LIBRARY MANAGER

DOCUMENTATION

Juin 2019 – Version 1.1

Le projet LibraryManager est un système d'information basé sur un web service SOAP délivrant des services dédiés à la gestion d'une base de données de prêts de livre pour une bibliothèque.

1. TABLE DES MATIERES

2. INFORMATIONS GENERALES

1.1. COMPOSITION DU PROJET

Web service (LibraryManager-WS)

Le web service fournit à l'ensemble du système plusieurs opérations permettant à la fois la gestion des prêts et également la gestion d'autres informations de la base de données tels que les membres, les livres ou les catégories de livres.
Le WS utilise Apache CXF pour générer les classes Java à partir des fichiers WSDL décrivant chaque service. La connexion à la base de données est gérée à l'aide Hibernate. Les modèles créés implémentent des annotations permettant la création automatique des tables dans la base de données (voir la section Configuration).

Application Web (LibraryManager-WebApp)

L'application Web permet aux utilisateurs d'accéder à une interface graphique pour gérer les données. Elle créée à l'aide de Spring MVC et n'accède aux informations qu'à travers le WS qui lui fournit les fichiers WSDL pour créer un client.

Batch de relance par email (LibraryManager-Batch)

Le batch est utilisée pour envoyer un mail aux membres n'ayant pas rendu leurs livres empruntés à temps. Il récupère les prêts concernés et les utilise afin d'agréger les données nécessaires à l'envoi des mails de relance. Le batch ne se connecte à la base de données qu'à travers le WS. Une fois lancé, il renouvelle l'envoi de mails à intervalle régulier (configuré dans le batch par défaut à un lancement quotidien).

1.2. TECHNOLOGIES

Le Web Service, le batch et l'application web se basent sur Spring Framework 5.1.3, notamment l'injection de dépendance, et Maven 3.6.0, pour la gestion des dépendances.

Le Web Service utilise en plus Hibernate 5.4.0 pour la gestion de la relation avec la base de données PostgreSQL 42.2.5, notamment la gestion des requêtes et transfert de données via l'EntityManager. Il utilise également Apache CXF 3.2.7 pour la gestion de la génération des classes Java et des interfaces à partir des fichiers WSDL.

L'application web utilise Spring MVC pour implémenter le modèle MVC et la gestion des requêtes HTTP, ainsi qu'Apache CXF pour générer les classes clientes du web service.

Le batch est construit à l'aide de Spring Batch 4.1.0 et Javax Mail 1.6.2 pour la gestion des Jobs et l'envoi des mails de relance aux utilisateurs.

3. CONCEPTION

1.3. FONCTIONNALITES

Gestion des Prêts

Le service dédié aux prêts propose des opérations CRUD ainsi que des méthodes pour modifier les attributs liés au rendu et à l'extension du prêt.

Gestion des Livres

Le service dédié aux livres permet de gérer les opérations CRUD ainsi que des méthodes pour rechercher les livres dans la base de données selon leurs attributs "Category", "Author" et "Title".

Gestion des Catégories

Le service dédié aux catégories permet de gérer les occurrences de la table "categories" de la base de données via des méthodes CRUD.

Gestion des Membres

Le service dédié aux membres permet de gérer les occurrences de la table "members" de la base de données via des méthodes CRUD.

Récupération des prêts en retard

Le service dédié à la récupération des prêts en retard propose une seule opération permettant au batch de récupérer les occurrences de "loan" dont l'attribut "DateEnd" est antérieur à la date de la requête.

Authentification

Le service dédié à l'authentification propose les opérations de connexion et déconnexion à l'application web via un couple identifiant/mot de passe. L'opération utilise BCrypt pour vérifier que le mot de passe hashé correspond au mot de passe envoyé par l'utilisateur.

Gestion des utilisateurs

L'un des services web proposés permet à un administrateur de se connecter à une interface d'administration afin de gérer les comptes utilisateurs : affichage, ajout, modification et suppression.

1.4. CONTEXTE

Environnement

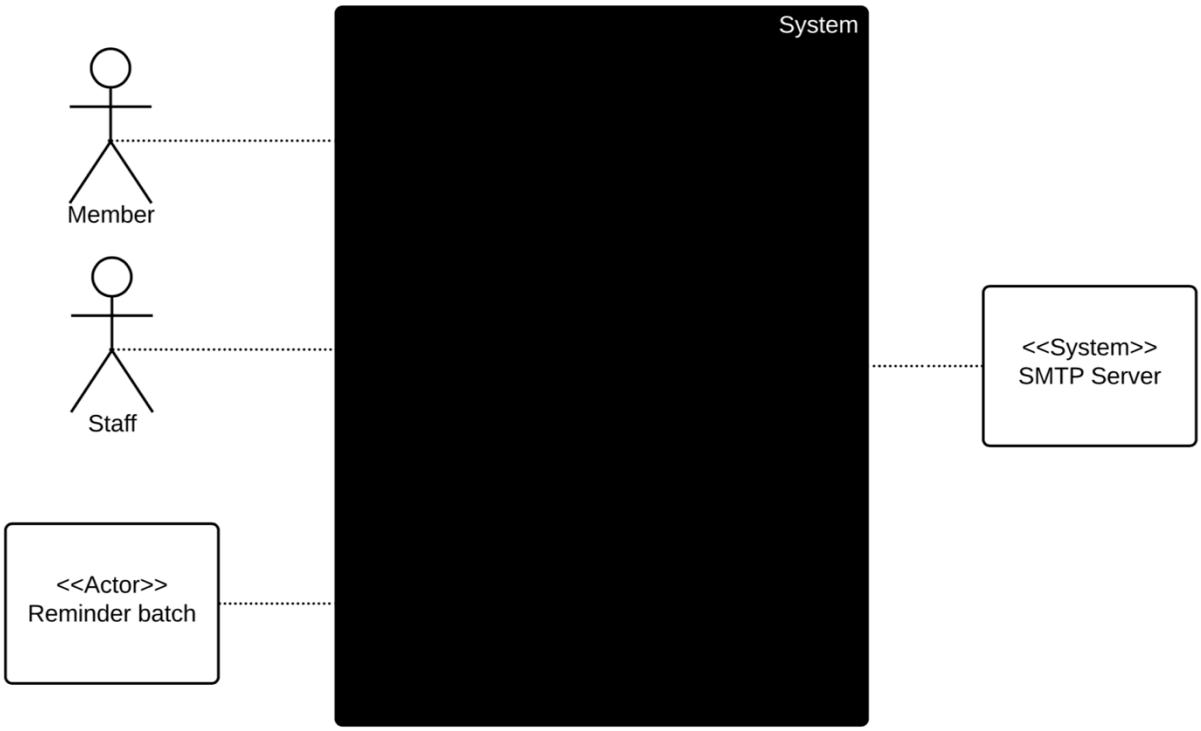

Figure 1 : Diagramme de contexte

Diagramme de package

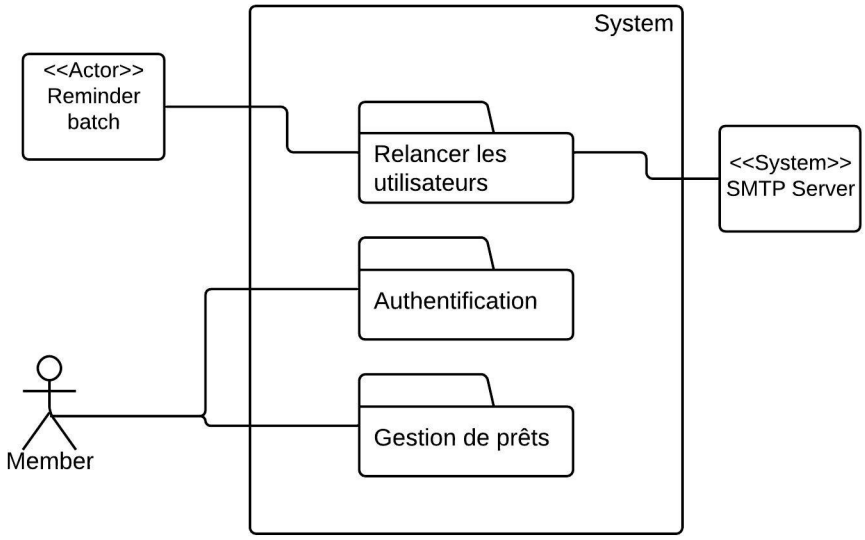

Figure 2 : Diagramme de package

4. DOMAINE FONCTIONNEL

1.5. MODELE CONCEPTUEL DE DONNEES

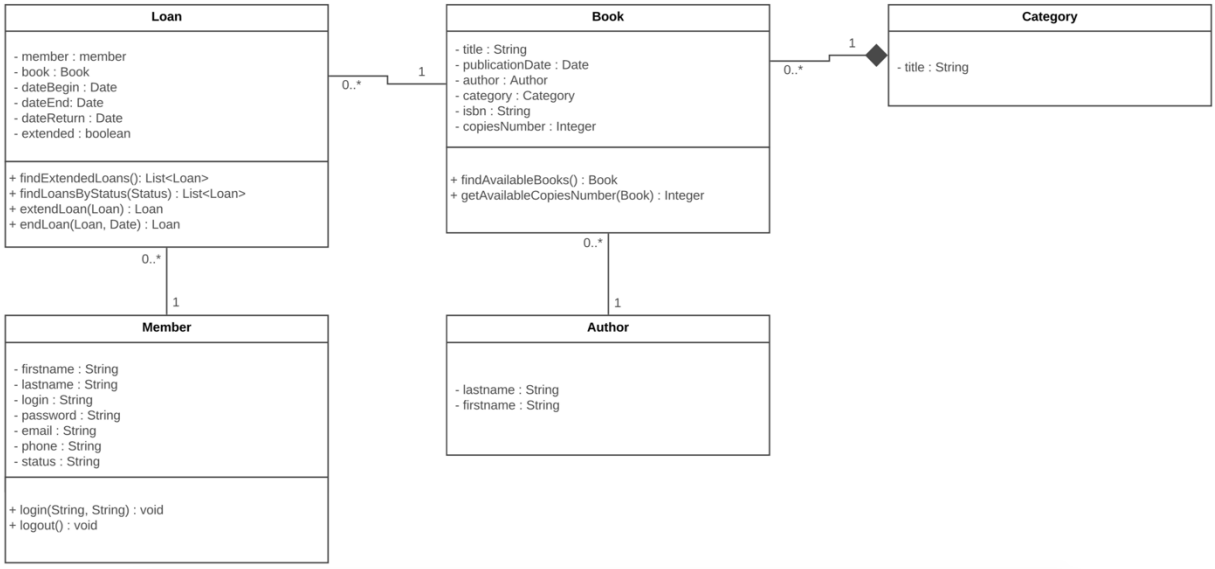

Figure 3 : Diagramme de classe

1.6. MODELE PHYSIQUE DE DONNEES

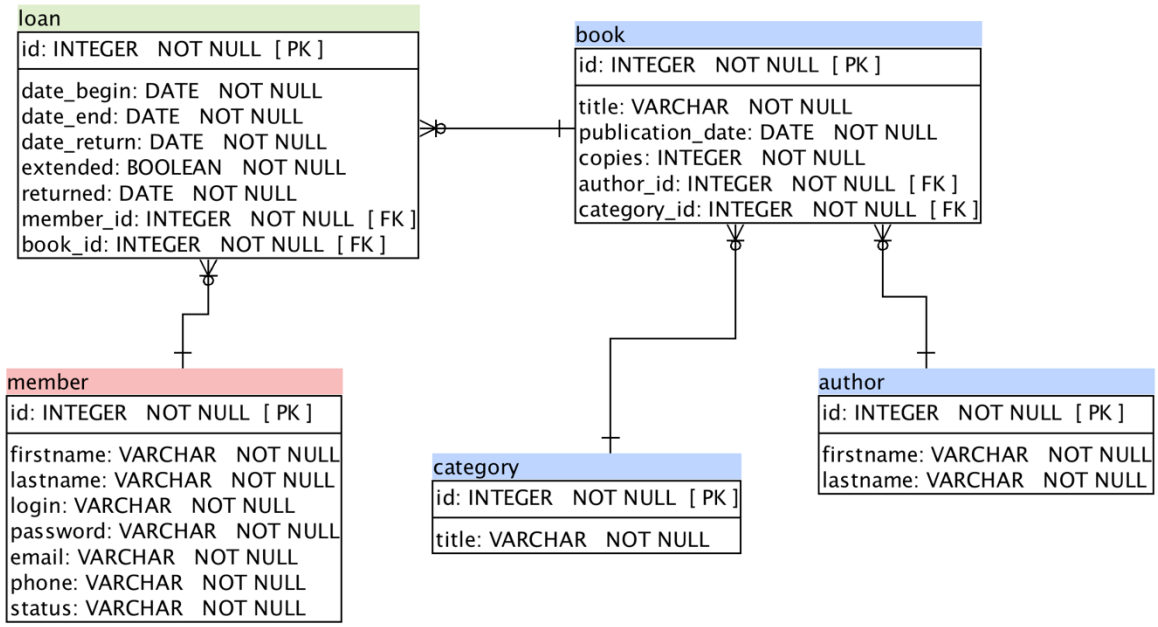

Figure 4 : Modèle physique de données

5. CONFIGURATION

La configuration du web service, de l'application et du batch repose sur les URLs d'accès aux fichiers WSDL et à la base de données.

Configuration du Web Service

1 - Base de données

La base de données utilisée est PostgreSQL 10. Il suffit de la créer, Hibernate se chargera de créer l'ensemble des tables et des séquences au premier lancement du Web Service.

En cas de nécessité, le script SQL de création est disponible avec les données de test

L'accès à la base de données est à configurer dans le fichier context.xml (module Webservice) et dans le fichier db.properties (module Consumer) :

context.xml

```
url="jdbc:postgresql://localhost:5432/librarymanager_db"
driverClassName="org.postgresql.Driver"
username="postgres"
password="admin"
```

db.properties

```
url=jdbc:postgresql://localhost:5432/librarymanager_db
driverClassName=org.postgresql.Driver
username=postgres
password=admin
```

Des modifications dans ces fichiers nécessitent également la mise à jour de la ressource JNDI pour le serveur de déploiement Tomcat (version utilisée : 9.0.13) dans les fichiers context.xml (module Webservice) et dans le fichier web.xml (module WebService) :

context.xml

```
name="jdbc/LMWSDB"
```

web.xml

```
<resource-ref>
 <res-ref-name>jdbc/LMWSDB</res-ref-name>
 <res-type>javax.sql.DataSource</res-type>
 <res-auth>Container</res-auth>
</resource-ref>
```

2 - WSDL

Pour générer les classes et interfaces Java nécessaires aux clients du web service, plusieurs fichiers POM.xml sont à modifier selon le modèle de déploiement utilisé. Ces fichiers possèdent les URLs pour accéder aux fichiers WSDL. Par défaut, l'URL se base sur lien vers localhost :
http://localhost:8090/librarymanager/ws/?wsdl*

Ces URLs sont à modifier dans les fichiers de configuration POM.xml dans les modules *Model* de l'application web et du batch. Ainsi ils pourront chacun utiliser les contrats WSDL fournis par Apache CXF pour générer les classes, les interfaces et les implémentations nécessaires à l'utilisation des services.

3 - Génération des éléments Java

Pour générer les classes, interfaces et implémentations, le Web Service doit être compilé puis le serveur de déploiement du web service doit être lancé. Une fois disponible, une page avec l'ensemble des fichiers WSDL disponibles va s'ouvrir.

Pour générer les mêmes éléments pour le batch et l'application web, tout en laissant tourner le serveur du web service, il faut à nouveau compiler le batch et l'application pour que la génération via Apache CXF se lance automatiquement.

Une fois les classes générées, toutes les applications sont pleinement utilisables et connectées au web service.

4 - Configuration du Batch

Le Batch nécessite la configuration d'un serveur SMTP et d'un compte pour envoyer les emails de relance aux utilisateurs.
Dans le fichier config.properties, 4 lignes sont à modifier selon le fournisseur (par défaut, le batch est configuré pour GMail) :

```
spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=
spring.mail.password=
```

6. DEPLOIEMENT

Avec Tomcat

Chaque composant du projet est à déployer sur un serveur Tomcat dédié.

Web Service

Le web service est déployé avec le package WAR généré en utilisant la commande *mvn package*
Le serveur Tomcat doit être configuré avec un port différent afin de laisser le port *8080* à l'application web.
Par défaut, le serveur Tomcat du Web Service est déployé sur le port *8090* et sur le chemin */librarymanager/*.

Application web

A l'instar du Web Service, l'application, web est déployé à l'aide du package WAR généré par la commande mvn package (ou via l'IDE) et d'un serveur Tomcat configuré sur port *8080* (par défaut) sur le chemin */librarymanager/*.

Avec Docker

Le déploiement via Docker utilise 4 containers (BDD, WebService, Maven et WebApp). Avant de lancer la création des images, il faut créer le network dédié à l'application avec la commande :

```
docker network create lmnetwork
```

Pour la base de données

Le container dédié à la base de données est créé à partir du Dockerfile présent dans le dossier db dans le repository GitHub. L'image utilisée se base sur celle de PostgreSQL 10.6 diponible sur Docker Hub. Le Dockerfile contient les identifiants pour créer la base de données et l'administrateur.

Pour créer l'image, utiliser la commande :
docker build -t db /chemin/vers/db/

Pour lancer le container, utiliser la commande :

docker run --rm -d --network lmnetwork -p 5433:5432 --name db db

Afin d'insérer les premières données, il est possible d"utiliser un outil comme DBVisualizer pour lancer le script data.sql présent dans le repository.

Pour le Web Service

Le container dédié au web service s'appuie sur l'image de Tomcat 9.0.14 fournie par Docker Hub. La première étape consiste à créer l'archive WAR de l'application via l'IDE ou la commande :

mvn clean package

La seconde étape est la création de l'image. Le Dockerfile est présent dans le dossier de l'application *LibraryManager-WS*. Il suffit de lancer la commande :

docker build -t webservice /chemin/vers/LibraryManager-WS/

Une fois lancée, Docker va créer une image à partir de Tomcat, puis dupliquer l'archive WAR dans le dossier webapps de l'installation de Tomcat dans le container.
La dernière étape consiste à lancer (en fond) le serveur avec la commande :

docker run --rm -d --network lmnetwork -p 8090:8080 --name webservice webservice

L'ensemble des opérations disponibles et des lien vers les WSDL sera accessible sur l'URL http://localhost:8090/webservice de la machine locale

Il est indispensable que le container db soit démarré

Pour compiler l'Application Web (utilisant le network de Docker)

L'application web complique le déploiement : si l'on souhaite compiler depuis l'IDE ou un terminal de la machine local, il faudra configurer sur l'adresse http://localhost:8090 mais celle-ci ne sera pas disponible dans le container et lèvera une erreur Connection refused.
Le POM.xml du module Model est à configurer avec l'adresse http://webservice:8080/webservice (si la configuration de base est gardée)

Si la configuration est modifiée, il faudra mettre à jour l'URL en fonction des noms du container et de l'image.

Aucune image n'est nécessaire pour la compilation via Maven. Placez-vous dans le dossier de l'application web "LibraryManager-WebApp" et lancez la commande :

docker run --rm -d -v "$PWD":/usr/local/lmwebapp --network lmnetwork --name lmcompile -w /usr/local/lmwebapp maven:3.6.0-jdk-8 mvn clean package

Cette commande va copier le dossier de l'application en tant que volume et lancer *mvn clean package* dans le dossier indiqué par le paramètre "-w".

L'archive WAR nécessaire au lancement de l'application a été généré dans le dossier /webapp/target comme si la commande package avait été lancée depuis l'IDE ou dans un terminal de l'hôte.

Il est indispensable que le container webservice soit démarré

Pour l'Application Web

A l'instar du container du Web Service, celui de l'Application Web est également basé sur Tomcat 9.0.14-jre8. Le Dockerfile, présent dans le dossier "LibraryManager-WebApp" du répository, va se baser sur l'image depuis Docker Hub puis copier l'archive WAR packagée par Maven (dans l'étape précédente) dans le dossier webapps de l'installation de Tomcat.
L'image est créée à l'aide de la commande :

docker build -t webapp /chemin/vers/LibraryManager-WebApp

Le container est à lancer avec la commande :

docker run --rm -d --network lmnetwork -p 8888:8080 --name webapp webapp

> Il est indispensable que le container webservice soit démarré

L'application est disponible à l'adresse *http://localhost:8888/webapp*

Utiliser Docker-Compose
Il est possible d'utiliser Docker-Compose pour créer les images, containers et les déployer automatiquement.

Il est impératif que la Web Application ait été compilée à l'aide d'un container Docker pour que le docker-compose puisse déployer correctement l'ensemble des éléments

Dans un Terminal, déplacez-vous jusque dans le dossier LibraryManager et utilisez les commandes :

docker-compose build --no-cache
docker-compose up

L'ensemble des containers sur les ports définis dans les fichiers de configuration XML des différentes applications.

7. CAS D'UTILISATIONS

1.7. UTILISER UN COMPTE UTILISATEUR

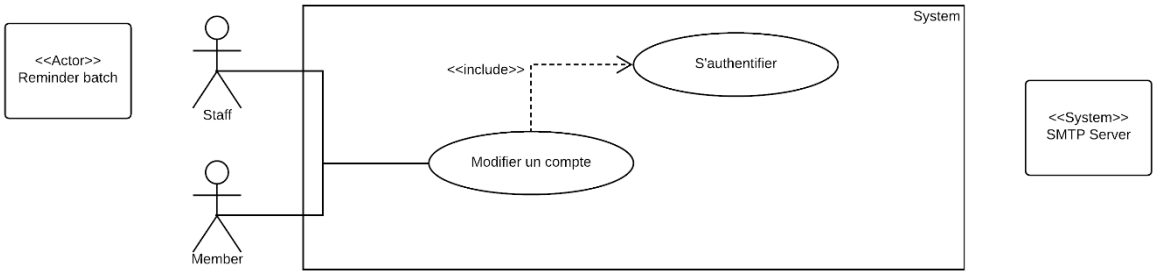

Figure 5 : Diagramme de cas d'utilisation « Utiliser un compte utilisateur »

Identification

Numéro : 1

Nom : Utiliser un compte utilisateur
Acteurs : Staff, Membre
Description : Un Membre ou un Staff peut modifier un compte.

Auteur : Anthony T.
Date : 24/10/2018

Préconditions : Le Membre ou le Staff a un compte enregistré dans la base de données.
Démarrage : Le Membre ou le Staff se connecte à l'application web.

Description des scenarii

Scénario :

1. Le *Membre* / *Staff* se connecte à l'application web
2. Le *Membre* / *Staff* renseigne son identifiant et son mot de passe
3. Le *système* vérifie la concordance des éléments dans la base de données
4. Le *Membre* / *Staff* accède à son compte
5. Le *Membre* / *Staff* modifie les informations de son compte

Scénarii alternatifs :

1. B. Le *Membre* / *Staff* quitte l'application
2. B. Le *Membre* / *Staff* annule
4. B. Le *système* refuse l'accès
4. C. Le *Membre* / *Staff* quitte l'application
5. B. Le *Membre* / *Staff* quitte l'application

Fin :

- Scénario nominal : Le *Membre* / *Staff* a modifié ses informations.
- Scénario d'exception : *Membre* / *Staff* quitte l'application (points 1, 2 ou 4) ou accède à d'autres pages du site (points 4 ou 5).

Post-conditions

- **Scénario nominal** : Le *Membre* / *Staff* est connecté et les nouvelles informations sont enregistrées dans la base de données.
- **Scénario d'exception** : Le *Membre* / *Staff* est connecté mais ne modifie aucune donnée ou Le *Membre* / *Staff* n'est pas connecté. Aucune information n'est modifiée ou enregistrée en base de données.

Compléments

Question(s) en attente :

- Dans quelle application du système s'effectue l'enregistrement des nouveaux utilisateurs ?

1.8. GERER UN PRET

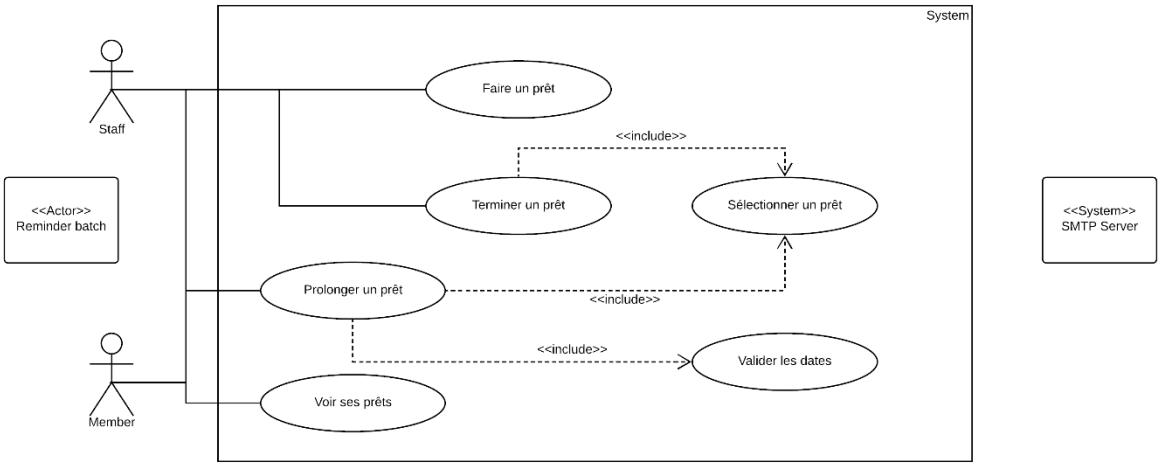

Figure 6 : Diagramme de cas d'utilisation « Gérer un prêt »

Identification

Numéro : 2
Nom : Gérer un prêt
Acteurs : Staff, Membre
Description : Un Membre a la possibilité de voir ses prêts et de les prolonger.

Auteur : Anthony T.
Date : 24/10/2018

Préconditions : Le Membre doit être connecté.
Démarrage : Le Membre se rend sur son compte.

Description des scenarii

Scénario :

1. Le *Membre* affiche ses prêts
2. Le *Membre* sélectionne un prêt
3. Le *Membre* prolonge son prêt
4. Le *système* valide la date de prolongation
5. Le *système* enregistre la prolongation

Scénarii alternatifs :

1. B. Le *Membre* quitte la page
2. B. Le *Membre* quitte la page
3. B. Le *Membre* annule sa modification
4. B. Le *système* ne valide pas la date de prolongation
5. B. Le *système* n'enregistre pas les modifications

Fin :

- Scénario nominal : Le *système* a validé et enregistré la date de prolongation du prêt.
- Scénario d'exception : Le *système* n'a pas validé les dates (point 4) ou le *Membre* a annulé (points 1, 2 ou 3). Le *système* n'a enregistré aucune prolongation (point 5).

Post-conditions

- Scénario nominal : Le *Membre* a prolongé un de ses prêts et ne peut plus le prolonger.
- Scénario d'exception : Aucune nouvelle donnée n'a été enregistrée en base de données.

Compléments

Question(s) en attente :

- La prolongation se fait en jours, en semaines ou par mois ?

2.1. RELANCER LES UTILISATEURS

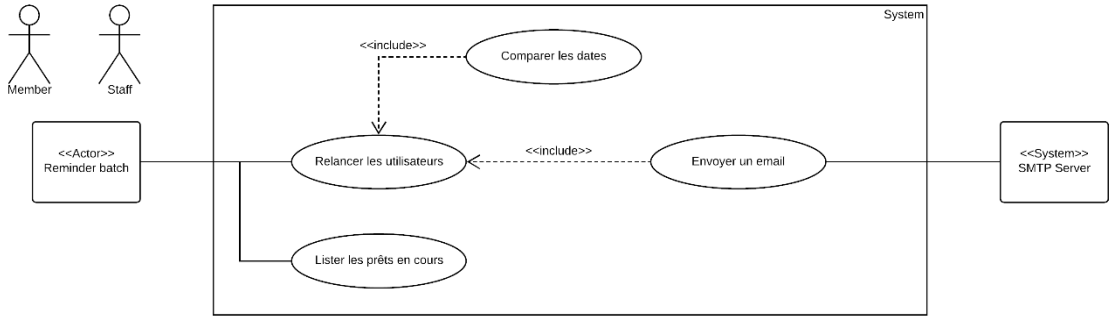

Figure 7 : Diagramme de cas d'utilisation « Relancer les utilisateurs »

Identification

Numéro : 3
Nom : Relancer les utilisateurs
Acteur : Reminder Batch
Description : Le Reminder Batch relance les Membre ayant un prêt se terminant via email.

Auteur : Anthony T.
Date : 24/10/2018

Préconditions : Le lancement du batch est automatisé et connecté à l'application web.
Démarrage : Le script du Reminder Batch se lance.

Description des scenarii

Scénario :

1. Le *Reminder Batch* liste les prêts en cours
2. Le *Reminder Batch* compare les dates de fin à la date du jour
3. Le *Reminder Batch* relance les utilisateurs,
4. Le *système* fait appel au *SMTP Server* pour envoyer les emails
5. Le *SMTP Server* envoie les emails

Scénarii alternatifs :

1. B. Le *Reminder Batch* ne trouve aucun prêt en cours
3. B. Le *Reminder Batch* ne relance aucun utilisateur

Fin :

- Scénario nominal : Le *Reminder Batch* envoie des emails de relance aux utilisateurs via le *SMTP Server.*
- Scénario d'exception : Le *Reminder Batch* ne trouve aucun prêt, aucun email de relance n'est envoyé.

Post-conditions

- **Scénario nominal** : Les emails de relance sont envoyés.
- **Scénario d'exception** : Aucun email de relance n'est envoyé.

Compléments

Question(s) en attente :

- A quelle fréquence doit se lancer le Reminder Batch ?

3.1. GERER LES MEMBRES

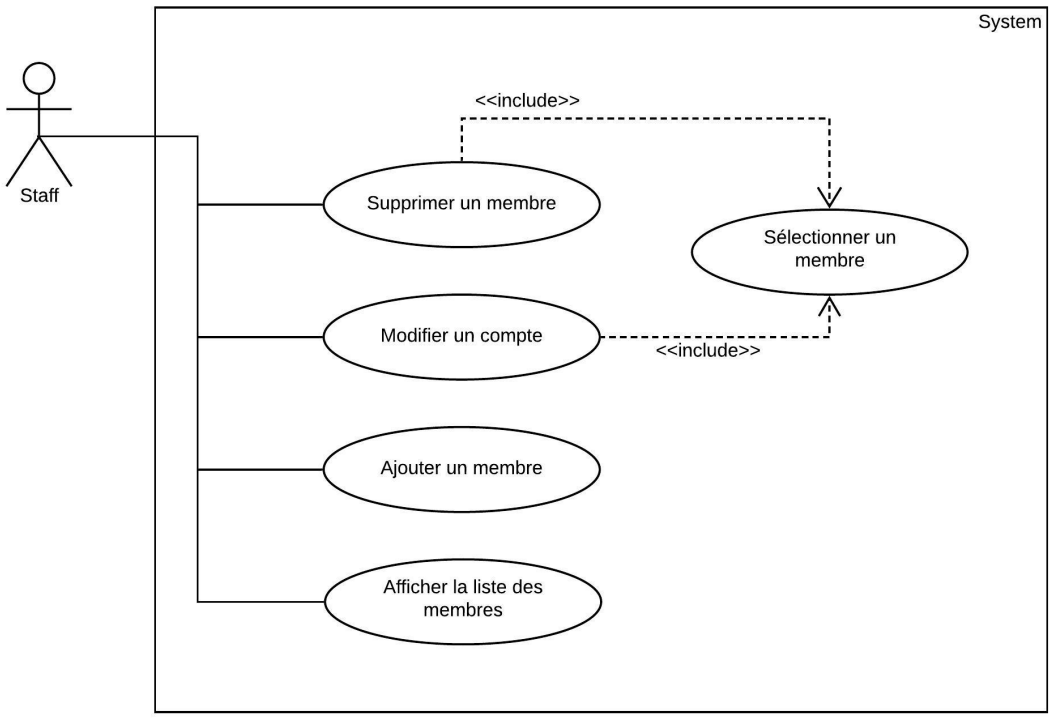

Figure 8 : Diagramme de cas d'utilisation « Gérer les membres»

Identification

Numéro : 4
Nom : Gérer les membres
Acteur : Staff
Description : Un Staff peut ajouter, modifier ou supprimer un Membre

Auteur : Anthony T.
Date : 05/06/2019

Préconditions : Le Staff doit être authentifié

Démarrage : Le Staff accède à l'interface administration

Description des scenarii

Scénario :

1. Le *Staff* affiche la liste des membres inscrits
2. Le *Staff* sélectionne un membre
3. Le *Staff* modifie un membre

Scénarii alternatifs :

2. B. Le *Staff* ajoute un membre
3. B. Le *Staff* supprime un membre

Fin :

- Scénario nominal : Le *Staff* a modifié un membre.
- Scénario d'exception : Le *Staff* a supprimé un membre (point 2) ou a ajouté un nouveau membre (point 3).

Post-conditions

- **Scénario nominal** : Le membre a été modifié et ses changements persistés dans la base de données
- **Scénario d'exception** : Le membre ajouté et persisté en base de données (point 2) ou le membre supprimé a été retiré de la base de données

Compléments

8. ARCHITECTURE

4.1. WEB SERVICE

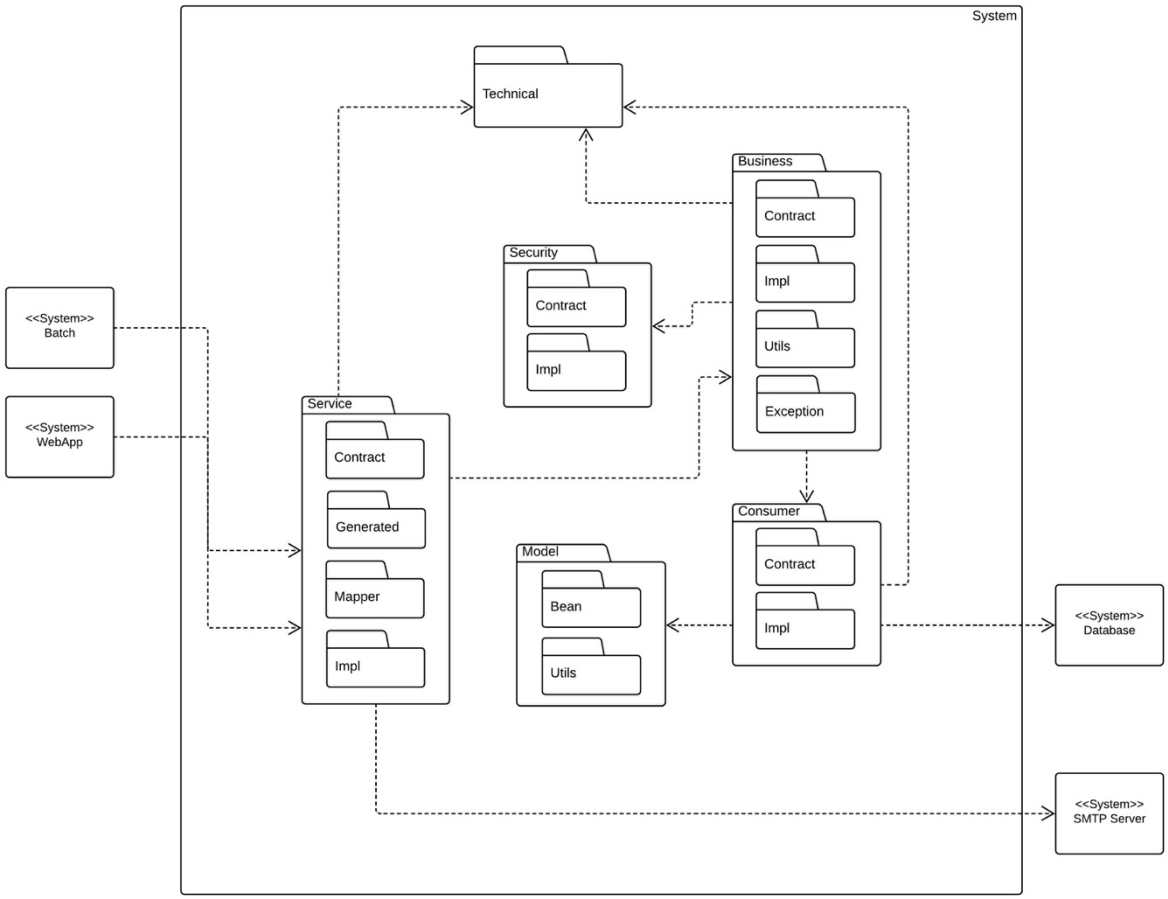

Figure 9 : Diagramme de package du web service

4.2. WEB APPLICATION

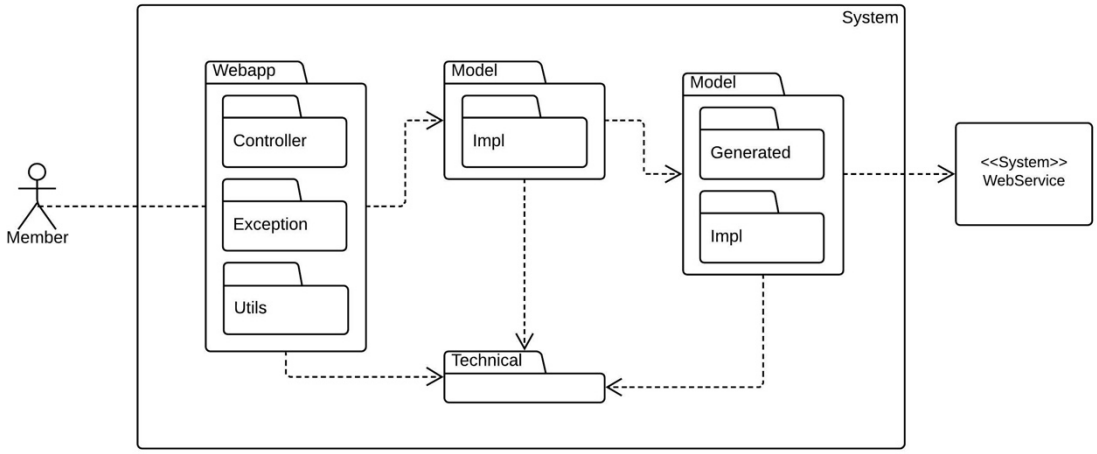

Figure 10 : Diagramme de package de l'application web

4.3. BATCH

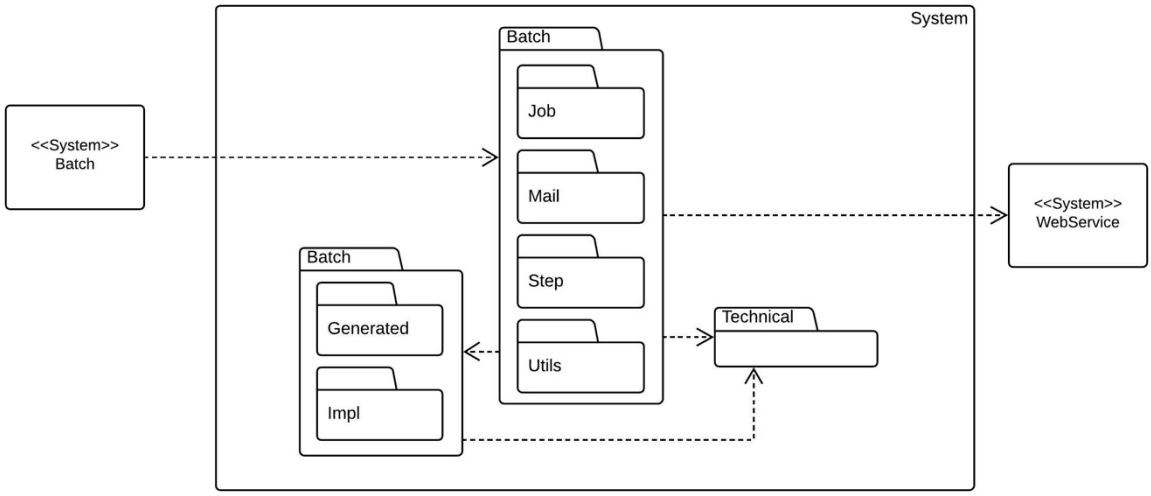

Figure 11 : Diagramme de package du batch

9. CONTACT

Auteur : Anthony Tazzari
Email : anthony.tazzari@gmail.com

Version : 1.1
Date : Juin 2019

Repository GitHub : https://github.com/getantazri/LibraryManager-P10