POLYTECH TOURS
64 avenue Jean Portalis
37200 TOURS, FRANCE

Rapport de TP

Online Book Borrowing System

2025-2026

Professeur encadrant : Shokouh Ghulam Sakhi
Équipe : Aithamadi Outman

Table des matières

1 Description de l'architecture MVC et des composants clés

L'application web est conçue selon le patron de conception **MVC (Modèle-Vue-Contrôleur)**, garantissant une séparation claire entre la logique métier, la présentation et le contrôle des interactions utilisateur.

1.1 Le Modèle (JavaBeans)

Le modèle regroupe les classes responsables de la gestion des données et de la logique métier, indépendamment de l'interface utilisateur. Ces classes sont placées dans le package `fr.univtours.polytech.tpeval.model`.

1.1.1 Book.java

La classe **Book** représente une entité métier fondamentale. Elle encapsule les informations d'un livre (`ISBN`, `title`, `author`, `availableCopies`, `description`, `format`). Elle propose également des méthodes de gestion du stock (`increaseCopies()`, `decreaseCopies()`). Cette classe implémente `Serializable` afin de permettre la persistance des objets en session utilisateur.

1.1.2 BorrowedList.java

Cette classe gère la collection de livres empruntés par un utilisateur donné (stockée dans la session). Elle fournit la méthode métier `calculateTotalCost()` qui applique la règle de tarification suivante :

— Livre physique : **10 € par mois**
— Livre en ligne : **5 € par mois**

Elle offre également des opérations de manipulation de la liste d'emprunts : `addBook()`, `returnBook()`, `clear()`.

1.1.3 Library.java

La classe **Library** joue le rôle de catalogue global partagé entre tous les utilisateurs. Elle agit comme une *Data Access Layer* simulée, en utilisant un `ConcurrentHashMap<String, Book>` pour stocker les livres accessibles via leur ISBN. Le choix d'un **ConcurrentHashMap** au lieu d'un simple `HashMap` ou `Hashtable` s'explique par :

— La nécessité d'assurer une **sécurité des accès concurrents** dans un environnement multi-utilisateur (plusieurs requêtes simultanées).
— Une meilleure **performance** que `Hashtable`, car les verrous sont segmentés et non globaux.
— Une garantie de **visibilité mémoire cohérente** sans avoir à synchroniser explicitement les blocs de code.

Le catalogue est initialisé via un **bloc statique**, simulant un ensemble de livres disponibles dès le démarrage de l'application.

1.2 Le Contrôleur(Servlets)

Les servlets constituent la couche **Contrôleur** de l'architecture MVC. Elles servent de médiateur entre les vues (JSP) et le modèle (JavaBeans). Chaque servlet gère une partie spécifique du flux de navigation et de la logique métier de l'application.

1.2.1 BorrowServlet : gestion des emprunts et retours

La classe **BorrowServlet** est le cœur de la logique métier utilisateur. Elle est mappée sur les routes `/reserve`, `/return` et `/borrowed`. Son rôle est de gérer les trois actions principales suivantes :

- **Afficher la liste des livres empruntés** via `doGet()`.
- **Emprunter un livre** via `POST` sur `/reserve`.
- **Retourner un livre** via `POST` sur `/return`.

Structure et logique interne :

- À chaque requête, la servlet récupère la session utilisateur et initialise la liste d'emprunt `BorrowedList` si elle n'existe pas encore.
- L'action à exécuter est identifiée grâce à `getServletPath()`.
- Lorsqu'un utilisateur réserve un livre, la servlet :
  1. Vérifie que des exemplaires sont disponibles.
  2. Diminue le stock global dans `Library`.
  3. Crée une copie du `Book` pour l'ajouter à la `BorrowedList`.
- Lorsqu'un utilisateur retourne un livre :
  1. Le livre est supprimé de la `BorrowedList`.
  2. Le stock global de la bibliothèque est incrémenté.

Gestion des erreurs et cohérence : La servlet envoie un message d'erreur à la vue si :

- Le livre recherché n'existe pas (`404 Livre non trouvé`).
- Aucun exemplaire n'est disponible pour l'emprunt.

Cette approche garantit la cohérence entre le stock global et les listes d'emprunts de chaque utilisateur, même en environnement multi-utilisateur.

1.2.2 CatalogServlet : affichage du catalogue global

La classe **CatalogServlet**, mappée sur `/catalog`, assure l'affichage de l'ensemble des livres disponibles.

Fonctionnement :

- Lorsqu'une requête GET est reçue, elle interroge le modèle `Library` pour récupérer les livres disponibles via la méthode statique `getAvailableBooks()`.
- La liste est ensuite placée dans l'attribut de requête `catalog`.
- Enfin, la requête est transférée vers la page `catalog.jsp` pour affichage dynamique à l'aide de `JSTL`.

Exemple de traitement :

```
request.setAttribute("catalog", Library.getAvailableBooks());
request.getRequestDispatcher("/catalog.jsp").forward(request,
response);
```

1.2.3 CheckoutServlet : validation et confirmation des emprunts

La classe **CheckoutServlet**, mappée sur `/checkout`, gère la dernière étape du processus — la validation de la commande.

Règles métier implémentées :

- Chaque utilisateur ne peut pas emprunter plus de **2 livres** simultanément.
- Le coût total mensuel est calculé selon le format du livre :
  - Livre physique : **10 €**
  - Livre numérique : **5 €**

Comportement des méthodes :

- `doGet()` :
  1. Vérifie la présence d'une session et d'une liste d'emprunts valide.
  2. Applique la règle métier de limite d'emprunt.
  3. Calcule et transmet le coût total à la vue `checkout.jsp`.
- `doPost()` :
  1. Supprime la `BorrowedList` de la session.
  2. Affiche un message de confirmation.
  3. Redirige l'utilisateur vers `index.jsp`.

Principe de nettoyage de session : Cette étape finale assure la libération de la mémoire de session après validation, évitant ainsi toute persistance non souhaitée entre les sessions utilisateurs successives.

Annotation Servlet :

```
@WebServlet("/checkout")
```

1.3 La Vue (JSP & JSTL)

Les pages JSP constituent la **couche de présentation** du modèle MVC. Elles sont responsables de l'affichage des données issues du modèle et transmises par les servlets, en garantissant une séparation nette entre la logique métier et la logique d'affichage.

Chaque page JSP correspond à une étape du parcours utilisateur et communique avec une servlet spécifique.

1.3.1 catalog.jsp — Affichage du catalogue global

Cette page est associée à la **CatalogServlet** et affiche la liste complète des livres disponibles.

Points clés :

- Récupère la liste `catalog` depuis les attributs de requête définis dans la servlet.
- Parcourt les livres avec `<c:forEach>` pour en afficher le titre, l'auteur, le format et le nombre de copies disponibles.
- Utilise la balise `<c:choose>` pour afficher soit un bouton *"Emprunter"* (si des copies sont disponibles), soit un bouton *"Indisponible"* désactivé.
- Intègre un lien direct vers la page `/borrowed` afin que l'utilisateur puisse consulter ses emprunts.

Extrait représentatif :

```
<c:forEach var="book" items="${catalog}">
    <td>${book.title}</td>
    <td>${book.author}</td>
    <td>${book.format}</td>
    <td>${book.availableCopies}</td>
```

1.3.2 borrowed.jsp — Liste des livres empruntés

Cette page correspond à la **BorrowServlet**. Elle affiche les livres empruntés par l'utilisateur et permet de retourner un livre via un formulaire POST vers `/return`.

Points clés :

- Récupère la liste `borrowedList` depuis la session utilisateur (`sessionScope.borrowedList`).
- Vérifie la présence de livres avec :

```
<c:if test="${empty borrowedList or empty borrowedList.borrowedBooks}">
```

- Calcule dynamiquement le coût d'un livre selon son format :

```
<c:set var="cost" value="${book.format eq 'physical' ? 10 : 5}" />
```

- Permet le retour de chaque livre via un formulaire :

```
<form action="<c:url value='/return' />" method="post">
    <input type="hidden" name="isbn" value="${book.isbn}">
    <button type="submit">Retourner</button>
</form>
```

1.3.3 checkout.jsp — Validation et récapitulatif

Cette page est liée à la **CheckoutServlet** et représente la dernière étape du parcours utilisateur.

Points clés :

- Affiche un tableau récapitulatif des livres empruntés et de leur coût.
- Calcule le **total mensuel** reçu depuis l'attribut `totalCost` défini dans la servlet.
- Utilise la balise `<fmt:formatNumber>` pour afficher les prix avec deux décimales.
- Gère les messages d'erreur et de confirmation :
  - Message d'erreur si l'utilisateur dépasse la limite d'emprunts.
  - Message de validation après finalisation.

Extrait représentatif :

```
<tfoot>
    <tr>
        <td colspan="2"><strong>Total Mensuel:</strong></td>
        <td><strong>€<fmt:formatNumber value="${requestScope.totalCost}" /></strong><
    </tr>
</tfoot>
```

1.3.4 index.jsp — Page d'accueil

La page d'accueil agit comme point d'entrée de l'application. Elle propose des liens de navigation vers :

- le catalogue des livres ;
- la liste des livres empruntés ;
- la page de paiement (`checkout`).

e parfaite séparation entre logique de contrôle et affichage.
float caption

2 Conception et modélisation UML

Cette section présente la conception de l'application à travers plusieurs diagrammes UML illustrant les principales interactions entre les composants du système.

2.1 Diagramme de cas d'utilisation

Le diagramme de cas d'utilisation met en évidence les interactions entre l'utilisateur et le système. Il montre les fonctionnalités principales offertes par l'application, notamment :

- Consulter le catalogue des livres.
- Ajouter un livre au panier d'emprunt.
- Valider l'emprunt.
- Consulter la liste des livres empruntés.
- Retourner un livre.

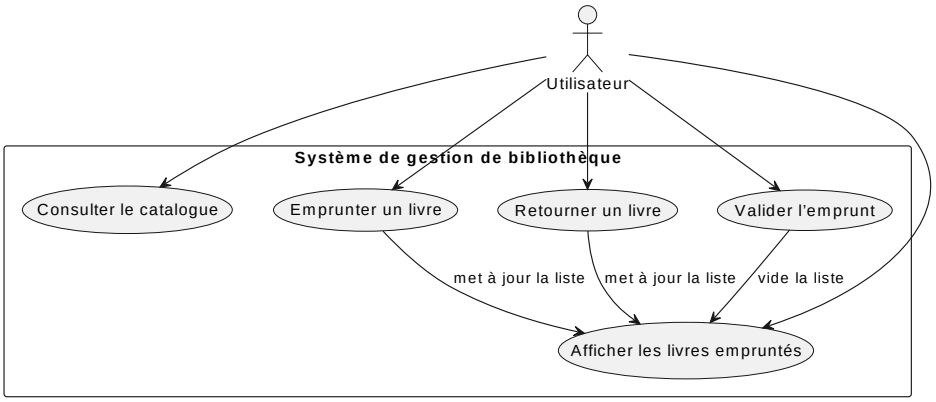

FIGURE 1 – Diagramme de cas d'utilisation principal de l'application

2.2 Diagramme de classes

Le diagramme de classes représente la structure interne du projet et les relations entre les classes principales. On y retrouve notamment les entités `Book`, `BorrowedList`, `Library`, ainsi que les contrôleurs correspondants.

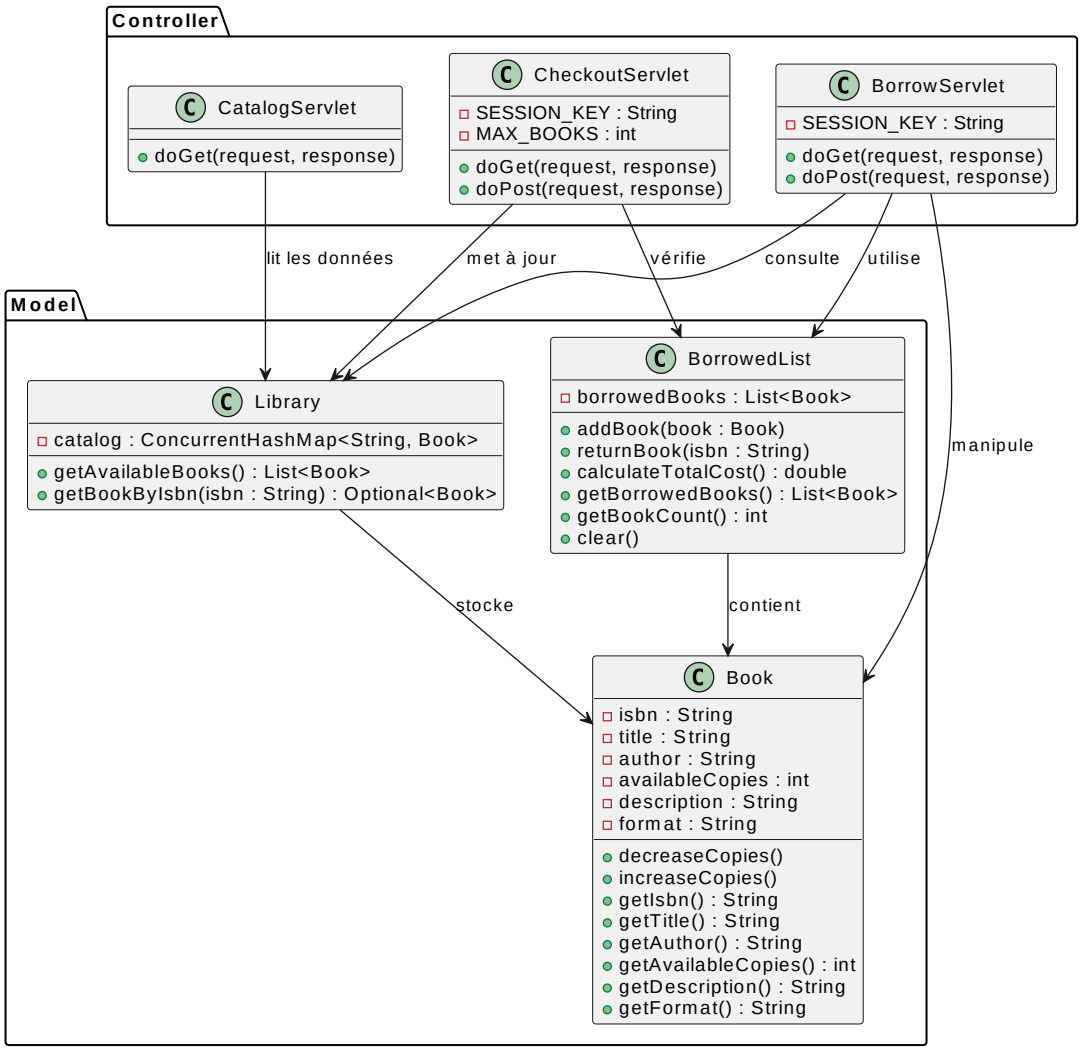

FIGURE 2 – Diagramme de classes du projet

2.3 Diagramme de séquence

Ce diagramme illustre le déroulement typique d'un emprunt de livre. Il montre les échanges entre l'utilisateur, le contrôleur, le modèle (`Library`, `BorrowedList`) et la vue JSP correspondante.

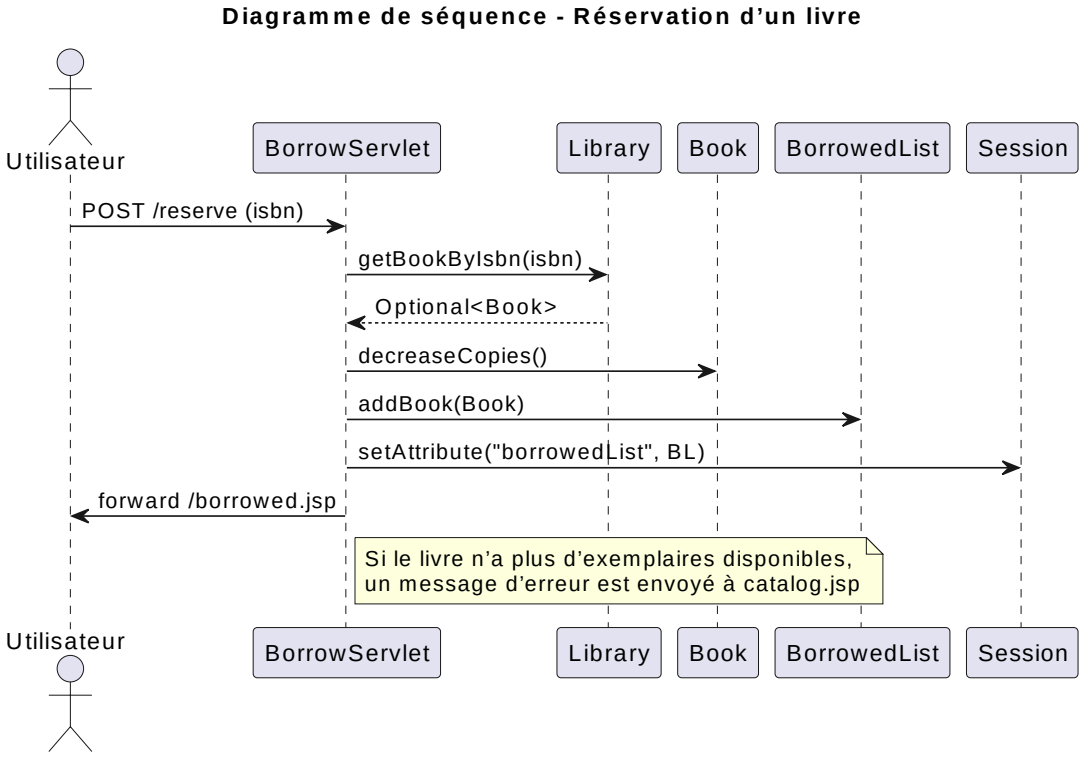

FIGURE 3 – Diagramme de séquence : scénario d'emprunt d'un livre

3 Captures d'écran des exercices

3.1 Exercice 1 : Catalogue des livres (`/catalog`)

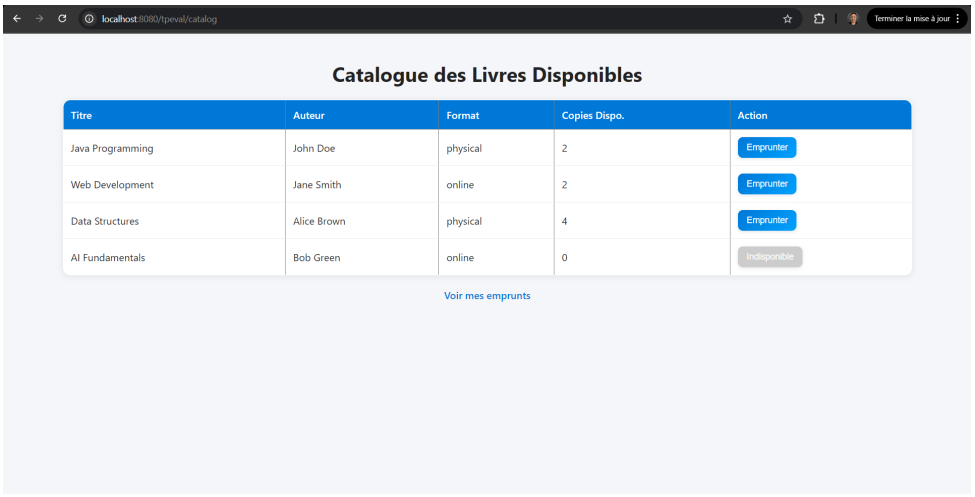

FIGURE 4 – Page `/catalog` affichant la liste des livres disponibles et les boutons "Emprunter"

3.2 Exercice 2 : Panier d'emprunt (`/reserve`)

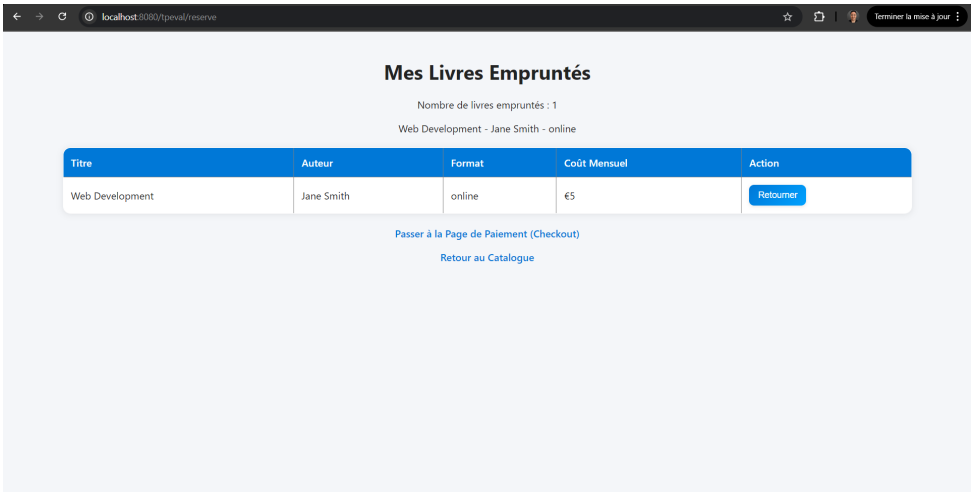

FIGURE 5 – Page `/reserve` affichant le panier d'emprunt avec les livres sélectionnés

3.3 Exercice 3 : Validation - Première étape (`/checkout`)

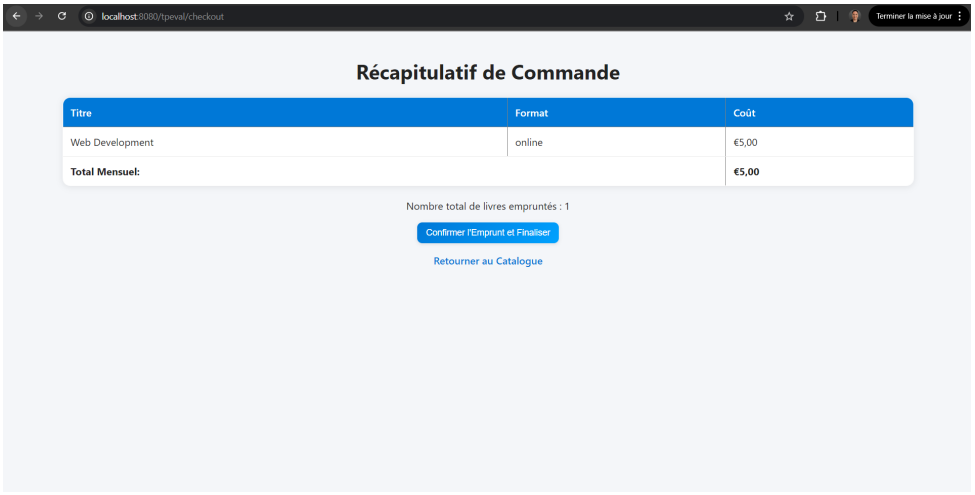

FIGURE 6 – Page `/checkout` affichant la liste récapitulative des livres à emprunter

3.4 Exercice 4 : Validation - Message d'erreur (`/checkout`)

FIGURE 7 – Message d'erreur affiché lorsqu'un utilisateur tente d'emprunter plus de deux livres

3.5 Exercice 5 : Validation - Confirmation de l'emprunt (`/checkout`)

FIGURE 8 – Page `/checkout` affichant la confirmation de l'emprunt après validation

3.6 Liste des livres empruntés (`/borrowed`)

FIGURE 9 – Page `/borrowed` affichant la liste des livres actuellement empruntés et les options de retour

3.7 Vidéo de démonstration

La démonstration complète du projet est disponible en vidéo au lien suivant :

`https://youtu.be/fUhsNUTXJlw`

Le code source complet du projet est disponible sur GitHub :

`https://github.com/aithamadioutman/Tp_Book_Reservation`

3.7.1 Comment exécuter le projet

Pour utiliser l'application sur votre machine locale, suivez ces étapes :

1. Ouvrez le projet dans VS Code ou Eclipse.
2. Exécutez la commande suivante pour construire le fichier WAR :

```
mvn clean package
```

3. Déployez le fichier `tpeval.war` dans le dossier `webapps` de Tomcat 10.
4. Démarrez le serveur Tomcat.
5. Accédez à l'application via l'URL suivante :

```
http://localhost:8080/tpeval
```