'a pas réussi à tenir le planning pour des raisons de manque d'expérience dans ce domaine, cependant le client étant satisfait de ce résultat partiel, celui-ci aimerait le voir reporter à l'année suivante pour de futurs EI4.

This management software project was carried out as part of the second year of engineering at Istia. It was designed and developed by a team of three students, Julien Raillard Aloïs Bretaudeau and Mickael Meneux for the Ken'go Judo Club. They first designed the project as a whole, including a specification sheet written in collaboration with club leader Marc Saunot and a document listing the technologies used. Once these documents were presented to the project manager and accepted, development could begin. Finally, Ken'go Software contains two tabs, one for licensee search and the other for adding, deleting, editing and re-entering them in the database. The team was not able to keep the schedule due to lack of experience in this area, however the client was satisfied with this partial result and would like to see it postponed to the following year for prospective EI4.