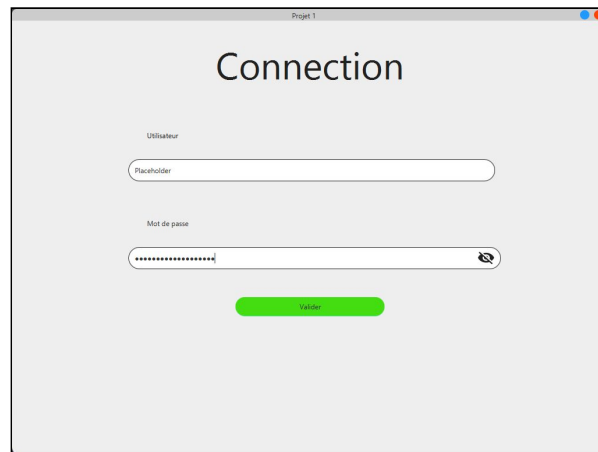


Compte Rendu Jalon 3

par Benjamin Mermod, Raphaël Dézé, Tom Martin, Tom Touzé, Adrien Collot

Quel est l'avancement du projet ?

- Système de login

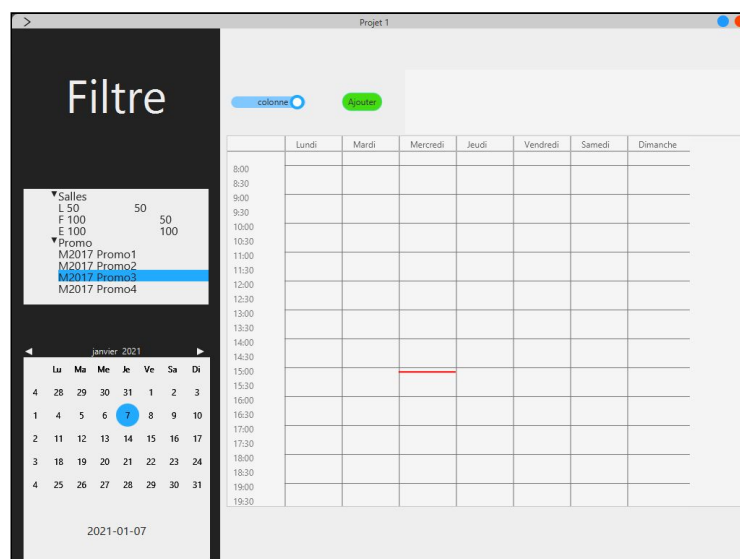


Le système de login permet de se connecter en vérifiant les valeurs de username et le mot de passe (ces deux champs sont uniques à chaque compte).

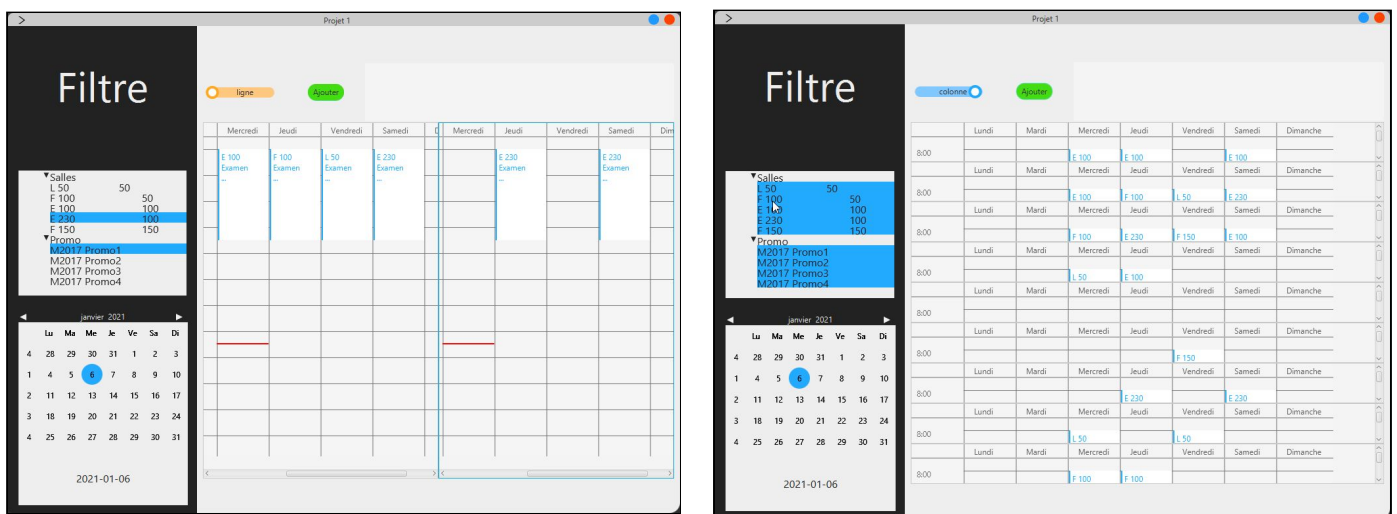
Les données relatives aux comptes sont inscrites dans la table 'login' de la base de données. Cette table contient non seulement le username ainsi que le mot de passe mais également l'adresse mail (qui sert notamment à notifier de la création d'un compte et récupérer le mot de passe lors de sa création), ainsi que le mot de passe (en clair, non hashé par manque de temps. Il aurait fallu comparer le hash contenu dans la base de données avec le String haché donné par l'utilisateur à chaque connexion).

Par ailleurs, si les données rentrées dans l'interface sont incorrectes, l'interface notifie l'utilisateur par un message d'erreur.

- Visualisation des examens sur une semaine donnée



L'interface graphique affiche les examens contenus dans une semaine donnée. Il est possible de changer la semaine concernée grâce à un calendrier présent en bas à gauche de l'interface. De plus, il est possible de changer la disposition des éléments entre deux modes, les jours ou les horaires en abscisse, représenté par un switch "Colonne/Ligne" en haut de la page. L'utilisateur peut par ailleurs filtrer les résultats s'affichant sur l'interface grâce au tree présent à gauche. Ces filtres sont de diverses natures, entre autres par salle, par promotion d'élèves, par matière et par type d'examen. Chaque élément de chacune des catégories citées au dessus peuvent être sélectionnés indépendamment pour plus de clarté. Ainsi, si l'utilisateur choisit deux filtres, les résultats de ces filtres seront affichés dans deux emplois du temps différents, séparés sur la hauteur de l'interface.



Pour terminer, est disposé au-dessus de l'emploi du temps présentant les examens le bouton "Ajouter", permettant d'ajouter un examen, selon plusieurs critères notés dans la catégorie suivante. Il devrait y avoir quelques boutons supplémentaires tels que "Ajout de Salle", "Modification d'examen" et "Modification de Salle" entre autres pour changer les propriété d'éléments déjà présents dans la base de données.

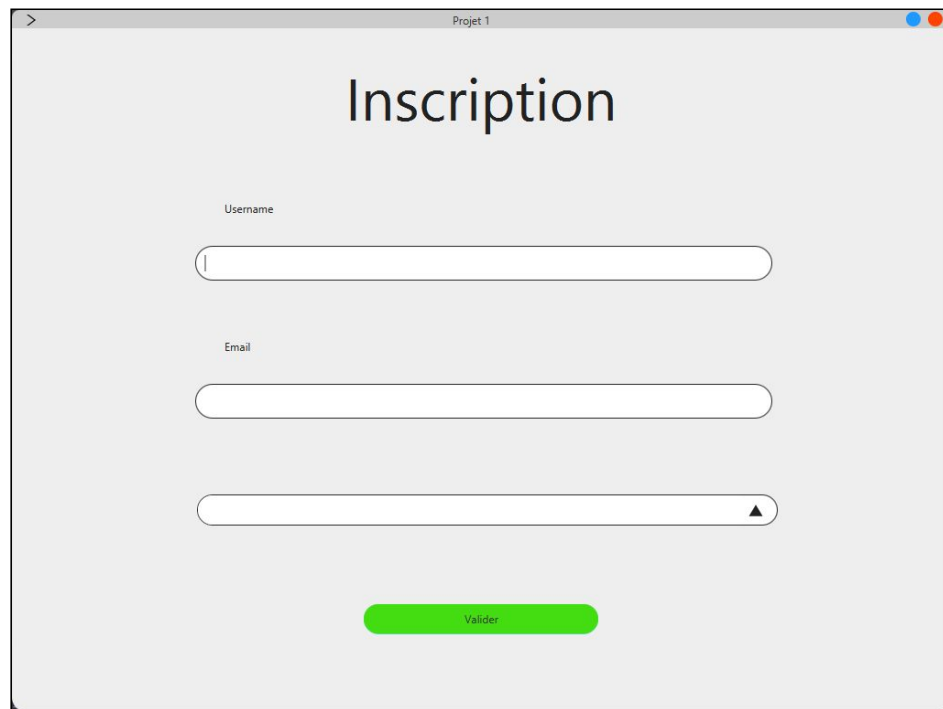
- Ajout d'examens

L'ajout d'examen permet de rajouter un examen sur le calendrier en spécifiant la matière, les élèves, le type d'examen, la durée en minutes.

Les élèves sont présentés dans une liste tree similaire à celle des filtres, et sont organisés par filière et par ordre alphabétique (par nom de famille). Cela permet à l'utilisateur d'ajouter au choix une filière dans son intégralité, ou bien élève par élève. La conception de la base de données traitant chaque élève comme une contrainte unique, un examen peut tout aussi bien contenir une promotion entière qu'une promotion plus un élève ponctuel, ou bien un

nombre défini par les besoins de l'utilisateur d'élèves d'autant de promotions différentes que nécessaires. Il n'est cependant pas possible actuellement d'ajouter des contraintes de temps et de salles, quand bien même le script permettant l'envoi d'informations par un fichier exam dans optaplanner inclut cette possibilité (cette impossibilité provient directement des problèmes que nous rencontrons avec l'implémentation d'optaplanner et non ni l'envoi ni la sauvegarde des données vers celui-ci).

- Inscription d'un membre par l'admin

The image shows a web browser window with the title 'Projet 1'. The main heading is 'Inscription'. Below it, there are three input fields: the first is labeled 'Username', the second is labeled 'Email', and the third is a dropdown menu with a small upward-pointing triangle on the right. At the bottom of the form is a green button labeled 'Valider'.

L'inscription d'un nouveau membre par l'administrateur permet aux comptes ayant le rôle admin de créer un nouveau compte (chose impossible pour un utilisateur extérieur, le but de l'application n'est pas de pouvoir créer son propre compte mais bien de se le faire assigner par l'administration), en renseignant le nom du compte ainsi que l'adresse mail de référence. Cette dernière permet à l'utilisateur concerné d'être notifié de la création de son compte ainsi que son type de compte, et de son mot de passe (généré aléatoirement, l'administrateur n'a aucun accès ni contrôle sur le mot de passe), le tout instantanément par mail utilisant la librairie Javax Mail ainsi que les serveurs de google avec une adresse Gmail composée pour l'occasion.

Il aurait été intéressant d'ajouter la fonction mot de passe oublié (envoyé par mail) ou réinitialisation du mot de passe (une fois connecté), mais ce n'est pas disponible dans la version actuelle du projet.

- Conversion des examens depuis la DB vers un fichier exam

Chaque examen créé est automatiquement ajouté dans la base de données. Cependant, hormis les examens créés avec une contrainte de salle ou de temps (non disponibles actuellement), les examens sont créés avec uniquement comme propriété de temps et d'espace leur longueur ainsi que les élèves présents (sans compter la matière et le type). Ainsi, c'est le rôle d'Optaplanner d'agencer chaque examen selon des critères définis en amont et présenter la meilleure configuration. Cependant, il n'est évidemment pas possible de donner à la boîte noire qu'est Optaplanner une vue provenant de la base de donnée, et le formatage (permettant de mettre à la bonne syntaxe des éléments données) a été au cœur de la réalisation de ce projet. Ainsi la première étape consiste en la conversion des divers examens et de leurs propriété en un fichier d'une syntaxe particulière, un fichier exam. Ce dernier est créé par la classe CreationFich, et, en partant d'une liste d'objets Examen (utilisés à de nombreuses reprises dans le code, car contenant toutes les informations nécessaires), va créer en sortie le fichier exam correspondant avec la syntaxe adéquate selon les informations suivantes:

- Une liste d'examens attribuée définitivement.
- Une liste de créneaux attribués définitivement.
- Une liste de salles attribuées définitivement.
- Une liste de contraintes sur les examens attribuée définitivement.
- Une liste de contraintes de salle sur les examens attribuée définitivement.

Cependant, comme expliqué précédemment, Optaplanner est une boîte noire, à savoir que le programme n'a accès qu'à une entrée et une sortie sans contrôle sur le processus. Cela devrait être un avantage, cependant au point actuel du projet, ledit processus n'est pas implémenté, et nous avons rencontré un nombre très important d'erreurs, ainsi que d'incohérences que ce soit des résultats divergents de manières importantes pour une même entrée, selon la machine, des temps d'exécutions irréalistes ou encore des problèmes d'installations.

- Conversion d'un XML de sortie d'optaplanner vers DB

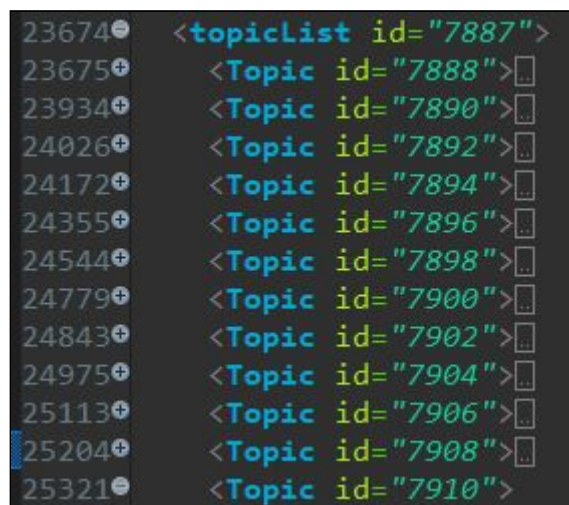
Cependant, la sortie d'Optaplanner est connue, ainsi il est possible de convertir les données de sortie en informations nécessaires pour la base de données, et ainsi d'affecter à chaque examen un créneau et donc une place définie dans l'emploi du temps. Ces informations temporaires (supprimées et ré-inscrites à chaque ajout d'examen, donc à chaque traitement par Optaplanner) sont contenues dans la table Liaison. Cette table permet de faire le lien entre un examen, un créneau et une contrainte (dans la grande majorité des cas, une

contrainte d'élève, et en un cas unique, une contrainte de salle, déterminée par Optaplanner).

Pour récupérer les informations, la classe XMLParser utilise le DOM Parser natif de Java, permettant entre autres de récupérer les nodes, sous-nodes, et contenus. La syntaxe du XML étant statique, il est facile de déterminer où chercher quelles données, mais peu clair à présenter dans le code.

Il faut savoir dans un premier temps que le système d'ID présent dans le fichier XML est commun à toutes les nodes, à savoir qu'il s'agit d'un entier qui s'auto-incrémente depuis le premier élément jusqu'au dernier peu importe le jeu de données. Ainsi le premier objectif, pour faire le lien entre les ID présentes dans la base de données et les ID présentes dans le fichier XML de sorti d'Optaplanner est de récupérer les ID correspondantes aux listes des topics (les examens), des periods (créneaux), ainsi que des rooms (salles). Ces listes contenant un nombre indéfini d'éléments, il suffit de faire pour la plupart le numéro de début de liste (par exemple 2000 pour topicList, 7000 pour periodList et 7500 pour roomList) et d'aller dans un second temps récupérer les informations relatives aux examens (liste LeadingExam), qui contient à la fois l'ID d'un examen, sa salle et son créneaux attribué. Une fois toutes ces valeurs récupérées, il suffit de faire la différence (sauf dans le cas de topicList pour une raison qui sera expliquée plus bas). Ainsi une liste provenant de LeadingExam de la syntaxe (2001,7005,7560) correspondra en réalité à l'examen 1 au créneau 5 et dans la salle 60.

Dans le cas de topicList en revanche, les ID de chaque examen sont espacées d'une ID supplémentaire, correspondant à la liste interne d'élèves y participant. Ainsi, le traitement qui part de l'ID de la topicList vers l'ID présente dans la base de données est légèrement différent de celles des deux autres listes.



```
23674● <topicList id="7887">
23675+   <Topic id="7888">..
23934+   <Topic id="7890">..
24026+   <Topic id="7892">..
24172+   <Topic id="7894">..
24355+   <Topic id="7896">..
24544+   <Topic id="7898">..
24779+   <Topic id="7900">..
24843+   <Topic id="7902">..
24975+   <Topic id="7904">..
25113+   <Topic id="7906">..
25204+   <Topic id="7908">..
25321● <Topic id="7910">
```

Fichier XML de sortie dans le node "topicList"

La dernière étape est, une fois toutes les informations récupérées et les ID trouvées d'envoyer les informations dans la base de données.

Quels sont les tests unitaires ?

Le but d'un test unitaire étant de comparer le résultat attendu par le test d'une méthode et le résultat réel donné dans une classe avec son résultat choisi de manière encadrée et optimisée par JUnit, nous avons couvert les principales fonctions.

Voici une liste de courtes explications :

- AuthentificationTest

```
void testAuthentification() {
    Authentification a = new Authentification("Tom", "deusvult"); //admin
    Authentification a1 = new Authentification("scolarite", "scolarite"); //scolarite
    Authentification a2 = new Authentification("secretariat", "secretariat"); //secretariat

    Authentification b = new Authentification("Tom", "faux");
    Authentification c = new Authentification("faux", "deusvult");
    Authentification d = new Authentification("faux", "faux");
    Authentification e = new Authentification("faux", null);
    Authentification f = new Authentification(null, "faux");

    assertEquals(true, a.getAutorise());
    assertEquals(false, b.getAutorise());
    assertEquals(false, c.getAutorise());
    assertEquals(false, d.getAutorise());
    assertEquals(false, e.getAutorise());
    assertEquals(false, f.getAutorise());

    assertEquals(3, a.getStatut());
    assertEquals(2, a1.getStatut());
    assertEquals(1, a2.getStatut());
    assertEquals(0, b.getStatut());
}
```

La classe Authentification permet de comparer les données entrées par l'utilisateur dans l'interface (à savoir l'username et le mot de passe) et les données présentes dans la base de données. Pour qu'un utilisateur soit identifié, il faut que son nom d'utilisateur et son mot de passe correspondent (soit dans la même ligne, pas uniquement dans la table). Dans un premier temps, sont créés dans la classe AuthentificationTest trois nouveaux objets avec des valeurs correspondantes à celles de la base de données, suivi de cinq cas qui correspondent à différentes situations de refus (respectivement un username bon, mais un mot de passe faux, l'inverse, deux valeurs qui ne correspondent à rien, puis soit le username soit le mot de passe non renseigné).

Il est ainsi attendu que les 3 premiers objets retournent vrai lors du test de la méthode getAutorise().

Dans un second temps le programme teste la méthode getStatut(), qui dans le cas des 3 objets fonctionnels retourne la valeur attendue (l'ID correspondant au statut du compte), et 0 dans le cas du test où l'objet ne réussit pas dans un premier temps la connexion (valeur définie par défaut).

- CreationFichTest

Pour tester la Creation de fichier d'import d'Optaplanner (sous le format .exam)

la classe test créer 5 fichiers .exam

Les deux premiers ont des paramètres identiques mais le dossier d'arrivé est différent du premier

le troisième à une liste d'examen différente.

Le quatrième à une liste de créneau différente.

Le cinquième à une liste de salle différente.

```
String doss1 = "data/testexam1.exam";
String doss2 = "data/testexam2.exam";
String doss3 = "data/testexam3.exam";
String doss4 = "data/testexam4.exam";
String doss5 = "data/testexam5.exam";

CreationFich a = new CreationFich(le1,lc1,ls1,doss1);
CreationFich b = new CreationFich(le1,lc1,ls1,doss2);
CreationFich c = new CreationFich(le2,lc1,ls1,doss3);
CreationFich d = new CreationFich(le1,lc2,ls1,doss4);
CreationFich e = new CreationFich(le1,lc1,ls2,doss5);
a.creerFile();
b.creerFile();
c.creerFile();
d.creerFile();
e.creerFile();
```

La classe test stocke ensuite ligne par ligne les différents fichiers créés.

```
InputStream ips1=new FileInputStream(doss1);
InputStreamReader ipsr1=new InputStreamReader(ips1);
BufferedReader br1=new BufferedReader(ipsr1);
while ((b1 = br1.readLine()) != null) {
    r1 += b1;
}
br1.close();
InputStream ips2=new FileInputStream(doss2);
InputStreamReader ipsr2=new InputStreamReader(ips2);
BufferedReader br2=new BufferedReader(ipsr2);
while ((b2 = br2.readLine()) != null) {
    r2 += b2;
}
br2.close();
InputStream ips3=new FileInputStream(doss3);
InputStreamReader ipsr3=new InputStreamReader(ips3);
BufferedReader br3=new BufferedReader(ipsr3);
while ((b3 = br3.readLine()) != null) {
    r3 += b3;
}
br3.close();
```

On compare ensuite les string en eux . Deux string égaux veulent dire que les fichiers .exam sont identiques

```
assertEquals(true,r1.equals(r2));
assertEquals(false,r1.equals(r3));
assertEquals(false,r1.equals(r4));
assertEquals(false,r1.equals(r5));
```


- VerifTest

Le but de la classe Verif est de s'assurer lors de la création d'un compte par l'administrateur que ni l'email, ni le nom d'utilisateur est déjà utilisé par un autre compte. Ainsi la classe VerifTest test la méthode Verification dans les cas suivants:

Username et email déjà utilisés, username déjà utilisé, email déjà utilisé et aucun élément utilisé.

Ainsi les résultats attendus sont respectivement vrai, vrai, vrai et false, vrai correspondant à une correspondance d'un des éléments dans la base de données, donc un refus de créer le compte à l'échelle du programme.

- ExamenTest

La classe ExamenTest teste son constructeur et les différents ajout unitaire et par liste des composantes de l'examen qui sont : les élèves , les créneaux , les contraintes d'examen et les contraintes de salles.

- EtudiantTest

Test simple du constructeur Étudiant et de ses attributs.

```
@Test
void testEtudiant() throws NamingException, SQLException {
    Etudiant a = new Etudiant(21903697, "martin", "tom", "info", 2);
    assertEquals(21903697, a.getNumeroetu());
    assertEquals("martin", a.getNom());
    assertEquals("tom", a.getPrenom());
    assertEquals("info", a.getPromo());
    assertEquals(2, a.getAnnee());
}
```

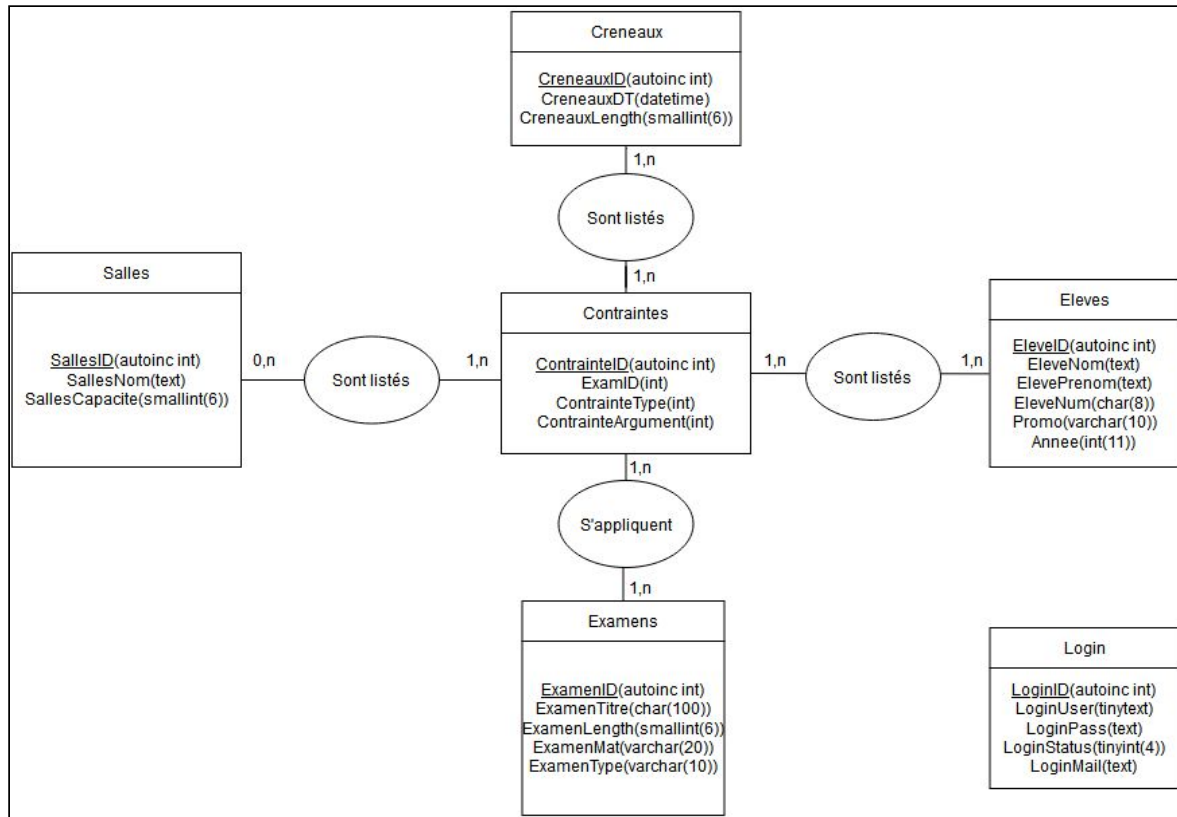
- CreneauHoraireTest

La classe CreneauHoraire permet de modifier le format de la date du format fourni par le type SQL DATETIME en un format exploitable par le fichier exam, dont l'utilité est expliquée plus haut.

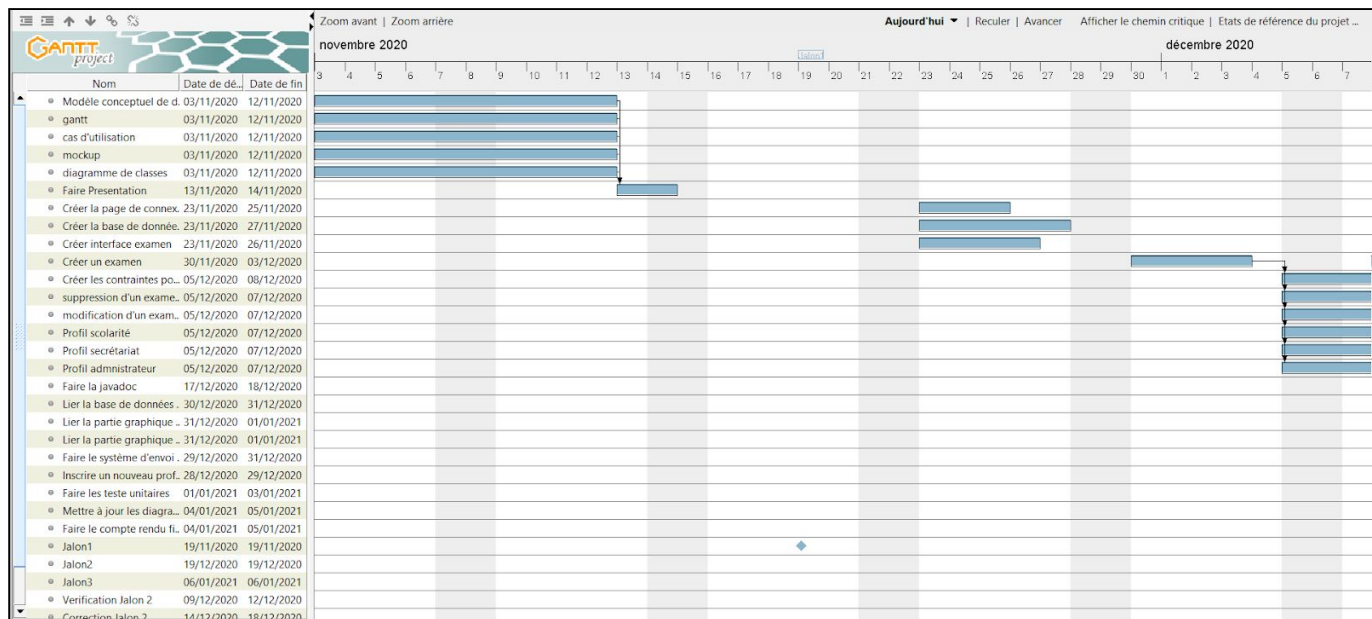
Le principe de la classe test est ici de s'assurer du bon fonctionnement des calculs (de la méthode CreneauHoraire) prenant comme entrée un créneau horaire de la syntaxe HH:MM:SS et une durée en minutes et le bon fonctionnement de la syntaxe de sortie (après conversion des dates depuis la base de données) notamment pour s'assurer qu'aucune information avec une syntaxe incorrecte soit notée dans le fichier d'entrée d'Optaplanner ce qui l'empêcherait de fonctionner.

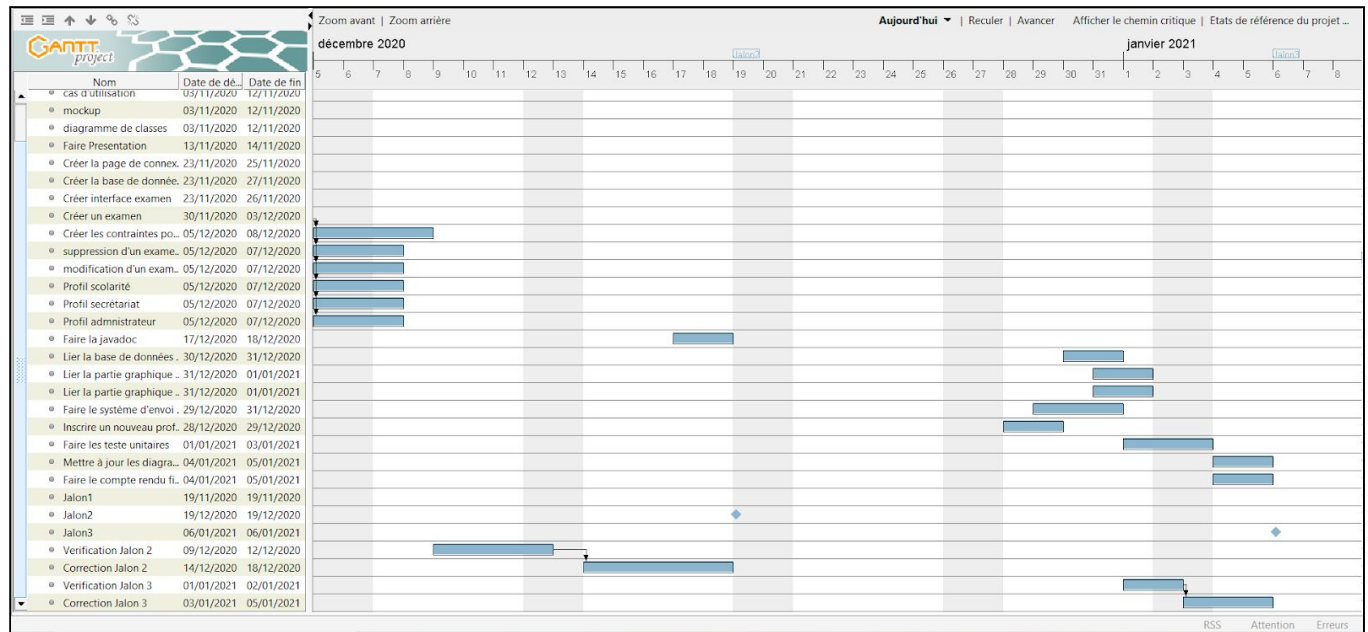
Versions mises-à-jour des graphiques :

MCD:



GANTT:





Répartition des tâches :

Raphaël DEZE :

Je me suis occupé de réaliser le diagramme de cas d'utilisation principalement pour le Jalon 1 ainsi que de l'appliquer de manière à garder une structure logique dans le code du projet lors de la production de celui-ci (majoritairement durant les Jalons 2 et 3).

Je me suis également occupé de l'algèbre relationnelle des requêtes principales déterminées avec Benjamin Mermod lors des premières sessions de travail du Jalon 2, dans le but d'obtenir des relations logiques des communications importantes entre les tables.

Par ailleurs, dans le but de permettre la création de compte par l'Administrateur et la réinitialisation en cas de mot de passe oublié (non implémentée actuellement), j'ai créé la classe permettant d'envoyer des mails, basé sur la librairie Javax Mail, sur une adresse gmail créée pour l'occasion, et utilisant le protocole SMTP (type de protocole permettant l'envoi sécurisé de mail).

Tom MARTIN :

Durant la première partie du projet jusqu'au jalon 1, je me suis occupé du diagramme de classe et des premières recherches documentaires et pratiques sur le solveur optaplanner.

Durant la seconde phase j'ai participé dans le code java pour la création du fichier .exam (fichier import d'Optaplanner). J'ai aidé dans la structure de la Base de Données, à faire les requêtes principales et les traduire en algèbre et calcul relationnel.

J'ai mis en place avec mes camarades les envois de données entre l'application java et la base de données

Et enfin , j'ai participé dans la récupération des données de BD pour l'affichage et la création de nouveaux fichiers d'import .exam.

Je me suis aussi occupé des classes de tests et du système fonctionnel de login .

Tout au long du projet , j'ai essayé d'utiliser optaplanner dans notre projet , sans grand succès pour l'instant ,j'espère y arriver pour le jour de la présentation orale.

Adrien COLLOT :

Durant le jalon 1 je me suis occupé de la gestion du temps grâce au Gantt. Ce dernier a évolué avec nos besoins et a été modifié au fil du temps. Je me suis occupé principalement de la documentation en Javadoc du côté fonctionnel de l'application et ai aussi assisté à la création de la classe CreationFich avec Tom Martin lors du jalon 2. Cette classe permet de créer un fichier sous un format exploitable par le solveur Optaplanner.

Lors du jalon 3 j'ai également participé à la création de XMLParser, qui est expliqué plus haut, et me suis chargé des explications sur l'avancement de notre projet et des tests unitaires avec Benjamin Mermod.

Benjamin MERMOD : chargé de base de données,

J'ai tout d'abord commencé par établir le MCD avant de créer la base de données, le tout durant le Jalon 1.

Le Jalon 2 a été une période pendant laquelle beaucoup d'éléments initialement présents dans la base ont été modifiés, d'où la présence d'un MCD mis-à-jour dans ce document.

Certaines classes comme liaison et contraintes ont vues le jour avec la compréhension d'Optaplanner et la manière dont les données allaient être formatées (il était plus simple d'adapter la DB en fonction de la syntaxe d'entrée plutôt que de devoir passer par un script de formatage lourd).

Je me suis assuré de l'intégrité des données et de la bonne utilisation de la base tout le long de la conception de la partie fonctionnelle du projet.

Une grande partie du Jalon 3 a également été occupée par la conception de la classe XMLParser après de nombreux essais sur DOM Parser, afin de traduire les informations sortant d'Optaplanner (en fichier XML), dont le procédé est expliqué plus haut dans ce document.

J'ai également relié le système d'envoi de mail créé par Raphaël dans le système de création de compte par l'administrateur, et m'est également occupé des requêtes relatives à

l'ajout d'informations dans la table liaison et login (respectivement pour XMLParser et la création de nouveaux comptes par l'administrateur).

Pour terminer je me suis chargé d'écrire la quasi intégralité des textes explicatifs à propos de notre avancement dans le projet pour ce document ainsi que les paragraphes explicatifs à propos des tests unitaires Authentification, Verif et CreneauHoraire.

Tom TOUZÉ : chargé de l'interface graphique,

La première étape était de se renseigner sur JavaFX, ses méthodes et sa syntaxe générale. Je voulais trouver une syntaxe en java, afin de pouvoir appliquer les méthodes vues lors des cours.

Sur internet, la plupart des codes ouverts utilisait FXML (un langage pour javafx plus proche du XML) mais cela ne convenait pas à mes attentes, je me suis donc renseigné sur optaplanner et ses classes d'affichage, j'en ai déduit l'utilisation de Swing. Par le passé j'ai déjà eu l'opportunité de faire des interfaces utilisateur complexes avec les librairies standard, en utilisant le principe des composants, ce qui permet de personnaliser les éléments graphiques pour l'application.

Avec swing, il y a la possibilité de créer une fenêtre (Frame) avec un constructeur, j'ai donc cherché un moyen d'y parvenir avec javafx, ensuite j'ai créé quelques composants de base comme le panel, le textfield, le texte, le titre. A ce moment là, ce ne sont que des ébauches et des dessins, les codes java sont fait au minimum, l'idée étant surtout de tester les capacités, quelque milliseconde d'attentes au lancement, mais pour le reste pas de soucis.

J'ai donc repris le code depuis le début (pour refaire la hiérarchie et surtout factoriser ce qui pouvait) une fois les premier composant fait j'ai étudié les différents composant de javafx et les types de variables que l'utilisateur pourrait donner, pour choisir les éléments à ajouter. J'ai donc essayé les composants en les écoutants d'abord naturellement avec la méthode "add", mais malheureusement, l'utilisation de la class frame faisant que la création des éléments de base de javafx, générer une erreur à l'initialisation de Toolkit, c'est pour cela que certains composants ont dû être refaits sans les composants identiques dans JavaFX.