

AUZANNEAU Maxime
CAFFY Cédric
G32

Création d'un logiciel de gestion d'aéroclub



Année 2015-2016

Sommaire

Introduction	3
I. La partie « analyse » du projet	4
1. Cahier des charges.....	4
a. La gestion des membres de l'aéroclub	4
b. La gestion des avions de l'aéroclub.....	5
c. La gestion du retour d'un vol	5
d. La gestion des paiements.....	5
e. L'aide à la préparation d'un vol (optionnel).....	6
2. L'analyse UML	7
a. Le diagramme de Use Case	7
b. Description détaillée de chaque Use Case.....	8
c. Le diagramme de classe	10
3. La base de données.....	12
4. Les maquettes	12
5. Les outils utilisés	13
a. L'analyse UML	13
b. Le maquettage de l'application	13
c. Le langage de programmation	13
d. La base de données.....	15
e. L'IDE (Integrated Development Environment)	15
f. Le gestionnaire de version.....	16
II. Le développement de notre application.....	16
1. Le design pattern MVC	16
2. L'implémentation d'un controller	18
3. La gestion de la base de données en java	19
a. Le design pattern Singleton.....	19
b. Le design pattern DAO.....	19
c. Les requêtes préparées.....	22
d. La gestion des exceptions	22
III. Le projet	23
1. L'avancement du projet.....	23
2. Les difficultés rencontrées.....	23
a. JavaFX.....	23
b. Les bonnes pratiques de la programmation	24
c. Le travail en équipe	24
Conclusion.....	25
Annexes.....	Erreur ! Signet non défini.

Introduction

Dans le cadre de notre projet de première année de cycle d'ingénieur en informatique et modélisation, nous avons souhaité faire un logiciel de gestion d'aéroclub.

Un aéroclub est une association à but non lucratif qui a pour but de permettre la pratique du pilotage d'avions légers.

Notre logiciel doit alors permettre de gérer tout ce qui est en rapport avec les avions et les membres qui composent l'aéroclub.

En première partie, nous présenterons la partie analyse de notre projet (analyse UML, outils utilisés...). En deuxième partie nous expliquerons comment nous avons développé notre application. Enfin, nous aborderons les difficultés rencontrées au sein du projet.

I. La partie « analyse » du projet

L'analyse est une phase très importante au sein d'un projet. Elle permet de faire un lien entre les développeurs et l'utilisateur final, mais permet également d'avoir une vue d'ensemble du projet afin que les développeurs puissent avoir un même socle de départ.

1. Cahier des charges

a. La gestion des membres de l'aéroclub

Le logiciel doit permettre à l'utilisateur de saisir et sauvegarder toutes les informations nécessaires à l'identification d'un membre de l'aéroclub. Chaque membre possède :

- Un nom
- Un prénom
- Une adresse
- Un email
- Un numéro de téléphone
- Une date de naissance
- Une date de péremption de sa visite médicale classe B
- 0,1 ou plusieurs brevets
- Un numéro d'instructeur

i. Le brevet

Un membre n'a pas le droit de voler seul à bord sans avoir un brevet. Le brevet est en réalité un diplôme délivré par un instructeur habilité suite à la réussite du vol de contrôle.

Il existe plusieurs types de brevet :

- Le brevet de base (BB) qui permet au pilote de piloter seul dans un rayon de 30 km autour de l'aérodrome de départ. Le pilote peut aller plus loin si l'instructeur lui en donne l'autorisation.
- Le PPL (Private Pilote Licence) qui permet au pilote de piloter seul ou accompagné de passagers n'importe où dans le monde entier.
- Le LAPL (Light Aircraft Pilot Licence) qui est un PPL allégé dans le sens où le pilote ne peut voyager qu'en Europe.
- Le CPL (Commercial Pilot Licence) qui est le brevet obligatoire pour pouvoir se faire rémunérer sur chaque vol (Les instructeurs ont tous un CPL).

ii. La visite médicale

Chaque membre doit passer une visite médicale d'aptitude au pilotage afin de pouvoir piloter les avions du club.

iii. Instructeur

Un instructeur est un membre du club ayant reçu une autorisation (par les autorités compétentes) pour pouvoir donner des leçons au sol et en vol à un élève.

L'instructeur possède un numéro d'instructeur sous la forme FI-XXXXXXXXXXXX.

b. La gestion des avions de l'aéroclub

Les membres de l'aéroclub peuvent louer des avions.

Chaque avion possède :

- Un nom (Ex : Robin DR-400, Cessna 172)
- Un type (Voyage, voltige, école, remorquage)
- Une immatriculation (Sous la forme X-XXXX avec X une lettre de A à Z)
- Une autonomie (en régime de croisière)
- Une capacité de réservoir
- Un nombre de places
- Une masse maximale à ne pas dépasser
- Un centrage (Le centre de gravité de l'avion varie en fonction du poids et de la répartition du chargement). Le calcul du centrage doit être fait avant chaque vol, si le centre de gravité dépasse une limite avant ou une limite arrière, l'avion devient dangereux.
- Un coût à l'heure de vol
- Une disponibilité (Loué, disponible, en réparation)
- Une vitesse de croisière

Le logiciel doit pouvoir stocker toutes les informations de chaque avions de l'aéroclub. Ces informations devront être utilisées pour l'aide à la préparation d'un vol ou pour la réservation d'un avion.

c. La gestion du retour d'un vol

Après chaque vol, le pilote doit rentrer les informations de son vol dans le logiciel. La saisie du temps de vol est le plus important car c'est lui qui va déterminer le prix que devra payer le pilote pour son vol.

Le logiciel doit pouvoir enregistrer :

- La date et heure du vol
- Les différentes étapes du vol (Aérodrome de départ, aérodromes d'escale, aérodrome de destination)
- Le temps de vol entre chaque étape
- L'instructeur avec lequel le pilote a volé
- Le nombre de passagers présents

Après la saisie de ces informations, le logiciel doit calculer combien le pilote doit payer à l'aéroclub pour son vol.

d. La gestion des paiements

Le pilote ne doit pas être en négatif, la survie du club en dépend.

L'utilisateur doit pouvoir saisir un paiement. Plusieurs types de paiement sont possibles :

- Chèque bancaire
- Espèces

La date du paiement, le nom et le prénom du pilote et le montant payé doivent être enregistrés.

i. Chèque bancaire

Le logiciel doit enregistrer les informations suivantes pour le dépôt d'un chèque :

- La banque à débiter
- Le numéro de chèque

ii. Espèces

Le pilote peut payer directement en espèce (après sa saisie, il ira mettre ses espèces dans la caisse de l'aéroclub prévue à cet effet).

e. L'aide à la préparation d'un vol (optionnel)

Chaque pilote « conscientieux » doit préparer son vol avant de partir en navigation. Après préparation, le pilote doit avoir avec lui un log de navigation (Figure 1).

Notre logiciel doit alors permettre d'aider le pilote à préparer son vol en générant ce log de navigation en fonction des informations saisies.

Notre logiciel doit aussi être capable de calculer le centrage de l'avion (en fonction de son chargement) afin que le pilote puisse savoir s'il peut voyager comme prévu en toute sécurité ou s'il va devoir adapter son chargement...

Figure 1: Un log de navigation

2. L'analyse UML

L'analyse UML (Unified Modeling Language) permet de fournir une méthode normalisée afin de visualiser la conception d'un système. Nous avons opté pour ce type d'analyse car nous connaissons son efficacité de par notre expérience antérieure.

a. Le diagramme de Use Case

Ce diagramme permet de décrire les cas d'utilisations du logiciel. Il permet de répondre de manière synthétique aux questions « Que peut faire le logiciel ? », « Qui peut faire telle ou telle action ? », « Qu'est-il nécessaire de faire pour effectuer une action ? ».

Voici le diagramme de Use Case que nous avons réalisé :

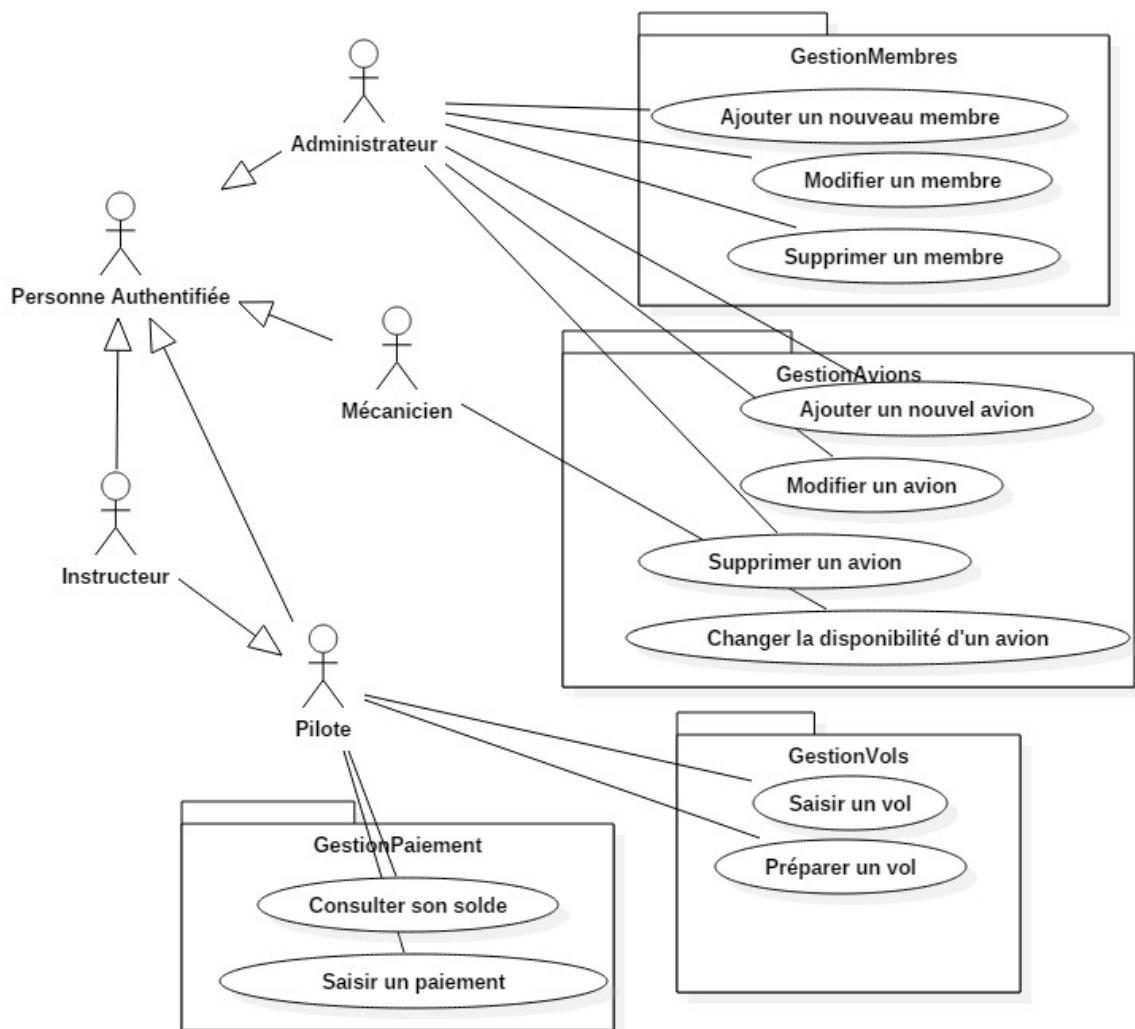


Figure 3: Le diagramme de Use Case du logiciel

b. Description détaillée de chaque Use Case

i. Use case « Ajouter un nouveau membre »

Objectif : Inscription d'un nouveau membre dans le logiciel

Acteur principal : l'administrateur

Préconditions : L'administrateur doit être authentifié

Post conditions : Un nouveau membre sera ajouté

ii. Use case « ajouter un nouvel avion »

Objectif : Ajouter un nouvel avion dans le logiciel

Acteur principal : l'administrateur

Préconditions : L'administrateur doit être authentifié

Post conditions : Un nouvel avion sera ajouté à la flotte de l'aéroclub

iii. Use case « Saisir un vol »

Objectif : Saisie d'un vol lors du retour d'un pilote

Acteur principal : le pilote

Préconditions : le pilote doit être authentifié

Post conditions : Le compte pilote sera débité du montant de l'heure de vol

iv. Use case « Saisir un paiement »

Objectif : Saisie d'un paiement pour réapprovisionner le compte pilote du membre

Acteur principal : le pilote

Préconditions : le pilote doit être authentifié

Post conditions : le compte pilote sera crédité du montant de l'heure de vol

v. Use case « Supprimer un membre »

Objectif : Supprimer un membre de la base de données

Acteur principal : l'administrateur

Préconditions : l'administrateur doit être authentifié et le membre doit exister

Post conditions : Toutes les informations relatives au membre seront supprimées.

vi. Use case « Supprimer un avion »

Objectif : Suppression d'un avion de la base de données

Acteur principal : l'administrateur

Préconditions : l'administrateur doit être authentifié et l'avion doit exister

Post conditions : Toutes les informations relatives à l'avion seront supprimées

vii. Use case « Consulter un solde »

Objectif : Consulter le solde d'un membre

Acteur principal : le membre

Préconditions : Le membre doit être authentifié

Post conditions : aucune

viii. Use case « Préparer un vol »

Objectif : Aider le pilote à préparer son futur vol

Acteur principal : le pilote

Préconditions : Le pilote doit être authentifié

Post conditions : aucune

ix. Use case « Changer la disponibilité d'un avion

Objectif : Permet au mécanicien de changer l'état d'un avion (en état de vol, en réparation, en révision)

Acteur principal : le mécanicien

Préconditions : le mécanicien doit être authentifié

Post conditions : aucune

c. Le diagramme de classe

Le diagramme de classe permet d'avoir une vue d'ensemble des différentes classes métiers de notre application. Il permet également de faciliter l'implémentation de ces classes car les liaisons entre chaque classe sont visibles d'un seul coup d'œil.

Tout d'abord, nous avons dû réaliser un diagramme de classe « orienté utilisateur » c'est-à-dire qu'il ne contient aucun type informatique.

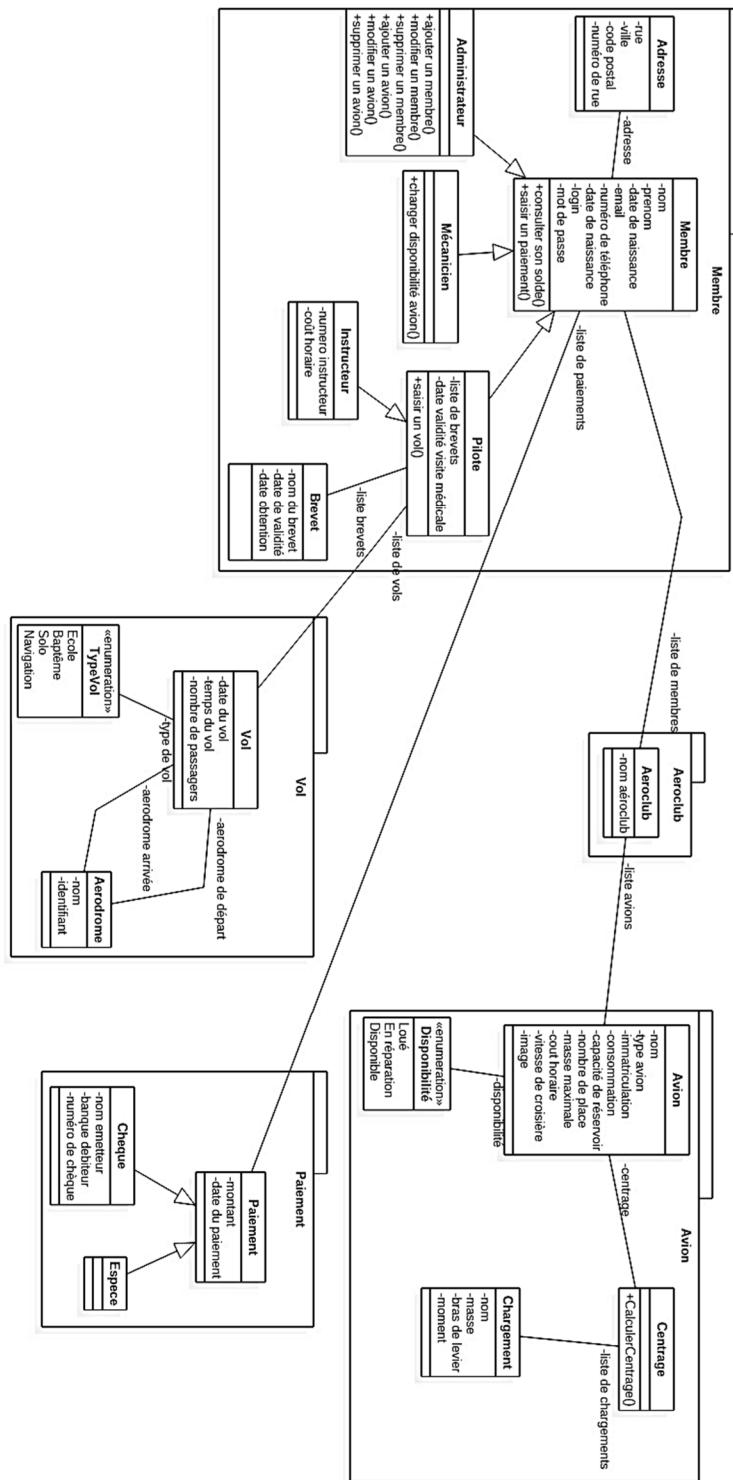


Figure 4: Notre diagramme de classe orienté utilisateur

Après avoir créé le diagramme de classe utilisateur, nous avons pu créer le diagramme de classe qui va nous servir pour le développement de notre application : il contient tous les types primitifs ou objets qui seront utilisés.

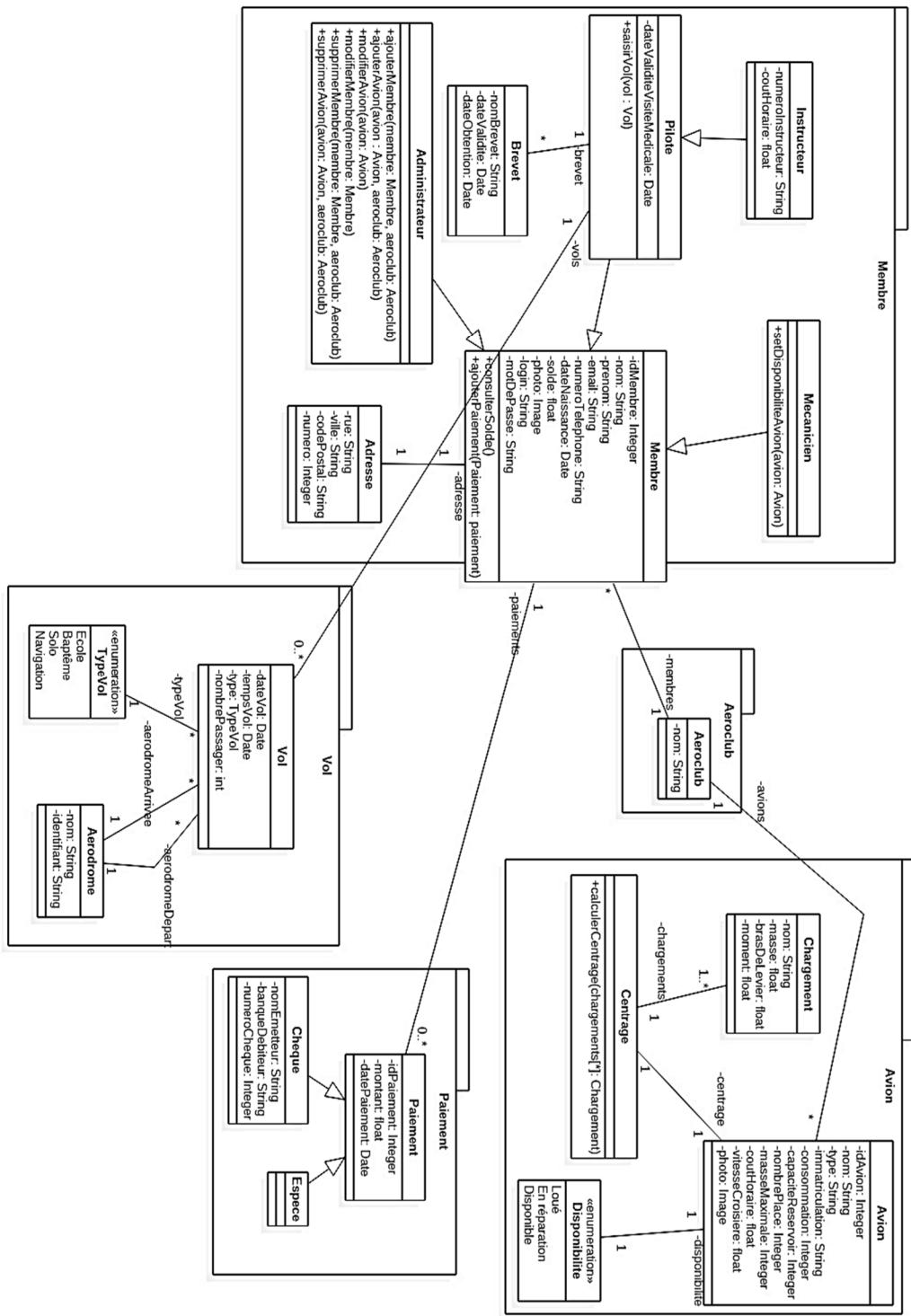


Figure 5:Le diagramme de classe "développeur"

3. La base de données

Nous avons réalisé la base de données à l'aide du diagramme de classe UML. Bien entendu, nous avons dû adapter certaines choses car le diagramme de classe UML n'est pas adapté pour faire de la base de données.

Voici le Modèle Physique de Données (MPD) que nous avons créé :

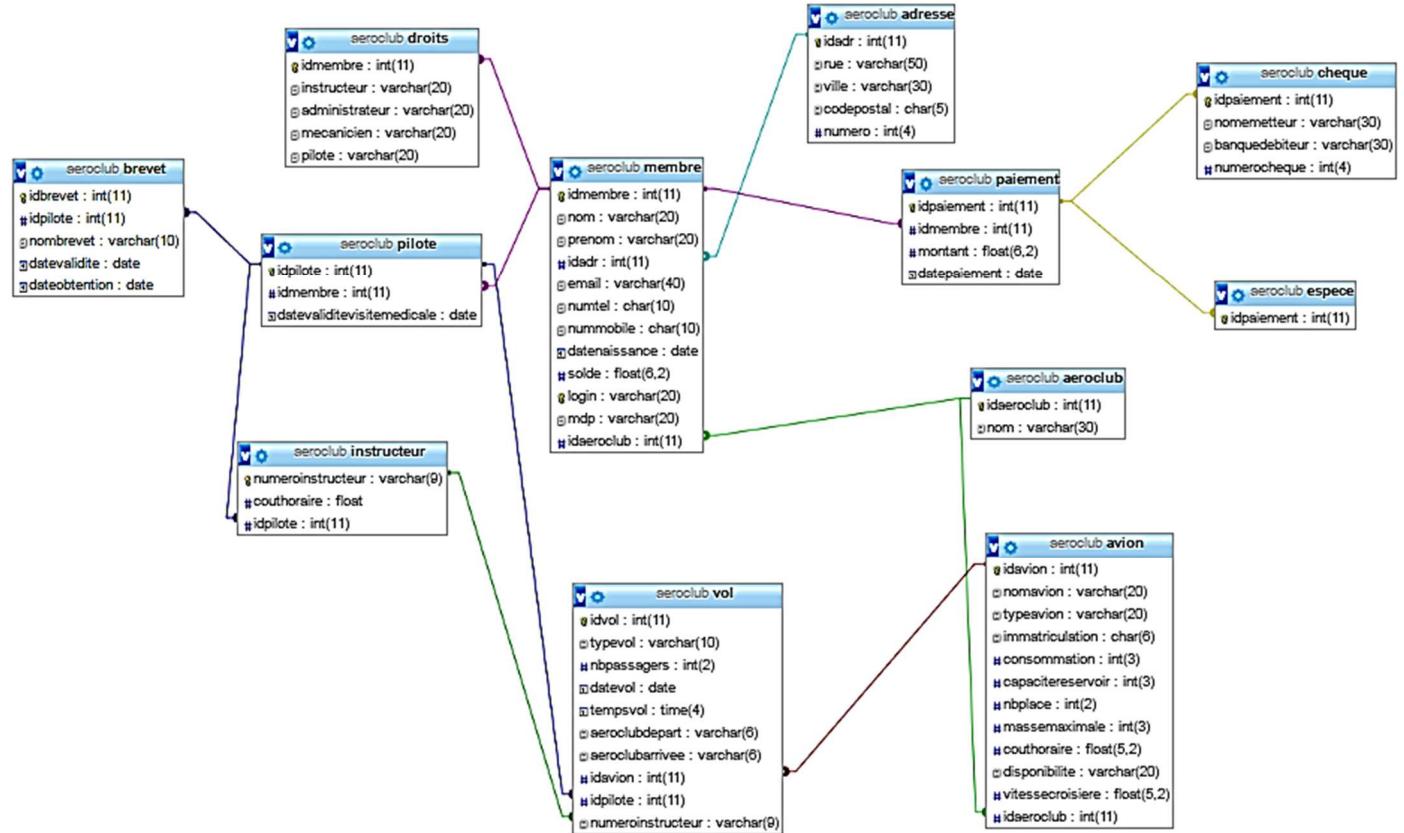


Figure 6: Le MPD de notre base de données

On remarque que la base de données ne contient pas de table « Mécanicien » ou « Administrateur ». En effet, nous n'avons aucune information à stocker sur ce type d'objets. Etre mécanicien ou administrateur permet d'accéder à des fonctionnalités de l'application, ces droits sont dans la table « droits ».

4. Les maquettes

Notre application disposant d'une interface graphique, nous avons dû élaborer les maquettes de ces interfaces.

Ces maquettes sont données en annexe.

5. Les outils utilisés

a. L'analyse UML

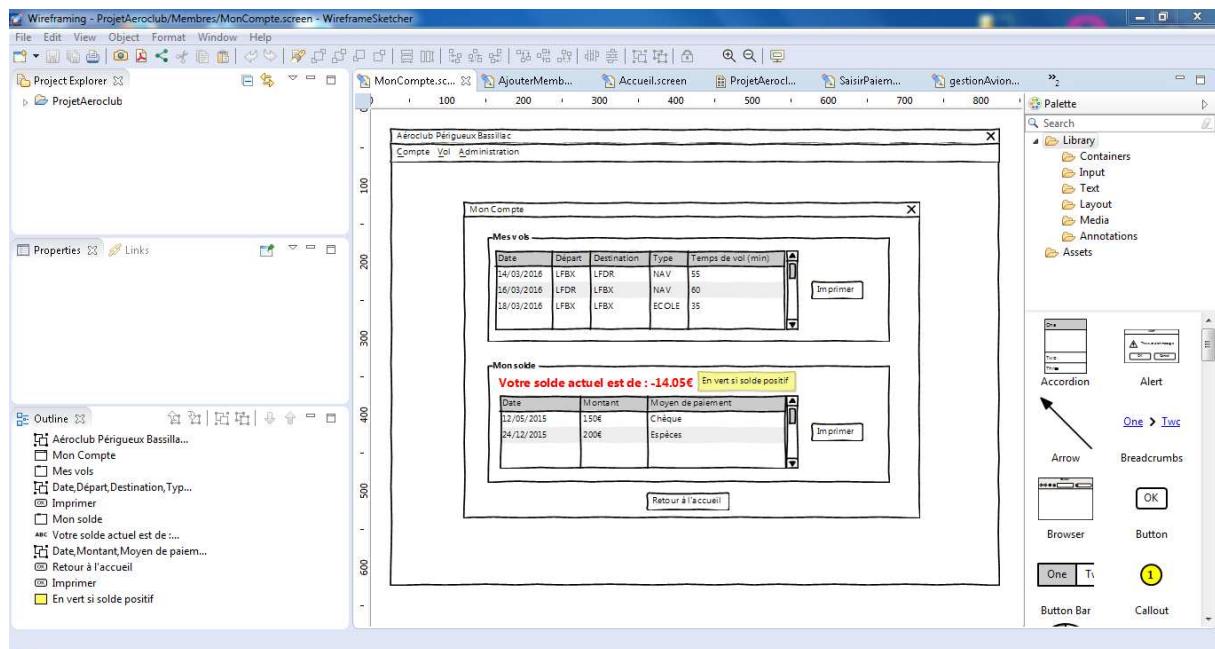
Pour l'analyse UML, nous avons choisi d'utiliser le logiciel StarUML qui permet de réaliser des diagrammes par « Drag'n'Drop ».



Figure 7:StarUML

b. Le maquettage de l'application

Afin de réaliser les maquettes de l'application, nous avons utilisé WireframeSketcher qui est un logiciel qui ressemble à Eclipse et permet de faire des maquettes par « Drag'n'Drop » très facilement.



c. Le langage de programmation

Pour le développement de notre application, nous avons choisi le langage JAVA qui est orienté objet, et que l'on a appris lors de notre DUT.



Figure 8: Java

Pour la réalisation des interfaces graphiques, nous avons utilisé le framework JavaFX et Scene Builder qui permet de réaliser les interfaces par « Drag'n'Drop ».

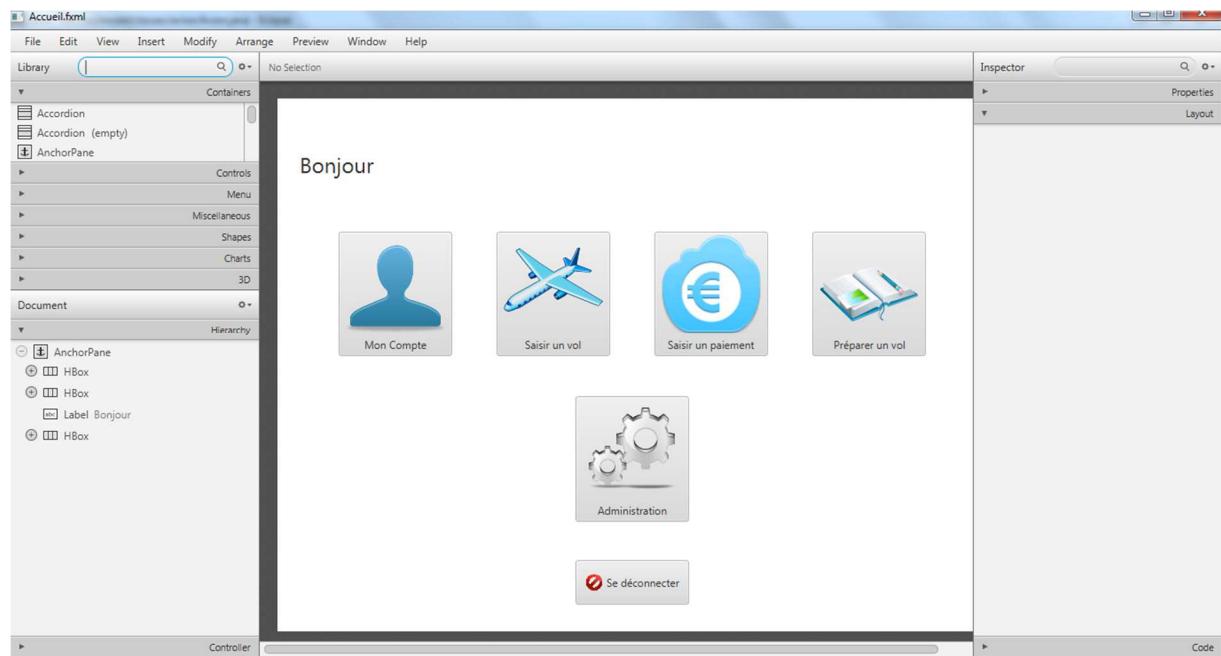


Figure 9:L'interface de la page d'accueil faite avec SceneBuilder

L'affichage réel :

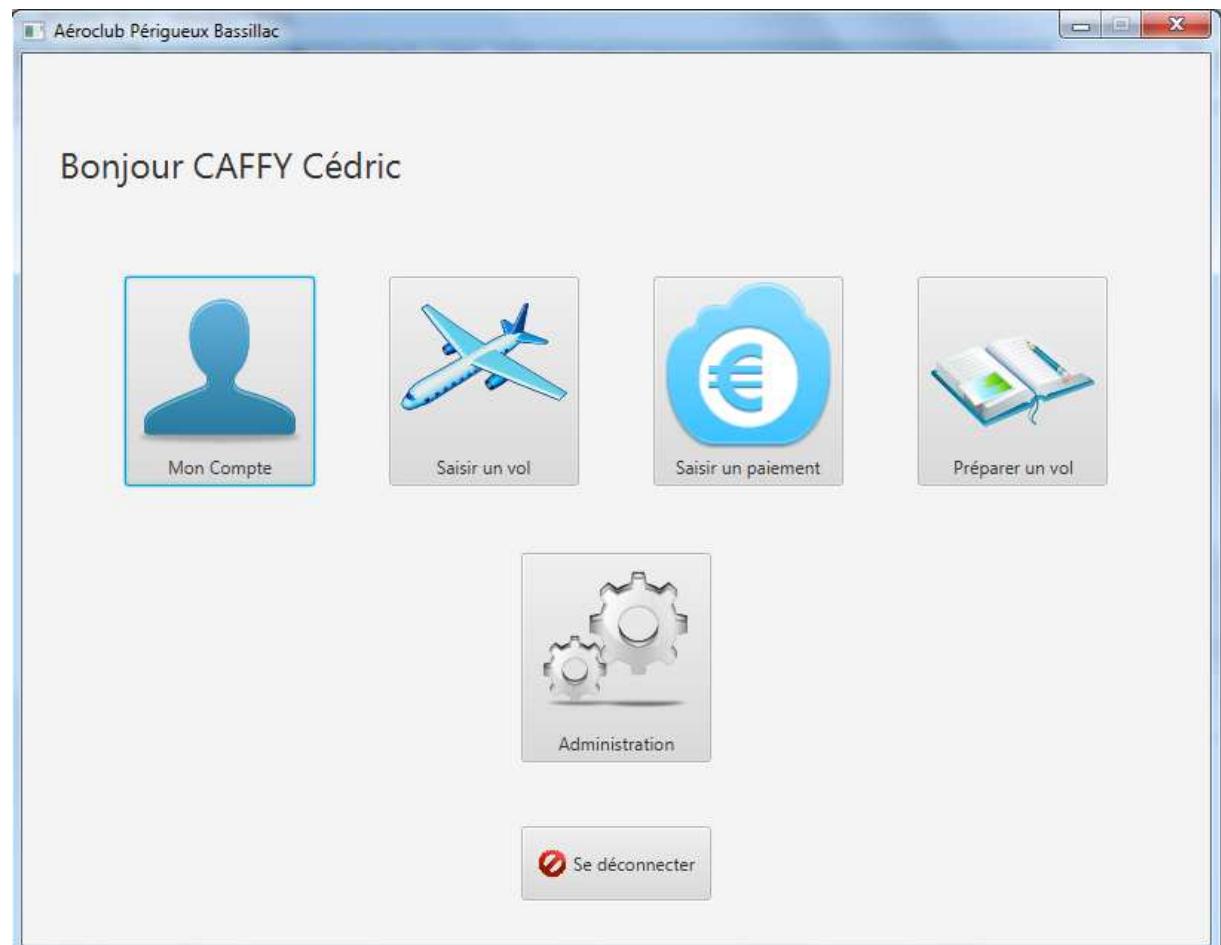


Figure 10 : L'interface réelle de la page d'accueil

d. La base de données

Nous avons choisi d'utiliser le SGBD (Système de Gestion de Base de Données) MySQL pour notre base de données.



Figure 11: MySQL

C'est un SGBD que nous avons déjà utilisé lors de précédents projets ce qui a facilité sa mise en œuvre.

Afin de consulter le contenu de notre base de données, nous avons utilisé l'application web PhpMyAdmin qui permet de gérer l'administration de la base de données de manière graphique et intuitive.

A screenshot of the PhpMyAdmin interface. The left sidebar shows a tree view of databases: 'Nouvelle base de données', 'aeroclub' (selected), 'information_schema', 'mysql', 'performance_schema', and 'sys'. Under 'aeroclub', there are tables: 'adresse', 'aeroclub', 'avion', 'brevet', 'cheque', 'droits', 'espece', 'instructeur', 'membre', 'Paiement', 'Pilote', and 'vol'. The main panel shows the 'Structure' tab for the 'aeroclub' table. The table structure is as follows:

Table	Action	Lignes	Type	Interclassement	Taille	Perte
adresse	★ Afficher Structure Rechercher Insérer Vider Supprimer	3	InnoDB	utf8_bin	16 Kio	-
aeroclub	★ Afficher Structure Rechercher Insérer Vider Supprimer	1	InnoDB	utf8_bin	16 Kio	-
avion	★ Afficher Structure Rechercher Insérer Vider Supprimer	3	InnoDB	utf8_bin	32 Kio	-
brevet	★ Afficher Structure Rechercher Insérer Vider Supprimer	5	InnoDB	utf8_bin	32 Kio	-
cheque	★ Afficher Structure Rechercher Insérer Vider Supprimer	1	InnoDB	utf8_bin	16 Kio	-
droits	★ Afficher Structure Rechercher Insérer Vider Supprimer	3	InnoDB	utf8_bin	16 Kio	-
espece	★ Afficher Structure Rechercher Insérer Vider Supprimer	4	InnoDB	utf8_bin	16 Kio	-
instructeur	★ Afficher Structure Rechercher Insérer Vider Supprimer	2	InnoDB	utf8_bin	32 Kio	-
membre	★ Afficher Structure Rechercher Insérer Vider Supprimer	3	InnoDB	utf8_bin	48 Kio	-
Paiement	★ Afficher Structure Rechercher Insérer Vider Supprimer	6	InnoDB	utf8_bin	32 Kio	-
Pilote	★ Afficher Structure Rechercher Insérer Vider Supprimer	3	InnoDB	utf8_bin	32 Kio	-
vol	★ Afficher Structure Rechercher Insérer Vider Supprimer	0	InnoDB	utf8_bin	64 Kio	-
12 tables Somme		34	MyISAM	latin1_swedish_ci	352 Kio	0 0

Figure 12: L'interface graphique de PhpMyAdmin

e. L'IDE (Integrated Development Environment)

L'IDE que nous avons choisi d'utiliser est Eclipse qui est un IDE très complet permettant de développer en Java rapidement (auto-complétion, debugger...).



Figure 13:Eclipse

f. Le gestionnaire de version

Il est arrivé que nous ayons à travailler simultanément sur le même projet, mais sur des machines différentes. Nous avons donc dû utiliser un logiciel qui permet de gérer facilement les versions de notre application.

Ce logiciel est GitHub.



Figure 14: GitHub

II. Le développement de notre application

Une fois la partie analyse terminée, nous avons pu commencer le développement de notre logiciel. Dans cette partie, nous allons détailler les techniques de programmations utilisées pour mettre en place certaines fonctionnalités de notre logiciel.

1. Le design pattern MVC

La bonne pratique pour le développement d'une application graphique est d'utiliser le design pattern MVC.

Ce design pattern permet de séparer de manière distincte les différentes couches de l'application : Model (Couche métier), View (Couche graphique), Controller (Fait le lien entre la couche graphique et la couche métier).

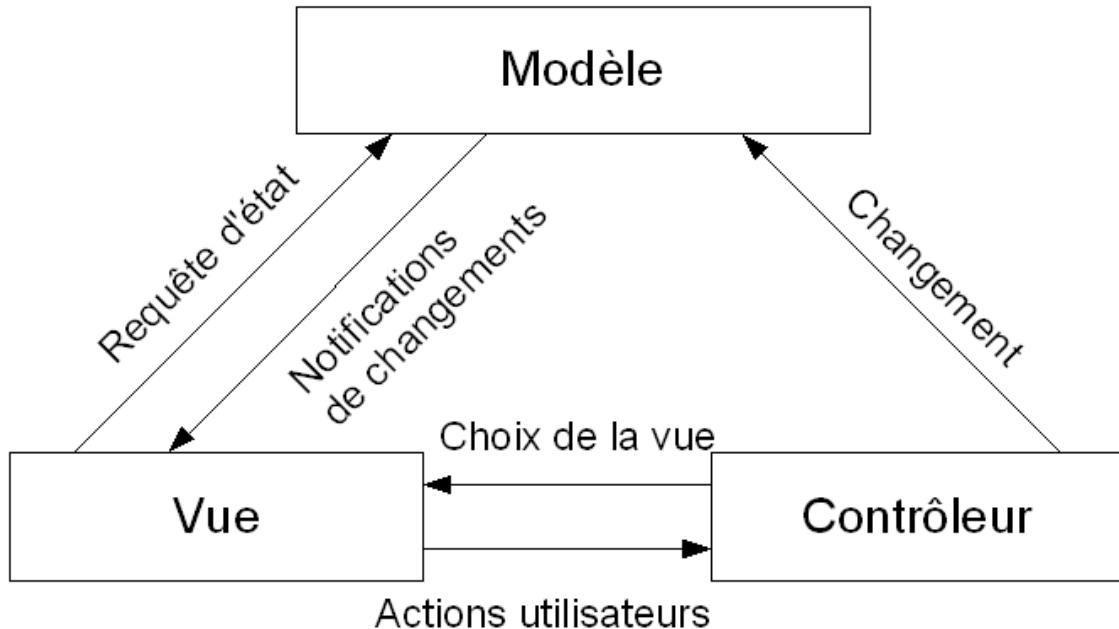


Figure 15: Le design pattern MVC

Après avoir réalisé l'interface graphique par « Drag'n'Drop » sur SceneBuilder, un fichier .fxml est généré. Il contient du code xml qui représente l'interface graphique réalisée. Un fichier fxml est créé pour chaque interface graphique. Ce fichier représente la couche « Vue » du design pattern MVC.

```
<ImageView fitHeight="114.0" fitWidth="112.0" pickOnBounds="true" preserveRatio="true">
    <image>
        <Image url="@images/iconeAvion.png" />
    </image>
</ImageView>
```

Figure 16: Exemple de code généré par SceneBuilder pour l'affichage d'une image

La couche « Controller » est réalisée par des classes java qui sont liées au fichier .fxml grâce à SceneBuilder.



Figure 17: Le lien entre la vue et le controller est fait via Scene Builder

La couche « Model » est simplement l'implémentation du diagramme de classe en java.

2. L'implémentation d'un controller

Le controller va permettre de lier l'interface graphique et la couche métier de l'application. Il va permettre de gérer ce que doit faire chaque élément de la fenêtre après un clic de l'utilisateur. Par exemple l'action qui va suivre le clic sur un bouton est réalisée par le controller.

Après avoir lié l'interface avec la classe du controller comme Figure 17, nous pouvons indiquer via l'interface graphique de Scene Builder quelle méthode le controller doit exécuter suite à un clic de souris sur un composant.

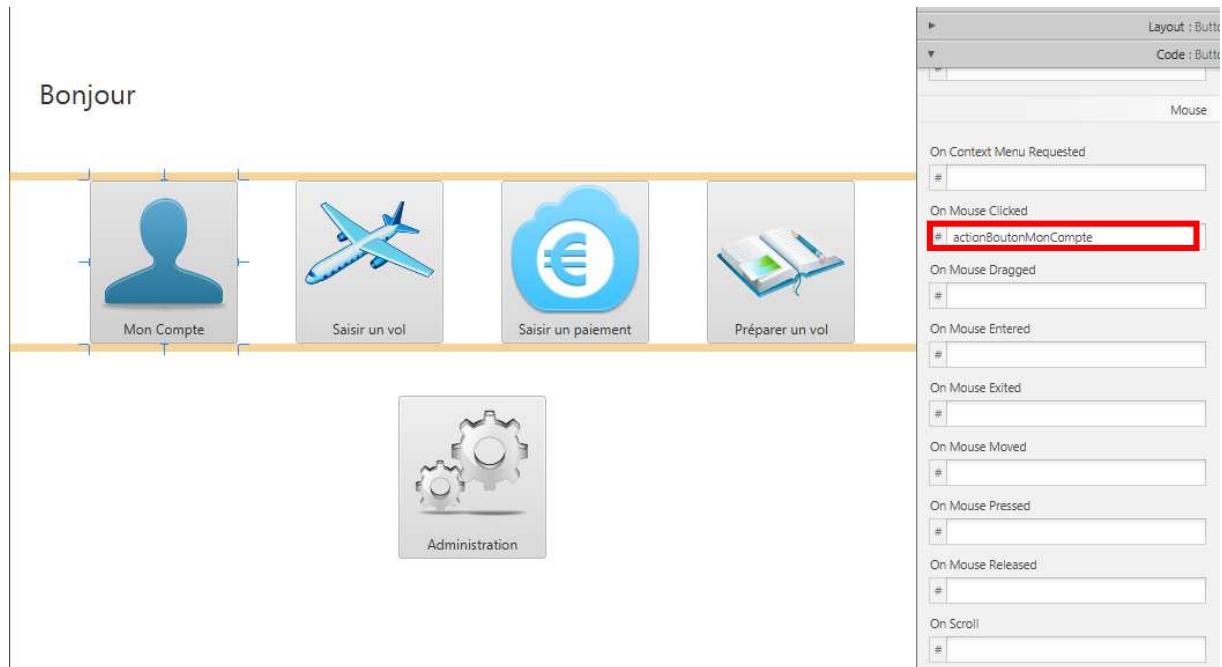


Figure 18: On indique l'action qui suit le clic du bouton mon compte

Voici l'implémentation de la méthode `actionBoutonMonCompte` qui sera exécutée au clic du bouton « Mon Compte » :

```
/*
 * Action qui suit le click sur le bouton Mon Compte
 */
@FXML /*Annotation @FXML pour dire au controller de lire dans le fichier
.fxml
private void actionBoutonMonCompte(){
    mainApp.afficherEcranMonCompte(this.membre);
}
```

Le controller va demander à l'application principale d'afficher l'écran correspondant à « Mon Compte ». Le fichier fxml de l'interface « Mon Compte » sera alors chargé et l'application affichera l'écran « Mon Compte ».

Chaque interface possède un seul et unique controller.

3. La gestion de la base de données en java

Afin de connecter la base de données à notre application java, il a fallu tout d'abord installer le driver JDBC adapté à MySQL.

Ce driver contient les bibliothèques qui permettent de réaliser toutes les interactions possibles avec la base de données.

a. Le design pattern Singleton

Nous voulons être sûrs qu'au cours de l'utilisation de l'application, une seule instance de la connexion à la base de données soit présente.

Ceci ne peut être fait qu'à l'aide du design pattern Singleton.

Le principe de ce design pattern est le suivant :

C'est une classe contenant une méthode qui permet d'instancier cette classe uniquement si elle n'est pas déjà instanciée.

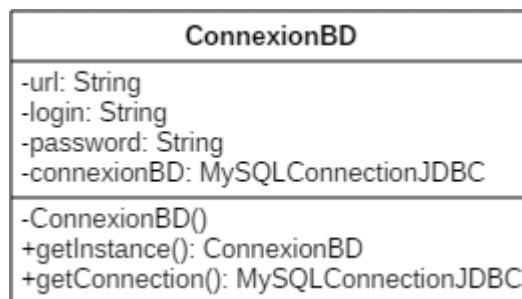


Figure 19: Le design pattern Singleton pour notre connexion à la base de données

La connexion à la base de données se fait selon le principe suivant :

- Lecture des informations de connexion depuis un fichier Propertie
- Chargement du driver de connexion
- Création de l'instance de la connexion

b. Le design pattern DAO

Ce design pattern permet de lier la couche métier de notre application à la base de données. C'est dans les classes qui composent ce design pattern que seront écrites les requêtes SQL.



Figure 20 : Représentation du design pattern DAO

Chaque DAO est développée selon le design pattern Adapter qui permet d'adapter les DAO selon le SGBD utilisé.

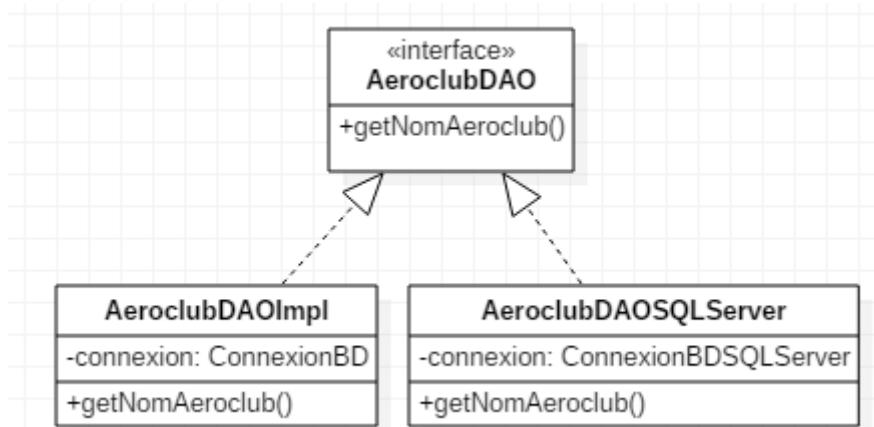


Figure 21 : Exemple illustrant le design pattern adapter

Voici le code contenant l'implémentation de AeroclubDAOImpl qui permet de faire des requêtes sur le SGBD MySQL :

```
public class AeroclubDAOImpl implements AeroclubDAO{
    private ConnexionBD connexion;
    private static String requeteGetNomAeroClub="SELECT nom FROM Aeroclub";

    /**
     * Constructeur
     * @param connexion la connexion a la BDD
     */
    public AeroclubDAOImpl(ConnexionBD connexion){
        this.connexion=connexion;
    }

    /**
     * Recupere le nom de l'aeroclub
     */
    @Override
    public String getNomAeroclub() throws DAOException{
        Connection connexion=null; //Contient la connexion a la BDD
        PreparedStatement preparedStatement=null; //contiendra la requete
        preparee SQL
        ResultSet resultSet=null; //Contient le resultat de la requete
        String nomAeroclub=null; // Contient le nom de l'aeroclub
        try{
            connexion=this.connexion.getConnexion(); //Connexion a la BDD
            //Initialisation de la requete preparee

            preparedStatement=DAOUtilitaire.initialiserRequetePreparee(connexion,AeroclubDAOImpl.requeteGetNomAeroClub,true,(Object)null);
            //Execution de la requete
            resultSet=preparedStatement.executeQuery();
            if(resultSet.next()){
                //Resultat trouve
                nomAeroclub=resultSet.getString("nom");
            }
        }catch(SQLException e){
            throw new DAOException(e);
        }finally{
            //Fermeture propre de la connexion, du resultSet et du
            preparedStatement
        }
        DAOUtilitaire.fermeturesSilencieuses(resultSet,preparedStatement,connexion);
    }
    return nomAeroclub;
}
}
```

c. Les requêtes préparées

Afin d'éviter toute injection SQL d'un utilisateur malveillant, nous avons choisi d'utiliser ce qu'on appelle des requêtes préparées.

Exemple d'une requête préparée :

« INSERT INTO PAIEMENT (idmembre,montant,datepaiement) VALUES (?, ?, ?) »

Les points d'interrogation seront remplacés par leur valeurs plus tard après vérifications de ces valeurs.

Pour cela, nous avons créé une méthode statique dans une classe DAOUtilitaire qui permet d'initialiser une requête préparée :

```
public static PreparedStatement initialiserRequetePreparee(Connection connexion,
String sql, boolean returnGeneratedKeys, Object... objets ) throws SQLException
{
    PreparedStatement preparedStatement = connexion.prepareStatement(
sql, returnGeneratedKeys ? Statement.RETURN_GENERATED_KEYS :
Statement.NO_GENERATED_KEYS );
    if(objets[0]!=null){
        for ( int i = 0; i < objets.length; i++ ) {
            //Remplacement des points d'interrogations par les objets
            //passés en paramètres
            preparedStatement.setObject( i + 1, objets[i] );
        }
    }
    return preparedStatement;
}
```

Cette méthode a un nombre de paramètres variable.

Les paramètres de cette requête sont la connexion, la requête SQL, un booléen permettant de savoir si on souhaite retourner une valeur telle que l'identifiant du dernier élément ajouté dans la BDD, et un paramètre objets qui va contenir les objets que l'on va mettre à la place des points d'interrogation de la requête préparée.

La méthode retourne le preparedStatement qui est l'objet qui va permettre d'exécuter la requête préparée.

d. La gestion des exceptions

Il est possible que des erreurs de connexion ou de résultat de requête surviennent. En effet, si le SGBD MySQL n'est pas démarré, aucune requête ne peut être exécutée.

Pour gérer ce type d'erreurs nous avons créé des classes d'exceptions :

- DAOConfigurationException qui survient lorsque le fichier Propertie n'a pas pu être lu ou lorsque le driver de connexion est introuvable
- SQLException qui survient lorsqu'il y'a un problème de requête SQL ou de connexion à la base de données.

III. Le projet

1. L'avancement du projet

Le projet est quasiment fini.

Toute la partie administration c'est-à-dire l'ajout, la suppression et la modification d'un membre ou d'un avion fonctionne.

La gestion des droits de l'application fonctionne également : Seul un administrateur peut avoir accès à l'interface d'administration.

La saisie d'un paiement est également fonctionnelle, l'utilisateur peut saisir un paiement par chèque ou par espèce.

L'implémentation de la page « Mon Compte » est quasiment terminée, il reste à calculer le solde du membre en fonction des heures de vol effectuées et des paiements versés.

La saisie d'un vol est terminée, l'utilisateur peut saisir un vol avec ou sans instructeur et saisir les différentes étapes de son vol sur la même interface.

Il reste deux fonctionnalités à implémenter : l'ajout des brevets pour un pilote et le module d'aide à la préparation d'un vol.

2. Les difficultés rencontrées

a. JavaFX

JavaFX est un framework puissant pour faire de l'interface graphique. Cependant, il faut quand même l'avoir pratiqué avant d'être à l'aise et développer rapidement. Il nous a donc fallu suivre des tutoriels et faire des applications « basiques » afin de comprendre le fonctionnement de JavaFX.

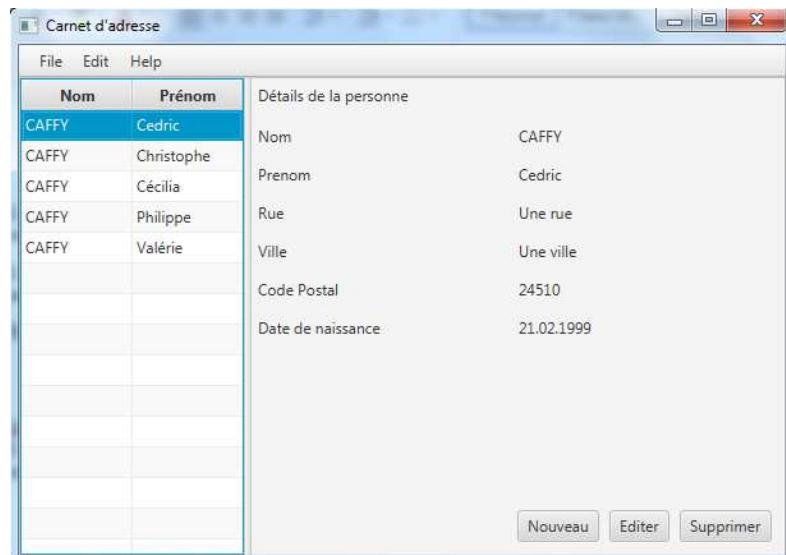


Figure 22 : Une application réalisée pour se former en JavaFX

Ce temps de formation n'a pas été négligeable, nous ne pouvions pas commencer le projet sans avoir appris à utiliser JavaFX.

b. Les bonnes pratiques de la programmation

En DUT, nous avons appris les bases de beaucoup de langages de programmation. Les bases ne permettent pas de développer de manière « propre » une application. En effet, l'utilisation du design pattern MVC ou encore DAO n'était pas une habitude car nous n'avions pas le temps de faire de gros projets encadrés.

Il a donc fallu nous renseigner sur des sites fiables tels que OpenClassrooms qui montrent quelques bonnes pratiques de la programmation orientée objet.

c. Le travail en équipe

Nous avions tous les deux eu des rythmes de travail différents et il était difficile de travailler en même temps sur le projet.

i. *Les tâches effectuées par chacun*

Cédric

- Cahier des charges + Analyse complète UML (diagramme de Use Case, diagramme de classe, maquettes)
- Implémentation des classes métiers
- Réalisation de toutes les interfaces avec Scene Builder + controllers + tests avec valeurs entrées en dur dans l'application
- Création des popups de confirmation et d'erreurs
- Connexion à la base de données (Design pattern Singleton + installation driver JDBC) + gestion des exceptions + fichier Propertie de connexion
- Design pattern DAO « de base »
- Gestion de la connexion d'un membre
- Remplissage des combobox de l'interface de saisie d'un vol et gestion du contenu de ces combobox en fonction des choix de l'utilisateur
- Implémentation de l'interface de saisie de paiement (contrôles de saisies + insertion dans la base de données)

Maxime

- Implémentation de la base de données (Création des tables)
- Réalisation des triggers de mise à jour du solde du membre
- Réalisation de la partie administration :
 - Ajout, modification, suppression d'un membre dans la base de données + DAO correspondantes + contrôles de saisie
 - Ajout, modification, suppression d'un avion + DAO correspondantes + contrôles de saisie
- Finition de l'interface de saisie d'un vol : saisie des étapes, contrôles de saisie et insertion dans la base de données.
- Gestion des droits de l'application : Ajouts dans la base de données et restrictions d'accès aux fonctionnalités de l'application

Conclusion

Notre application sera sans doute utilisée par de petits aéroclub qui n'ont pas les moyens de se payer une application « professionnelle » et qui utilisent encore le papier pour la saisie des retours de vol, des paiements...

Ce projet nous a été bénéfique car il a permis l'apprentissage / le perfectionnement d'une technologie de développement (Java/JavaFX). Il nous a également permis de nous rendre compte qu'un petit projet comme celui-là demande beaucoup de temps et d'implication.

Le logiciel devra sûrement être modifié afin de pouvoir correspondre à 100% aux exigences des aéroclubs (Un module de gestion de la comptabilité de l'aéroclub pourrait être demandé par exemple).



Window



Connexion

Login :

cecaffy

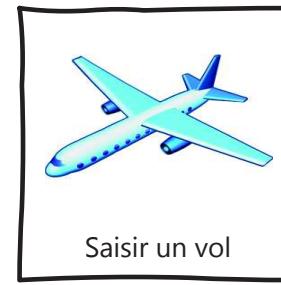
Mot de passe :

Valider

Quitter



Accueil





Mon Compte



Mes vols

Date	Départ	Destination	Type	Temps de vol (min)
14/03/2016	LFBX	LFDR	NAV	55
16/03/2016	LFDR	LFBX	NAV	60
18/03/2016	LFBX	LFBX	ECOLE	35

[Imprimer](#)

Mon solde

Votre solde actuel est de : -14.05€

En vert si solde positif

Date	Montant	Moyen de paiement
12/05/2015	150€	Chèque
24/12/2015	200€	Espèces

[Imprimer](#)[Retour à l'accueil](#)



Accueil



Mon Compte



Saisir un vol



Saisir un paiement



Préparer un vol



Administration

Saisir un vol

**Avion**

F-HPGX - Cessna 172 ▼

Date

06/03/2016 □

Type de vol

Instruction ▼

Instructeur

Jean Michel DELPECH ▼

Nombre de passagers

1 ▼

Etapes

Aérodrome de départ	Aérodrome d'arrivée	Temps de vol (min)
LFBX	LFDS	35
LFDS	LFBX	40

Ajouter Etape

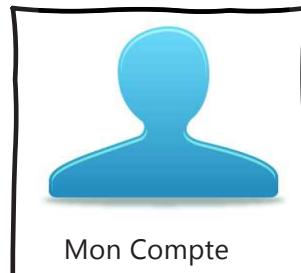
Supprimer Etape

Enregistrer

Annuler



Accueil



Mon Compte



Saisir un vol



Saisir un paiement



Préparer un vol



Administration

Saisir un paiement X

Pilote : Cédric CAFFY

Type de paiement Date : 06/03/2016

Chèque ▼

Montant Banque

250,00€ Crédit Agricole

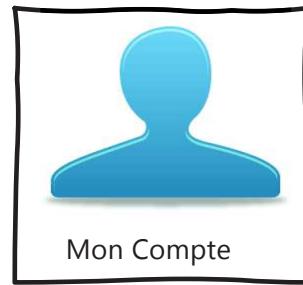
Numéro de chèque

00000254

Enregistrer Paiement Annuler

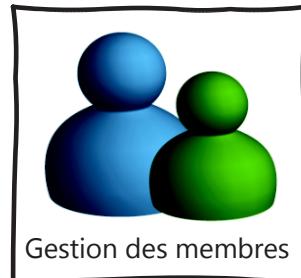


Accueil





Administration



Gestion des membres



Gestion des avions

Gestion des membres

X

Nom	Prénom	Tél	Email
CAFFY	Cédric	06.07.25.25.36	cedric.caffy@etu.isima.fr
AUZANNEAU	Maxime	06.56.69.98.65	maxime.auzanneau@coucou.fr
CHABROL	Michèle	06.07.08.09.10	chabrol@isima.fr
BOUCHALOIS	Florian	06.96.56.54.51	flbouchalo1@isima.fr

Ajouter un
Membre

Editer un
Membre

Supprimer un
membre

Revenir à l'accueil



Ajout d'un membre



Informations personnelles

Nom : AUZANNEAU Prénom : Maxime

Adresse : Chez Maxime 63000 Clermont Ferrand

Email : maxime.auzanneau@coucou.fr

Téléphone : 05.55.56.57.58 Mobile : 06.07.08.09.10

Date de naissance : 31/12/1994

Brevets et visites médicales

Type Brevet	Date d'obtention	Date de validité
PPL(A)	06/03/2016	06/03/2018

 Ajouter brevet Supprimer brevet Instructeur

Numéro d'instructeur : FI-0000001

 Administrateur Mécanicien Enregistrer le membre Annuler

Gestion des membres

X

Nom	Prénom	Tél	Email
CAFFY	Cédric	06.07.25.25.36	cedric.caffy@etu.isima.fr
AUZANNEAU	Maxime	06.56.69.98.65	maxime.auzanneau@coucou.fr
CHABROL	Michèle	06.07.08.09.10	chabrol@isima.fr
BOUCHALOIS	Florian	06.96.56.54.51	flbouchalo1@isima.fr

Ajouter un
Membre

Editer un
Membre

Supprimer un
membre

Revenir à l'accueil



Modification d'un membre

**Informations personnelles**

Nom : AUZANNEAU Prénom : Maxime

Adresse : Chez Maxime 63000 Clermont Ferrand

Email : maxime.auzanneau@coucou.fr

Téléphone : 05.55.56.57.58 Mobile : 06.07.08.09.10

Date de naissance : 31/12/1994

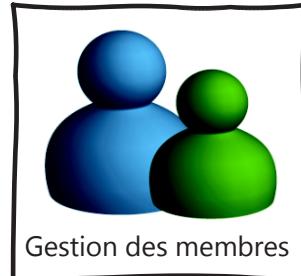
Brevets et visites médicales

Type Brevet	Date d'obtention	Date de validité
PPL(A)	06/03/2016	06/03/2018

 Ajouter brevet Supprimer brevet Instructeur Numéro d'instructeur : FI-0000001 Administrateur Mécanicien Modifier le membre Annuler



Administration



Gestion des membres



Gestion des avions

Gestion des avions

X

Immatriculation	Nom
F-GCNP	Cessna 152
F-HPGX	Cessna 182
F-GIKO	DR-400

[Ajouter un Avion](#)[Editer un Avion](#)[Supprimer un Avion](#)[Revenir à l'accueil](#)

Ajouter un avion



Informations générales

Nom : DR-400

Immatriculation : F-GLDG

Type : Voyage

Coût horaire : 145€

Nombre de places : 4



Ajouter photo

Navigation

Capacité de réservoir (L) : 110

Masse maximale (kg) : 1100

Vitesse de croisière (kt) : 100

Autonomie (l/h) : 25

Informations de centrage

Enregistrer l'avion

Annuler

Gestion des avions

X

Immatriculation	Nom
F-GCNP	Cessna 152
F-HPGX	Cessna 182
F-GIKO	DR-400

Ajouter un
Avion

Editer un
Avion

Supprimer un
Avion

Revenir à l'accueil

Modifier un avion



Informations générales

Nom : DR-400

Immatriculation : F-GLDG

Type : Voyage

Coût horaire : 145€

Nombre de places : 4



Ajouter photo

Navigation

Capacité de réservoir (L) : 110

Masse maximale (kg) : 1100

Vitesse de croisière (kt) : 100

Autonomie (l/h) : 25

Informations de centrage

Disponibilité

 Disponible En réparation

Modifier l'avion

Annuler

Ajouter un avion



Informations de centrage



Intitulé

Nom

Type

Nom

Nom

Intitulé	Masse(kg)	Bras de levier	Moment (m x kg)
Avion vide	570	0.297	169.90
Commandant de bord	75	0.410	30.750
Copilote	80	0.410	32.8

Ajouter ligne

Supprimer ligne

Enregistrer

Annuler

Enregistrer

Annuler



Accueil

