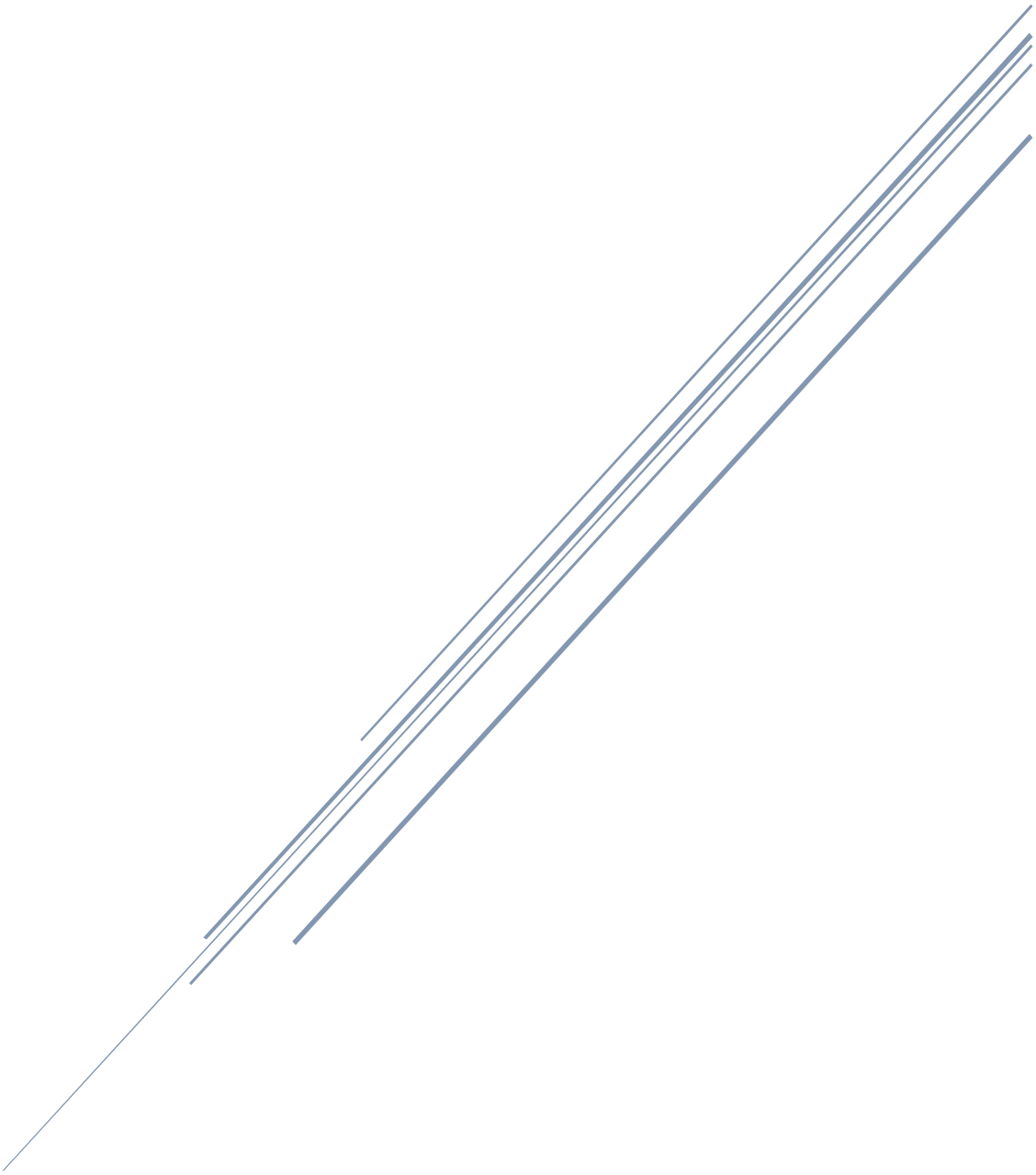
DOCUMENTATION TECHNIQUE

Projet Liborrow



P4 Open Classrooms
Benoît AUBIN

Table des matières

1 Description du domaine fonctionnelle :

1.1 Rappel besoins client :

Le projet a pour but le développement et la mise en production du système d'information d'une bibliothèque municipal. Les attentes en termes de fonctionnalités sont les suivantes :

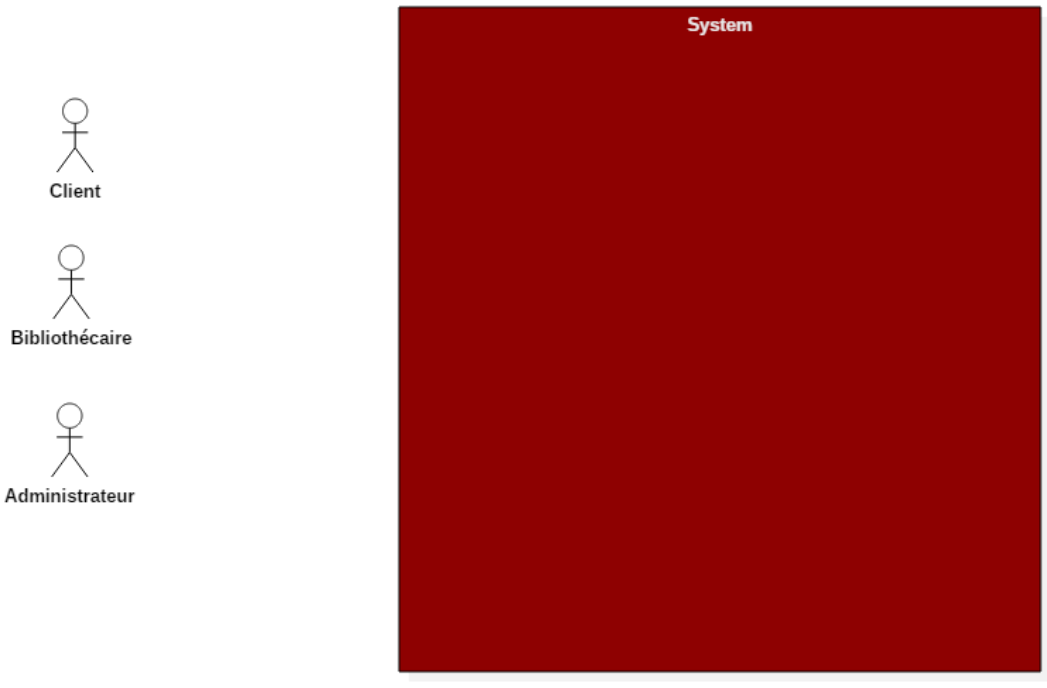
- Gérer les emprunts (créer, supprimer, relancer, finaliser).
- Rechercher des ouvrages et consulter leur disponibilité.
- Prolonger un prêt.
- Consulter les prêts en cours.

A ces fins, ce système devra être composé de trois éléments principaux :

- Un webservice permettant de consommer les données de la base de données PostgreSQL (également à réaliser). Ce dernier contiendra toute la logique destinée à réaliser les fonctionnalités précédemment listées.
- Une application web destiné uniquement aux clients de la bibliothèque et leur permettant de gérer leurs prêts, ainsi que de rechercher des ouvrages et consulter leur disponibilité.
- Un batch destiné à la relance des utilisateurs ayant réalisés des prêts dont la date de rendu est dépassée.

Le client a également émis le besoin d'une application mobile et d'un outil de gestion des prêts destiné à l'usage des bibliothécaires qui ne seront pas réalisés dans un premier e t dont la présente documentation ne couvre pas le sujet.

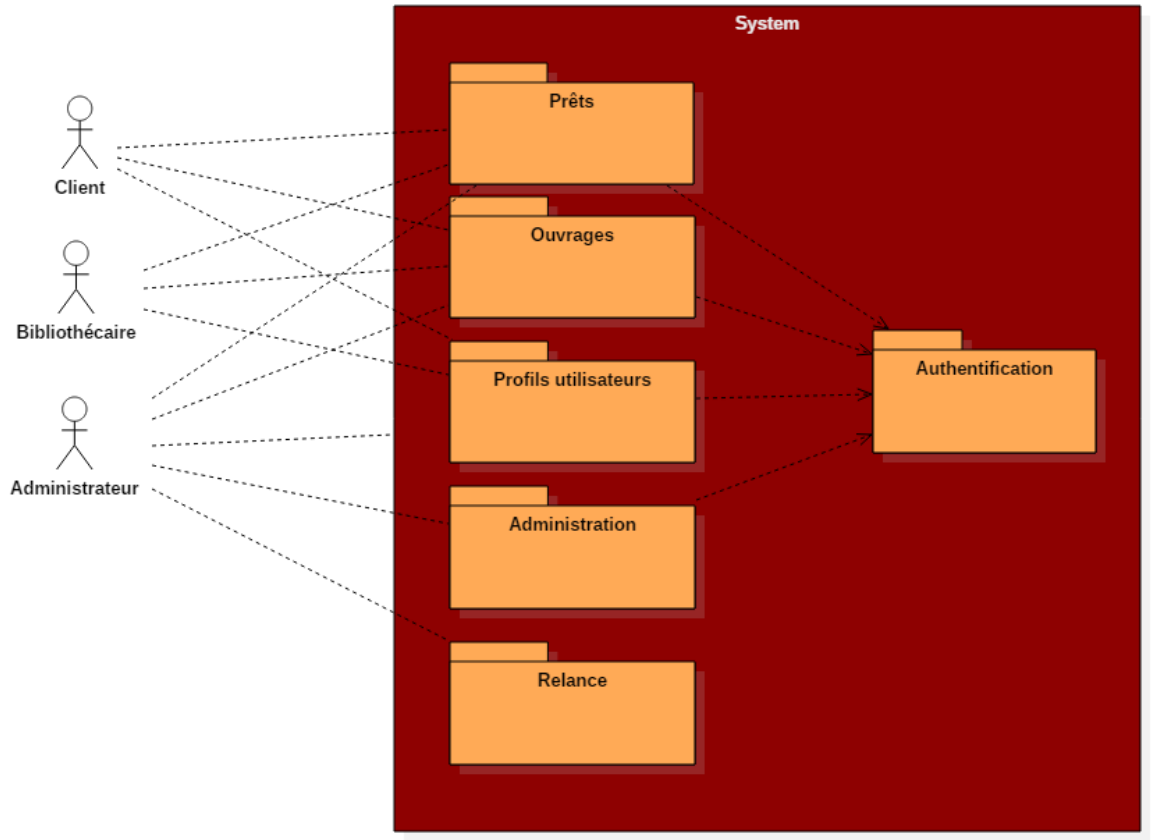
1.2 Futurs utilisateurs :



A la vue des fonctionnalités demandées il est en effet assez simple de déterminer le premier type d'utilisateur de l'application à savoir les clients de la bibliothèque. On ajoutera également deux profils d'administration à savoir les bibliothécaires et les administrateurs. La détermination de ces trois types de rôles est réalisée à partir du besoin émis :

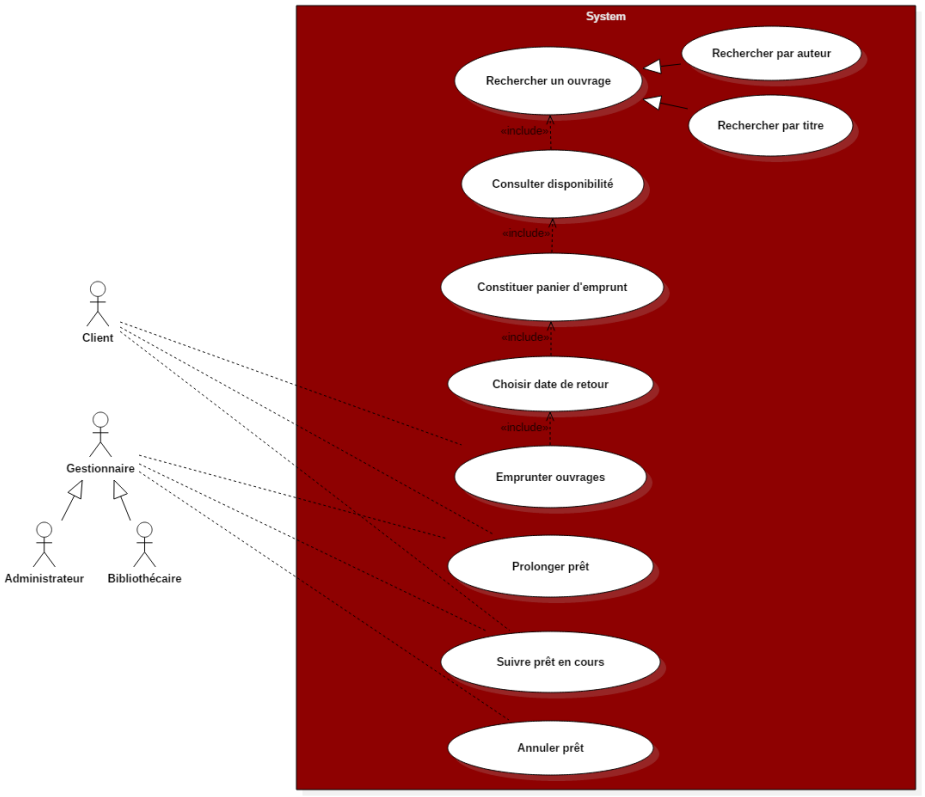
- Le rôle client : ce dernier correspond à un profil d'utilisateur sans aucun droit d'administration ayant accès uniquement à l'application web du système d'informations. Ces derniers pourront ainsi rechercher des ouvrages sur le site, consulter leurs prêts en cours et éventuellement les prolonger.
- Le rôle bibliothécaire : ce premier niveau de droits d'administration offre la possibilité aux bibliothécaires de gérer les prêts (créer, supprimer, modifier, finaliser) et ouvrages (créer, modifier supprimer) de la bibliothèque. Il ne permet néanmoins pas d'assurer l'administration technique de la solution.
- Le rôle administrateurs : Les administrateurs possèdent tous les droits sur la solution ainsi que la possibilité supplémentaire de gestion des utilisateurs (création, modification, suppression). Ils peuvent également modifier la date d'exécution du batch et lancer immédiatement le job.

1.3 Fonctionnalités :

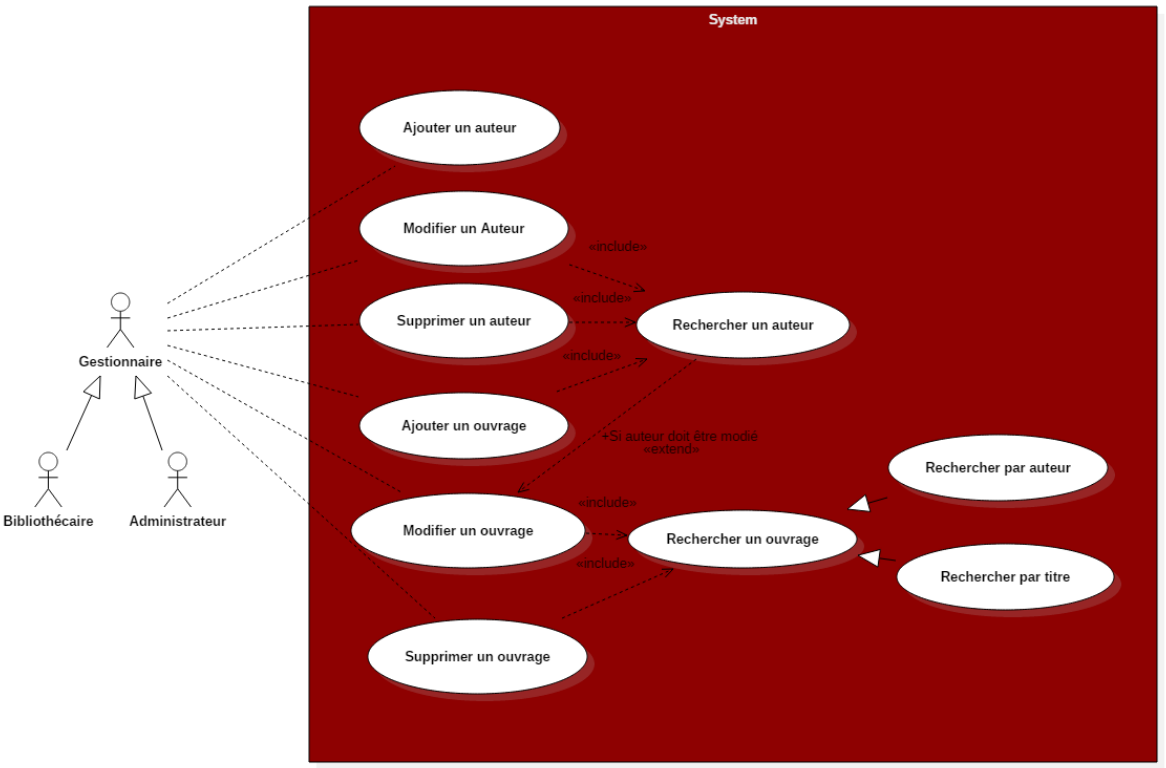


Les fonctionnalités de la solution ont été organisés en six grandes catégories de fonctionnalités comme on peut le voir sur le schéma précédent :

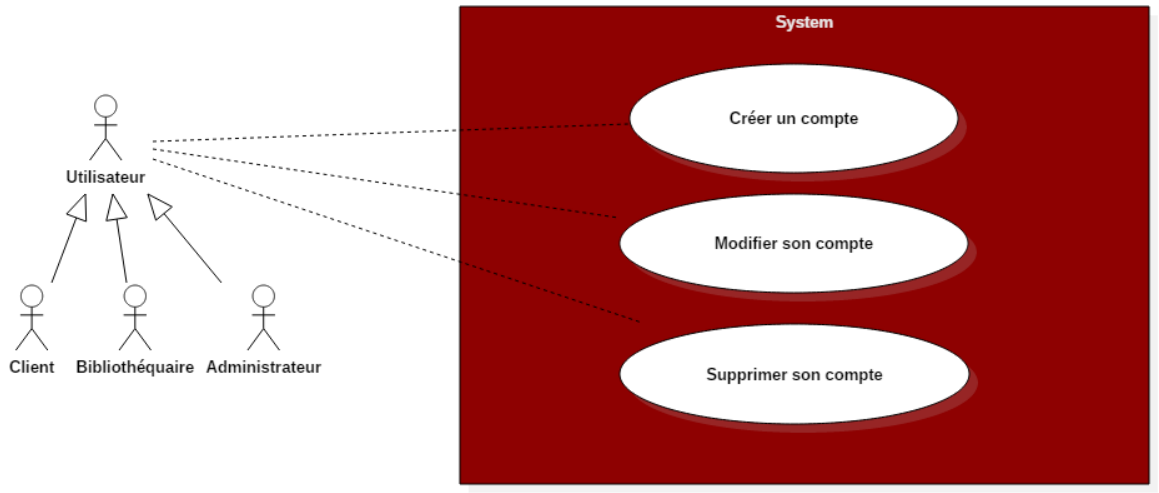
- Gestion des prêts : regroupe toutes les fonctionnalités concernant les prêts excepté la relance des utilisateurs en retard : création de prêt, suivi de prêt, prolongation de prêt ou annulation de prêt. Ces fonctionnalités sont offertes à tous les utilisateurs.



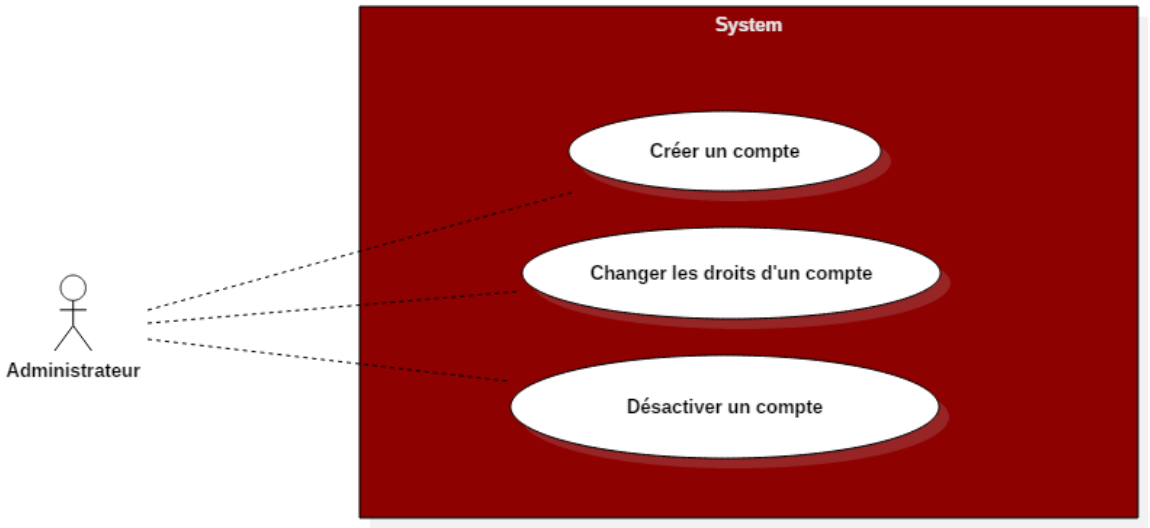
- Gestion des ouvrages : La gestion des ouvrages s'oriente sur deux axes à savoir la gestion des auteurs et la gestion de l'ouvrage en lui-même. La gestion des auteurs rassemble les fonctionnalités de création, modification et suppression d'auteurs d'ouvrages. La partie gestion d'ouvrage comprend les mêmes fonctionnalités pour un ouvrage ainsi que des fonctionnalités de recherche selon différents critères (par auteur, par titre…). Les fonctionnalités de recherche sont accessibles à tous les utilisateurs tandis que les fonctionnalités de gestion sont réservées aux bibliothécaires ainsi qu'aux administrateurs.



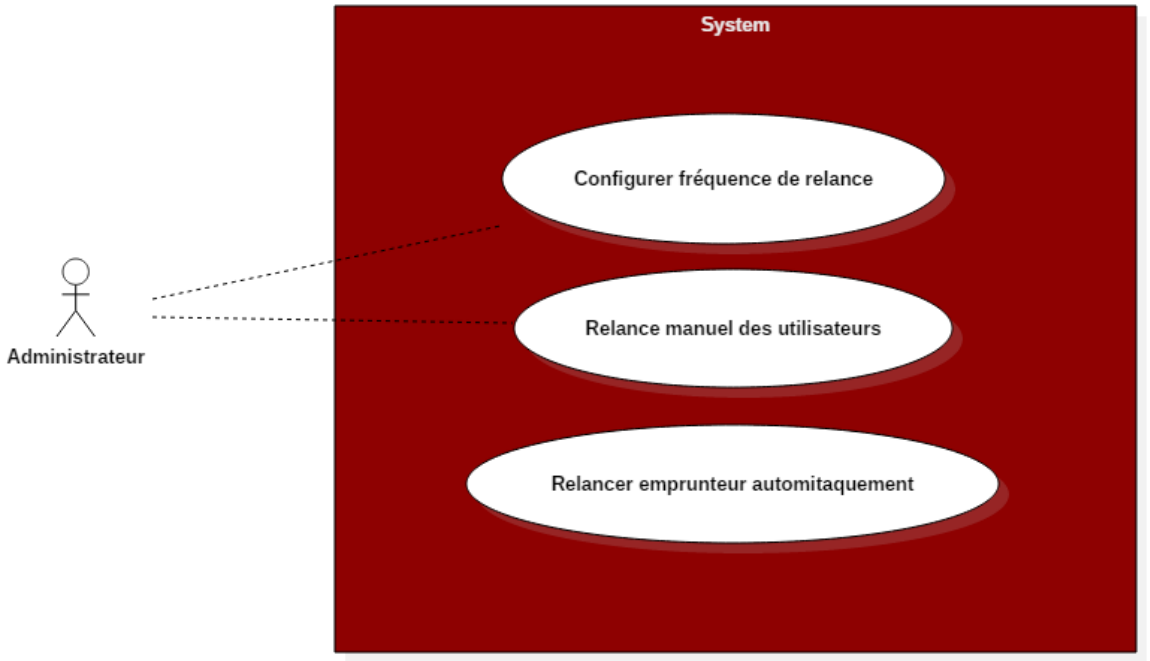
- Gestion des utilisateurs : La gestion des utilisateurs comprend la création, modification et suppression de son profil utilisateur. Ces fonctionnalités sont accessibles à tus les utilisateurs.



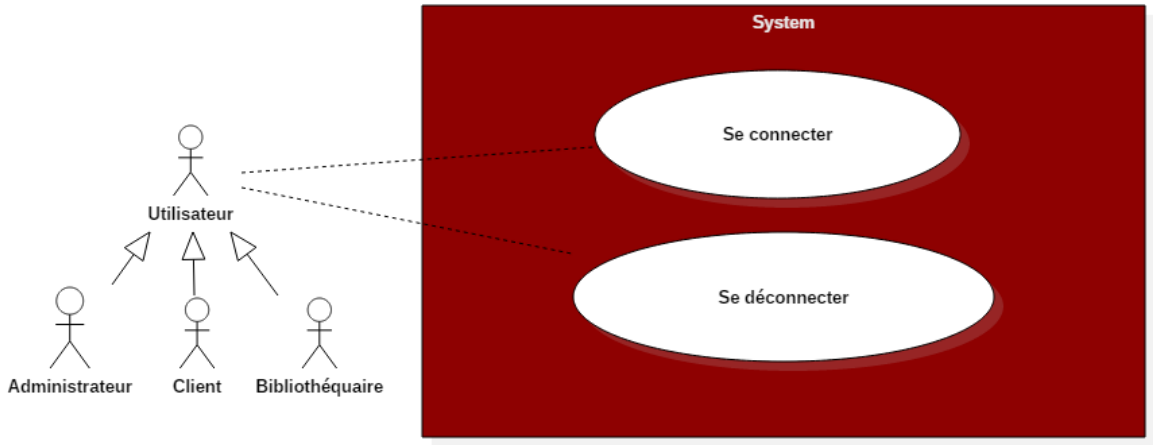
- Administration de la solution : La partie administration n'est accessible qu'aux administrateurs et leur offre la possibilité de créer un compte, de modifier les droits d'un compte et de désactiver un compte.



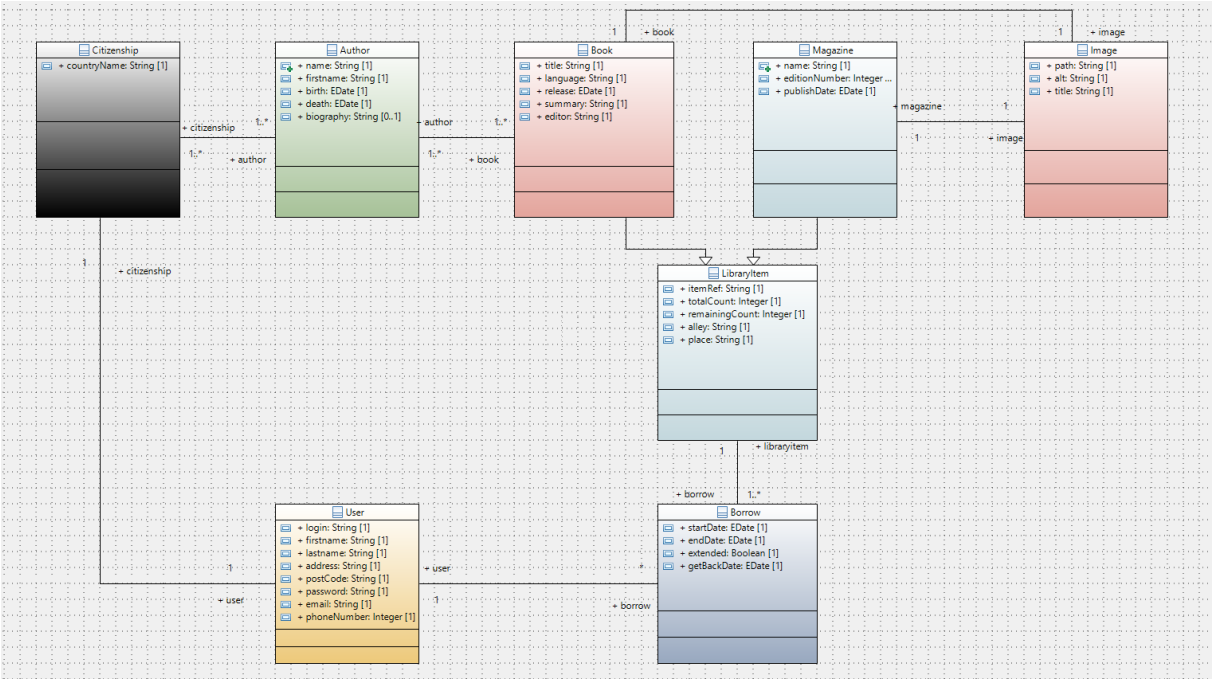
- Relance des prêts : Cette partie, ne concernant uniquement que les relances de prêts en retard est constitué de deux fonctionnalités : relancer les prêts en retard (lancement du batch) et configuration du timestamp de relance. Ces deux fonctionnalités ne sont accessibles qu'aux administrateurs.



- Authentification : Cette partie concerne l'authentification des utilisateurs sur l'interface web du système d'information. Elle est donc accessible à tous les utilisateurs et comprend les fonctionnalités de connexion et de déconnexion des utilisateurs.



1.4 Diagramme de classe :

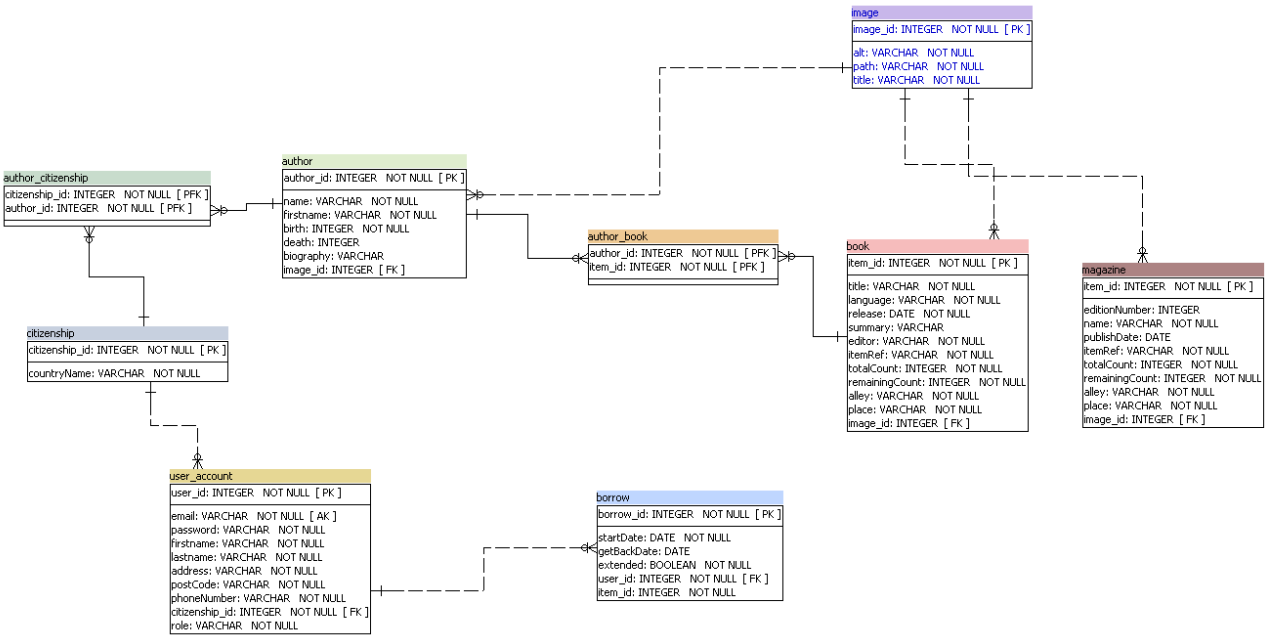


Les besoins en fonctionnalités précédemment établis nous ont permis de dresser ce diagramme de classes comportant huit classes :

- La classe User dresse la liste des caractéristiques d'un utilisateur du système d'information. Un utilisateur peut ainsi posséder 1 ou plusieurs nationalité (classe Citizenship) et avoir réaliser un ou plusieurs emprunts.
- La classe borrow décrit les informations d'un emprunt. Un emprunt est lié à un item de la bibliothèque et à un utilisateur.
- La classe item dresse les informations générales d'un objet de la bibliothèque. On relève deux types d'item les livre (classe book) et les revues (classe magazine). Un item est lié à zéro ou plusieurs emprunts.

- La classe book dresse les caractéristiques des livres de la bibliothèque. Elle hérite de la classe item. Un livre peut avoir été écrit par un ou plusieurs auteurs et un livre est lié à zéro à une image (couverture de l'ouvrage sur le site).
- La classe magazine fait l'état des différentes informations décrivant une revue de la bibliothèque. Elle hérite de la classe item (et par extension de ses attributs. Une revue peut être lié à zéro ou une image (couverture de la revue).
- La classe image contient toutes les informations nécessaires à la description et l'affichage d'une image sur le site.
- La classe auteur décrit les caractéristiques d'un auteur de livres. Un auteur peut avoir écrit un ou plusieurs livres et être d'une ou plusieurs nationalités.
- La classe nationalité contient une liste exhaustive des pays du monde. Une nationalité peut être lié à zéro ou plusieurs auteurs ou à zéro ou plusieurs utilisateurs.

1.5 Modèle physique de donnée :



La base de données utilisée pour le projet est une base de données PostgreSQL. Ce type de base de données est intéressante de par sa simplicité de mise en œuvre et de ses performances optimisés (client assez léger). Au niveau du modèle de physique de données on retrouve une table par classe de notre diagramme uml excepté trois exceptions :

- L'héritage entre la classe item et les classes book et magazine est réalisé grâce à la stratégie d'héritage Hibernate Single Table. Cette stratégie impose que les tables correspondant aux classes filles possèdent tous les champs de la classe dont elles héritent et que la clé primaire de ces tables soit la même (ici intitulé item_id). Cela permet d'augmenter les performances d'Hibernate et de simplifier la mise en place des relations avec les autres entités. Ainsi la clé étrangère présente dans la table borrow sera la clé item_id permettant à hibernate de déterminer si le prêt concerne un magazine ou un livre.
- Les deux particularités suivantes concernent les tables author_book et author_citizenship qui sont nécessaires pour assurer ne relation many to many entre les tables auteur et livre pour la première, et entre auteur et nationalité pour la seconde. Ainsi on retrouvera dans

ces deux tables uniquement les identifiants des deux tables quels associe qui seront clé primaire et clé étrangère.

2 Solution technique :

2.1 JDK :

L'utilisation d'un JDK version 8 (ou 1.8) a été décidé pour favoriser la vitesse de développement offert par les nouvelles fonctionnalités de cette version du kit de développement java. Cette version a été préférée à la version 9 plus récente mais encore peu fiable et peu compatible avec certains outils.

2.2 Gestion des entités :

L'accès à la base de données est réalisé à l'aide d'un driver JDBC pour PostgreSQL, mais la gestion des entités (c'est-à-dire des objets java contenant les données) est réalisée par l'ORM Hibernate. Un ORM est un outil permettant de simplifier le mapping des objets java en fonction des données en base. Hibernate va donc permettre à l'aide d'un système d'annotation de mapper nos objets java avec les données en base en transcrivant les requête JPA (ou JPQL) s'appliquant sur nos objets dans un dialecte compréhensible par la base à savoir ici PostgreSQL. Cela simplifie ainsi les échanges entre le code java et la base de données ainsi que le travail du développeur tout en offrant des performances optimales.

Dans le projet Hibernate a été couplé à Spring Data et son système de crud repository permettant de de générer automatiquement les méthodes de création suppression et modification des entités ainsi que des méthodes génériques tel que les méthodes de récupération par id ou de récupération de toutes les entités. Ces repository ont également été utilisés pour créer des fonctions personnalisées en écrivant uniquement la requête JPQL correspondante. Les méthodes de recherche ont néanmoins nécessité l'utilisation d'une méthode de récupération plus classique de récupération des données via Hibernate et JPQL.

Les entités Hibernate possèdent néanmoins un inconvénient pour le projet à savoir l'impossibilité de les transférer directement via un webservice du fait de la création d'un xml sans fin lors du transfert d'un tel objet. Ainsi pour palier à ce problème les objets transmis par le webservice ont été transformer en DTO (Data Transfer Object) en tout points similaires en termes d'attributs que les entités leurs correspondant, mais permettant de gérer leurs dépendances vis-à-vis des autres entités d'une manière compréhensible par le protocole SOAP. Ainsi pour chaque entités (dans le package entité du modèle du projet webservice) correspond un objet DTO correspondant (dans le package DTO du modèle du projet webservice) et une classe de transformation permettant de passer d'une entité à un DTO et inversement (dans le package utils.transformer du module modèle du projet webservice). Les objets transitant entre le webservice et les clients seront donc toujours de type DTO (que ce soit pour le batch ou l'interface web).

2.3 Packaging, gestion des dépendances et architecture multi-modules :

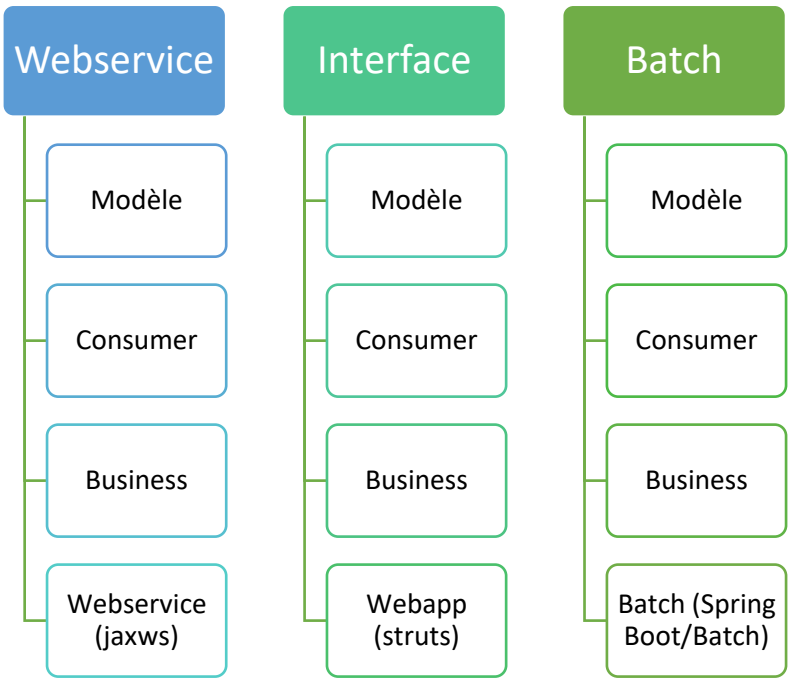
Maven, outil de gestion de build et de dépendances java a été utilisé. Cette solution a permis la mise en œuvre aussi bien d'une architecture développée en plusieurs modules (architecture multi-tiers ou multi-modules) les différentes briques logicielles du projet mais également d'intégrer à ces derniers les jars et autre dépendances nécessaires à leur développement et à leur fonctionnement.

Ainsi, pour cette solution, trois projets Maven ont été créés : batch, interface web et webservice, chacun découpé en quatre modules Maven (sous-projet Maven) :

- Module modèle : ce module contient les classes décrivant le modèle fonctionnel de la solution. Ce dernier n'est décrit qu'une seule fois dans le projet webservice et généré

via un plugin Maven à partir des WSDL des webservices SOAP pour les projets batch et interface web. Ce module est accessible à tous les autres modules du projet Maven.

- Module Consumer : Ce module contient la logique de récupération des données. Pour le projet webservice ce module se connecte directement à la base de données pour récupérer les informations demandées tandis que les deux autres projets utilisent ce module pour consommer l'un des webservices exposés dans le projet webservice. Ce module n'est accessible que via le module business.
- Module Business : Ce module contient la logique métier propre à l'application. C'est dans ce module que l'on va retrouver les appels au module consumer mais également les différents traitements des données nécessaires avant leur transmission à la couche applicative. Ce module n'est accessible que via la couche applicative.
- Module applicatif : Ce module diffère grandement selon les projets Maven de la solution. Pour la partie webservice il contient la logique d'exposition des webservices ainsi que la configuration nécessaire au fonctionnement de JAXWS. Pour la partie Batch il contient les classes de description des différents steps du job ainsi que la configuration de spring batch. Enfin pour la partie web interface il contient les différents éléments nécessaires à l'affichage des différentes pages web (contrôleurs, JSP, CSS, JS…). C'est dans ce module qu'est réuni la configuration spring via le fichier bootstrapContext.xml.



2.4 L'exposition des webservices (JAXWS) :

Le développement des webservices et leur exposition est réalisé selon la méthode bottom-up grâce à l'utilisation de JAXWS. Cet utilitaire java simplifie. Cette méthode consiste à générer le fichier wsdl de communication entre les clients et le webservice après avoir développé la logique dans le webservice. Elle fait opposition à la méthode top-down qui consiste à écrire le wsdl avant d'avoir implémenté toute logique dans le service. JAXWS permet ainsi d'exposer très simplement les méthodes de notre webservice via des annotions. Il se chargera ensuite lors du déploiement du war du webservice de générer le ou les wsdl nécessaire à son fonctionnement.

Le modèle ainsi que les classes et interface nécessaires pour l'appel de ce dernier dans le client sera alors automatisé lors du build du module modèle du client. Néanmoins pour assurer un build sans erreurs il sera nécessaire de déployer le webservice avant de lancer le build du client.

2.5 La mise en place du pattern MVC et d'un design responsive :

Pour implémenter le pattern MVC sur la partie interface (Modèle vue contrôleur) la technologie impsoée était struts 2. Ce framework Java EE permet via de nombreux outils d'accélérer le développement de cette partie. Les actions struts font office de contrôleur et permettent l'orientation de l'utilisateur en fonction des différents évènements ainsi que l'appel aux différentes fonctions du module business. En fonction des résultats retournés par le ou les contrôleurs struts va ensuite retourner une vue au format html ou jsp. Struts permet également via ses nombreux intercepteurs de réagir automatiquement à différents évènements comme la soumission d'un formulaire, la connexion d'un utilisateur… Struts fournit également une libriairie de type taglib utilisable dans les jsp. Cette dernière fournie de nombreuses fonctions permettant une mise en page évolutive de différents éléments (fonctions de types jstl).

Le développement du design responsive de l'interface web s'est quant à lui orienté autour de bootstrap 4. La très récente dernière version du framework développé par twitter offre au développeur la possibilité de d'axer son design autour de quatre points de ruptures et offre une mise en page simplifiée et accélérer grâce à des éléments tel que les cards ou encore les navbars. L'interface développé est en effet adaptée à tous les types de terminaux. L'implémentation de bootstrap implique également l'utilisation du framework javascript JQuery très utile pour accélérer la dynamisation des pages web du site (utilisé notamment lors du développement de la partie recherche de l'interface).

2.6 Le développement du batch :

Le développement du batch a été réalisé grâce notamment au framework Spring et plus spécifiquement grâce à l'outil spring Batch. Ce dernier permet de structurer les différents jobs développés mais également d'en accélérer leur exécution. Chaque job est subdivisé en step dont la logique est stockée dans des classes implémentant l'interface tasklet fournie par spring batch. Des décideurs peuvent également être développé pour déterminer la possibilité de passer au step suivant, de relancer le step précédent ou de tout simplement stopper l'exécution du job.

Le batch de rappelle des utilisateurs est ainsi organisé en un unique job de deux steps et un décideur. Le premier step récupère les prêts en retard et dresse la liste des utilisateurs à rappeler et leur fait correspondre la liste de leurs prêts en retard. Le décideur va ensuite déterminer si la liste ainsi retournée est vide et va arrêter l'exécution du job dans ce cas, ou va lancer l'exécution du job suivant dans tous les autres cas. Le dernier step consiste à l'envoi pur et simple d'un mail à chaque utilisateur de la liste contenant la liste de ses prêts en retard. Ce dernier step bénéficie également de la puissance du framework spring et plus particulièrement de spring mail qui simplifie l'envoi de mail.

3 Mise en œuvre du système :

3.1 Docker :

Docker est un outil de gestion conteneur très pratique permettant de simplifier le déploiement des différents composants de notre système d'information. Docker repose sur un système d'images

permettant de paramétrer les différents conteneurs en se basant soit sur une image existante (il est par exemple possible de récupérer une image avec un serveur web déjà configuré dessus) soit en créant son image de toute pièce. Chaque image est configurée et créée à partir d'un fichier Dockerfile présent dans le dossier src/main/docker des modules applicatifs de chaque projet Maven du projet.

Une fois les images créées il est alors possible de lancer un container à partir de chaque image. Chaque conteneur contiendra alors les différents composants configurés dans le Dockerfile de l'image. Le lancement des conteneurs peut être réalisé soit sur la machine virtuelle docker présente lors de l'installation soit sur une machine virtuelle créée (machine virtuelle vitualbox pour les système MAC, Linux et Windows sauf pour windows 10 qui intègre hyper v nativement et ne nécessite pas d'autres hyperviseur pour faire fonctionner docker).

Les images docker du système d'information sont organisées comme suit :

- Image liborrow-webservice-db : image contenant un serveur postgreSQL ainsi que la base de données liborrow avec le jeu de données intégré.
- Image liborrow-webservice-web : image contenant un serveur web glassfish5 hébergeant le webservice du SI. Doit être lancé après le container liborrow-webservice-db.
- Image liborrow-webinterface : image contenant un serveur web glassfish5 hébergeant l'interface web du système d'information. Doit être déployé après le container liborrow-webservice-web.
- Image liborrow-callagainbatch : image contenant le jar springboot démarré hébergeant le batch de rappel des utilisateurs du SI. Doit être déployé après le container liborrow-webservice-web.

3.2 Déploiement du SI :

3.2.1 Installation de docker :

Comme vu dans la partie docker il est nécessaire d'installer docker avant de réaliser les étapes de déploiements. Pour cela se rendre sur [urlDocker] et télécharger docker pour votre système d'exploitation (système de déploiement conseillé windows 10 ou windows server 2012 ou 2016).

3.2.2 Déploiement :

- Etape 1 : Télécharger sur le repo github les fichiers sources du projet.
- Etape 2 : Ajouter les fichiers de configuration comme suit :
  - Rendez-vous dans le dossier src/main/resources/com/liborrow/webapp du module webapp du webservice et ajouter le ficher jdbc.properties comme suit :

```
jdbc.driverClassName=org.postgresql.Driver
jdbc.url=jdbc:postgresql://ADRESSE_DE_VOTRE_SERVEUR_BDD/db_liborrow
jdbc.username=UTILISATEUR_DE_VOTRE_BDD
jdbc.password=MDP_DE_VOTRE_SERVEUR_BDD
hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
```

  - Rendez vous dans le dossier src/main/resources/com/liborrow/callagainbatch du module batch du batch et ajouter le ficher mail.properties comme suit :

```
liborrow.mail.host=smtp.gmail.com
liborrow.mail.port=587
liborrow.mail.sender.username= VOTRE_ADRESSE_MAIL_GOOGLE_D_ENVOI
liborrow.mail.sender.password= VOTRE_MOT_DE_PASSE
liborrow.mail.smtp.auth=true
```

```
liborrow.mail.smtp.ttls=true

Pour les SMTP différents de google :

liborrow.mail.host=NOM_SERVEURSMTP
liborrow.mail.port=587
liborrow.mail.sender.username= VOTRE_ADRESSE_MAIL_GOOGLE_D_ENVOI
liborrow.mail.sender.password= VOTRE_MOT_DE_PASSE
A adapter selon votre configuration
liborrow.mail.smtp.auth=true ou false
liborrow.mail.smtp.ttls=true ou false
```

- Rendez vous dans le dossier src/main/resources/com/liborrow/callagainbatch du module batch du batch et ajouter le ficher scheduler.properties comme suit :

```
#Executed every 10 seconds
com.liborrow.scheduler.delayDev= *\/10 * * * * *
#Executed every Sunday 12:00 am
com.liborrow.scheduler.delayProd= 0 0 12 * * SUN
```

- Etape 3 : Lancer le build maven du projet webservice (mvn clean package ou mvn clean install).
- Etape 4 : Ouvrez une fenêtre powershell administrateur et créez une machine virtuelle docker dédier au SI :
  - Pour Windows 10 : **docker-machine create -d hyperv --hyperv-virtual-switch "myswitch" vmLiborrow**
  - Pour Linux/MAC et autres versions de Windows : **docker-machine create --driver virtualbox vmLiborrow**
- Etape 5 : Vérifier la génération de votre machine : **docker-machine ls**
  - Ajouter dans votre fichier host (pour windows C:\Windows\System32\drivers\etc) l'alias liborrow pour l'adresse ip de votre machine nouvellement créée (présente dans la ligne url du tableau généré par le docker-machine ls).
  - Modifier les fichiers docker-compose CHEMIN_VERS_LE_DOSSIER\Liborrow\liborrow-callagainbatch\liborrow-callagainbatch-batch\docker-compose.yml et CHEMIN_VERS_LE_DOSSIER\Liborrow\liborrow-webinterface\liborrow-webinterface-webapp\docker-compose.yml :

```
extra_hosts:
      - "liborrow:IP_DE_VOTRE_MACHINE "
```

```
Administrateur : Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. Tous droits réservés.

PS C:\WINDOWS\system32> docker-machine ls
NAME      ACTIVE   DRIVER    STATE     URL                         SWARM   DOCKER        ERRORS
default   -        hyperv    Stopped                                       Unknown
myvm1     -        hyperv    Running   tcp://192.168.1.24:2376             v18.02.0-ce
PS C:\WINDOWS\system32>
```

- Etape 6 : Sélectionner votre vm docker :
  - **docker-machine env vmLiborrow**

```
PS C:\WINDOWS\system32> docker-machine env myvm1
$Env:DOCKER_TLS_VERIFY = "1"
$Env:DOCKER_HOST = "tcp://192.168.1.24:2376"
$Env:DOCKER_CERT_PATH = "C:\Users\Ben\.docker\machine\machines\myvm1"
$Env:DOCKER_MACHINE_NAME = "myvm1"
$Env:COMPOSE_CONVERT_WINDOWS_PATHS = "true"
# Run this command to configure your shell:
# & "C:\Program Files\Docker\Docker\Resources\bin\docker-machine.exe" env myvm1 | Invoke-Expression
```

  - Copier coller et exécuter la commande qui s'affiche du type : **& "C:\Program Files\Docker\Docker\Resources\bin\docker-machine.exe" env vmLiborrow | Invoke-Expression**
  - **docker-machine ls** de nouveau pour vérifier que le shell docker sera désormais activé sur la machine virtuelle créée (* dans la case active si la commande a fonctionnée).

```
PS C:\WINDOWS\system32> & "C:\Program Files\Docker\Docker\Resources\bin\docker-machine.exe" env myvm1 | Invoke-Expression
PS C:\WINDOWS\system32> docker-machine ls
NAME      ACTIVE   DRIVER    STATE     URL                         SWARM   DOCKER        ERRORS
default   -        hyperv    Stopped                                       Unknown
myvm1     *        hyperv    Running   tcp://192.168.1.24:2376             v18.02.0-ce
```

- Etape 7 : Déployer la base de données et le webservice :
  - **cd CHEMIN_VERS_LE_DOSSIER\Liborrow\liborrow-webservice\liborrow-webservice-webapp\src\main\docker\db.**
  - **docker build -t liborrow-webservice-db .** (Création de l'image de la bdd).
  - Copier le war du webservice généré dans le dossier target du module webapp du projet liborrow-webservice dans le dossier CHEMIN_VERS_LE_DOSSIER\Liborrow\liborrow-webservice\liborrow-webservice-webapp\src\main\docker\web.
  - **cd CHEMIN_VERS_LE_DOSSIER\Liborrow\liborrow-webservice\liborrow-webservice-webapp\src\main\docker\web.**
  - **docker build -t liborrow-webservice-web .** (Création de l'image du webservice).
  - **cd CHEMIN_VERS_LE_DOSSIER\Liborrow\liborrow-webservice\liborrow-webservice-webapp**
  - **docker-compose up (lance les deux containers de la bdd et du webservice).**
- Etape 8 : Déployer la webinterface et le batch :
  - Réaliser un mvn clean package ou mvn clean install sur les deux projets liborrow-webinterface et liborrow-callagainbatch.
  - Copier le war de la webinterface générer dans le dossier target du module webapp du projet dans le dossier :

CHEMIN_VERS_LE_DOSSIER\Liborrow\liborrow-webinterface\liborrow-webinterface-webapp\src\main\docker

- **cd CHEMIN_VERS_LE_DOSSIER\Liborrow\liborrow-webinterface\liborrow-webinterface-webapp\src\main\docker**
- **docker build -t liborrow-webinterface .** (Création de l'image de la webinterface).
- **cd CHEMIN_VERS_LE_DOSSIER\Liborrow\liborrow-webinterface\liborrow-webinterface-webapp**
- **docker-compose up (lance le container de la webinterface).**
- Copier le jar du batch généré dans le dossier target du module batch du projet liborrow-callagainbatch dans le dossier : CHEMIN_VERS_LE_DOSSIER\Liborrow\liborrow-callagainbatch\liborrow-callagainbatch-batch\src\main\docker
- **cd CHEMIN_VERS_LE_DOSSIER\Liborrow\liborrow-callagainbatch\liborrow-callagainbatch-batch\src\main\docker**
- **docker build -t liborrow-callagainbatch . (Création de l'image du batch).**
- **cd CHEMIN_VERS_LE_DOSSIER\Liborrow\liborrow-callagainbatch\liborrow-callagainbatch-batch**
- **docker-compose up (lance le container du batch).**

- **Etape 9 : Se rendre sur les différents éléments pour vérifier leur fonctionnement :**
  - **http://liborrow:8080/liborrow-webservice-webapp**
  - **http://liborrow:80/liborrow-webinterface/login.action**
  - **http://liborrow:8888/callagainJob (lance le job)**