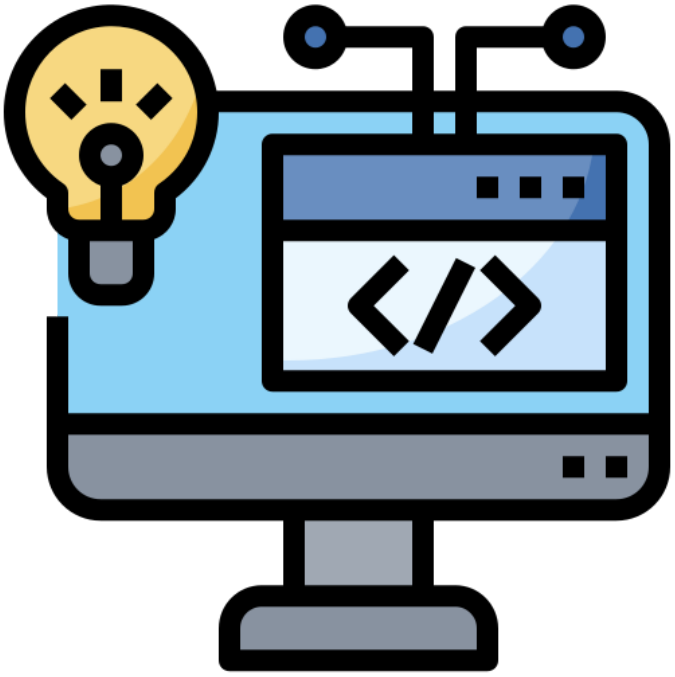
21 Mai 2020



RAPPORT PROJET

CATALOGUE ENSC

PROGRAMMATION AVANCÉE

BINET COLINE - PERRIER ALBAN

BORDEAUX INP

ENSC 1A – Groupe 1

SOMMAIRE

INTRODUCTION

Dans le cadre du module de Programmation avancée, nous avons effectué un
catalogue de projets effectués à l'ENSC.

L'objectif de ce rapport est de présenter notre travail et justifier nos choix de conception. Il détaillera ainsi les différentes fonctionnalités que nous avons implémentées, la manière dont celles-ci ont été développées, notre gestion de projet ainsi que, pour conclure, un bilan.

Le code source de ce projet est joint à ce rapport et aussi disponible à cette adresse : https://github.com/aperrier004/Catalogue_ENSC.

I. PRÉSENTATION DU PROJET

Ce projet de catalogue présente plusieurs fonctionnalités accessibles depuis le menu de l'application. Celles-ci sont réparties dans 3 sous-menus différents :

- Recherche de projets
  - Affichage du catalogue complet
  - Le tri et l'affichage de projets répondant à un critère particulier
- Gestion de projet
  - L'ajout de nouveaux projets
  - La suppression d'un projet
  - La modification de certaines informations d'un projet
- Gestion des données du projet
  - Réinitialisation de la Base de Données
  - Ajout d'objets en Base de Données

On suppose qu'un projet du catalogue est un projet terminé, rendu et noté. La liste de projets affichés (catalogue complet ou filtré) présente le résumé des projets. Il est ensuite possible de sélectionner un projet particulier pour prendre connaissance de toutes les informations qui lui sont relatives, les supprimer ou les modifier.
Le projet dans sa globalité fonctionne grâce à une base de données en local, nous utilisons SQLite, qui nous permet grâce à un unique fichier d'avoir une base données très simple. Elle nous permet de stocker toutes nos informations sur les projets, mais aussi sur les autres données (livrables, matières, élèves, professeurs, ...).

I.1. Affichage du catalogue

Le catalogue complet de projets stockés en BD peut être affiché.

I.2. Tri du catalogue

Le catalogue peut être trié en fonction de 8 critères :

- Par année de réalisation
- Par étudiant
- Par promotion
- Par type de projet (transpromotion, transdisciplinaire, projet de fin d'année, projet web, etc)
- Par matière
- Par mot-clef
- Par enseignant référent
- Par intervenant externe (personne non enseignant intervenant dans le projet comme tuteur ou client par exemple)

Chacun des critères génère une liste de valeurs possibles. Cette liste est générée automatiquement en consultant la base de données, elle permet ainsi de savoir quelles sont les promotions existantes, les matières existantes, etc.

Dans le cas des étudiants, la liste pourrait être très longue si l'on suppose que tous les étudiants de l'école, toutes promotions confondues soient en BD. Nous demandons donc de sélectionner au préalable la promotion de l'étudiant souhaité afin de réduire la liste.

I.3. Ajout de projet

Un projet peut-être ajouté à la liste des projets existants. Cet ajout implique qu'il soit sauvegardé en base de données.
On suppose que tous les *enseignants*, les *matières*, les *types de projets* ainsi que les *étudiants* existent déjà en base de donnée au moment d'ajouter un nouveau projet. L'utilisateur devra les sélectionner depuis une liste générée automatiquement. S'il souhaite en ajouter de nouveau, il devra au préalable les créer dans le menu de gestion des données (voir partie III.3).
Le reste des informations devra être renseigné manuellement.
Grâce à toutes ces informations est créée un objet *Projet*. Ce nouvel objet va se sauvegarder en base de données via une requête *Insert*.

I.4. Suppression de projet

Le menu de gestion de projet donne également accès la fonctionnalité de suppression d'un projet.
Pour ce faire, et pour éviter qu'un nombre potentiellement incommensurable de projets soit proposé à l'utilisateur, il lui est demandé à de sélectionner un critère de tri pour affiner la recherche.
Une fois le catalogue filtré, l'utilisateur doit renseigner le projet qu'il souhaite consulter et ce sera, depuis la page de consultation, qu'il aura la possibilité de le supprimer.

I.5. Modification d'un projet

La fonctionnalité de modification est similaire à celle de suppression : il est demandé à l'utilisateur un critère de tri pour filtrer le catalogue, puis il doit saisir le numéro projet qu'il souhaite voir en détail afin d'avoir accès aux différentes options de modification.

Celles-ci sont les suivants :

- Modifier le titre du projet
- Modifier le type du projet
- Modifier l'année du projet
- Modifier la note du projet
- Modifier les étudiants qui ont participé au projet.

Exception faite du cas des étudiants (voir partie III.4), les informations à modifier sont mises à jour en base de données par le biais d'une requête *update*.

I.6. Réinitialisation de la Base de Données

Cette fonctionnalité permet de remettre la base de données à son état initial.
Elle commence par supprimer toutes les tables (et donc leur contenu) du fichier .sqlite, pour ensuite créer de nouveau une structure avec les données initiales écrites dans le fichier bd.cs

I.7. Ajout d'un objet en Base de Données

Cette fonctionnalité permet d'ajouter un objet en Base de Données. Cet ajout est nécessaire si l'on souhaite créer un projet utilisant l'une de ces informations.
On peut notamment ajouter :

- Une matière
- Un livrable
- Un élève
- Un professeur
- Un intervenant

II. NOTICE

II.1. Installation

L'application ne demande aucune installation particulière. Au premier lancement l'application va générer le fichier de base de données en local ainsi qu'installer les dépendances (package SQLite), cela peut éventuellement prendre un peu de temps.
Si toutefois un problème au niveau du using System.Data.SQLite survenait , il sera peut-être nécessaire de procéder à son ajout manuellement. Voici les ressources à suivre :
https://system.data.sqlite.org/index.html/doc/trunk/www/downloads.wiki
https://system.data.sqlite.org/index.html/doc/trunk/www/faq.wiki#q8

II.2. Utilisation

Une fois lancée, l'application s'ouvre sur un menu donnant accès aux différentes catégories de fonctionnalités que nous avons implémentées.

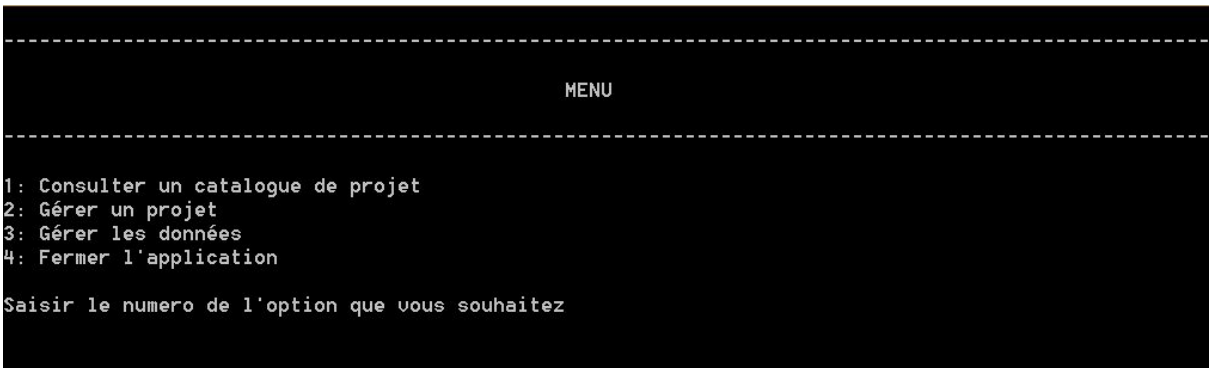


Figure 1 : Menu de lancement de l'application

Si l'option 1 est choisie, l'utilisateur a accès au menu catalogue :

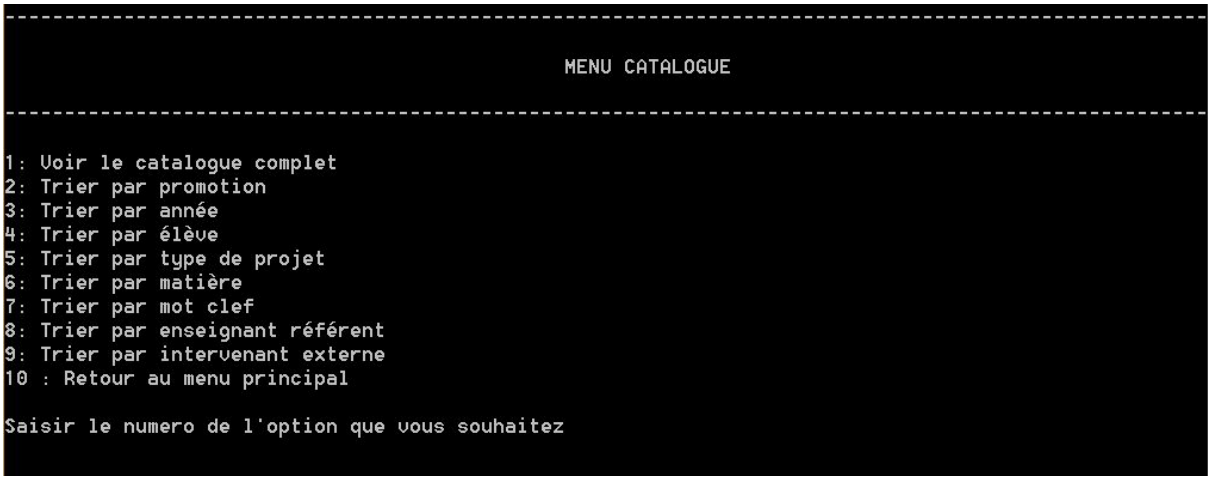


Figure 2 : Menu Catalogue

Les différentes options permettent d'effectuer un tri particulier sur les projets existants en BD afin d'afficher ceux correspondants.
L'affichage d'un projet résumé ressemble à cela :

```
========== Projet 4==========
Titre du projet : Projet Programmation Avancée
Type du projet : Projet Programmation Avancée
Année : 2018
Note : 16
Etudiant(s) :
Nom : Dilleux
Prénom : Eulalie
Promotion : 2018

Nom : Pevencie
Prénom : Susan
Promotion : 2018


Si vous souhaitez connaître le détail d'un projet, entrer l'id de celui-ci, sinon appuyez sur entrée
```

Figure 3 : Résumé d'un projet

On peut connaître le détail d'un des projets affichés en entrant son id/numéro. Cet affichage ressemble à cela :

```
Titre du projet : Projet Programmation Avancée
Type du projet : Projet Programmation Avancée
Année : 2018
Note : 16
Sujet Libre : Non
Titre du sujet : Dictionnaire enigmes
Professeur référent : Nom : Favier
Prénom : Pierre-Alexandre
Matières enseignées(s) :
Libellé de la matière : Programmation avancée
Code de la matière : C0GSFPA0

Nombre total d'intervenants : 3 personnes.
Date de Début : 23/03/2018 00:00:00
 Date de Fin : 18/05/2018 00:00:00
Livrables attendus :
Nature du livrable :Rapport
Date du rendu :05/18/2018 10:04
Note :15
Nature du livrable :Code source
Date du rendu :05/18/2018 10:04
Note :17
Liste des intervenants externes :
Mots Clef :
programmation avancee
Enigmes
Projet collectif : Oui
Etudiant(s) :
Nom : Dilleux
Prénom : Eulalie
Promotion : 2018

Nom : Pevencie
Prénom : Susan
Promotion : 2018



0: pour revenir à l'affichage de votre catalogue
1: pour supprimer le projet
2: pour modifier le projet
3: pour revenir au menu principal


Saisir le numero de l'option que vous souhaitez
```

Figure 4 : Détails d'un projet

Une fois le détail affiché, il est alors possible de supprimer ou modifier ce projet.

L'option 2 du menu principal permet d'afficher ce menu de gestion de projet :

Figure 5 : Menu Projet

L'option 1 permet la création de projet. Le déroulé de cette fonctionnalité est une suite de demandes d'informations dans un format spécifique. Tant que le format entré par l'utilisateur ne correspond pas à ce qui est attendu, il est impossible de passer à la suite.

L'option 2 permet de modifier un projet. Après avoir trié et sélectionné le projet souhaité, différentes options nous sont proposées :

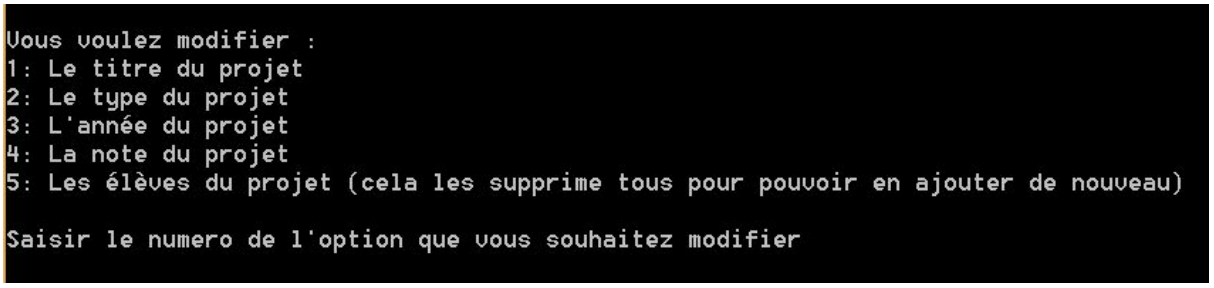


Figure 6 : Menu de paramètres à modifier

Pour les options 1, 2, 3 et 4, il est demandé à l'utilisateur de rentrer une nouvelle chaîne de caractères pour modifier le champ.
Pour l'option 5, la liste des élèves est générée afin que l'utilisateur puisse sélectionner ceux ou celui qui ont participé au projet.

Figure 7 : Menu Données

Depuis le menu principal, sélectionner le menu 3 amènes à ce sous-menu. Il est possible de remettre la BD à son état initial ou bien ajouter des objets dans la BD. Ces objets sont nécessaires pour l'ajout de projets.

III. CHOIX TECHNIQUES

III.1. Gestion des données

Nous avons souhaité sauvegarder et gérer nos données en utilisant une base de données. En effet, celle-ci nous semblait permettre une récupération des informations bien plus efficace que si nous étions passés par un fichier texte. Les requêtes SQL permettent de filtrer facilement les projets en une seule instruction tandis qu'une gestion par fichier texte aurait demandé de récupérer chaque ligne, de les caster pour sélectionner et déterminer la valeur d'un attribut, puis de les analyser, comparer, etc. Ceci additionné au fait que chacun de nos objets / tables dispose d'un nombre parfois assez conséquent d'attributs a justifié notre choix.
Outre l'avantage certain concernant la manipulation de données que nous offre une base de données, la bibliothèque SQL Lite nous permet d'enregistrer les modifications de la base de données (ajout, modification, suppression) d'une session à une autre.

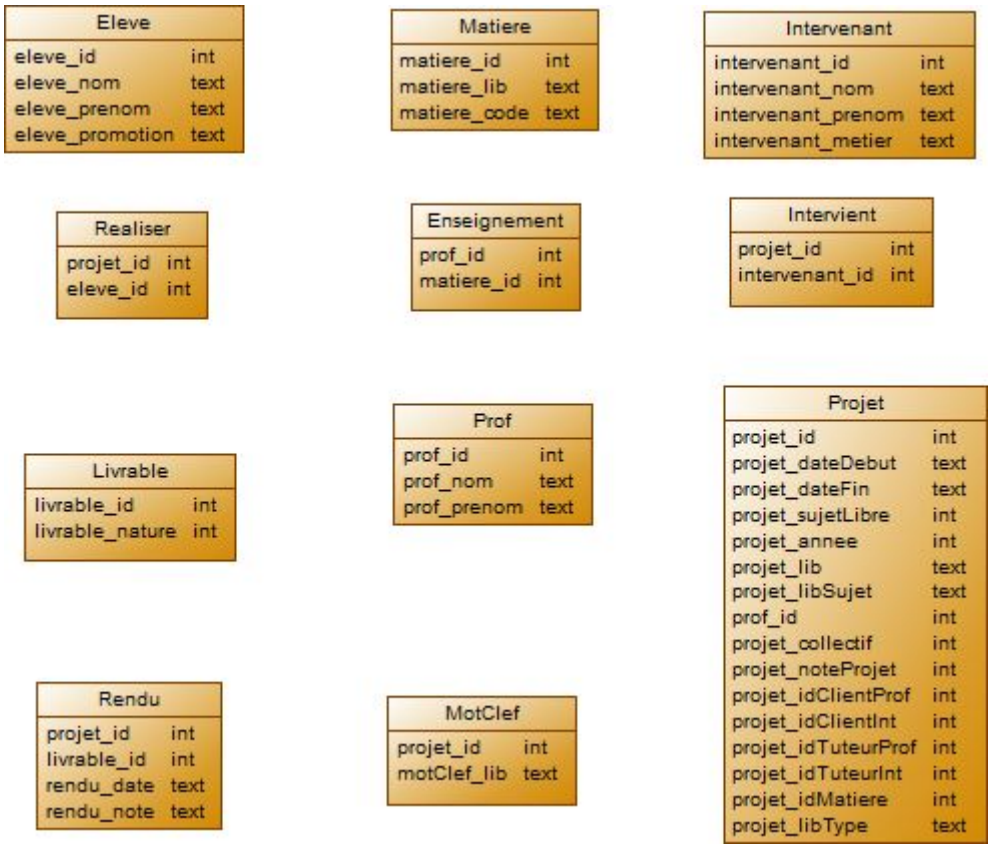


Figure 8 : Schéma de la structure de la BD

Notre base de données ne correspond pas exactement à la structure de nos objets en C#.
Nous avons en effet créé des tables intermédiaires (*Realiser*, *Enseignement*, *Rendu*, *Intervient*) qui nous permettent d'associer à une instant X un nombre variables d'entités Y. Nous l'utilisons lorsqu'il faut associer un enseignant à plusieurs matières (table intermédiaire : *enseignement*) ou un étudiant à plusieurs projets (table intermédiaire : *réaliser*).

Une autre particularité de cette structure est l'absence de contraintes ou de clés (primaires ou étrangères). Nous avons fait ce choix pour simplifier l'utilisation de l'application et de SQLite, ne voulant ni alourdir notre charge de travail, ni ralentir notre progression sur le projet.

Dans le but de simplifier la notion d'héritage(cf. le diagramme UML) pour notre BD, nous avons décidé de simplement dupliquer des champs : les attributs nom et prénom de la classe parente *Personne* se retrouvent alors dans les tables *Eleve*, *Prof* et *Intervenant* qui correspondent aux classes filles. De la même manière, nous avons simplifié la notion de client ou de tuteur pour un *projet Professionnel* en dupliquant ces champs. Dans un projet, si le client est un enseignant, alors la colonne *projet_idClientInt* (qui prendrait l'identifiant du client si celui-ci est un intervenant externe) aura pour valeur 0.
Concernant les types des colonnes, ils correspondent aux types utilisés par SQLite (une variable "DateTime" en C# devient un "text" en base de données SQLite).

III.2. Architecture

Nous avons été amenés à revoir notre architecture initiale. (cf. diagramme UML initial)
Celle-ci en effet s'est vue trop complexe sur certains aspects pour pouvoir être correctement implémentée et exploitée durant l'usage de l'application. Ce que nous avions prévu concernant les objets relatifs à Projet (classes *TypeProjet*, *ProjetProfessionnel*, *ProjetAppliqué* et les classes qui en héritent) a été simplifié : il ne reste plus que les *ProjetProfessionels* et les *ProjetsAppliqués*. Toutes les classes qui héritaient de ces deux types de projet ont été supprimées (transpromotion, GCCO, RAO, etc) car, à notre échelle, nous ne les trouvions pas pertinentes. Dans un contexte plus concret, il aurait toutefois été idéal de les garder pour une gestion plus précise des projets.
Nous avons également supprimé les classes *Informatique* et *Cognitique* que nous trouvions finalement peu intéressantes comme critères de tris du catalogue *(*cf diagramme UML final).

Nous avons également supprimé l'interface ITriable. Celle-ci aurait été intéressante si nous avions permis que notre application trie également les matières, les étudiants, ... Et pas uniquement les projets. Ne le faisant pas, elle n'avait plus d'utilité.

III.3. Afficher la liste de projets filtrée selon un critère

Depuis le menu, l'utilisateur a la possibilité de sélectionner la fonctionnalité qu'il souhaite. (voir Figure 1)

S'il souhaite consulter le catalogue, un sous-menu affiché par la fonction TrierProjet() (géré via un *switch*) lui permet de sélectionner le critère de son choix (consulter le catalogue complet ou trier selon un des critères énuméré). (voir Figure 2 et cf. Annexe Code)

À chaque cas du *switch* une fonction de la forme MenuTriCritere() est associée. (cf. Annexe Code)
Dans cette fonction se trouvent les différentes instructions nécessaires pour déterminer les valeurs du critère choisi.
Tout d'abord, nous exécutons une requête (*SELECT distinct X FROM Table*) en base de données afin de déterminer les valeurs possibles pour le critère donné (quelles sont les promotions existantes, les matières, etc). Cette liste permettra à l'utilisateur de préciser la valeur du critère sans risquer de commettre une erreur de saisie.
Une fois que l'utilisateur a inscrit son choix dans la console, une seconde requête est exécutée dans la fonction CreerCatalogue() (cf. Annexe Code), afin de déterminer quels sont les projets y répondant.
Ladite requête retourne les données des projets qui sont alors créés dans une liste de projets avec la fonction CreerListeProjets(). Cette fonction permet de créer un à un tous les projets en les ajoutant à la liste. Cette liste

de projets peut alors être ajoutée à un objet *Catalogue* qui les recensera et permettra, ensuite, de les afficher à l'utilisateur.

Nous créons donc l'objet avant de l'afficher, de même que lorsque l'on veut sauvegarder un nouveau projet en base de données, nous commençons au préalable par créer l'objet *Projet* correspondant. Cette initiative est pensée dans une question de rigueur et de maintenabilité. L'utilisation du constructeur du *Projet* permet de vérifier que les données entrées sont bonnes et cohérentes avant d'effectuer toute manipulation de données en BD.

Cette fonctionnalité associée à la fonction TrierProjet() nous permet donc de la réutiliser pour d'autres fonctionnalités où il est utile de n'afficher que le projet voulu (pour modifier ou supprimer par exemple).
D'autre part, nous avons voulu séparer les différents cas selon des critères en appelant des fonctions propres à chacune afin d'avoir l'opportunité de rajouter potentiellement d'autres critères par la suite. Il suffirait ainsi d'ajouter un cas au *Switch* et une fonction *MenuTriCritere()* qui nous renverrait des strings que l'on peut utiliser dans les *data1* ou *data2* afin de créer le catalogue correspondant. Il y a cependant des cas comme le tri par élève, enseignant et intervenant où il est intéressant d'utiliser des fonctions avec des paramètres *out* afin d'avoir plusieurs variables renvoyées (nom et prénom de la personne), justifiant l'utilisation d'un data2.

III.4. Modification de projets

Afin de modifier un projet, nous faisons appel à la fonction ModifierProjetBD() (cf. Annexe Code) qui affiche les 5 options de modification (voir Figure 6). Selon l'option choisie, une requête SQL du type *update* sera exécutée afin de mettre à jour le champ associé.

Un seul cas particulier ne suit pas ce schéma-là : la modification d'élèves participant à un projet. En effet, pas de requête *update* pour eux, mais une suppression des données associées puis un ajout de lignes dans une table. Ce choix s'explique notamment par le fonctionnement de notre base de données. En effet, pour associer un étudiant à un projet, nous employons une table intermédiaire : *Réaliser* qui associe *projet_id* à *eleve_id.* Cette table nous permet d'associer un nombre variable d'étudiants à un projet. Un avantage qui peut vite devenir un inconvénient quand on souhaite modifier la liste des étudiants ayant travaillé sur un projet en particulier, car il faut, pour chacun d'eux, demander à l'utilisateur ce qu'il souhaite faire (le supprimer, le garder ou le modifier, s'il veut le modifier, il faut afficher de nouveau la liste complète des élèves pour qu'il choisisse celui souhaité, etc). Il y aurait eu beaucoup d'interactions avec l'utilisateur pour pas grand-chose. Supprimer la liste et la recréer est plus rapide et moins contraignant.

III.5. Ajout d'un objet en BD

Afin de permettre à l'utilisateur de pouvoir ajouter de nouveaux objets, comme une matière, un élève ou un professeur, nous avons créé un "Menu Données" (cf. Figure 7). Il permet de faire appel aux fonctions d'ajouts d'objets. Par exemple pour ajouter une matière, on appelle la fonction AjouterMatiere() (cf. Annexe Code) qui récupère le dernier id créé avec ChercherDernierIdMatiere() pour ensuite demander les valeurs des attributs d'un objet Matiere. Une fois cela fait il sera possible de créer un objet Matière avec son constructeur, objet que l'on envoie dans la fonction AjouterMatiereBD(), qui, elle, s'occupe de vérifier si cet objet n'existe pas déjà, et si tel est le cas, de l'ajouter en BD.

III.6. Développement

L'ensemble du projet a été testé au fur et à mesure de son développement par le biais de tests unitaires et de tests fonctionnels.
Nous avons essayé de limiter au maximum les entrées utilisateurs inattendus, notamment grâce à la fonction ValiderChaîne(). Cette fonction prend en paramètres un type, une chaîne et facultativement une liste. Selon le type on crée un objet Regex correspondant au format que l'on attend dans la chaîne. On vérifie si ce Regex et cette chaîne correspondent, si oui, on regarde si une liste a été donnée. Cette liste contiendra les valeurs déjà existantes en BD pour le type demandé. On vérifie ainsi si le contenu de la chaîne existe bien. Par exemple, dans le cas où l'on demande d'entrer le numéro d'un élève à ajouter, cette fonction permet de vérifier que, d'une part, la chaîne saisie correspond à un numéro, et d'autre part que ce numéro fasse partie des élèves qui lui ont été affichés.

De plus, afin de vérifier le bon fonctionnement de notre application, nous avons mené un plan de test selon diverses situations (cf. Annexe 1).

IV. GESTION DU PROJET

IV.1. Plannings

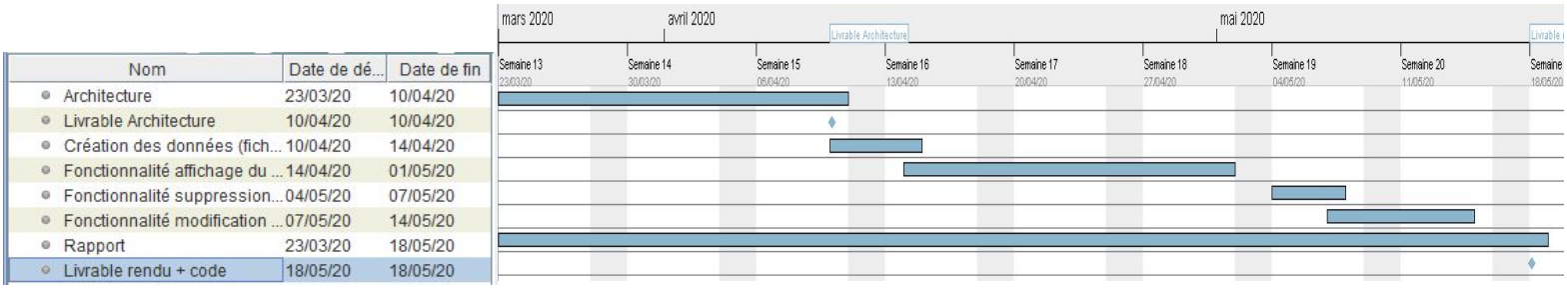


Figure 9 : Planning initial

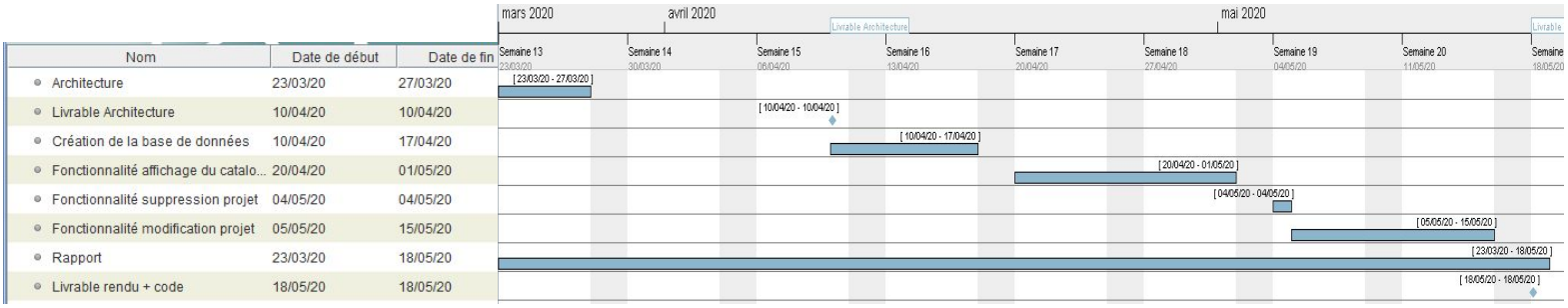


Figure 10 : Planning final

Le planning initial a été réfléchi en prenant en considération les autres projets sur lesquels nous travaillions en parallèle, raison pour laquelle certaines tâches peuvent sembler longues.
Nous avons quelques écarts. Le temps consacré à la gestion des données a été plus long du fait d'un changement de stratégie (nous avons délaissé les fichiers textes pour une gestion par base de données).
La suppression de projets a été très rapide à traiter, au contraire des deux autres fonctionnalités (affichage et modification) qui sont plus conséquentes.

IV.2. Fonctionnalités hiérarchisées

Nous avons, au début du projet, effectué la hiérarchie de fonctionnalités suivante en pensant à ce qui devait être réalisé en priorité pour obtenir un produit minimum viable :

- Afficher une liste de projet filtré selon un critère
- Créer un projet
- Supprimer un projet
- Modifier un projet
- Réinitialiser la BD
- Ajouter des données dans la BD

Nous avons adapté le planning en fonction de cette hiérarchie.
Nous avons finalement eu la possibilité de développer chacune de ces fonctionnalités.

V. CONCLUSION

V.1. Difficultés rencontrées

Nous avons pu rencontrer des difficultés, notamment concernant les vérifications de ce qu'entre l'utilisateur.
Par exemple, pour l'ajout d'élèves à un projet, il faut vérifier que, selon que le projet soit collectif ou non, le nombre d'élèves soit cohérent. Il faut donc que l'ajout s'arrête automatiquement si le projet est individuel dès la sélection du premier élève.
En revanche s'il est collectif, il faut faire attention que l'utilisateur rentre au moins 2 élèves, et qu'il ne rentre pas plusieurs fois le même.
Nous avons donc surmonté ce problème avec une variable qui nous permet d'avoir le minimum d'élèves à ajouter (2 dans le cas d'un projet collectif), variable que l'on décrémente à chaque ajout.

Concernant la vérification d'ajout en doublon, nous avons modifié nos fonctions d'insertion d'objets en BD afin de vérifier en BD qu'un objet avec les paramètres souhaités n'existe pas déjà (par exemple le *projet_id* et l'*èlève_id* dans la table *réaliser*). Si c'est le cas, l'insertion n'a pas lieu, la fonction renvoie *false* comme valeur de retour (auparavant ces fonctions ne renvoyaient rien, mais désormais elles nous permettent de savoir si une insertion à bien été réalisé ou non).

Une autre difficulté que nous avons pu rencontrer est celle de la fermeture volontaire de l'application alors que celle-ci est en cours de création de projet par exemple.
En effet, la création d'un objet projet nécessite forcément de créer des objets intermédiaires au préalable (livrables, élèves, ect). L'objet projet ne

peut être créé que si tous les objets qui lui serviront d'attribut seront instanciés.
De fait, si la console se ferme en milieu de processus, la BD peut alors avoir dans ses tables, par exemple dans *réaliser*, la présence d'un *projet_id* avec un *eleve_id* alors qu'aucun projet avec ce *projet_id* n'existe. Cela est possible car l'id que l'on utilise pour ajouter dans la BD est simplement l'id max trouvé dans la table projet + 1. Donc lorsque l'on créé un projet, les élèves étant ajoutés avant le projet, si l'on quitte avant la fin de la création, rien n'a été ajouté dans la table projet et donc l'id récupéré pour la prochaine création est le même, ce qui pose évidemment des problèmes (on ne peut même plus ajouter de nouveaux élèves au bout d'un moment puisqu'il y a la vérification de doublon).
Notre solution a donc été de créer une fonction *VerifierDonneesApresFermetureBrusque()* qui vérifie pour toutes les tables concernées (par un potentiel ajout d'un *projet_id* avant que le projet se crée) s'il existe des données dans ces tables avec un *projet_id* supérieur à celui maximum présent dans la table projet. Si un résultat est trouvé, alors la ligne de la table est supprimée. Cela nous permet d'éviter toutes incohérences dans nos tables.

V.2. Perspectives et améliorations

Plusieurs améliorations sont envisageables afin de rendre ce catalogue à la fois complet et intuitif.

Nous avons fait le choix de prioriser le développement des fonctionnalités plutôt que le design. Nous avons toutefois conscience qu'utiliser notre application avec la console n'est pas le plus pratique pour l'utilisateur. L'emploi de WinForms aurait rendu son utilisation plus agréable pour l'utilisateur et, côté développement, cela nous aurait simplifié la tâche. Nous aurions pu exploiter les avantages de la programmation évènementielle pour rendre les fonctionnalités plus accessibles (avoir accès aux fonctionnalités

de modification et de suppression d'un projet directement depuis le détail d'un projet sans avoir à revenir au menu par exemple).

Nous songeons également qu'il puisse être intéressant de disposer de davantage de possibilités de modifications, qui se limitent à cinq paramètres, ainsi que de pouvoir modifier les autres objets liés au catalogue (les enseignants, les intervenants externes, les livrables, etc).

VI. ANNEXES

SOMMAIRE DES ANNEXES

Annexe 1 : TESTS

No. Test	Fichier ou écran cible	Nom du test	But du test	Résultat attendu	Résultat actuel
1	Console	Lancement de l'application	Vérifier que le message d'accueil s'affiche et qu'en appuyant sur une touche on passe à l'écran d'après	On est redirigé vers le menu principal	Comme prévu
2	Console	Menu principal	Vérifier que si l'on entre autre chose que 1,2,3 ou 4 la console nous redemande toujours d'entrer une option	La demande de saisie se répète	Comme prévu
3	Console	Menu principal	Vérifier que si l'on entre 1 on soit redirigé sur le menu catalogue	On est redirigé vers le menu catalogue	Comme prévu
4	Console	Menu principal	Vérifier que si l'on entre 2 on soit redirigé sur le menu de gestion d'un projet	On est redirigé vers le menu projet	Comme prévu
5	Console	Menu principal	Vérifier que si l'on entre 3 on soit redirigé sur le menu de gestion des données	On est redirigé vers le menu données	Comme prévu
6	Console	Menu principal	Vérifier que si l'on entre 4 on soit redirigé vers une confirmation de fermeture de l'application	On est redirigé vers une demande de confirmation	Comme prévu
7	Console	Quitter	Vérifier qu'en appuyant sur la touche "O" la console se ferme/ l'application s'arrête	L'application s'arrête	Comme prévu
8	Console	Quitter	Vérifier qu'en appuyant sur la touche "N" on est redirigé sur le menu principal	On est redirigé vers le menu principal	Comme prévu
9	Console	Menu catalogue	Vérifier que si l'on entre autre chose que 1,2,3,4,5,6,7,8,9 ou 10 la console nous redemande toujours d'entrer une option	La demande de saisie se répète	Comme prévu
10	Console	Menu catalogue	Vérifier que si l'on entre 1 on affiche tous les projets de la BD	Tous les projets sont affichés	Comme prévu

11	Console	Menu catalogue	Vérifier que si l'on entre 2 on affiche les promotions d'élèves présent en BD et demande d'en saisir une	Les promotions sont affichés et la saisie aussi	Comme prévu
12	Console	Tri par Promotion	Vérifier que si l'on entre autre chose que les promotions affichés la demande de saisie se répète	La demande de saisie se répète	Comme prévu
13	Console	Tri par Promotion	Vérifier que si l'on entre une des années de promotion affichés, les projets réalisés par des élèves de cette promotion soient affichés	Les projets s'affiche correctement	Comme prévu
14	Console	Menu catalogue	Vérifier que si l'on entre 3 on affiche les années de projet présente en BD et demande d'en saisir une	Les années sont affichés et la saisie aussi	Comme prévu
15	Console	Tri par Année	Vérifier que si l'on entre autre chose que les années affichées la demande de saisie se répète	La demande de saisie se répète	Comme prévu
16	Console	Tri par Année	Vérifier que si l'on entre une des années de projet affichés, les projets réalisés cette année soient affichés	Les projets s'affiche correctement	Comme prévu
17	Console	Menu catalogue	Vérifier que si l'on entre 4 on affiche les promotions des élèves présent en BD et demande d'en saisir une	Les promotions sont affichées et la saisie aussi	Comme prévu
18	Console	Tri par Eleve	Vérifier que si l'on entre autre chose que les id d'élèves affichés la demande de saisie se répète	La demande de saisie se répète	Comme prévu
19	Console	Tri par Eleve	Vérifier que si l'on entre un des id affichés, les projets ayant cet élève en réalisatieur soient affichés	Les projets s'affiche correctement	Comme prévu
20	Console	Menu catalogue	Vérifier que si l'on entre 5 on affiche les types de projets présent en BD et demande d'en saisir une	Les types de projet sont affichées et la saisie aussi	Comme prévu

21	Console	Tri par TypeProjet	Vérifier que si l'on entre autre chose que le le libellé d'un type de projet affichés la demande de saisie se répète	La demande de saisie se répète	Comme prévu
22	Console	Tri par TypeProjet	Vérifier que si l'on entre un des libellé affichés, les projets ayant ce type de projet soient affichés	Les projets s'affiche correctement	Comme prévu
23	Console	Menu catalogue	Vérifier que si l'on entre 6 on affiche les matières présentes en BD et demande d'en saisir une	Les matières sont affichées et la saisie aussi	Comme prévu
24	Console	Tri par Matière	Vérifier que si l'on entre autre chose que le le libellé d'une matière affichée la demande de saisie se répète	La demande de saisie se répète	Comme prévu
25	Console	Tri par Matière	Vérifier que si l'on entre un des libellé affichés, les projets ayant un prof référant enseignant cette matière soient affichés	Les projets s'affiche correctement	Comme prévu
26	Console	Menu catalogue	Vérifier que si l'on entre 7 on affiche les mots clefs présent en BD et demande d'en saisir un	Les mots clefs sont affichés et la saisie aussi	Comme prévu
27	Console	Tri par Mot Clef	Vérifier que si l'on entre autre chose que le le libellé d'un mot clef affiché la demande de saisie se répète	La demande de saisie se répète	Comme prévu
28	Console	Tri par Mot Clef	Vérifier que si l'on entre un des libellé affichés, les projets ayant pour mot clef celui entré soient affichés	Les projets s'affiche correctement	Comme prévu
29	Console	Menu catalogue	Vérifier que si l'on entre 8 on affiche les enseignants présent en BD et demande d'en saisir un	Les enseignants sont affichés et la saisie aussi	Comme prévu
30	Console	Tri par Enseignant	Vérifier que si l'on entre autre chose que l'id d'un enseignant affiché la demande de saisie se répète	La demande de saisie se répète	Comme prévu
31	Console	Tri par Enseignant	Vérifier que si l'on entre un des id affichés, les projets ayant pour enseignant celui entré soient affichés	Les projets s'affiche correctement	Comme prévu

32	Console	Menu catalogue	Vérifier que si l'on entre 9 on affiche les intervenants présent en BD et demande d'en saisir un	Les intervenants sont affichés et la saisie aussi	Comme prévu
33	Console	Tri par Intervenant	Vérifier que si l'on entre autre chose que l'id d'un intervenant affiché la demande de saisie se répète	La demande de saisie se répète	Comme prévu
34	Console	Tri par Intervenant	Vérifier que si l'on entre un des id affichés, les projets ayant pour mot clef celui entré soient affichés	Les projets s'affiche correctement	Comme prévu
35	Console	Menu catalogue	Vérifier que si l'on entre 10 on est redirigé sur le menu principal	On est redirigé correctement	Comme prévu
36	Console	Menu Projet	Vérifier que si l'on entre autre chose que les options affichés la demande de saisie se répète	La demande de saisie se répète	Comme prévu
37	Console	Créer un projet	Vérifier que le projet se créée bien	Le projet s'ajoute correctement	Comme prévu
38	Console	Créer un projet	Vérifier que les entrées utilisateur soient vérifiés et que les demandes de saisie se répète si l'utilisateur n'entre pas quelque chose d'attendu	Les demandes de saisies se répètent	Comme prévu
39	Console	Modifier un projet	Vérifier que le projet se modifie bien	Le projet se modifie correctement	Comme prévu
40	Console	Modifier un projet	Vérifier que les entrées utilisateur soient vérifiés et que les demandes de saisie se répète si l'utilisateur n'entre pas quelque chose d'attendu	Les demandes de saisies se répètent	Comme prévu
41	Console	Supprimer un projet	Vérifier que le projet se supprime bien	Le projet se modifie correctement	Comme prévu
42	Console	Supprimer un projet	Vérifier que les entrées utilisateur soient vérifiés et que les demandes de saisie se répète si l'utilisateur n'entre pas quelque chose d'attendu	Les demandes de saisies se répètent	Comme prévu

43	Console	Menu Données	Vérifier que si l'on entre autre chose que les options affichés la demande de saisie se répète	La demande de saisie se répète	Comme prévu
44	Console	Ajouter une matière	Vérifier que la matière se crée bien	La matière s'ajoute correctement	Comme prévu
45	Console	Ajouter un livrable	Vérifier que le livrable se crée bien	Le livrable s'ajoute correctement	Comme prévu
46	Console	Ajouter un élève	Vérifier que l'élève se crée bien	L'élève s'ajoute correctement	Comme prévu
47	Console	Ajouter un prof	Vérifier que le prof se crée bien	Le prof s'ajoute correctement	Comme prévu
48	Console	Ajouter un intervenant	Vérifier que l'intervenant se crée bien	L'intervenant s'ajoute correctement	Comme prévu
49	Console	Revenir au menu principal	Vérifier que l'on revient bien au menu principal	On est redirigé correctement	Comme prévu
50	MaBaseDeDonnees.sqlite	Remettre la BD à l'état initial	Vérifier que le fichier se reniatilise bien	La BD est correctement mise à l'état initial	Comme prévu
51	MaBaseDeDonnees.sqlite	Première exécution du projet	Vérifier que le fichier se crée bien	La BD est correctement créée	Comme prévu

Annexe 2 : Diagramme UML initial

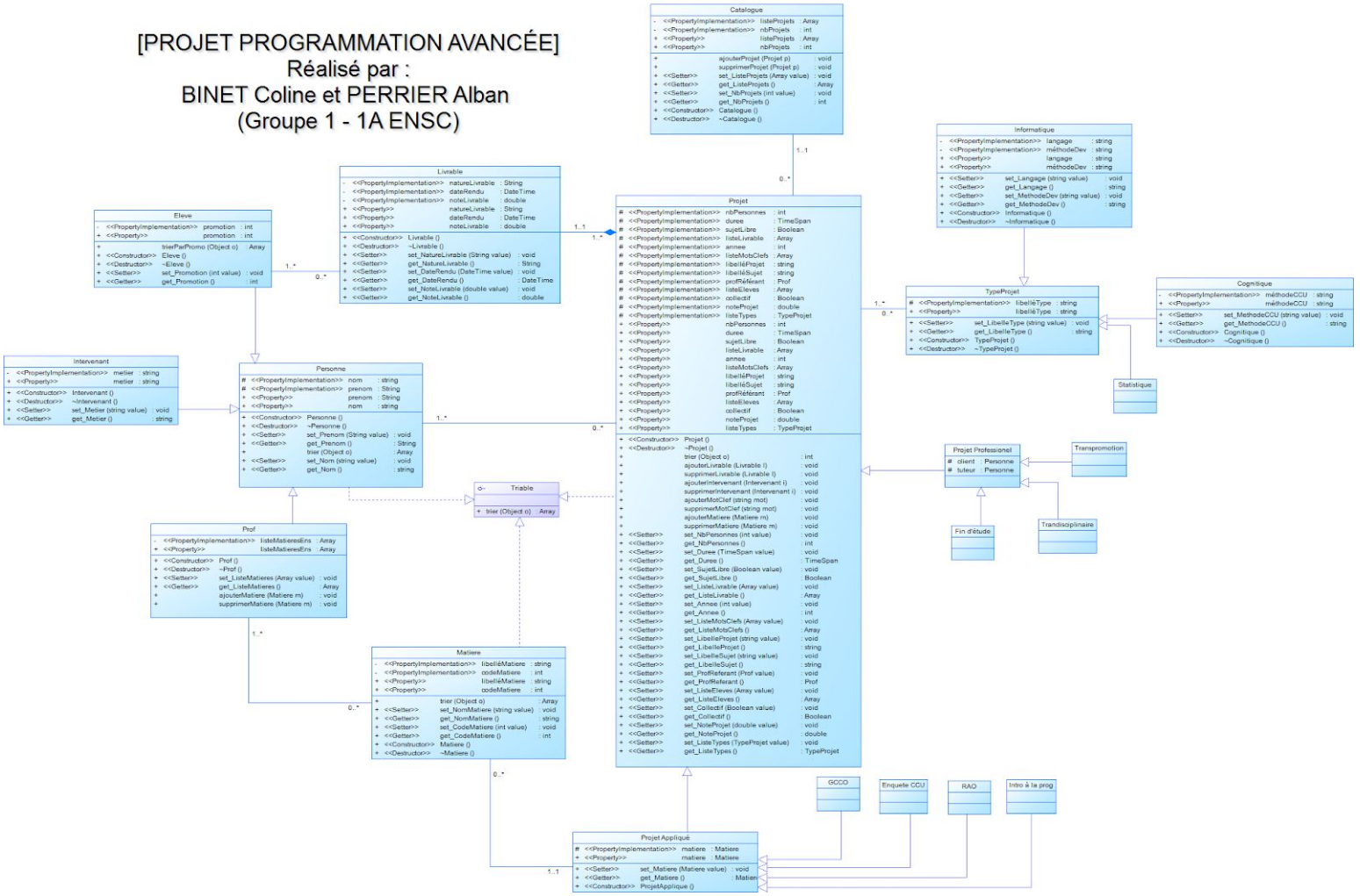


Diagramme disponible à ce lien :

https://drive.google.com/file/d/1UBqdMHKp0P6zkh7T0Yo_DlEG-r-TeENe/view?usp=sharing

Annexe 3 : Diagramme UML final

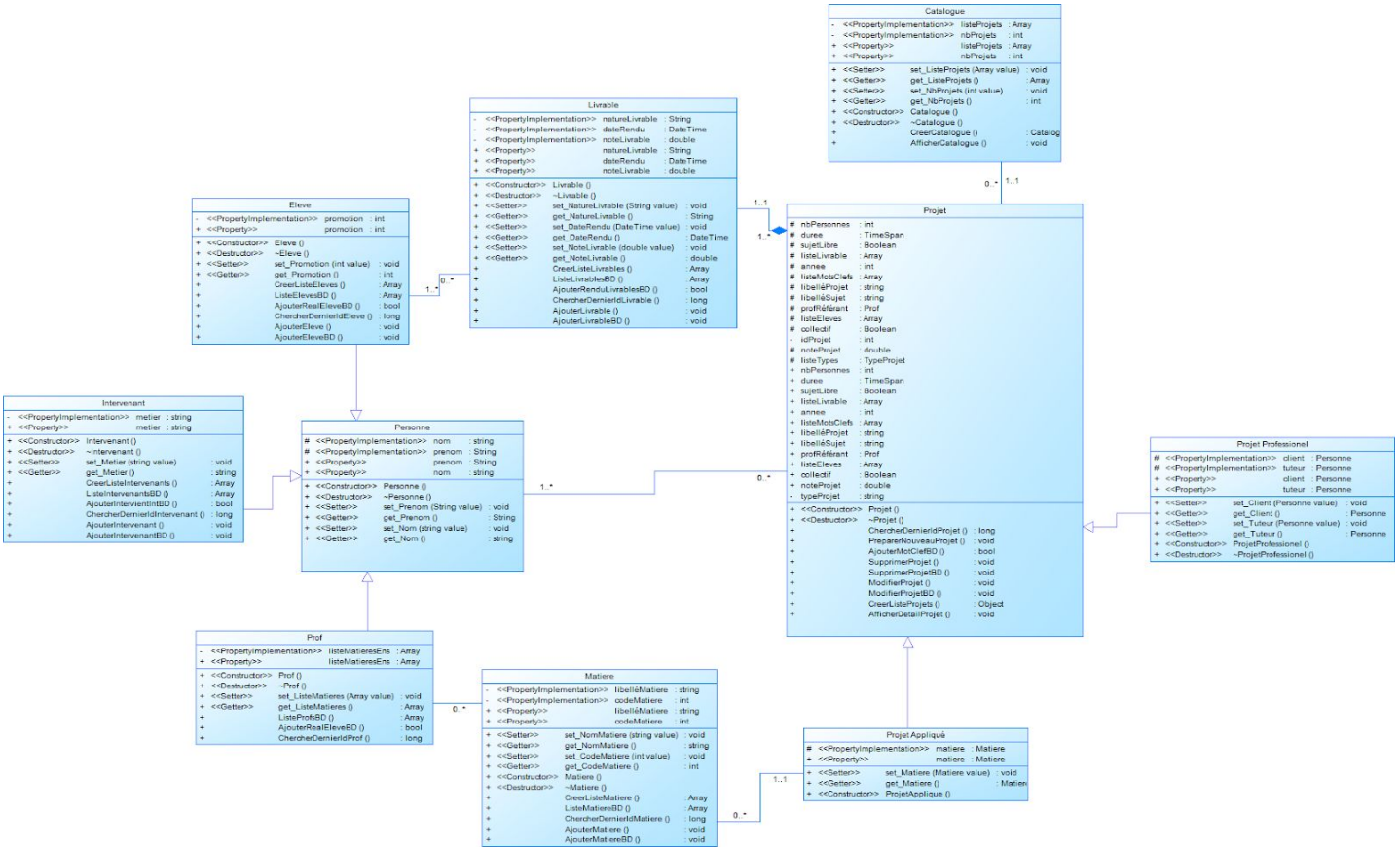


Diagramme disponible à ce lien :

https://drive.google.com/file/d/1nIeYja_L-JDb1iJ-yZGNRb5CYTwzCzSQ/view?usp=sharing

Annexe 4 : Code

AfficherMenuCatalogue()

```
// But : Permet d'afficher le menu de visualisation du catalogue
public static void AfficherMenuCatalogue()
{

    // Affichage

Console.WriteLine("\n-----------------------------------------------------------------------------------------------------
------");
    Console.WriteLine("                                                    MENU CATALOGUE
         ");

 Console.WriteLine("-----------------------------------------------------------------------------------------------------
-----");

    // On crée un objet Catalogue après avoir demandé les critères de tri à l'utilisateur
    Catalogue cat = TrierProjet();
    // On affiche le catalogue
    Catalogue.AfficherCatalogue(cat, maConnexion);

}
```

TrierProjet()

```
// But : Permet d'afficher à l'utilisateur les méthodes de tri et de recherche de projet
// Retourne : un objet Catalogue
public static Catalogue TrierProjet()
{
    // Options de tri
    Console.Write("1: Voir le catalogue complet \n" +
                "2: Trier par promotion \n" +
                "3: Trier par année \n" +
                "4: Trier par élève \n" +
                "5: Trier par type de projet \n" +
                "6: Trier par matière \n" +
                "7: Trier par mot clef \n" +
                "8: Trier par enseignant référent \n" +
                "9: Trier par intervenant externe \n" +
                "10 : Retour au menu principal\n");


    // Verifications de l'entree clavier + Feedback de ce que rentre l'utilisateur
    bool saisieIsValid = false;
    int caseSwitch = 0;
    while (!saisieIsValid)
    {
        Console.WriteLine("\nSaisir le numero de l'option que vous souhaitez");
        string saisie = Console.ReadLine();
        if (int.TryParse(saisie, out caseSwitch))
        {
            caseSwitch = int.Parse(saisie);
            if (caseSwitch > 0 && caseSwitch < 11)
            {
                saisieIsValid = true;
            }
        }
    }

    // Permet d'appeler la fonction permettant de realiser le tri demandé
    // les data vont recupérer les valeurs de tri pour un critere donné
    string critere = "", data1 = "", data2 = "";
```

```
    // Permet d'appeler la fonction permettant de realiser le tri demandé
    // les data vont recupérer les valeurs de tri pour un critere donné
    string critere = "", data1 = "", data2 = "";
    // Selon le choix on appelle un menu différent et on donne les valeurs aux variables
    switch (caseSwitch)
    {
        case 1: // Voir le catalogue complet
            // On ne fait rien
            break;
        case 2: // Trier par promotion
            int promo = MenuTriPromo();
            data1 = promo.ToString();
            critere = "promotion";
            break;
        case 3: // Trier par annee
            int annee = MenuTriAnnee();
            data1 = annee.ToString();
            critere = "annee";
            break;
        case 4: // Trier par élève
            critere = "eleve";
            string nomEleve, prenomEleve;
            MenuTriEleve(out nomEleve, out prenomEleve);
            data1 = nomEleve;
            data2 = prenomEleve;
            break;
        case 5: // Trier par type de projet
            critere = "typeProjet";
            string typePro = MenuTriTypeProjet();
            // 1 = Projet Pro, 2 = Projet Appli
            data1 = typePro;
            break;
        case 6: // Trier par matière
            critere = "matiere";
            string matiere = MenuTriMatieres();
            data1 = matiere;
            break;
        case 7: // Trier par mot clef
            critere = "motClef";
            string motClef = MenuTriMotsClef();
            data1 = motClef;
            break;
        case 8: // Trier par enseignant référent
            critere = "prof";
            string nomEnseignant, prenomEnseignant;
            MenuTriEnseignant(out nomEnseignant, out prenomEnseignant);
            data1 = nomEnseignant;
            data2 = prenomEnseignant;
            break;
        case 9: // Trier par intervenant
            critere = "intervenant";
            string nomIntervenant, prenomIntervenant;
            MenuTriIntervenant(out nomIntervenant, out prenomIntervenant);
            data1 = nomIntervenant;
            data2 = prenomIntervenant;
            break;
        case 10: // Afficher Menu principal
            AfficherMenu();
            break;
    }

    // On appelle la fonction creeCatalogue qui va consulter la BD pour recuperer les projets conrrespondant au
critere donne
    Catalogue cat = Catalogue.CreerCatalogue(maConnexion, critere, data1, data2);
    return cat;
}
```

MenuTriCritere()

Exemple avec la matière pour critère :

```
// But : Permet d'afficher le menu de tri par matieres
// Retourne : une chaine de caractère du nom de la matière
static string MenuTriMatieres()
{
    Console.WriteLine("\nSelectionner la matière souhaitée : \n\n");
    string matiere = "";
    // Requête permettant d'afficher, dans un switch, toutes les matieres recensees et de choisir celle souhaitée
    var cmdMatiere = new SQLiteCommand(maConnexion);
    cmdMatiere.CommandText = "SELECT DISTINCT matiere_lib, matiere_code FROM matiere";
    cmdMatiere.Prepare();
    SQLiteDataReader readerMatiere = cmdMatiere.ExecuteReader();

    ArrayList matiereLib = new ArrayList();
    //Compteur
    while (readerMatiere.Read())
    {
        // Affichage des informations de la matière
        Console.WriteLine("- " + readerMatiere["matiere_code"] + " : " + readerMatiere["matiere_lib"]);

        // On ajoute à la liste
        matiereLib.Add((string)readerMatiere["matiere_lib"]);
    }

    do
    {
        Console.WriteLine("Entrer le libellé de la matiere");
        matiere = Console.ReadLine();
    } while (!ValiderChaine("", matiere, matiereLib));

    return matiere;
}
```

CreerCatalogue()

```
// But : Permet de créer un objet catalogue
// Paramètres : Objet de connexion SQLite, un critere, et deux champs de données
// Retourne : Un objet catalogue
public static Catalogue CreerCatalogue(SQLiteConnection maConnexion, string critere, string data1, string data2)
{
    // Requete préparée
    var cmd = new SQLiteCommand(maConnexion);
    // Requête SQL selon le critère
    switch (critere)
    {
        case "annee":
            // Critère de tri = par année
            // transformation de la valeur donnee en argument en la valeur attendue en BD
            int annee = Int32.Parse(data1);

            // Requete
            cmd.CommandText = "SELECT * FROM projet WHERE projet_annee = @annee";
            cmd.Parameters.AddWithValue("@annee", annee);
            break;

        case "eleve":
            // Critère de tri = par eleve
            // Requete pour recuperer l'eleve
            var cmdEleve = new SQLiteCommand(maConnexion);
            cmdEleve.CommandText = "SELECT * FROM eleve WHERE eleve_nom = @nom AND eleve_prenom = @prenom";
            cmdEleve.Parameters.AddWithValue("@nom", data1);
            cmdEleve.Parameters.AddWithValue("@prenom", data2);

            cmdEleve.Prepare();
            //execute la requete et mets les lignes dans le reader.
            SQLiteDataReader readerEleve = cmdEleve.ExecuteReader();
            //pour chaque ligne
            while (readerEleve.Read())
            {
                // Recupere les projets que l'élève a realisés
                var cmdReal = new SQLiteCommand(maConnexion);
                cmdReal.CommandText = "SELECT * FROM realiser WHERE eleve_id = @eleve_id";
                cmdReal.Parameters.AddWithValue("@eleve_id", readerEleve["eleve_id"]);

                cmdReal.Prepare();
                SQLiteDataReader readerReal = cmdReal.ExecuteReader();
                if (readerReal.HasRows)
                {
                    //pour chaque ligne
                    while (readerReal.Read())
                    {
                        // Requete pour recuperer les infos du projet
                        cmd.CommandText = "SELECT * FROM projet WHERE projet_id = @projet_id";
                        cmd.Parameters.AddWithValue("@projet_id", readerReal["projet_id"]);
                    }
                }
                else
                {
                    // Requete par défaut
                    cmd.CommandText = "SELECT * FROM realiser WHERE eleve_id = @eleve_id";
                    cmd.Parameters.AddWithValue("@eleve_id", readerEleve["eleve_id"]);
                }
            }

            break;
```

```
case "promotion":
    // Requete pour recuperer les élèves appartenant à la promotion passee en parametre
    var cmdPromotion = new SQLiteCommand(maConnexion);
    cmdPromotion.CommandText = "SELECT * FROM eleve WHERE eleve_promotion = @promotion";
    cmdPromotion.Parameters.AddWithValue("@promotion", data1);

    cmdPromotion.Prepare();
    SQLiteDataReader readerPromotion = cmdPromotion.ExecuteReader();
    while (readerPromotion.Read()) // On parcourt ces élèves
    {
        // Recup les projets que l'éleve appartenant à cette promotion ont realisés
        var cmdReal = new SQLiteCommand(maConnexion);
        cmdReal.CommandText = "SELECT * FROM realiser WHERE eleve_id = @eleve_id";
        cmdReal.Parameters.AddWithValue("@eleve_id", readerPromotion["eleve_id"]);

        cmdReal.Prepare();
        SQLiteDataReader readerReal = cmdReal.ExecuteReader();
        if (readerReal.HasRows)
        {
            while (readerReal.Read())
            {
                // Requete pour le projet
                cmd.CommandText = "SELECT * FROM projet WHERE projet_id = @projet_id";
                cmd.Parameters.AddWithValue("@projet_id", readerReal["projet_id"]);
            }

        }
        else
        {
            // par défaut
            cmd.CommandText = "SELECT * FROM realiser WHERE eleve_id = @eleve_id";
            cmd.Parameters.AddWithValue("@eleve_id", readerPromotion["eleve_id"]);
        }

    }
    break;
case "motClef":
    // Requete pour recuperer les projets associés au mot clef
    var cmdMotClef = new SQLiteCommand(maConnexion);
    cmdMotClef.CommandText = "SELECT * FROM motClef WHERE motClef_lib = @mot";
    cmdMotClef.Parameters.AddWithValue("@mot", data1);

    cmdMotClef.Prepare();
    SQLiteDataReader readerMotCLef = cmdMotClef.ExecuteReader();
    while (readerMotCLef.Read())
    {
        // recuperer les projets qui ont pour mot clef celui donnée
        cmd.CommandText = "SELECT * FROM projet WHERE projet_id = @projet_id";
        cmd.Parameters.AddWithValue("@projet_id", readerMotCLef["projet_id"]);
    }
    break;

case "typeProjet":
    // Requete
    cmd.CommandText = "SELECT * FROM projet WHERE projet_libType = @typeP";
    cmd.Parameters.AddWithValue("@typeP", data1);
    break;

case "matiere":
    // Requete pour recuperer les matieres
    var cmdMatiere = new SQLiteCommand(maConnexion);
    cmdMatiere.CommandText = "SELECT * FROM matiere WHERE matiere_lib = @matiere";
    cmdMatiere.Parameters.AddWithValue("@matiere", data1);

    cmdMatiere.Prepare();
    SQLiteDataReader readerMatiere = cmdMatiere.ExecuteReader();
    while (readerMatiere.Read())
    {
        // Requete pour trouver les projets
        cmd.CommandText = "SELECT * FROM projet WHERE projet_idMatiere = @matiere_id";
        cmd.Parameters.AddWithValue("@matiere_id", readerMatiere["matiere_id"]);
    }
    break;
```

```
        case "prof":
            // Requete pour recuperer les profs
            var cmdProf = new SQLiteCommand(maConnexion);
            cmdProf.CommandText = "SELECT * FROM prof WHERE prof_nom = @prof_nom AND prof_prenom = @prof_prenom";
            cmdProf.Parameters.AddWithValue("@prof_nom", data1);
            cmdProf.Parameters.AddWithValue("@prof_prenom", data2);

            cmdProf.Prepare();
            SQLiteDataReader readerProf = cmdProf.ExecuteReader();
            while (readerProf.Read())
            {
                // Requete pour trouver les projets
                cmd.CommandText = "SELECT * FROM projet WHERE prof_id = @prof_id";
                cmd.Parameters.AddWithValue("@prof_id", readerProf["prof_id"]);
            }
            break;

        case "intervenant":
            // Requete pour recuperer les intervenants
            var cmdInt = new SQLiteCommand(maConnexion);
            cmdInt.CommandText = "SELECT * FROM intervenant WHERE intervenant_nom = @intervenant_nom AND
intervenant_prenom = @intervenant_prenom";
            cmdInt.Parameters.AddWithValue("@intervenant_nom", data1);
            cmdInt.Parameters.AddWithValue("@intervenant_prenom", data2);

            cmdInt.Prepare();
            SQLiteDataReader readerInt = cmdInt.ExecuteReader();
            while (readerInt.Read())
            {
                // Requete pour trouver les projets
                cmd.CommandText = "SELECT * FROM projet WHERE projet_idClientInt = @intervenant_id OR
projet_idTuteurInt = @intervenant_id";
                cmd.Parameters.AddWithValue("@intervenant_id", readerInt["intervenant_id"]);
            }

            break;

        default:
            // Requete pour trouver tous les projets
            cmd.CommandText = "SELECT * FROM projet";
            break;
    }

    // on execute la requête
    cmd.Prepare();
    SQLiteDataReader reader = cmd.ExecuteReader();

    // On cree la liste de projets en fonctions des resultats obtenus precedents inscrits dans le reader
    List<Projet.Projet> listeProjets = Projet.Projet.CreerListeProjets(maConnexion, reader);
    // On créé un catalogue recensant les projets triés selon le ou les criteres precisés
    Catalogue cat = new Catalogue(listeProjets);
    // On retourne le catalogue
    return cat;

}
```

CreerListeProjets()

```
// But : Permet de créer une liste de projet
// Paramètres : Objet de connexion SQLite, un objet reader
// Retourne : Une liste contenant les projets
public static List<Projet> CreerListeProjets(SQLiteConnection maConnexion, SQLiteDataReader reader)
{
    List<Projet> listeProjets = new List<Projet>();
    while (reader.Read())
    {
        int projet_id = (int)reader["projet_id"];
        // Créer les livrables associées
        ArrayList listeLivrables = Livrable.CreerListeLivrables(projet_id, maConnexion);
        // Créer les mots clefs associées
        ArrayList listeMotsClefs = Program.CreerListeMotsClefs(projet_id, maConnexion);
        // Crée le prof référant
        Personne.Prof profReferant = Prof.CreerProf(projet_id, maConnexion);
        // Crée la liste des élèves
        ArrayList listeEleves = Eleve.CreerListeEleves(projet_id, maConnexion);
        // Crée la lise des intervenants
        ArrayList listeIntervenants = Intervenant.CreerListeIntervenants(projet_id, maConnexion);

        Boolean sujetLibre = false;
        if ((int)reader["projet_sujetLibre"] == 1) sujetLibre = true;

        Boolean collectif = false;
        if ((int)reader["projet_collectif"] == 1) collectif = true;


        Projet p = new Projet((int)reader["projet_id"], DateTime.Parse((string)reader["projet_dateDebut"]),
DateTime.Parse((string)reader["projet_dateFin"]), sujetLibre,
            listeLivrables, (int)reader["projet_annee"], listeMotsClefs, (string)reader["projet_lib"],
(string)reader["projet_libSujet"],
        profReferant, listeEleves, collectif, (double)reader["projet_noteProjet"], (string)reader["projet_libType"],
listeIntervenants);

        listeProjets.Add(p);
    }

    return listeProjets;

}
```

ModifierProjetBD()

```
// But : Permet de modifier un projet en BD
// Paramètres : Objet de connexion SQLite, un long projet_id
public static void ModifierProjetBD(SQLiteConnection maConnexion, long projet_id)
{
    // On demande à l'utilisateur ce qu'il veut modifier
    Console.Write("\nVous voulez modifier : \n" +
                "1: Le titre du projet \n" +
                "2: Le type du projet\n" +
                "3: L'année du projet\n" +
                "4: La note du projet\n" +
                "5: Les élèves du projet (cela les supprime tous pour pouvoir en ajouter de nouveau)\n");

    // Verifications de l'entree clavier
    bool saisieIsValid = false;
    int caseSwitch = 0;
    while (!saisieIsValid)
    {
        Console.WriteLine("\nSaisir le numero de l'option que vous souhaitez modifier");
        string saisie = Console.ReadLine();
        if (int.TryParse(saisie, out caseSwitch))
        {
            caseSwitch = int.Parse(saisie);
            if (caseSwitch > 0 && caseSwitch < 6)
            {
                saisieIsValid = true;
            }
        }
    }

    // Pour savoir quand la chaine entrée de modification correspond à ce que l'on attend
    bool modif = true;
    string newChaine = "";
    if( caseSwitch != 5) // Si on ne modifie pas les élèves
    {
        do
        {
            Console.WriteLine("\nEntrer votre modification : ");

            newChaine = Console.ReadLine();
            if (newChaine.Length > 1 || caseSwitch == 4)
            {
                if (caseSwitch == 3 && Program.ValiderChaine("annee", newChaine)) // si la modif concerne l'année
                {
                    modif = false;
                }
                else if (caseSwitch == 4 && Program.ValiderChaine("note", newChaine)) // si la modif concerne le
type
                {
                    modif = false;
                }
                else if(caseSwitch != 3 && caseSwitch != 4)// Pour les autres cas qui n'attendent pas de chaine
particulière
                {
                    modif = false;
                }
            }
        } while (modif);
    }

    int nbLigne = 0;
```

```
    // Requete préparée pour modifier la table selon le choix de l'utilisateur
    var cmd = new SQLiteCommand(maConnexion);
    switch (caseSwitch)
    {
        case 1: // Titre du projet
            // Requete
            cmd.CommandText = "UPDATE projet SET projet_lib = @new WHERE projet_id = @projet_id";
            break;
        case 2: // Type du projet
            // Requete
            cmd.CommandText = "UPDATE projet SET projet_libType = @new WHERE projet_id = @projet_id";
            break;
        case 3: // Annnee du projet
            // Requete
            cmd.CommandText = "UPDATE projet SET projet_annee = @new WHERE projet_id = @projet_id";
            break;
        case 4: // Note du projet
            // Requete
            cmd.CommandText = "UPDATE projet SET projet_noteProjet = @new WHERE projet_id = @projet_id";
            break;
        case 5: // Eleves du projet
            // Requete pour supprimer tous les élèves ayant réalisé ce projet
            cmd.CommandText = "DELETE FROM realiser WHERE projet_id = @projet_id";
            cmd.Parameters.AddWithValue("@projet_id", projet_id);

            cmd.Prepare();
            cmd.ExecuteNonQuery();

            // Ensuite on ajoute les nouveaux élèves
            ArrayList idEleve = Eleve.ListeElevesBD(maConnexion);

            // On regarde le nombre minimum d'élèves à ajouter = savoir si un projet est collectif ou individuel
            var cmdCollectif = new SQLiteCommand(maConnexion);
            cmdCollectif.CommandText = "SELECT * FROM projet WHERE projet_id = @projet_id";
            cmdCollectif.Parameters.AddWithValue("@projet_id", projet_id);
            cmdCollectif.Prepare();
            SQLiteDataReader readerCollectif = cmdCollectif.ExecuteReader();

            // Variable pour savoir si c'est collectif ou individuel
            int collectif = 0;
            while (readerCollectif.Read())
            {
                collectif = (int)readerCollectif["projet_collectif"]; // Si 0 : individuel, si  1 : collectif
            }

            int nbEleveAAjouter = 1;
            if (collectif == 1) nbEleveAAjouter = 2;

            string rep = "";

            // Compteur pour vérifier que l'on entre au moins un élève
            bool encore = true;
            bool ajout = true;
            while (encore)
            {
                if (nbEleveAAjouter <= 0 && collectif != 1) encore = false; // S'il s'agit d'un projet individuel
                else
                {
                    do
                    {
                        Console.WriteLine("\nEntrer le numéro de l'élève que vous souhaitez ajouter au projet,
appuyez sur entrée pour arrêter d'en ajouter");
                        rep = Console.ReadLine();
                        if (nbEleveAAjouter <= 0 && rep.Equals("")) break;

                    } while (!Program.ValiderChaine("id", rep, idEleve));

                    if (!rep.Equals(""))
                    {
                        // On regarde les eleve_id déjà présent dans la table Réaliser
                        var cmdReal = new SQLiteCommand(maConnexion);
                        cmdReal.CommandText = "SELECT * FROM realiser WHERE projet_id = @projet_id AND eleve_id =
@eleve_id";
                        cmdReal.Parameters.AddWithValue("@projet_id", projet_id);
                        cmdReal.Parameters.AddWithValue("@eleve_id", Int32.Parse(rep));
                        cmdReal.Prepare();
                        SQLiteDataReader readerReal = cmdReal.ExecuteReader();
                        if (readerReal.Read()) // Si un résultat a été trouvé c'est que l'élève a déjà été ajouté
                        {
                            ajout = false;
                            Console.WriteLine("Cet élève a déjà été ajouté");
                        }
                        else
```

```
                        }
                        else
                        {
                            ajout = true;
                        }
                    }

                    if (!rep.Equals("") && ajout) // Si la réponse n'est pas vide et que l'on peut ajouter
                    {
                        // Pour recupérer les données de l'élève
                        var cmdEleve = new SQLiteCommand(maConnexion);
                        cmdEleve.CommandText = "SELECT * FROM eleve WHERE eleve_id = @eleve_id";
                        cmdEleve.Parameters.AddWithValue("@eleve_id", Int32.Parse(rep));
                        cmdEleve.Prepare();
                        SQLiteDataReader readerEleve = cmdEleve.ExecuteReader();
                        while (readerEleve.Read())
                        {
                            // lien sur realiser pour la BD
                           Eleve.AjouterRealEleveBD(maConnexion, projet_id, (int)readerEleve["eleve_id"]);
                            Console.WriteLine("Eleve ajouté");
                        }

                        // On décremente le compteur d'élève à ajouter au minimum
                        nbEleveAAjouter--;
                    }
                    else if (nbEleveAAjouter <= 0 && rep.Equals("")) encore = false; // modifier le nombre minimum
d'élève a été rentré et on décide d'arreter
                }
            }
            break;
    }

    if(caseSwitch != 5) // si la modif ne concerne pas les élèves
    {
        cmd.Parameters.AddWithValue("@new", newChaine);
        cmd.Parameters.AddWithValue("@projet_id", projet_id);

        cmd.Prepare();
        nbLigne = cmd.ExecuteNonQuery();
    }

    if (nbLigne == 1) // Vérification que la modification a été réalisée avec succès
    {
        Console.WriteLine("\nModification réalisée");
    }
    else if (caseSwitch != 5)
    {
        Console.WriteLine("\nErreur dans la modification");
    }
}
```

AjouterMatiere()

```
// But : Permet d'ajouter un matière
// Paramètres : Objet de connexion SQLite
public static void AjouterMatiere(SQLiteConnection maConnexion)
{
    // dernier id matiere
    long lastId = ChercherDernierIdMatiere(maConnexion) + 1;

    // Nom de la matière
    string nomMatiere = "";
    do
    {
        Console.WriteLine("Entrer le nom de la matière que vous souhaitez ajouter : ");
        nomMatiere = Console.ReadLine();
    } while (nomMatiere.Length < 1);

    // Code de la matière
    string code = "";
    do
    {
        Console.WriteLine("Entrer le code de la matière correspondant :");
        code = Console.ReadLine();
    } while (code.Length < 1);
    // Création de l'objet Matière
    Matiere m = new Matiere((int)lastId, nomMatiere, code);
    // On appelle la fonction pour ajouter une matière en BD
    AjouterMatiereBD(maConnexion, m);
}
```

ChercherDernierIdMatiere()

```
// But : Permet de chercher le dernier id de la table matiere
// Paramètres : Objet de connexion SQLite
// Retourne : un long id de la matière
static long ChercherDernierIdMatiere(SQLiteConnection maConnexion)
{
    // Id de la matiere
    long lastId = 0;
    // Requête pour chercher le dernier id
    var cmdId = new SQLiteCommand(maConnexion);
    cmdId.CommandText = "SELECT MAX(matiere_id) as matiere_idMax FROM matiere";
    cmdId.Prepare();
    SQLiteDataReader readerId = cmdId.ExecuteReader();

    while (readerId.Read())
    {
        lastId = (long)readerId["matiere_idMax"];
    }
    return lastId;
}
```

AjouterMatiereBD()

```
// But : Permet d'ajouter une matière en BD
// Paramètres : Objet de connexion SQLite, un objet Matiere
public static void AjouterMatiereBD(SQLiteConnection maConnexion, Matiere m)
{
    // On vérifie que la matière n'a pas déjà été ajoutée
    var cmd = new SQLiteCommand(maConnexion);
    cmd.CommandText = "SELECT * FROM matiere WHERE matiere_code = @codeM";
    cmd.Parameters.AddWithValue("@codeM", m.CodeMatiere);
    cmd.Prepare();
    SQLiteDataReader reader = cmd.ExecuteReader();

    bool existe = false;

    while (reader.Read())
    {
        existe = true;
    }
    reader.Close();
    if (existe)
    {
        Console.WriteLine("Ajout impossible, la matières existe déjà");
    }
    else // la matière n'existe pas
    {
        // Insertion de données dans la table matiere
        cmd.CommandText = "INSERT INTO matiere VALUES (@idM, @nomM, @codeM)";
        cmd.Parameters.AddWithValue("@idM", m.IdMatiere);
        cmd.Parameters.AddWithValue("@nomM", m.NomMatiere);
        cmd.Parameters.AddWithValue("@codeM", m.CodeMatiere);
        cmd.Prepare();
        if (cmd.ExecuteNonQuery() == 1)
        {
            Console.WriteLine("Insertion correctement effectuée");
        }
        else
        {
            Console.WriteLine("Une erreur est survenue dans l'insertion");
        }
    }
}
```

ValiderChaine()

```
// But : Permet de valider une chaîne de caractères rentrée par l'utilisateur selon le type de valeur attendu
// Paramètres : le type de la chaîne attendu, la chaîne entrée par l'utilisateur, et facultativement la liste des
valeurs possibles
// Retourne : Vrai si la chaîne est bonne, faux sinon
public static bool ValiderChaine(string type, string chaine, ArrayList liste = null)
{
    // Initialisation de l'objet Regex
    Regex myRegex = new Regex("");
    // Initialisation du booléen
    bool verif = true;
    // Selon le type
    switch (type)
    {
        case "date":
            // format : YYYY-MM-DD
            myRegex = new Regex(@"^\d{4}-((0[1-9])|(1[012]))-((0[1-9]|[12]\d)|3[01])$");
            break;
        case "bool":
            // format : 0 ou 1
            myRegex = new Regex(@"^[0-1]$");
            break;
        case "annee":
            // format : YYYY
            myRegex = new Regex(@"^\d{4}$");
            break;
        case "id":
            // format : chiffres
            myRegex = new Regex(@"^[0-9]*$");
            break;
        case "note":
            // >0 <20
            myRegex = new Regex(@"^([0-9]|1[0-9]|20)$");
            break;
        case "":
            // Pas de format particulier attendu
            break;
    }

    if (!type.Equals(""))
    {

        // Si la chaîne correspond au Regex
        verif = myRegex.IsMatch(chaine);
    }

    if (verif) // Si la chaîne est bonnne
    {
        if (liste != null) // On regarde si le paramètre liste est null
        {
            if(type.Equals("id") || type.Equals("annee")) // si le type est un id
            {
                // On regarde si l'id entré par l'utilisateur est présent dans cette chaîne ou pas
                verif = liste.Contains(Int32.Parse(chaine));
            }
            else
            {
                verif = liste.Contains(chaine);
            }
        }
    }
    // Retourne true ou false selon la vérification
    return verif;
}
```

VerifierDonneesApresFermetureBrusque()

```
// But : Permet de Supprimer les données incohérentes dans la BD
// Paramètres : Objet de connexion SQLite
public static void VerifierDonneesApresFermetureBrusque(SQLiteConnection maConnexion)
{
    // Tableau contenant le nom des tables que l'on veut vérifier
    string[] table = new string[] { "rendu", "realiser", "intervient", "motClef" };

    long projet_id = Projet.Projet.ChercherDernierIdProjet(maConnexion);
    // On passe à l'id supérieur qui en théorie n'existe pas
    projet_id++;

    foreach (string element in table) // Pour chaque élément dans la table
    {
        // On regarde si une table a un projet_id supérieur au dernier projet ajouté
        string sql = "SELECT * FROM " + element + " WHERE projet_id = " + projet_id;
        // Requete préparée
        SQLiteCommand cmd = new SQLiteCommand(sql, maConnexion);
        // Que l'on éxecute
        SQLiteDataReader reader = cmd.ExecuteReader();
        // Si un résultat a été trouvé
        if(reader.Read())
        {
            // On supprime les données
            string sql1 = "DELETE FROM " + element + " WHERE projet_id = " + projet_id;
            // Requete préparée
            SQLiteCommand commande0 = new SQLiteCommand(sql1, maConnexion);
            // Que l'on éxecute
            commande0.ExecuteNonQuery();
        }
    }
}
```