KyLibrary – P10 Openclassrooms

version 2.0-RELEASE

GitHub : https://github.com/Kybox/KyLibrary2

Liste des interventions :

Refactoring du code

Modification du système d'authentification :

Ajout d'un système de gestion d'authentification par token auto-généré (Universal Unique Identifier) lors du login utilisateur, et stocké dans l'objet session (classe SessionAware de Struts2).

Le token permet au web-service d'obtenir l'identité de l'utilisateur effectuant une requête auprès du web-service, et d'adapter la réponse en fonction du niveau d'autorisation accordé.

Dès lors que la session côté client (via Struts2) n'est plus valide, le token est automatiquement détruit par le web-service. De même, si la session est encore active coté client, mais que le token à expiré, ce dernier est automatiquement détruit et un code HTTP 498 (Token expired/invalid) est retourné en tant que réponse à la requête, ce qui supprime automatique la session côté client.

Cette modification a nécessité l'ajout d'une relation « token_storage » en base de données (voir le modèle physique de données en dernière page du document)

Ajout d'un système de réservation d'ouvrage

Aperçu de la méthode du web-service permettant la réservation d'un ouvrage :

```java
@Override
@WebMethod
public int reserveBook(String token, String isbn) {

    Map<String, Object> tokenData = userService.checkTokenData(token);
    if(tokenData.get(TOKEN_ACTIVE) == Boolean.FALSE)
        return TOKEN_EXPIRED_INVALID;

    Optional<BookEntity> optBookEntity = bookService.findBookByIsbn(isbn);
    if(!optBookEntity.isPresent())
        return BAD_REQUEST;

    BookEntity book = optBookEntity.get();
    UserEntity user = (UserEntity) tokenData.get(USER_FROM_TOKEN);
    String bookIsbn = book.getIsbn();

    if(book.getAvailable() > ZERO || book.getBookable() == Boolean.FALSE)
        return FORBIDDEN;

    List<ReservedBook> reservedBookList = bookService.findAllReservedBooksByUser(user);
    for(ReservedBook reservation : reservedBookList) {
        if (bookIsbn.equals(reservation.getBook().getIsbn()) && reservation.isPending())
            return FORBIDDEN;
    }

    List<BorrowedBook> borrowedBookList = bookService.findAllBorrowedBooksUnreturnedByUser(user);
    for(BorrowedBook loan : borrowedBookList){
        if(bookIsbn.equals(loan.getBook().getIsbn()))
            return FORBIDDEN;
    }

    ReservedBook reservedBook = new ReservedBook();
    reservedBook.setUser(user);
    reservedBook.setBook(book);
    reservedBook.setReserveDate(LocalDateTime.now());
    reservedBook.setPending(true);

    bookService.saveReservedBook(reservedBook);

    checkReservationStatus(reservedBook.getBook());

    return OK;
}
```

Ensemble des demandes relatives au ticket #1 réalisées :

- La possibilité pour un usager (authentifié) de réserver n'importe quel ouvrage (à la condition que le nombre d'exemplaires disponibles est égal à zéro).

Condition :

```
if(book.getAvailable() > ZERO || book.getBookable() == Boolean.FALSE)
    return FORBIDDEN;
```

- La liste des réservations ne peut comporter qu'un maximum d'usager correspondant à 2x le nombre total d'exemplaires enregistrés en bibliothèque (multiplicateur définit dans le fichier de propriétés du web-service).

```
@Value("${settings.multiplierReservation}")
private int multiplierReservation;
```

```
private void checkReservationStatus(BookEntity book){

    int nbCopies = book.getNbCopies();
    int maxReservedBook = nbCopies * multiplierReservation;

    List<ReservedBook> reservedBookList = bookService
            .findAllReservedBooksByBookAndPendingTrueOrderByReserveDateAsc(book);

    if(reservedBookList.size() >= maxReservedBook) {
        book.setBookable(false);
    }
    else {
        book.setBookable(true);
        book.setReturnDate(getNextReturnDate(book));
    }

    bookService.saveBook(book);
}
```

- Il est impossible pour un usager de réserver un ouvrage qu'il est déjà en train d'emprunter.

```
List<BorrowedBook> borrowedBookList = bookService.findAllBorrowedBooksUnreturnedByUser(user);
for(BorrowedBook loan : borrowedBookList){
    if(bookIsbn.equals(loan.getBook().getIsbn())){
        return FORBIDDEN;
    }
}
```

Afin de garder une certaine cohérence dans la gestion des ouvrages, une autre condition a été ajoutée. Elle interdit l'addition de réservations d'un même ouvrage.

```
List<ReservedBook> reservedBookList = bookService.findAllReservedBooksByUser(user);
for(ReservedBook reservation : reservedBookList) {
    if (bookIsbn.equals(reservation.getBook().getIsbn()) && reservation.isPending()) {
        return FORBIDDEN;
    }
}
```

- Passé le délai de 48h (délai définit dans le fichier de propriété du batch), après notification d'une réservation disponible, cette dernière est supprimée. Code du batch :

```
for(int index = 0; index < total; index++){

    int position = index + 1;
    BookReserved bookReserved = bookList.get(index);
    for(int i = 0; i <= index; i++) emptyKey += emptyKey;

    BatchResult.put(emptyKey, RESERVATION_TITLE + position);

    LocalDateTime reservationDate = bookReserved.getReserveDate();
    DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd / MM / yyyy - HH:mm:ss");
    BatchResult.put(position + HYPHEN + RESERVATION_DATE, ON + reservationDate.format(formatter));

    String userFirstName = userList.get(index).getFirstName();
    String userLastName = userList.get(index).getLastName();
    String userEmail = userList.get(index).getEmail();

    String bookIsbn = bookReserved.getBook().getIsbn();
    String bookTitle = bookReserved.getBook().getTitle();
    String bookAuthor = bookReserved.getBook().getAuthor();

    BatchResult.put(position + HYPHEN + USER, userFirstName + SPACE + userLastName);
    BatchResult.put(position + HYPHEN + BOOK, bookTitle + BY + bookAuthor);

    if(bookReserved.isNotified()){

        LocalDateTime notificationDate = bookReserved.getNotificationDate();

        String stringDate = bookReserved.getNotificationDate().format(formatter);
        BatchResult.put(position + HYPHEN + USER_ALREADY_NOTIFIED, ON + stringDate);

        LocalDateTime expirationDate = notificationDate.plusHours(reservationExpire);

        if(expirationDate.isAfter(LocalDateTime.now())){

            long remainingTime = ChronoUnit.HOURS.between(LocalDateTime.now(), expirationDate);

            if(remainingTime == 0) {
                remainingTime = ChronoUnit.MINUTES.between(LocalDateTime.now(), expirationDate);
                BatchResult.put(position + HYPHEN + REMAINING_TIME_BEFORE_DELETION,
                        APPROXIMATELY + remainingTime + MINUTES);
            }
            else
                BatchResult.put(position + HYPHEN + REMAINING_TIME_BEFORE_DELETION,
                        APPROXIMATELY + remainingTime + HOURS);

            continue;
        }

        BatchResult.put(position + HYPHEN + CANCEL_RESERVATION, ATTEMPT_TO_CANCEL_RESERVATION);
        CancelReservation request = objectFactory.createCancelReservation();
        request.setIsbn(bookIsbn);
        request.setEmail(userEmail);
        request.setToken(token);

        CancelReservationResponse responseData = libraryService.cancelReservation(request);

        if(responseData.getResult() != HTTP_CODE_OK){
            BatchResult.put(position + HYPHEN + CANCEL_RESERVATION, ERROR + CANCEL_RESERVATION_ERROR);
            continue;
        }

        BookReserved nextBookReserved = responseData.getBookReserved();
        User nextUserReservation = responseData.getUser();

        if(nextBookReserved == null || nextUserReservation == null) {
            BatchResult.put(position + HYPHEN + NEXT_RESERVATION, NO_OTHER_RESERVATION);
            continue;
        }

        BatchResult.put(position + HYPHEN + NEXT_RESERVATION, ANOTHER_RESERVATION_EXISTS);
        BatchResult.put(position + HYPHEN + ADDING_A_RESERVATION, ADDING_THE_NEW_RESERVATION);

        bookList.add(nextBookReserved);
        userList.add(nextUserReservation);

        total ++;

        continue;
    }
```

- Dès lors, pour le web-service, deux fonctionnalités entrent en jeu :

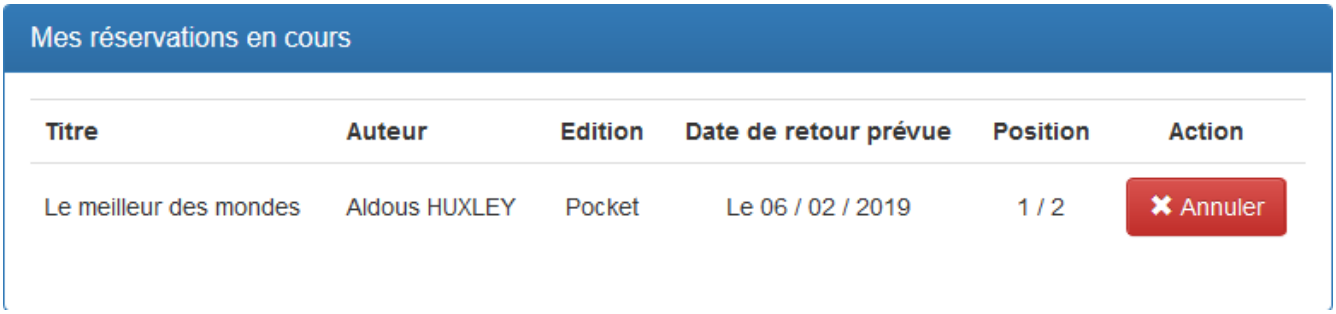
- Soit aucune autre réservation n'est enregistrée et donc l'ouvrage est de nouveau disponible à un emprunt en bibliothèque (option de réservation bloquée + incrémentation du nombre d'exemplaires disponibles).

- Soit une autre réservation de l'ouvrage est enregistrée, auquel cas elle prend alors la place de la précédente.

```
bookService.deleteReservedBook(optReservedBook.get());

if(!bookService.areThereAnyReservationForBook(book)){

    if(!book.getBookable() || book.getAvailableForBooking() == ZERO){
        response.setResult(INTERNAL_SERVER_ERROR);
        return response;
    }

    book.setAvailableForBooking(book.getAvailableForBooking() - ONE);
    book.setAvailable(book.getAvailable() + ONE);
    book.setBookable(false);
    book.setReturnDate(null);
    book.setNbReservations(book.getNbReservations() - ONE);

    bookService.saveBook(book);

    response.setResult(OK);
    return response;
}

// Define the nearest return date
List<ReservedBook> reservedBookList = bookService
        .findAllReservedBooksByBookAndPendingTrueOrderByReserveDateAsc(book);

LocalDate nearestReturnDate = LocalDate.now();
nearestReturnDate = nearestReturnDate.plusWeeks((defaultLendingPeriod * reservedBookList.size()));
book.setReturnDate(Date.valueOf(String.valueOf(nearestReturnDate)));

// Remove the last cancelled reservation
book.setNbReservations(book.getNbReservations() - ONE);

// Save
bookService.saveBook(book);

// Part reserved for the batch
user = (UserEntity) tokenData.get(USER_FROM_TOKEN);

if(user.getLevel().getId() <= MANAGER){

    // Create the batch response
    ReservedBook reservedBook = reservedBookList.get(FIRST_ONE_LIST);

    BookReserved bookReserved = (BookReserved) Reflection.EntityToWS(reservedBook);
    bookReserved.setBook((Book) Reflection.EntityToWS(reservedBook.getBook()));

    response.setBookReserved(bookReserved);
    response.setUser((User) Reflection.EntityToWS(reservedBook.getUser()));
    response.setResult(OK);

    return response;
}

response.setResult(OK);
return response;
```

- Lors d'une recherche d'ouvrage, pour ceux indisponibles, la date de retour prévue la plus proche et le nombre de personnes ayant réservé l'ouvrage doivent être mentionnés.

- L'usager doit pouvoir avoir une liste des réservations qu'il a en cours avec pour chaque ouvrage la prochaine date de retour prévue et sa position dans la liste d'attente ainsi que la possibilité d'annuler sa réservation.



Fin des différentes fonctionnalités relative au ticket #1

Correction du bug relatif au ticket #2

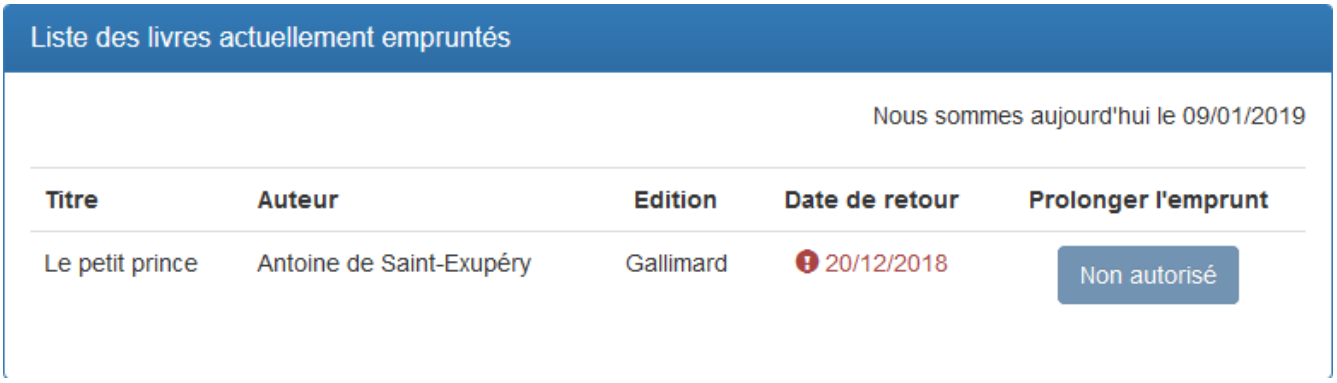
Rappel :

« Nous avons découvert un bug de la gestion des prolongations de prêt. En effet, un usager peut prolonger un prêt après la date butoir. Il ne doit pas être possible pour l'usager de prolonger un prêt si la date de fin de prêt est dépassée. »

Code web-service méthode extendBorring :

```
LocalDate returnDate = borrowedBook.getReturnDate().toLocalDate();

if(returnDate.isAfter(LocalDate.now())){
    response.setResult(FORBIDDEN);
    return response;
}
```

Coté client, une vérification est également apportée afin de bloquer la requête :



Bug relatif au ticket #2 solutionné

Ensemble des demandes relatives au ticket #3 :

Rappel :

Nous aimerions, en parallèle de la correction de la gestion des prolongations des prêts (cf ticket #2), ajouter un mécanisme de rappel aux usagers des prêts arrivant bientôt à expiration. Les usagers qui le souhaitent pourront activer une option (dans leur profil sur l'application web), qui leur enverra un mail de rappel 5 jours calendaires avant la fin de leur prêt.

Code du web-service :

```java
@Override
@WebMethod
public int updateAlertSenderStatus(String token, boolean status) {

    Map<String, Object> tokenData = userService.checkTokenData(token);
    if(tokenData.get(TOKEN_ACTIVE) == Boolean.FALSE){
        return TOKEN_EXPIRED_INVALID;
    }

    UserEntity user = (UserEntity) tokenData.get(USER_FROM_TOKEN);
    user.setAlertSender(status);
    userService.saveUser(user);
    return OK;
}
```

Rappel :

Le mail contiendra la liste de leurs prêts en cours qui arrivent à expiration dans 5 jours ou moins, avec la date d'expiration associée à chaque prêt.

Comme affiché, une indication à destination de l'usager l'informe qu'il peut prolonger son emprunt. Si l'emprunt en question a déjà été prolongé, l'information n'est pas indiquée.

Fin de la fonctionnalité relative au ticket #3

Modèle physique de données mis à jour