Page

Guide de Maintenance

DUSZYNSKI Laszlo, HIDA Nordine,
NIEF Yamato, PRUNIER Lucas,
BASSET Stéphane
C2

SOMMAIRE

1 - PARTIE CLIENT

I - Structure de l'application PAGE

Introduction

La structure globale de l'application peut être décomposée en trois parties principales conformément à l'architecture modèle-vue-contrôleur (MVC) : le Model, le Stockage, et la Vue. Ces parties sont détaillées ultérieurement.

Vous pouvez retrouver toute la documentation dans le dossier « Documentation/Doc PAGE » !

A - Model

Le modèle représente la logique métier de l'application, notamment la manière dont les données sont manipulées et traitées. Il comprend les classes et les fonctionnalités nécessaires pour gérer les étudiants, les notes ou encore les paramètres de l'application ...

B - Stockage

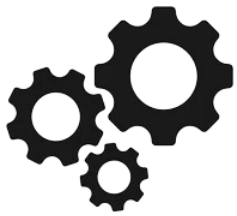
La partie stockage gère la persistance des données, c'est-à-dire comment les données sont sauvegardées et récupérées. Dans ce contexte, une base de données relationnelle SQL en lien avec le SGBD Oracle à été utilisé. Cette partie comprends les différents DAOs qui vont appeler l'API afin de récupérer ou sauver les étudiants, leurs notes, les promotions/années ou encore les utilisateurs.

De plus, un répertoire « StockageNoteConfidentielle » gère la BDD SQLite qui n'est pas encore utilisé dans l'application. Elle a pour but de sauver localement les informations médicales des étudiants qui ne peuvent être stocker sur la BDD Oracle (RGPD).

C - Vue

La partie vue regroupe deux parties liées à l'affichage de l'application : La partie « Ecran » regroupant l'ensemble des fenêtres WPF et la partie « Ressources » qui comprends les polices d'écriture, les images/pictogrammes et dans le dossier « Res » les ressources textuels (les textes de l'application en français et anglais).

II - Model et Logique



A - Classes générales

Les classes générales sont les classes représentant des éléments concrets du projet. On retrouve les classes :

1. Etudiant
2. Note
3. Promotion
4. Année
5. Utilisateur
6. Promotion

Ces classes regroupe toutes les informations disponibles sur ces éléments (le nom est explicite) et s'il faut par exemple modifier un type, rajouter un nouveau champ ou autre, c'est vers ces classes qui faut se diriger.

B - Classes techniques

Les classes techniques sont les classes qui représentes une fonction précisent dans l'application. On retrouve les classes :

I. Parametre

Comprend le chemin de génération du Word et la langue de l'application. S'il faut ajouter des paramètres c'est dans cette classe qui faut les ajouter. Vous pouvez ajouter la balise [DataMember] sur les propriétés que vous voulez sauvegarder.
C'est un **Singleton** ce qui permet d'avoir une seule instance utilisable dans toutes l'application (notamment pour savoir la langue utilisé).

II. JsonSerializerParametre

Permet de sérialiser les paramètres de l'application (qui on la balise [DataMember]). Les paramètres sont sérialisés en JSON dans le fichier « settings.json » (nom et chemin modifiable dans le constructeur).

Il **initialise** les paramètres (Langue → **Français** et path du Word → **Bureau)**. C'est modifiable dans la méthode « InitialiserParametre ».

III. LecteurExcel

Permet de lire le Excel avec la liste des étudiants lors de **l'importation** d'étudiants. Il utilise le nugget DocumentFormat.OpenXml (version 2.20.0) La lecture est brute, c'est-à-dire que la signification de chaque colonne est attribuée à la main en respectant l'exemple fournis par le gestionnaire

d'étudiant (OGE). Si le **format du Excel change** il faut modifier les « cell.CellReference » de la méthode **GetEtudiants**.

IV. Token

Le Token permet l'authentification d'un utilisateur avec son rôle. Il est généré à chaque reconnexion de l'utilisateur et est passé a chaque fenêtre afin de transmettre le rôle de l'utilisateur en cours.

Il devrait avoir un délai d'expiration de 10min et être rafraîchis à chaque action de l'utilisateur (pour éviter de se reconnecter toutes les 10min) **mais ce n'est pas encore implémenter**. Pour l'instant les token **n'expire donc pas** et il faudrait gérer leur suppression dans la partie stockage (voir III-A et III-B)

V. WordGenerateur

Cette classe permet de générer le Word avec toutes les notes des étudiants de la promotion en cours. Il utilise le nugget FreeSpire.Doc (version 11.6.0).

Si vous souhaitez **modifier le rendu du Word**, vous pouvez depuis la méthode **GenererWord** de la classe.
Pour **changer le nom du Word** il vous suffit de modifier la dernière ligne « doc.SaveToFile(CheminDuWord/NomQueVousVoulez) »

C - Classes du Patron Observateur

Nous utilisons le patron Observateur pour notifier les changements de la logique vers l'IHM (Vue).

Si une classe doit **prévenir** un observateur, elle doit **dériver** de la classe abstraite **Observable**.
Si une classe doit **observer** un Observable, elle doit implémenter là l'interface **IObservateur**.

« Notes », « Utilisateurs », « Etudiants » sont des exemples d'**Observable**. Ils permettent de notifier quand une modification est survenue dans une note, un utilisateur ou un étudiant.

Les classes qui implémentent **IObservateur** sont des **fenêtres** (voir IV – A).

D – Énumérations

Le dossier « Enumerations » regroupe toutes les **énumérations** utilisées dans l'application. Elles permettent d'éviter les chaînes de caractère ou autres types sources d'erreurs. Si vous souhaitez **rajouter** (ou retirer) des **nouvelles** catégories, confidentialité, de nouveau rôle ou encore une nouvelle langue, vous pouvez les **ajoutez** dans les **énumérations** correspondantes.

III – Stockage

A - Interface de DAO

Pour pérenniser les informations de l'application, nous utilisons des DAO (*data access object).* Afin de faciliter le **changement éventuel de moyen de stockage**, nous avons créer des interfaces de DAO pour chaque objet sauvable.
Vous pouvez facilement les retrouver grâce au préfixe « **I** » pour « **Interface** ».
Ces interfaces ont pour but de **regrouper les méthodes nécessaires** pour avoir un **DAO fonctionnel**. Si vous voulez changer de mode de stockage il suffit de créer un nouveau ObjetvouluDAO implémentant l'interface.

Par exemple INoteDAO regroupe les méthodes nécessaires pour gérer la pérennisation des notes, avec les méthodes de CRUD (pour créer, récupérer, modifier et supprimer une note).

B - Implémentation de DAO

Les DAO sont des classes qui implémentes les classes des interfaces. Pour l'instant l'application utilise une API qui va faire le lien entre PAGE et la base de données.

L'API est **hébergée en local** (sur le poste). Si vous souhaitez **l'héberger sur un serveur** par exemple il est **nécessaire** de changer les « string apiURL » **afin qu'il corresponde a l'url de votre API.**
(Attention certaine URL intégré directement les paramètres nécessaires, veillez à les réintégrer à vos nouvelles URL)

C - Base de Données SQLite

Dans le dossier « StockageNoteConfidentielle » vous retrouverez un début d'implémentation de base de données SQLite qui a pour but **d'héberger localement les notes à caractère médical ou confidentiel** qui ne peuvent être héberger sur un serveur distant (cf RGPD).
Il faut créer cette **BDD local sur le poste de l'utilisateur** pour qu'il puisse garder en mémoire ces notes confidentielles.
Pour l'instant **toutes** les notes passent par **les DAO** pour rejoindre la BDD Oracle via l'API.

IV - Vue

A – Ecran

1 – Fenêtre classique

Toutes les fenêtres sont en **xaml**. Elles gèrent l'affichage des informations récupérer via la couche Stockage (voir III - Stockage).

Pour structurer les pages, des **Grids** sont utilisées mais elles ne permettent pas de gérer l'adaptation des différentes résolution (**Pas responsive**). Il faudrait appliquer un ratio sur les fenêtres afin qu'elle puisse conserver un aspect similaire sur tous les écrans.

Toutes les fenêtres sont affichées avec un « **ShowDialog** » ce qui permet de bloquer les autres fenêtres tant que la fenêtre en cours n'est pas fermée. Vous pouvez changer cette fonctionnalité en remplacent par des simples « **Show** ».

2 – Composant de fenêtre

Les fenêtres qui affichent **dynamiquement** des éléments (« FenetrePrincipal », « GestionUtilisateur », « InformationsSupplementaires ») utilisent des **UserComponent** qui permettent d'afficher les étudiants, utilisateurs ou encore les notes, dans des composants plus esthétiques et personnalisables.
Si vous souhaitez **modifier** l'affichage de ces derniers, c'est vers les fenêtres avec le suffixe « **Component.xaml** » qu'il faut se diriger.

3 – Pop-Up

Pour **valider** ou indiquer une **erreur** pendant les manipulations de l'utilisateur, des pop-ups apparaissent avec des **messages personnalisés**. Si vous souhaiter modifier leurs formats, allez voir la classe « **PopUp.xaml** ».

Pour créer un pop-up personnalise vous devez simplement lui préciser le **titre**, le **message** que vous souhaitez, ainsi qu'utiliser l'énumération « **TYPEICON** » qui rassemble les différents pictogrammes affichable (erreur, succès, information par exemple).

Vous pouvez aussi en ajouter en modifier **l'énumération** et en ajouter votre Picto en **ressources** (voir 2- Images et Pictogrammes).

B - Ressources Générales

1 - Police d'écriture (Fonts)

Toutes l'application respecte la c**harte graphique** lié au logo de l'IUT de Dijon. Elle utilise donc la police « **Open Sauce** One » et toutes ces déclinaisons (Bold, Italic ...). Vous pouvez retrouvez toutes ces polices dans le dossier « **Fonts** ».
Pour les utiliser il suffit de l'ajouter en ressource de votre fenêtre comme ceci :

```
    <Window.Resources>
        <FontFamily x:Key="OpenSauceOne">/Vue/Ressources/Fonts/#Open Sauce
One</FontFamily>
    </Window.Resources>
```

Et de l'appliquer comme **ressource** sur votre texte :

```
FontFamily="{StaticResource OpenSauceOne}"
```

Libre à vous d'en ajouter de nouvelles en respectant cette manipulation.

2 - Images et Pictogrammes

Vous pouvez retrouver toutes les images de l'application, notamment les notes (dans le dossier « **Notes** ») dans le dossier « **Img** ». Veillez à bien les ajouter comme ressources (ou contenu selon l'usage que vous en faites).

Les **images des notes** sont chargées par la classe « **RessourceManager**.cs » dans le dossier Ressources. Cette classe devrait à terme charger **toutes les images nécessaires** afin d'éviter de surcharger l'application et de fluidifier les affichages.
Vous pouvez donc ajoutez toutes les images que vous souhaitez charger en les ajoutant au dictionnaire « Images » qui associe une image au nom que vous définissez.

3 - Langue et ressources textuelles (Res)

L'application est disponible en anglais et en français. Vous trouverez les ressources textuelles dans le dossier « Ressources/Res ». Vous y trouverez deux fichiers : « StringResources.fr.xaml » pour le français et « StringResources.en.xaml » pour l'anglais. C'est fichier contiennes tous les texte/titre/label de l'application ordonnée par fenêtre où ils sont utilisés.

Si vous souhaitez modifier un texte il vous suffit de trouver la « Key » qui fait référence au texte souhaité et de le modifier dans **les deux** fichiers.

Pour utiliser une ressource textuelle vous pouvez simplement utiliser :

```
Content="{DynamicResource KeyDeVotrTexte}"
```

Si vous souhaitez ajouter une langue, il faut donc créer un nouveau fichier ressource sur un des exemples déjà présents en veillant d'avoir bien **toutes les keys** ET ajouter votre ficher ressources dans la méthode « **Notifier** » de la classe « **App.xaml.cs** »

2 – PARTIE SERVEUR

Vous pouvez retrouver toute la documentation dans le dossier « Documentation/ Doc API-PAGE » !

Structure de l'API

La structure globale de l'API peut être décomposée en trois parties principales : Model, Contrôleurs, Stockage. (voir Les packages de l'API)

A - Model

I. Classes Générales

Les classes « générales » du model sont les mêmes que du coté client. Si le model coté client **change**, il faut **aussi appliquer les modifications du coté serveur**, et inversement.

Pour plus d'informations voir Partie 1 II-Model A-Classe General.
Pour la conception voir Conception Model

II. La « Connection » à la base de données

La classe « **Connection** » a pour but d'ouvrir et fournir des connexions à la base de données Oracle.
Si vous souhaitez **changer** de base de données il faut **impérativement changer le « connexionString » de cette classe** qui présente les identifiants de connexion à la base de données **!**

De plus, il **serait préférable de ne plus utiliser ce string de connexion en dur**, mais de l'importer depuis un fichier protéger extérieure.

III. Les Managers

Les Manager font **le lien** entre les « Controller » et les « DAOOracle ».
Si vous souhaitez **changer d'implémentation de DAO**, il vous suffit de changer l'initialisation de l'attribut privé <classe>DAO.

Par exemple :

```
private IEtuDAO EtuDAO => new EtudiantDAOOracle();
```

Deviendrait

```
private IEtuDAO EtuDAO => new EtudiantDAOCEQUEVOUSVOULEZ();
```

Il suffit que votre nouveau DAO implémente correctement l'interface (voir C-Stockage)

B - Controller

Il existe **4 Controllers** pour chaque **classe du model** : Année, Etu, Note et Utilisateur. Ils reçoivent les requête GET et POST des DAO du client (voir stockage partie Client) et vont transmettre la demande aux différents managers associés.

Les requêtes servent essentiellement au **CRUD** des différents classes métier, mais vous pouvez en ajouter dans les bons controllers à votre guise en veillant à implémenter les méthodes qui en découle dans les **managers**.

Dans le cas où vous **voudriez envoyer une « Promotion » comme paramètre**, vous devez **décomposer cette dernière en deux entiers** (anneeDebut, et typeBUT) pour recréer la promotion avec ces derniers. (Une promotion étant un objet **complexe** qui ne peut être **sérialisé et envoyer** correctement par l'API)

Voici l'exemple de « GetAllNotesByPromo » du « NoteController »

```
        public ActionResult<Dictionary<string, IEnumerable<Note>>>
GetAllNotesByPromo(int anneeDebut, int typeBUT)
        {

            NOMPROMOTION nompromo = NOMPROMOTION.BUT1;
            switch (typeBUT)
            {
                case 1:
                    nompromo = NOMPROMOTION.BUT2;
                    break;
                case 2:
                    nompromo = NOMPROMOTION.BUT3;
```

```
                    break;

            }
            Promotion promo = new Promotion(nompromo, anneeDebut);
```

C – Stockage

Pour avoir des informations complémentaires, voir Conception Stockage et le Modèle Logique des Données

1. Interface DAO

Il existe une interface de DAO pour chaque classe métier « I<classe>DAO ».
Elles regroupent **toutes les méthodes nécessaires à un DAO**.
Si vous souhaitez **changer le type de la base de données ou modifier des méthodes**, il faudra passer par ces **interfaces**.

Pour plus d'informations sur les implémentations voir Implémentation DAO.

2. Implémentation DAO

Chaque interface de DAO a son **implémentation** « **<classe>DAOOracle** ».
Elles permettent de communiquer avec la base de données en utilisant des connexions (voir Connection).

Attention certaines requêtes n'utilisent pas de PreparedStatement et sont donc **vulnérable aux injections SQL**. Il faudrait corriger cela en **priorité** !

Petite particularité pour le « **NoteDAOOracle** », nous utilisons des **dictionnaires pour associer** les catégories, les confidentialités et les natures des **notes** à des entiers.
Veiller à bien les réutiliser afin d'éviter des erreurs dans les requêtes. Sinon une modification de la structure de la table Note pourrait être envisageable.

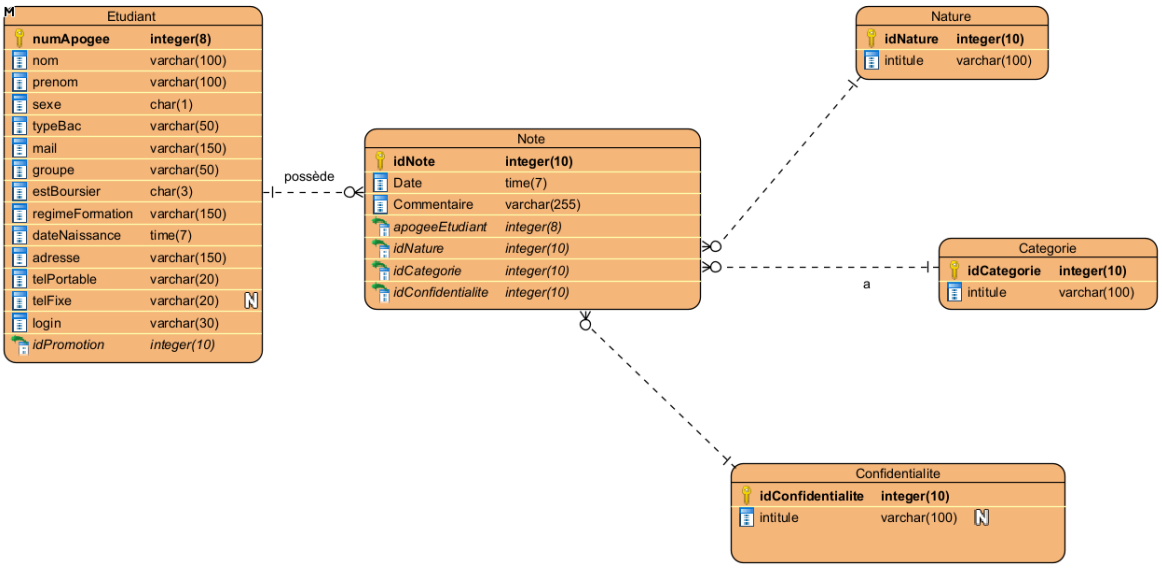
Vous avez accès au MLD des notes dans la partie « Etudiant et Note »

3 – Base de données Oracle

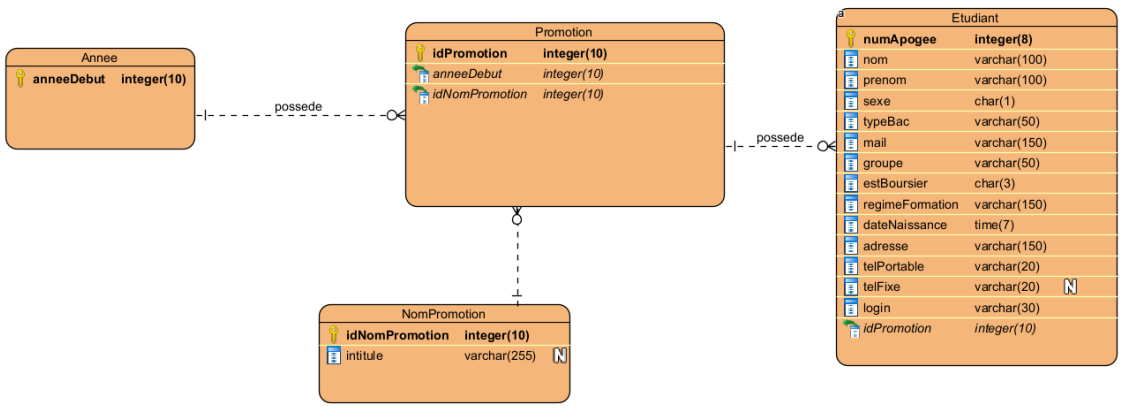
Script et Conception BDD

I – Modèle Logique des Données

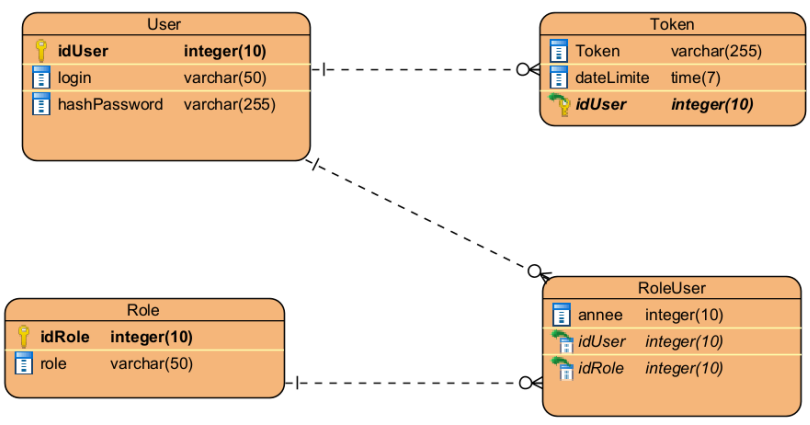
1 - Etudiant et Note



2 - Année et Promotion



3 - Utilisateur et Token



II – Script

Le script ci-dessous est le script **SQL** de la base de données de PAGE. Il est extrait du **SGBD Oracle** et **est configuré sur la database « IQ_BD_HIDA »**.

Si vous souhaitez redéployer la base de données il vous faudra adapter ce script pour votre database.

De plus, si vous utilisez un autre **SGBD veiller a ce que le SQL utilisé soit compatible.**

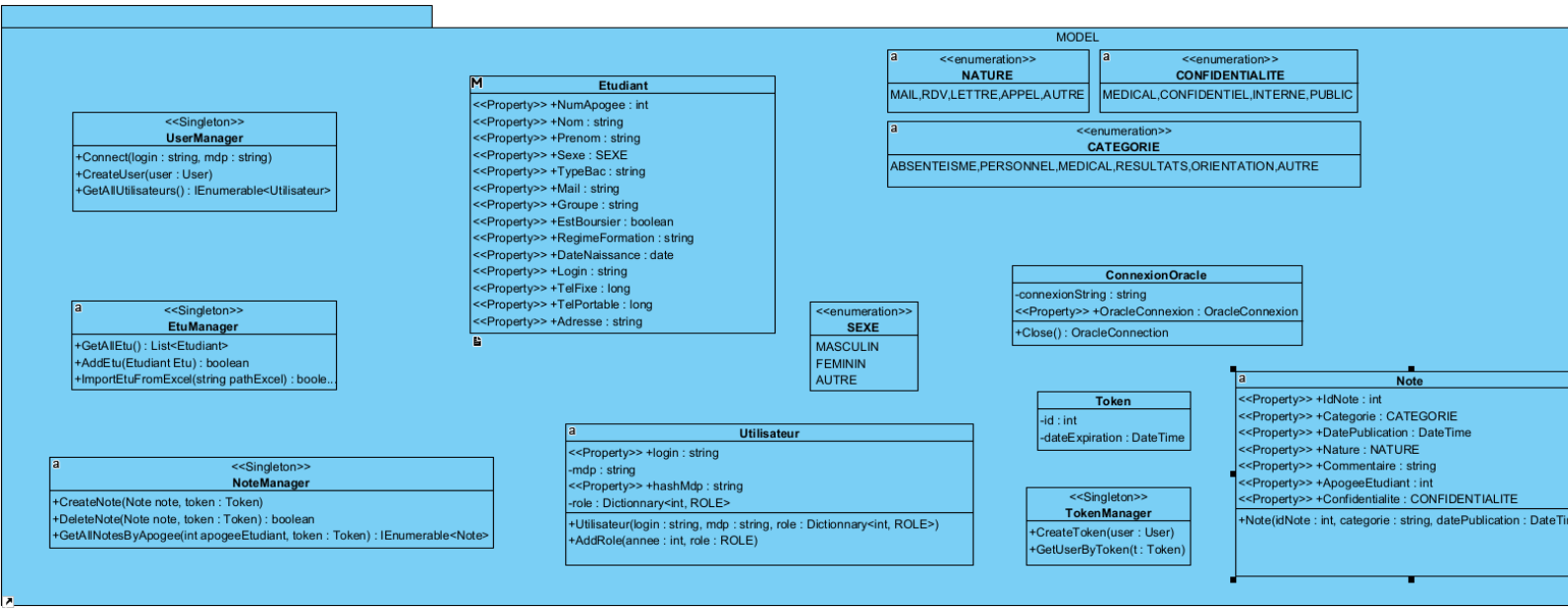
Vous pouvez accéder au script en cliquant ici

Annexe

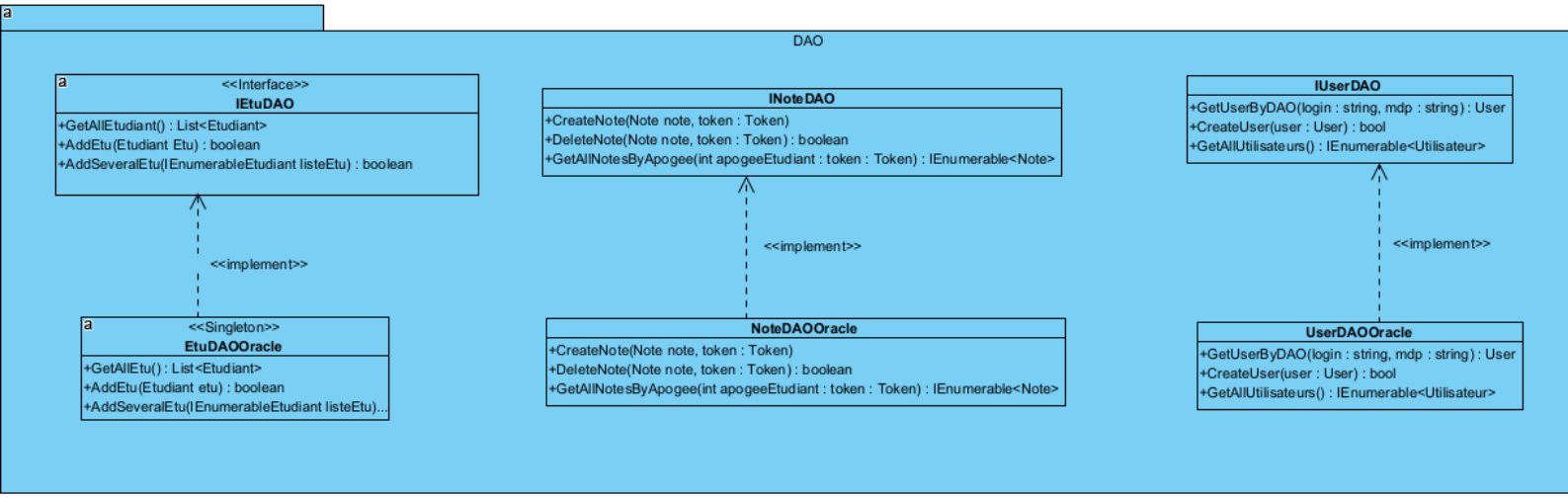
A - Conception

Les packages de l'API

Model



Stockage



B - Script Complet de la Base de Donnée

```
/* Etudiants */

CREATE TABLE Etudiant (
   numApogee NUMBER(8) PRIMARY KEY,
   nom VARCHAR2(50),
   prenom VARCHAR2(50),
   sexe CHAR(1),
   typeBac VARCHAR2(50),
   mail VARCHAR2(50),
   groupe VARCHAR2(50),
   estBoursier CHAR(3),
   regimeFormation VARCHAR2(50),
   dateNaissance DATE,
   adresse VARCHAR2(100),
   telPortable NUMBER(15),
   telFixe NUMBER(15),
   login VARCHAR2(8),
   CONSTRAINT chk_sexe CHECK (sexe IN ('M', 'F', 'A')),
   CONSTRAINT chk_bourse CHECK (estBoursier IN ('OUI', 'NON'))
);

/* NOTES */

-- Suppression de la table Note
DROP TABLE Note;

-- Suppression de la séquence NoteSeq
DROP SEQUENCE NoteSeq;

-- Création de la table des catégories
CREATE TABLE Categorie (
    idCategorie NUMBER(10) PRIMARY KEY,
    intitule VARCHAR2(50)
);

-- Insertion des données initiales pour les catégories
INSERT INTO Categorie (idCategorie, intitule) VALUES (0, 'ABSENTEISME');
INSERT INTO Categorie (idCategorie, intitule) VALUES (1, 'PERSONNEL');
INSERT INTO Categorie (idCategorie, intitule) VALUES (2, 'MEDICAL');
INSERT INTO Categorie (idCategorie, intitule) VALUES (3, 'RESULTATS');
INSERT INTO Categorie (idCategorie, intitule) VALUES (4, 'ORIENTATION');
INSERT INTO Categorie (idCategorie, intitule) VALUES (5, 'AUTRE');
```

```
-- Création de la table de confidentialités
CREATE TABLE Confidentialite (
    idConfidentialite NUMBER(10) PRIMARY KEY,
    intitule VARCHAR2(100)
);

-- Insertion des données initiales pour les confidentialités
INSERT INTO Confidentialite (idConfidentialite, intitule) VALUES (1, 'MEDICAL');
INSERT INTO Confidentialite (idConfidentialite, intitule) VALUES (2,
'CONFIDENTIEL');
INSERT INTO Confidentialite (idConfidentialite, intitule) VALUES (3, 'INTERNE');
INSERT INTO Confidentialite (idConfidentialite, intitule) VALUES (4, 'PUBLIC');

-- Création de la table des natures
CREATE TABLE Nature (
    idNature NUMBER(10) PRIMARY KEY,
    intitule VARCHAR2(50)
);

-- Insertion des données initiales pour les natures
INSERT INTO Nature (idNature, intitule) VALUES (1, 'MAIL');
INSERT INTO Nature (idNature, intitule) VALUES (2, 'RDV');
INSERT INTO Nature (idNature, intitule) VALUES (3, 'LETTRE');
INSERT INTO Nature (idNature, intitule) VALUES (4, 'APPEL');
INSERT INTO Nature (idNature, intitule) VALUES (5, 'AUTRE');

-- Création de la table des notes
CREATE TABLE Note (
    idNote NUMBER(10) PRIMARY KEY,
    titre VARCHAR2(50),
    datePublication DATE,
    commentaire VARCHAR2(255),
    idCategorie NUMBER(10),
    idNature NUMBER(10),
    idConfidentialite NUMBER(10),
    apogeeEtudiant NUMBER(8),
    CONSTRAINT fk_categorie FOREIGN KEY (idCategorie) REFERENCES
Categorie(idCategorie) ON DELETE SET NULL,
    CONSTRAINT fk_nature FOREIGN KEY (idNature) REFERENCES
Nature(idNature) ON DELETE SET NULL,
    CONSTRAINT fk_confidentialite FOREIGN KEY (idConfidentialite)
REFERENCES Confidentialite(idConfidentialite) ON DELETE SET NULL,
    CONSTRAINT fk_apogee FOREIGN KEY (apogeeEtudiant) REFERENCES
Etudiant(numApogee) ON DELETE SET NULL
);
```

```
-- Créez une séquence pour générer les valeurs auto-incrémentées
CREATE SEQUENCE NoteSeq
START WITH 1
INCREMENT BY 1
CACHE 10;

-- Création du déclencheur pour la séquence NoteSeq
CREATE OR REPLACE TRIGGER NoteSeqTrigger
BEFORE INSERT ON Note
FOR EACH ROW
BEGIN
    SELECT NoteSeq.NEXTVAL INTO :new.idNote FROM dual;
END;
/

select * from note;
delete from note;

INSERT INTO Note(idNote,titre, idCategorie, datePublication, idNature,
commentaire, apogeeEtudiant, idConfidentialite) VALUES(0,'Maladie la', 1,
TO_DATE('2023-11-23', 'YYYY-MM-DD'), 1, 'med abs mail', 30092005, 1);


  CREATE TABLE "IQ_BD_HIDA"."ROLESPAGE"
   (      "IDROLE" NUMBER(10,0),
          "INTITULEROLE" VARCHAR2(50 BYTE),
           PRIMARY KEY ("IDROLE")
  USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE
STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
DEFAULT)
  TABLESPACE "USERS"  ENABLE
   ) SEGMENT CREATION IMMEDIATE
  PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
 NOCOMPRESS LOGGING
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
DEFAULT)
  TABLESPACE "USERS" ;
```

```
  CREATE TABLE "IQ_BD_HIDA"."PROMOTION_ETUDIANT"
   (      "IDPROMOTION" NUMBER,
          "NUMAPOGEE" NUMBER,
           PRIMARY KEY ("IDPROMOTION", "NUMAPOGEE")
  USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE
STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
DEFAULT)
  TABLESPACE "USERS"  ENABLE,
           FOREIGN KEY ("IDPROMOTION")
            REFERENCES "IQ_BD_HIDA"."PROMOTION" ("IDPROMOTION") ENABLE,
           FOREIGN KEY ("NUMAPOGEE")
            REFERENCES "IQ_BD_HIDA"."ETUDIANT" ("NUMAPOGEE") ENABLE
   ) SEGMENT CREATION IMMEDIATE
  PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
 NOCOMPRESS LOGGING
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
DEFAULT)
  TABLESPACE "USERS" ;


  CREATE TABLE "IQ_BD_HIDA"."ROLEUTILISATEUR"
   (      "ANNEE" NUMBER(4,0),
          "IDUTILISATEUR" NUMBER(10,0),
          "IDROLE" NUMBER(10,0)
   ) SEGMENT CREATION IMMEDIATE
  PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
 NOCOMPRESS LOGGING
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
DEFAULT)
  TABLESPACE "USERS" ;


  CREATE TABLE "IQ_BD_HIDA"."NOMPROMOTION"
   (      "IDNOMPROMOTION" NUMBER,
          "INTITULE" VARCHAR2(4 BYTE),
           CHECK (intitule IN ('BUT1', 'BUT2', 'BUT3')) ENABLE,
```

```
          PRIMARY KEY ("IDNOMPROMOTION")
  USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE
STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
DEFAULT)
  TABLESPACE "USERS"  ENABLE
   ) SEGMENT CREATION IMMEDIATE
  PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
 NOCOMPRESS LOGGING
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
DEFAULT)
  TABLESPACE "USERS" ;


  CREATE TABLE "IQ_BD_HIDA"."PROMOTION"
   (      "IDPROMOTION" NUMBER,
          "ANNEEDEBUT" NUMBER,
          "IDNOMPROMOTION" NUMBER,
           PRIMARY KEY ("IDPROMOTION")
  USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE
STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
DEFAULT)
  TABLESPACE "USERS"  ENABLE,
          FOREIGN KEY ("ANNEEDEBUT")
           REFERENCES "IQ_BD_HIDA"."ANNEE" ("ANNEEDEBUT") ENABLE,
          FOREIGN KEY ("IDNOMPROMOTION")
           REFERENCES "IQ_BD_HIDA"."NOMPROMOTION" ("IDNOMPROMOTION")
ENABLE
   ) SEGMENT CREATION IMMEDIATE
  PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
 NOCOMPRESS LOGGING
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
DEFAULT)
```

```
  TABLESPACE "USERS" ;


  CREATE TABLE "IQ_BD_HIDA"."TOKEN"
   (      "IDUTILISATEUR" NUMBER(10,0),
          "TOKEN" VARCHAR2(255 BYTE),
          "DATELIMITE" DATE,
           PRIMARY KEY ("IDUTILISATEUR")
  USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE
STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
DEFAULT)
  TABLESPACE "USERS"  ENABLE
   ) SEGMENT CREATION IMMEDIATE
  PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
 NOCOMPRESS LOGGING
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
DEFAULT)
  TABLESPACE "USERS" ;


  CREATE TABLE "IQ_BD_HIDA"."UTILISATEUR"
   (      "IDUTILISATEUR" NUMBER(10,0),
          "LOGIN" VARCHAR2(50 BYTE),
          "HASHPASSWORD" VARCHAR2(255 BYTE)
   ) SEGMENT CREATION IMMEDIATE
  PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
 NOCOMPRESS LOGGING
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
DEFAULT)
  TABLESPACE "USERS" ;

  CREATE OR REPLACE EDITIONABLE TRIGGER
"IQ_BD_HIDA"."USERSEQTRIGGER"
BEFORE INSERT ON Utilisateur
FOR EACH ROW
BEGIN
```

```
    SELECT UserSeq.NEXTVAL INTO :new.idUtilisateur FROM dual;
END;

/
ALTER TRIGGER "IQ_BD_HIDA"."USERSEQTRIGGER" ENABLE;


  CREATE TABLE "IQ_BD_HIDA"."ANNEE"
   (      "ANNEEDEBUT" NUMBER,
           PRIMARY KEY ("ANNEEDEBUT")
  USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE
STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
DEFAULT)
  TABLESPACE "USERS"  ENABLE
   ) SEGMENT CREATION IMMEDIATE
  PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
 NOCOMPRESS LOGGING
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
DEFAULT)
  TABLESPACE "USERS" ;

  CREATE OR REPLACE EDITIONABLE TRIGGER
"IQ_BD_HIDA"."CREATEDEFAULTPROMOTIONS"
AFTER INSERT ON Annee
FOR EACH ROW
DECLARE
    v_idNomPromotion NUMBER;
BEGIN
    SELECT idNomPromotion INTO v_idNomPromotion
    FROM NomPromotion
    WHERE intitule = 'BUT1';

    IF v_idNomPromotion IS NOT NULL THEN
        INSERT INTO Promotion (idPromotion, anneeDebut, idNomPromotion)
VALUES (seq_promotion.nextval, :NEW.anneeDebut, v_idNomPromotion);
    END IF;

    SELECT idNomPromotion INTO v_idNomPromotion
    FROM NomPromotion
```

```
    WHERE intitule = 'BUT2';

    IF v_idNomPromotion IS NOT NULL THEN
        INSERT INTO Promotion (idPromotion, anneeDebut, idNomPromotion)
VALUES (seq_promotion.nextval, :NEW.anneeDebut, v_idNomPromotion);
    END IF;

    SELECT idNomPromotion INTO v_idNomPromotion
    FROM NomPromotion
    WHERE intitule = 'BUT3';

    IF v_idNomPromotion IS NOT NULL THEN
        INSERT INTO Promotion (idPromotion, anneeDebut, idNomPromotion)
VALUES (seq_promotion.nextval, :NEW.anneeDebut, v_idNomPromotion);
    END IF;
END;

/
ALTER TRIGGER "IQ_BD_HIDA"."CREATEDEFAULTPROMOTIONS" ENABLE;
```