Compte-rendu jalon 3

LEGUAY Emerance (FreyjaLgy sur Github)
MULLER Maxime (chasseur2dr sur Github)
MULARD Andreas (Kaputchino sur Github)
BROUARD Romain (romain327 sur Github)

Table des matières

Base de données

Dans le précédent compte rendu, il a été mentionné que de nombreuses modifications de la base de données ont dû être effectuées suite au rendez-vous avec M. Jiménès. Cela a engendré des modifications dans le code au niveau des requêtes, ainsi que des modifications sur le MCD et MLD pour que tous les documents concernant la base de données soient parfaitement à jour et éviter toute erreur liée à un problème d'orthographe. En effet, nous avons renommé plusieurs tables et attributs, notamment après avoir constaté que certaines tables du MLD sur lequel nous nous basions pour coder n'avaient pas de s, alors que celles de la BDD réelle en comportaient. Nous avons dû ajuster certaines tailles de données : par exemple, nous avons constaté que les titres d'ouvrage qui nous étaient fournis étaient beaucoup plus longs que ce que nous avions prévu. Une limite de 50 caractères était donc trop petite et empêchait le bon stockage de cette donnée.

Nous n'en avons pas eu l'occasion, mais il aurait été nécessaire de rencontrer à nouveau M. Jiménès car en effet, certaines données semblent étonnantes, comme un graveur responsable de la grande majorité des lettrines. En effet il existe dans la table t_bois un attribut graveur, mais il existe aussi une table corresp_bois_graveur qui lie les graveurs aux bois (incluant les lettrines), mais ces données semblent tout aussi étranges..
Enfin, nous avons extrait les métadonnées de la base t_bois. voir annexe.

Réalisation de l'interface graphique

Les fichiers FXML qui composent l'interface graphique ont été entièrement codés à l'aide du logiciel Scene Builder. C'est un éditeur de fichier FXML qui a l'avantage d'être extrêmement simple à utiliser, car il suffit de drag and drop les éléments composant l'interface là où on souhaite les placer, et le code est généré automatiquement.
L'interface se compose donc d'une trentaine de vues, chaque vue correspondante à une "page" de l'interface, chaque changement de scène étant considéré comme une page.
Le choix de coder une classe contrôleur par vue était purement une question de propreté. En effet, mettre tout le code permettant de contrôler l'interface dans la même classe aurait résulté en un fichier illisible de plus d'une centaine de méthodes dans lequel il aurait été compliqué de s'y retrouver, et donc compliqué à tester et à déboguer. La dernière version de Scene Builder est configurée pour JavaFX 19 néanmoins, nous avons utilisé JavaFX 17, ainsi nous avions des warning sans importance pour nous prévenir de la différence de version.
L'interface est quasi intégralement responsive, exception faite des pages FxInterface<objet métier> qui n'ont pas été retouchées.

Débogage

Ce jalon étant le dernier avant le rendu du projet, nous en avons consacré une bonne partie à faire du débogage. Nous avons donc vérifié les différentes classes pour s'assurer de leur bon fonctionnement : vérifier qu'on ne puisse pas faire d'injections SQL, vérifier qu'il y ait des conditions pour indiquer le comportement que le programme doit suivre lorsqu'un objet qu'il doit manipuler est null, et bien d'autres.
Ce sont probablement les DAO qui ont demandé le plus de travail, puisque ce sont des classes de fonctionnement cruciales à notre application qui n'ont pû être testées que tardivement suite aux nombreuses modifications de base de données et à certaines méthodes qui n'étaient pas encore totalement finies en début de jalon.
Les tests des DAO pouvaient se faire en parallèle de la progression sur l'interface graphique, mais les derniers tests de l'ensemble de l'application ont donc dû être réalisés à la toute fin après complétion de l'application.

En raison du temps passé à coder les DAOs, les tests de l'application et la recherche de bugs lors de son fonctionnement ont débuté très tard, pour la simple raison que nous n'avions pas une application pleinement fonctionnelle avant. En effet nous avions une interface graphique, des DAOs qui fonctionnaient quasiment tous, une base de données finale, mais rien de tout ça n'était encore lié. Nous avons simplement testé les différentes fonctionnalités, tenté d'obtenir des comportements anormaux, et vérifié que ce que nous obtenions était bien cohérent avec le fonctionnement attendu et la base de données.

En fin de jalon, le débogage s'est concentré sur l'interface. En effet, même si les DAO et objets métiers ont été testés durant ce jalon et le 2ème, cela ne garantit pas que l'interface est dépourvue de bugs, que ce soit propre à elles mêmes (par exemple un mauvais type d'event) ou alors issus de liaison entre les interfaces et les dao (comme un bouton lié à la mauvaise méthode, ou alors qu'un objet sous la mauvaise forme est fourni à une méthode).

Pour nous organiser nous avons mis en place un système de ticket inspiré de la méthode kanban : ainsi dès qu'un bug était trouvé, il était signalé pour éviter que deux personnes débugguent le même bug. Il était joint la méthode pour rencontrer le bug, sa gravité, son statut (en attente de correction, en cours de correction et par qui, corrigé). De plus, au pic d'activité les appels entre membres d'équipe étaient bi-journaliers.

Java

La version finale de cette application est codée majoritairement en java (plus de 97% d'après Github) et l'utilisation de ce langage en situation de projet nous a permis de résoudre des problèmes différents de ceux vus en cours, de découvrir de nouveaux objets souvent plus adaptés, tels que les StringBuilder pour concaténer des chaînes, bien plus optimisés qu'un simple + entre deux String.

Il a été nécessaire de modifier une partie des classes et des interfaces, c'est ainsi justement que l'exportTypeInterface est devenu une classe abstraite, pour simplifier la

création de nouvelles méthodes d'export. Pour le moment seul CSV est disponible, car nous avons défini comme non prioritaire la création d'autre méthode ainsi que la nécessité d'installer des bibliothèques externes qui auraient rendu l'installation pour le client plus long, sans apporter un avantage significatif grâce à la fonctionnalité d'export de phpMyAdmin.

Comme expliqué dans la partie non java ci-dessous, une partie de l'application est consacrée à la génération et l'affichage d'un nuage de tags. Le nuage est généré par python, mais la partie la plus compliquée à coder pour ce nuage fût une classe java. La partie nuage de tags marche de la manière suivante : la méthode tagAndSize de la classe TagDAO s'occupe de récupérer les tags contenus dans la base de données, ainsi que le nombre de lettrines qui leur sont associées. La classe cloudWordGenerator s'occupe de récupérer le résultat de la méthode tagAndSize (qui renvoie un String), de l'enregistrer au format texte pour que le programme python puisse générer le nuage, puis d'exécuter le programme python. Cette partie d'exécution du programme python était la plus difficile à coder car elle faisait appel à des objets Java jamais manipulés : les ProcessBuilder et les BufferedReader. Le ProcessBuilder permet de construire un processus en passant la commande et les paramètres du processus à construire en paramètre. Dans notre code donc, on a un

```
ProcessBuilder process = new ProcessBuilder("python",
"src/main/wordcloud/cloud.py").inheritIO();
```

La commande est donc en python, et le paramètre est le chemin du fichier python à exécuter. Cela revient à taper dans un terminal la commande :

python src/main/wordcloud/cloud.py

```
Process p = process.start();
```

Cette ligne permet de créer et lancer un processus à partir du ProcessBuilder créé précédemment.
Il faut ensuite lire les données récupérées par l'exécution du programme python. C'est le rôle du BufferedReader.

```
BufferedReader Buffered_Reader = new BufferedReader(
        new InputStreamReader(
                p.getInputStream()
        ));
```

Ces lignes permettent de récupérer dans le BufferedReader une chaîne de caractères, qui sont les octets résultant de l'exécution du processus, convertis en char par le InputStreamReader. On a donc un BufferedReader qui contient une chaîne de char.

```
String Output_line = "";
while ((Output_line = Buffered_Reader.readLine()) != null) {
   System.out.println(Output_line);
}
```

Une fois l'image obtenue et enregistrée en local, le contrôleur du nuage de tags, la classe FXNuageControleur s'occupe de l'afficher lorsqu'on clique sur le bouton nuage.

Une bonne proportion des classes est représentée par les contrôleurs des scènes fxml. Leur programmation n'était pas très longue et nous a permis de manipuler de nouveaux objets type Button, ListView...

Partie non java

Bien que la majorité de notre projet ait été codée en java, nous avons quelques fonctionnalités qui ont nécessité l'utilisation d'autres langages de programmation. Ainsi un programme permettant de générer un nuage de mots a été codé en python, et des scripts shell permettant d'exporter des données ainsi que d'installer les bibliothèques python nécessaires au bon fonctionnement du nuage de mots ont également été réalisés. Le python générant le nuage de mot est en réalité plutôt simple, il permet d'enregistrer une image créée à partir d'un fichier texte. L'image générée est le nuage de mot, créé grâce à la librairie wordcloud de python. L'image est ensuite enregistrée grâce à la librairie matplotlib.

Nom

Le nom "Projet Colmar" était censé être temporaire malgré la beauté pittoresque de la ville alsacienne, néanmoins nous n'avons jamais trouvé un nom cohérent. Nous laissons le soin à qui le souhaite de trouver un nom plus approprié.

Style et CSS

Il aurait dû être ajouté des thèmes à l'application, un white mode que l'on voit dans la quasi intégralité de l'application, un dark mode, que l'on a commencé à coder mais qui n'a pas été implémenté car trop précoce, et un mode vieux livre, entièrement customisé et en référence avec le thème de l'application. Néanmoins le développement de ce style à été sacrifié pour le débogage. Un fragment de ce style est visible sur la page de connexion.

Comparaison

A l'issue du premier jalon, nous avions présenté un diagramme de classes, résultat de notre réflexion sur la conception de l'application. Même si ce diagramme se voulait définitif, ce ne fut évidemment pas le cas et certaines classes imprévues au départ ont été implémentées. Par exemple, la classe cloudWordGenerator a été ajoutée un peu sur un coup de tête lorsque nous nous sommes rendu compte qu'un nuage de mots n'était pas très dur à implémenter, bien que cette option n'avait pas été envisagée au départ. On se retrouve donc avec un diagramme de classe un peu différent de celui pensé il y a quelques mois, et la différence entre les deux montre l'évolution de notre vision de l'application au fur et à mesure de l'avancement de son développement. Ces deux diagrammes sont disponibles en Annexe. En prenant du recul, il est clair que certaines

fonctionnalités n'ont pas été suffisamment réfléchies dès le départ, par exemple la méthode de connexion à la base de données ainsi que la connexion à l'application qui ont dû être modifiées à trois jours de la deadline pour ce dernier jalon. Ce sont tout autant de maladresses et d'imprécisions un peu partout dans le code qui nous ont amenées à devoir résoudre autant de bugs.
Par conséquent, la documentation précédemment fournie n'est plus d'actualité, suite aux modifications et ajouts de méthodes qui ont été réalisés lors de ce jalon.

Autorisation

Un système d'autorisation a été implémenté, il y a pour le moment 3 roles : admin, modérateur et chercheur. L'administrateur possède tous les droits, tandis que le modérateur ne peut créer de nouveaux utilisateurs, ni exporter les logs. Le chercheur a uniquement le droit de consulter les données.

Fonctionnement de l'application

Lancement :
Pour lancer l'application nous avons fait le choix d'utiliser un fichier configfile qui est lu au lancement. Il sert à créer la connexion à la base de données en obtenant l'adresse, le login et potentiellement le mot de passe. Si la connexion a pu être établie, alors l'application rentre dans la seconde phase.

Connexion :
Une fois que la connexion avec la base a pu être établie, l'utilisateur peut entrer son adresse e-mail et son mot de passe. Le mot de passe sera crypté et comparé avec celui stocké dans la base de données, et si les deux valeurs correspondent, alors le logiciel rentre dans la dernière phase.

Utilisation :
À partir de ce stade, l'utilisateur est "libre" il n'a plus de mot de passe à rentrer, et peut profiter de l'intégralité des fonctionnalités de l'application en fonction de son statut, sauf le nuage qui quant à lui dépend de python et de bibliothèques tierces.

Fin d'utilisation:
L'utilisateur peut utiliser le bouton de la fenêtre ou le bouton dans la barre des tâches, peu importe.

Licence

Pistes d'amélioration

Les changements qui suivent sont une liste des fonctionnalités, plus ou moins avancées, qui ont été pensées mais qui ont dû être sacrifiées pour manque de temps, ou difficulté d'implémentation et de création de l'interface. Elles sont aussi ici pour témoigner du recul vis à vis de notre projet.

- Sécurité
Avoir une seule connexion à la base de données est risquée, néanmoins on ne voit pas pourquoi quelqu'un voudrait pirater une base de données de lettrines accessible en ligne. Une faille plus critique est le stockage non crypté du mot de passe du login de la base de données dans le fichier configfile.

- Les styles CSS
Nous avons déjà parlé des styles CSS, une partie du code a d'ores et déjà été écrite, et commentée en vue d'une complétion ultérieure. Néanmoins, ce n'est pas une modification essentielle, mais l'aspect esthétique d'une application la rend plus accueillante donc plus facile à appréhender.

- Gestion des erreurs
Une meilleure gestion des erreurs, pour le moment quand une opération est illégale, l'application est vague voir inexistante .

- Redéfinition des autorisations
Les autorisations semblent un peu étranges, avec les chercheurs n'ayant en pratique que peu de droits. De même, un super-utilisateur unique qui pourrait supprimer les administrateurs devrait être envisagé si l'application est utilisée par une organisation regroupant de nombreuses personnes.

- Image
Pour le moment la mise en ligne des lettrines se fait soit sur un serveur à part, soit en mettant dans le fichier l'image, puis en notant le chemin d'accès. Un système plus simple d'utilisation pourrait être conçu.
Enfin un système pour afficher les images dans les ListViews serait un grand plus pour les lettrines.

-Recherche
Les outils de recherche actuels sont fonctionnels, néanmoins, ils peuvent être améliorés. La recherche par id est certes une solution, simple mais peu pratique. Le problème de l'usage de l'identifiant est aussi présent pour ajouter des liens entre les éléments comme lettrine - tag.
La recherche par métadonnées pour les lettrines : la recherche par métadonnées, que ce soit l'intitulé mais aussi la valeur, devrait être ajoutée. La recherche par métadonnées est codée mais pas implémentée dans l'interface.

-Interface
Quelques pages ne sont pas responsives, le simple ajout de quelques borderpane devrait être suffisant.

- Tags
La possibilité de taguer les ouvrages et personnes pourrait être un plus au niveau de la recherche (pouvoir taguer des lettrines, personnes et ouvrages du même tag pour constituer des groupes pourrait avoir un sens pour les chercheurs travaillant sur le sujet).

-Log
Une erreur de conception devrait être résolue : les logs sont au format DD-MM-YYYY alors que les heures, minutes et secondes sont nécessaires, nous avons fait le choix de ne pas modifier la base de données si peu de temps avant la deadline pour éviter les accès. De plus, un log local en cas de déconnexion avec la base pourrait être utile. Enfin les logs pourraient donner plus d'informations.

-Export
Il a été conçu une manière simple de créer de nouvelle méthode d'export, si un jour quelqu'un à la volonté de créer plus.

-Téléchargement des bibliothèques python
Les scripts shells et batch ont été écrits initialement pour pouvoir être exécutés au lancement de l'application, cela aurait permis d'automatiser le téléchargement des bibliothèques python. On n'a pas implémenté cette fonction par manque de temps.

Critique du travail

Un gros point négatif sur notre code tout au long de l'application est le manque de tests unitaires. En effet, nous avons testé seulement 4 classes, de manière peu rigoureuse, et la présence de nombreux bugs ainsi que le temps passé à déboguer les DAOs en sont une conséquence.

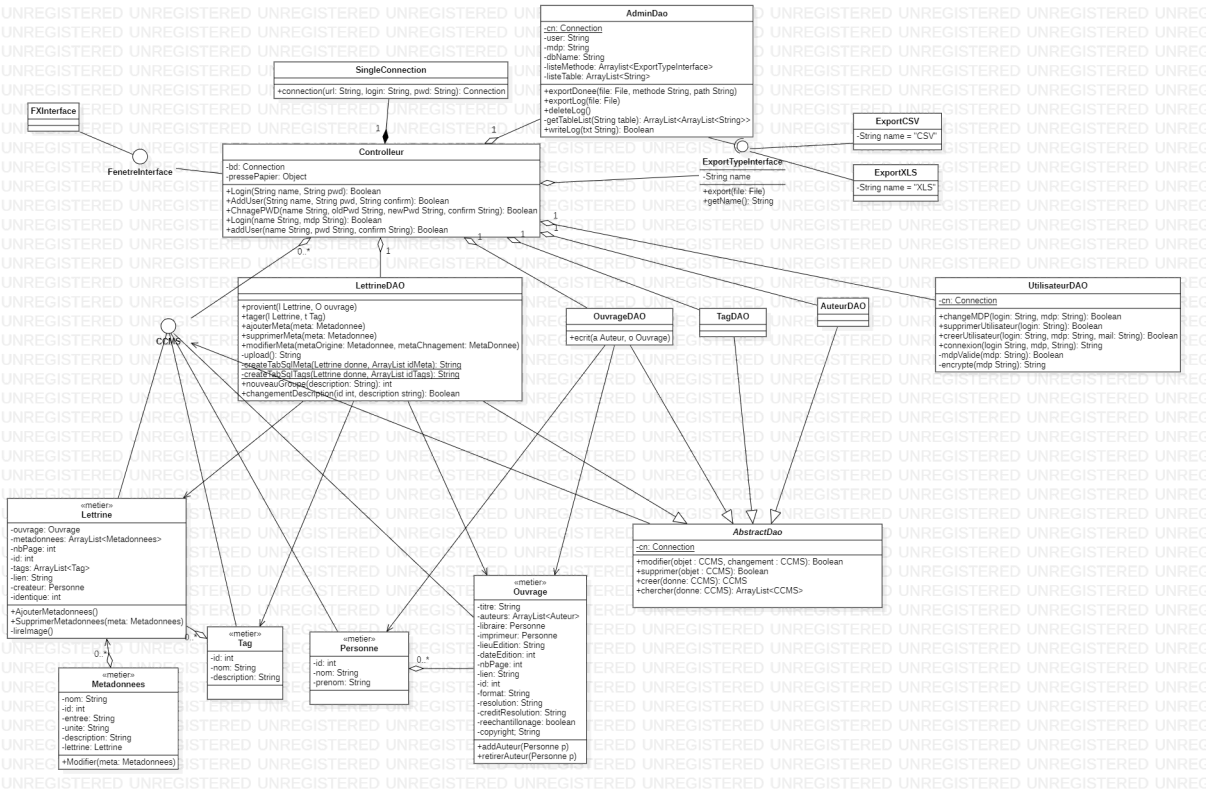
Conclusion

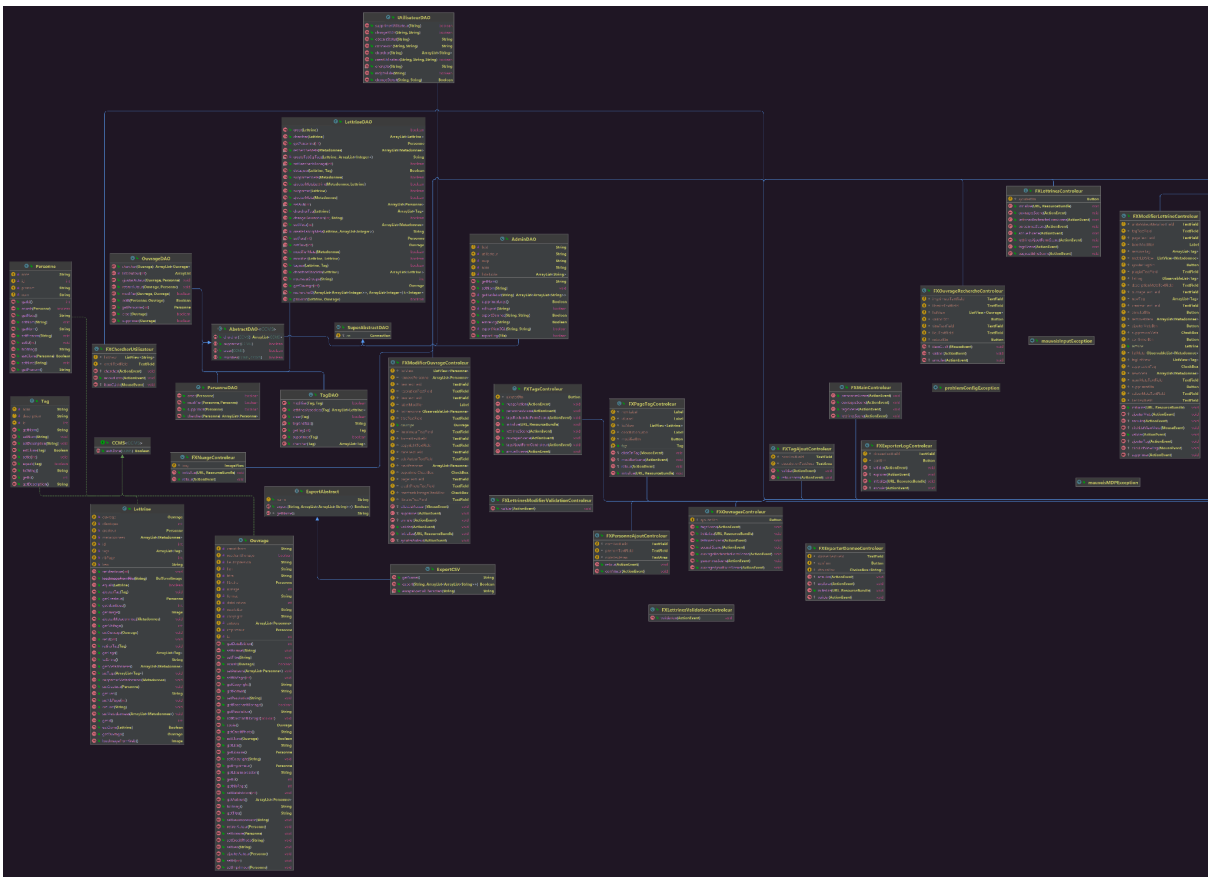
L'ambiance au sein du groupe était à l'image de la fin du jalon 2, organisée et structurée. La communication était claire et précise et les modifications apportées au projet étaient directement transmises au reste du groupe. Le système de ticket a grandement accéléré le débogage de l'application. Nous avons donc grandement appris de ce projet et progressé, et cela dans de nombreuses catégories : la communication, l'organisation, la création d'une application, ... Et ces compétences nous permettront lors de prochains projets d'être d'autant plus rapides et efficaces.

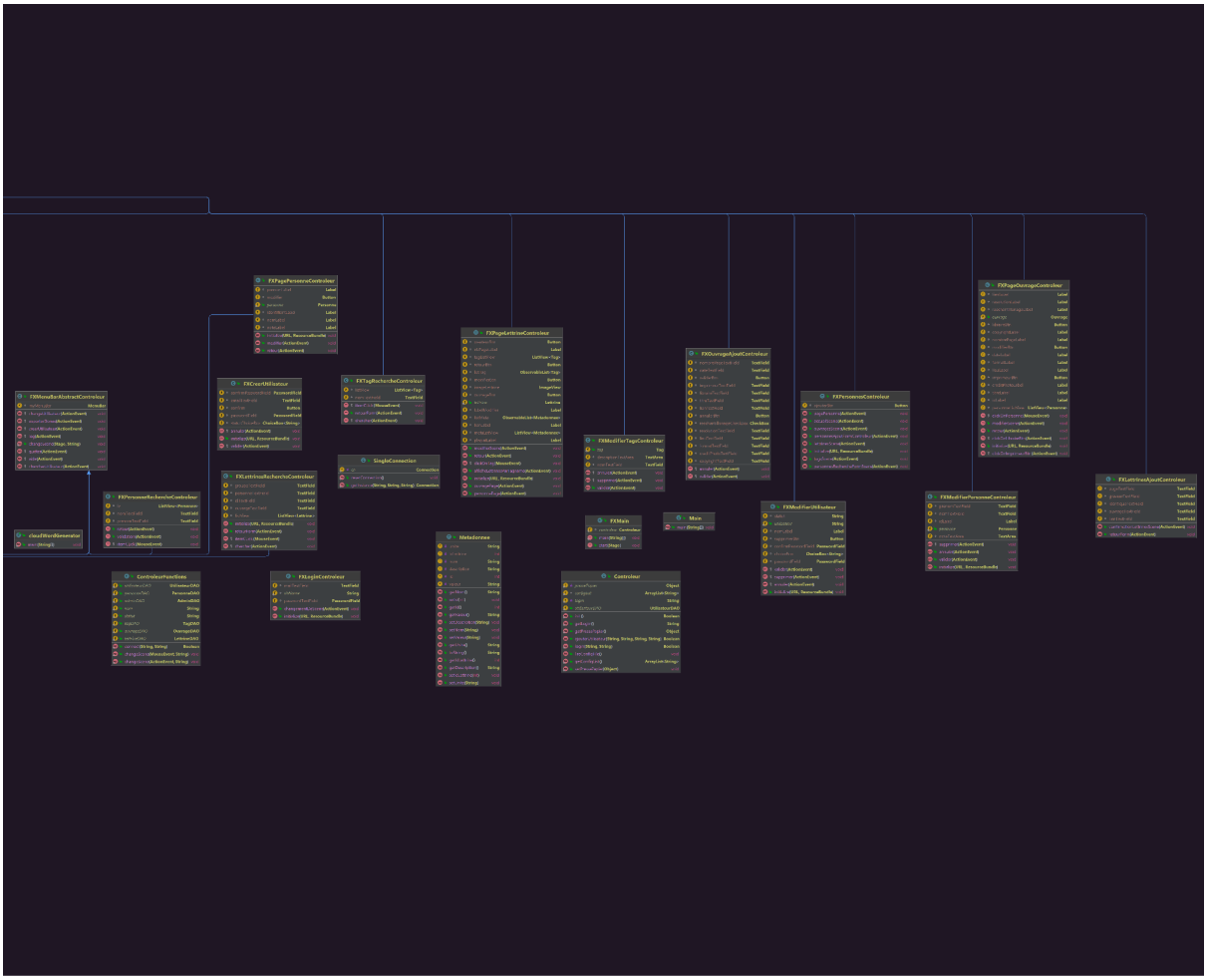
Annexe

Diagrammes de classes :

Ci-dessous les diagrammes de classes, le premier étant celui pensé à l'issue du premier jalon, le deuxième, généré par IntelliJ est celui , à l'issue de ce dernier jalon.







Le diagramme ci-dessus n'est pas très lisible car nous l'avons coupé en 2 à cause de sa largeur, il n'est donc pas très lisible mais une version complète en haute définition est disponible dans le dossier softwareEngineering.

Les requêtes pour insérer les métadonnées:

INSERT INTO `metadonnees`(`nom`, `valeur`,`idLettrine`)
SELECT "Lettrine Capital",`Lettrine_Capital`,`IdBois` FROM bdjimenes.`t_bois` WHERE `Type_Ornement`="Lettrine" and `Lettrine_Capital` IS NOT NULL;

INSERT INTO `metadonnees`(`nom`, `valeur`,`idLettrine`)
SELECT "Fond",`Lettrine_fond`,`IdBois` FROM bdjimenes.`t_bois` WHERE `Type_Ornement`="Lettrine" and `Lettrine_fond` IS NOT NULL;

INSERT INTO `metadonnees`(`nom`, `valeur`,`idLettrine`)
SELECT "Couleur",`Lettrine_Couleur`,`IdBois` FROM bdjimenes.`t_bois` WHERE `Type_Ornement`="Lettrine" and `Lettrine_Couleur` IS NOT NULL
;

```
INSERT INTO `metadonnees`( `nom`, `valeur`,`idLettrine`) SELECT
"Police",`Lettrine_Police`,`IdBois` FROM bdjimenes.`t_bois` WHERE
`Type_Ornement`="Lettrine" and `Lettrine_Police` IS NOT NULL
;
INSERT INTO `metadonnees`( `nom`, `valeur`,`idLettrine`) SELECT
"Alphabet",`Lettrine_Alphabet`,`IdBois` FROM bdjimenes.`t_bois` WHERE
`Type_Ornement`="Lettrine" and `Lettrine_Alphabet` IS NOT NULL
;INSERT INTO `metadonnees`( `nom`, `valeur`,`idLettrine`) SELECT
"Filet",`Lettrine_Filet`,`IdBois` FROM bdjimenes.`t_bois` WHERE
`Type_Ornement`="Lettrine" and `Lettrine_Filet` IS NOT NULL
;INSERT INTO `metadonnees`( `nom`, `valeur`,`idLettrine`) SELECT
"Devise",`Devise_transcription`,`IdBois` FROM bdjimenes.`t_bois` WHERE
`Type_Ornement`="Lettrine" and `Devise_transcription` IS NOT NULL
;INSERT INTO `metadonnees`( `nom`, `valeur`,`idLettrine`) SELECT
"Motif",`MotifMarqueTypo`,`IdBois` FROM bdjimenes.`t_bois` WHERE
`Type_Ornement`="Lettrine" and `MotifMarqueTypo` IS NOT NULL
;INSERT INTO `metadonnees`( `nom`, `valeur`,`idLettrine`) SELECT
"Technique",`Technique`,`IdBois` FROM bdjimenes.`t_bois` WHERE
`Type_Ornement`="Lettrine" and `Technique` IS NOT NULL
;INSERT INTO `metadonnees`( `nom`, `valeur`,`idLettrine`) SELECT "Particularites
graphiques",`Particularites_graphiques`,`IdBois` FROM bdjimenes.`t_bois` WHERE
`Type_Ornement`="Lettrine" and `Particularites_graphiques` IS NOT NULL
;INSERT INTO `metadonnees`( `nom`, `valeur`,`idLettrine`) SELECT "Ref
biblio",`Ref_biblio`,`IdBois` FROM bdjimenes.`t_bois` WHERE `Type_Ornement`="Lettrine"
and `Ref_biblio` IS NOT NULL
;INSERT INTO `metadonnees`( `nom`, `valeur`,`idLettrine`) SELECT
"commentaire",`commentaire_bois`,`IdBois` FROM bdjimenes.`t_bois` WHERE
`Type_Ornement`="Lettrine" and `commentaire_bois` IS NOT NULL
;INSERT INTO `metadonnees`( `nom`, `valeur`,`idLettrine`) SELECT "mot
clef",`mot_clef_description`,`IdBois` FROM bdjimenes.`t_bois` WHERE
`Type_Ornement`="Lettrine" and `mot_clef_description` IS NOT NULL
;INSERT INTO `metadonnees`( `nom`, `valeur`,`idLettrine`) SELECT
"Initiales",`Initiales`,`IdBois` FROM bdjimenes.`t_bois` WHERE `Type_Ornement`="Lettrine"
and `Initiales` IS NOT NULL
;INSERT INTO `metadonnees`( `nom`, `valeur`,`idLettrine`) SELECT "Date",`Date`,`IdBois`
FROM bdjimenes.`t_bois` WHERE `Type_Ornement`="Lettrine" and `Date` IS NOT NULL
;INSERT INTO `metadonnees`( `nom`, `valeur`,`idLettrine`) SELECT "Certitude
Attribution",`Certitude_attribution`,`IdBois` FROM bdjimenes.`t_bois` WHERE
`Type_Ornement`="Lettrine" and `Certitude_attribution` IS NOT NULL
;INSERT INTO `metadonnees`( `nom`, `valeur`,`idLettrine`) SELECT "Certitude
Mesure",`Certitude_mesure`,`IdBois` FROM bdjimenes.`t_bois` WHERE
`Type_Ornement`="Lettrine" and `Certitude_mesure` IS NOT NULL
```

;INSERT INTO `metadonnees`(`nom`, `valeur`,`idLettrine`) SELECT "Commentaire Attribution",`Commentaire_attribution`,`IdBois` FROM bdjimenes.`t_bois` WHERE `Type_Ornement`="Lettrine" and `Commentaire_attribution` IS NOT NULL
;INSERT INTO `metadonnees`(`nom`, `valeur`,`idLettrine`) SELECT "Hauteur",`hauteur`,`IdBois` FROM bdjimenes.`t_bois` WHERE `Type_Ornement`="Lettrine" and `hauteur` IS NOT NULL;