Projet 4 - Développez le nouveau système d'information de la bibliothèque d'une grande ville

---

OpenClassRooms - Arnaud Barbaria

11/02/2018

Contexte

Le service culturel d'une grande ville souhaite moderniser la gestion de ses bibliothèques. Pour cela, elle désire mettre à disposition de ses usagers, un système de suivi des prêts de leurs ouvrages.

Ce système comprendra :

- un site web (en responsive design) accessible aux usagers et permettant :
    - de rechercher des ouvrages et voir le nombre d'exemplaires disponibles
    - de suivre leurs prêts en cours. Les prêts sont pour une période de 4 semaines (durée configurable)
    - de prolonger un prêt. Le prêt d'un ouvrage n'est prolongeable qu'une seule fois. La prolongation ajoute une nouvelle période de prêt (4 semaines, durée de prolongation configurable) à la période initiale
- une application mobile iOS et Android fournissant les mêmes services que le site web
- une application spécifique pour le personnel des bibliothèque permettant, entre autres, de gérer les emprunts et le livres rendus
- un batch lancé régulièrement et qui enverra des mails de relance aux usagers n'ayant pas rendu les livres en fin de période de prêt

À vous de réaliser ce système !

Les applications pour smartphones et l'application du personnel est développé par une autre entreprise.

Spécification

Le client a souhaité que :

- le SGBD soit PostgreSQL
- l'application server soit une application JEE
- l'application cliente soit basée sur le framwork Apache Struts2
- une démarche SOA soit adopté
- les applications doivent être dockerisé

Description du projet

Ensemble des projets:

A la racine de l'ensemble des projets on retrouve les trois projets: le batch, le web-service ainsi que l'application cliente.
Il y a ensuite un fichier docker-compose.yml qui permet de dockeriser le projet.

On retrouve également un fichier .env dans lequel on peut définir des propriétés du projet qui seront lus par docker-compose.

Web-Service:

Dans le dossier projet_4_web_service on trouvera different fichier sql pour la mise en place de la base données et un fichier sh qui lance les différents sql dans le bon ordre.
A la racine du projet on retrouvera un fichier pom permettant de déclarer les propriétés du projet, les dépendances, les profiles et le build.
Enfin on trouve un fichier Dockerfile qui sera lancé directement par docker-compose.
Dans src/main/java il y a plusieurs package :

- io.ab.library: method main et classes de configuration.
- io.ab.library.model: représentation des entités de la base de données.
- io.ab.library.repository: comprend des extensions du CrudRepository de Spring permettant un accès simplifié à la base de données.
- io.ab.library.service: des interfaces décrivant les méthodes permettant de manipuler les données brutes extraites de la base de données.
- io.ab.library.service.impl: implémentations des interfaces de io.ab.library.service.
- io.ab.library.controller.soap: les différents endpoint du web-service.
- io.ab.library.controller.request/dto/response: représentation des entités telles que décrites dans WSDL.

Dans src/main/ressources on retrouve des fichiers xsd permettant de décrire le WSDL ainsi qu'un fichier application.properties pour déclarer propriétés de l'application.

Dans src/main/filters sont déclaré des propriétés du projet qui seront ajoutés aux propriétés de l'application. Ces propriétés seront redéfinies par docker-compose.

Web-App:

Comme dans le web-service on retrouve un pom.xml et un Dockerfile. Cependant, ici, le projet est séparé en différent modules.

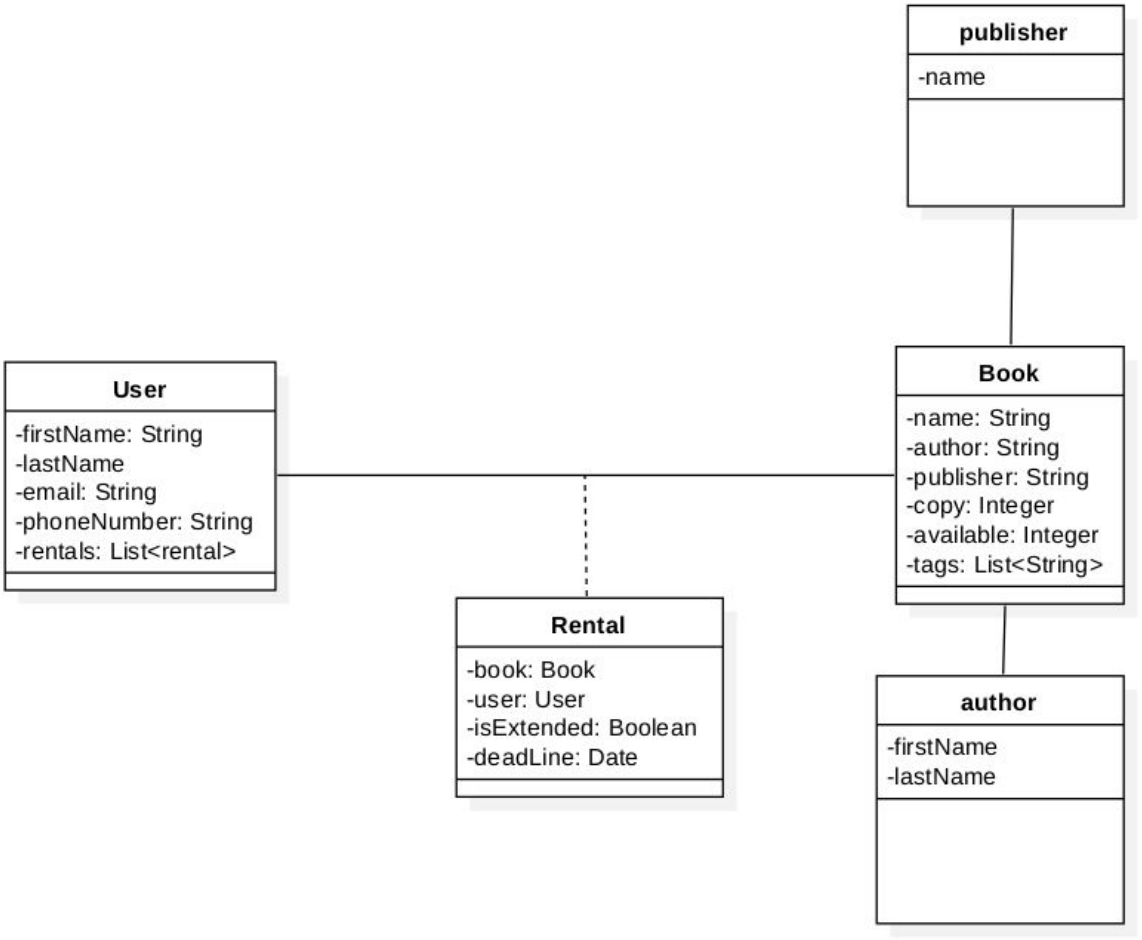
- library_webapp_client: définit les interfaces pour accéder au web-service.
- library_webapp_client_soap: implémente les interfaces, en l'occurence ici pour faire appel à un service soap.
- library_webapp_model_soap: les modèles tels que définis dans le WSDL.
- library_webapp_service: des services permettant de manipuler les modèles reçus
- library_webapp_controller: les classes struts2 permettant de renvoyer les vus à l'utilisateur.

Batch:

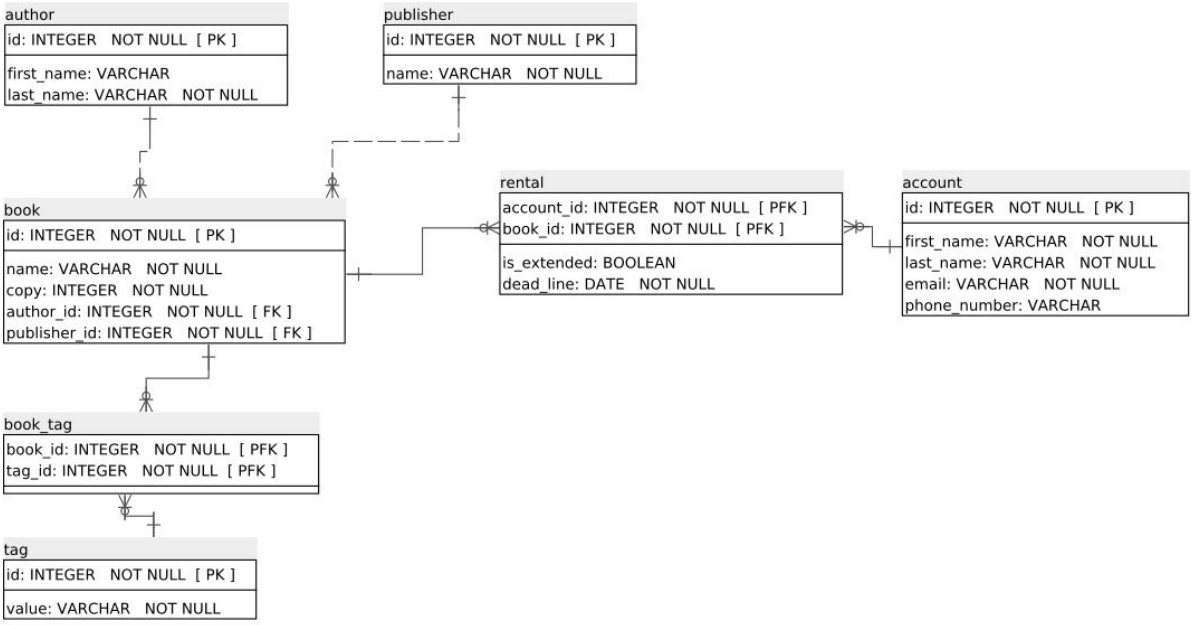
Une application java SpringBoot pour envoyer des mails de rappel aux usagers n’ayant pas ramener les livres à temps.
On retrouve quatre package dans src/main/java:

- io.ab.library.batch: class Main et class de configuration
- io.ab.library.batch.client: les classes appelant le WebService
- io.ab.library.batch.wsdl: les modèles définis dans le WSDL
- io.ab.library.batch.service: une classe métiers envoyant les mails.

UML


publisher
-name
User
-firstName: String
-lastName
-email: String
-phoneNumber: String
-rentals: List<rental>
Book
-name: String
-author: String
-publisher: String
-copy: Integer
-available: Integer
-tags: List<String>
Rental
-book: Book
-user: User
-isExtended: Boolean
-deadLine: Date
author
-firstName
-lastName

MPD


author
id: INTEGER NOT NULL [PK]
first_name: VARCHAR
last_name: VARCHAR NOT NULL
publisher
id: INTEGER NOT NULL [PK]
name: VARCHAR NOT NULL
book
id: INTEGER NOT NULL [PK]
name: VARCHAR NOT NULL
copy: INTEGER NOT NULL
author_id: INTEGER NOT NULL [FK]
publisher_id: INTEGER NOT NULL [FK]
rental
account_id: INTEGER NOT NULL [PFK]
book_id: INTEGER NOT NULL [PFK]
is_extended: BOOLEAN
dead_line: DATE NOT NULL
account
id: INTEGER NOT NULL [PK]
first_name: VARCHAR NOT NULL
last_name: VARCHAR NOT NULL
email: VARCHAR NOT NULL
phone_number: VARCHAR
book_tag
book_id: INTEGER NOT NULL [PFK]
tag_id: INTEGER NOT NULL [PFK]
tag
id: INTEGER NOT NULL [PK]
value: VARCHAR NOT NULL