ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR
Membre de HONORIS UNITED UNIVERSITIES

Rapport Bibliotheca

Matière : Développement Web

- Application Web de Gestion de Bibliothèque -

Réalisé par :
Kawtar BRIJA
Malak BERNAOUI
Yasmine BOUJNAH

Encadré par :
Mme Khadija TLEMÇANI

Date :
16 Janvier 2026

Année académique : 2025/2026

Contents

Remerciements 3

Résumé 4

1 Introduction générale 5

2 Analyse du cahier de charge 5

3 Technologies utilisées 5

3.1 Langages utilisés . . . 5

3.2 Librairies et outils . . . 6

3.3 Outils de développement . . . 6

4 Développement de l'application 6

4.1 Structure générale de l'application . . . 6

4.1.1 Organisation des fichiers . . . 6

4.1.2 Gestion de l'affichage des sections (SPA) . . . 6

4.2 Module 1 : Gestion des Livres (CRUD complet) . . . 6

4.2.1 Ajout et modification d'un livre . . . 6

4.2.2 Affichage de la liste . . . 7

4.2.3 Recherche et tri . . . 7

4.2.4 Suppression avec confirmation . . . 7

4.2.5 Sauvegarde dans le LocalStorage . . . 7

4.3 Module 2 : Gestion des Auteurs (CRUD simplifié) . . . 7

4.3.1 Ajout d'un auteur . . . 7

4.3.2 Affichage de la liste . . . 7

4.3.3 Suppression . . . 7

4.3.4 Lien entre auteurs et livres . . . 8

4.4 Gestion de l'authentification et des sessions . . . 8

4.4.1 Système d'inscription et de connexion . . . 8

4.4.2 Affichage utilisateur et horloge . . . 8

4.5 Module 3 : Dashboard & Intégration d'API . . . 8

4.5.1 Mise à jour des indicateurs KPI . . . 8

4.5.2 Visualisation avec Chart.js . . . 8

4.5.3 Intégration asynchrone avec une API publique . . . 9

5 Dashboard et visualisation des données 9

5.1 KPI de la bibliothèque . . . 9

5.1.1 Nombre total de livres . . . 9

5.1.2 Nombre d'auteurs . . . 9

5.1.3 Statut de l'API . . . 9

5.1.4 Dernier ajout . . . 9

5.2 Graphiques et statistiques . . . 10

5.2.1 Présentation des graphiques Chart.js . . . 10

5.2.2 Analyse des résultats affichés . . . 10

6 Intégration de l'API externe (Asynchrone) 10

6.1 Présentation de l'API OpenLibrary . . . 10

6.2 Utilisation de fetch() . . . 11

6.3 Traitement des données JSON . . . 11

6.4 Exploitation des données dans le dashboard . . . 11

6.5 Gestion des erreurs (try / catch) . . . 11

7 Tests et validation **12**

8 Difficultés rencontrées et solutions apportées **12**

9 Résultats et captures d'écran **12**

9.1 Présentation des interfaces finales 12

10 Conclusion **17**

REMERCIEMENTS

Nous tenons à exprimer notre profonde gratitude envers toutes les personnes qui ont contribué de près ou de loin à la réalisation de ce projet.

En premier lieu, nous adressons nos sincères remerciements à notre encadrante, Mme Khadija TLEMÇANI, pour son accompagnement précieux, ses conseils éclairés et sa disponibilité tout au long de ce projet. Son expertise et son soutien constant ont été déterminants dans la réussite de ce travail.

Nous remercions également l'École Marocaine des Sciences de l'Ingénieur (EMSI) pour l'excellente formation dispensée et pour les moyens pédagogiques mis à notre disposition. Les connaissances acquises durant notre parcours académique nous ont permis de mener à bien ce projet de développement web.

Nos remerciements vont aussi à l'ensemble de l'équipe pédagogique de la filière 3IIR pour leur enseignement de qualité et leur engagement à nous transmettre les compétences nécessaires à notre future carrière d'ingénieurs.

Enfin, nous nous remercions mutuellement, Kawtar BRIJA, Malak BERNAOUI et Yasmine BOUJNAH, pour le travail d'équipe, l'entraide et la bonne coordination qui ont caractérisé notre collaboration. Chacune a apporté ses compétences complémentaires, permettant ainsi l'aboutissement de ce projet dans les meilleures conditions.

Ce projet a été une expérience enrichissante tant sur le plan technique que sur le plan humain, et nous sommes fières du résultat obtenu grâce à la contribution de chacun.

Résumé

Le projet **Bibliotheca** est une application Web de type **Backoffice Dashboard** dédiée à la gestion d'une bibliothèque. Elle permet l'administration des **livres** et des **auteurs** à travers une interface structurée, moderne et intuitive, inspirée des systèmes de gestion professionnels.

L'application est conçue selon une architecture **Single Page Application (SPA)**, où l'ensemble des sections est géré dynamiquement à l'aide de **JavaScript Vanilla**. Elle met en œuvre la manipulation avancée du **DOM**, la gestion des événements utilisateurs et l'implémentation des fonctionnalités **CRUD** (ajout, affichage, modification et suppression des données). La persistance des informations est assurée grâce à l'utilisation du **LocalStorage**, garantissant la conservation des données après rechargement de la page.

Le module principal est consacré à la gestion complète des **livres**, incluant la recherche, le tri, l'affichage détaillé et la suppression avec confirmation. Un module secondaire permet la gestion simplifiée des **auteurs**, avec un lien logique entre les auteurs et les livres.

L'application intègre également un **Dashboard analytique** présentant des **indicateurs clés de performance (KPI)** tels que le nombre total de livres et d'auteurs, ainsi que des **graphiques statistiques** générés dynamiquement. Une partie asynchrone est implémentée via l'exploitation de l'API **OpenLibrary**, permettant d'enrichir les données locales et d'illustrer l'utilisation de la fonction fetch() et des promesses.

Ce projet constitue une solution complète de gestion de bibliothèque, combinant fonctionnalités métier, visualisation des données et bonnes pratiques du développement Web moderne.

Introduction générale

Ce projet a été réalisé dans le cadre du module **Développement Web** de la filière **3IIR** durant l'année universitaire **2025-2026**. Il consiste à développer une application **Front-End** de type **Backoffice Dashboard**, inspirée des interfaces professionnelles utilisées en entreprise.

L'application est conçue selon une architecture **Single Page Application (SPA)** et développée exclusivement avec **HTML5, CSS3 et JavaScript** . Elle permet la manipulation dynamique du DOM, la gestion des événements, la persistance des données via le **LocalStorage** et l'utilisation d'une **API externe**.

Le sujet choisi, **Bibliotheca**, concerne la gestion d'une bibliothèque à travers l'administration des **livres** et des **auteurs**. L'application intègre des fonctionnalités **CRUD**, un **Dashboard** avec des **KPI** et des graphiques statistiques, ainsi que l'exploitation de l'API **OpenLibrary**, offrant ainsi une solution simple, structurée et professionnelle de gestion de bibliothèque.

Analyse du cahier de charge

Le cahier de charge exige la réalisation d'une application Web de type **Backoffice Dashboard** permettant la gestion dynamique des données à travers des fonctionnalités **CRUD**, ainsi que l'affichage de statistiques et d'indicateurs clés de performance.

L'application doit respecter des contraintes techniques précises, notamment l'utilisation exclusive de **HTML5, CSS3 et JavaScript Vanilla**, une architecture **Single Page Application (SPA)**, la persistance des données via le **LocalStorage** et l'intégration d'une **API externe** à l'aide de la fonction fetch().

Dans ce cadre, le choix du sujet **Bibliotheca** s'avère pertinent, car il permet de mettre en œuvre un module principal de gestion des **livres** et un module secondaire dédié aux **auteurs**, tout en facilitant la création de KPI et l'exploitation de l'API **OpenLibrary**, conformément aux objectifs fonctionnels et pédagogiques du projet.

Technologies utilisées

Langages utilisés

- **HTML :** HTML5 a été utilisé pour structurer le contenu de l'application. Il a permis de créer les sections, formulaires et boutons nécessaires à l'interaction avec l'utilisateur.

- **CSS :** CSS3 a assuré la mise en forme et le design de l'interface. Les styles ont été appliqués pour améliorer l'ergonomie et la lisibilité des pages.

- **JavaScript :** JavaScript a géré la logique de l'application et les interactions dynamiques. Il a permis de manipuler le DOM pour mettre à jour le contenu en temps réel, gérer les événements utilisateurs et valider les formulaires. Les fonctions JS ont également servi à rendre le dashboard interactif et à calculer

les statistiques affichées.

Librairies et outils

- Aucune librairie CSS externe n'a été utilisée afin de respecter les contraintes du projet et de renforcer la maîtrise du CSS.

- La librairie **Chart.js** a été intégrée uniquement pour la partie **Dashboard**, permettant l'affichage de graphiques statistiques et d'indicateurs visuels dynamiques.

Outils de développement

Le développement de l'application a été réalisé à l'aide de l'éditeur **Visual Studio Code**. Les tests ont été effectués via un navigateur Web.

Le suivi du projet et la synchronisation du code ont été assurés par **Git** et **GitHub**, facilitant le contrôle de l'avancement et la validation du travail par l'encadrante.

Développement de l'application

Structure générale de l'application

4.1.1 Organisation des fichiers

L'application est organisée en trois fichiers principaux pour une séparation claire des responsabilités et une maintenance aisée :

- **index.html** : Interface unique combinant pages d'authentification et tableau de bord (login/inscription, sections Dashboard/Gestion Livres/Gestion Auteurs, modales)

- **style.css** : Thèmes et animations pour l'authentification et l'application (variables, sidebar, cards)

- **script.js** : Logique métier complète (authentification, gestion livres/auteurs, dashboard, Chart.js, intégration API OpenLibrary, horloge, persistance localStorage)

4.1.2 Gestion de l'affichage des sections (SPA)

Le principe de Single Page Application (SPA) est rigoureusement respecté. Toutes les sections (#dashboard, #books, #authors) sont contenues dans le fichier index.html et masquées par défaut avec la classe Bootstrap d-none.

La navigation est entièrement pilotée par JavaScript : au clic sur un lien de la sidebar, la section correspondante est affichée tandis que les autres sont masquées. Une classe CSS active est appliquée dynamiquement à l'élément de navigation actif pour un feedback visuel immédiat.

Cette approche permet une expérience utilisateur fluide, sans rechargement de page.

Module 1 : Gestion des Livres (CRUD complet)

4.2.1 Ajout et modification d'un livre

Un formulaire modal unique (accessible via le bouton *Nouveau Livre*) est utilisé à la fois pour l'ajout et la modification. Il contient les champs suivants :

- **Titre** (obligatoire)
- **Auteur** (obligatoire, saisie libre)
- **Année de publication** (nombre)
- **Genre** (sélection via `<select>` avec des options prédéfinies : Roman, Science-Fiction, etc.)

Un champ caché (#bookId) permet de différencier une création (id généré via Date.now()) d'une mise à jour. La validation des champs obligatoires est gérée par l'attribut HTML required.

4.2.2 Affichage de la liste

La liste des livres est présentée dans un tableau Bootstrap (`<table>`) avec les colonnes : Titre, Auteur, Année, Genre et Actions. Chaque ligne comporte des boutons pour **Modifier** et **Supprimer**.

Le rendu est dynamique : la fonction render() parcourt le tableau books et génère le HTML correspondant.

4.2.3 Recherche et tri

- **Recherche** : Un champ de texte (#searchBook) permet de filtrer en temps réel la liste des livres par titre ou par auteur. L'événement input déclenche un filtrage sur le tableau books.
- **Tri** : Un menu déroulant (#sortBook) permet de trier les livres par titre (ordre alphabétique) ou par année de publication (croissant/décroissant).

4.2.4 Suppression avec confirmation

La suppression d'un livre déclenche une boîte de dialogue native (confirm()) pour confirmation. Si l'utilisateur valide, le livre est retiré du tableau books et l'affichage est mis à jour.

4.2.5 Sauvegarde dans le LocalStorage

Après chaque opération de création, modification ou suppression, le tableau books est sérialisé en JSON et sauvegardé dans le LocalStorage sous la clé 'books'. Au chargement de la page, les données sont restaurées, garantissant la persistance des données entre les sessions.

</> Module 2 : Gestion des Auteurs (CRUD simplifié)

4.3.1 Ajout d'un auteur

Un formulaire modal simplifié permet d'ajouter un auteur par son nom complet. Le nom est ajouté au tableau authors après vérification de l'unicité.

4.3.2 Affichage de la liste

Les auteurs sont affichés sous forme de cartes (cards) Bootstrap organisées en grille. Chaque carte présente le nom de l'auteur et un bouton de suppression.

4.3.3 Suppression

La suppression d'un auteur suit le même principe que pour les livres, avec confirmation. Une vérification supplémentaire pourrait être implémentée pour empêcher la suppression d'un auteur associé à des livres

(intégrité des données).

4.3.4 Lien entre auteurs et livres

Bien que l'interface actuelle utilise un champ de saisie libre pour l'auteur d'un livre, l'architecture permet aisément d'implémenter un lien plus fort. Le tableau authors est stocké indépendamment et pourrait alimenter un élément `<select>` dans le formulaire des livres, garantissant la cohérence et évitant les doublons.

</> Gestion de l'authentification et des sessions

4.4.1 Système d'inscription et de connexion

Le module auth.js gère un système complet d'authentification basé sur le LocalStorage :

- **Inscription** : Vérifie l'unicité du nom d'utilisateur et la correspondance des mots de passe (minimum 6 caractères).
- **Connexion** : Vérifie les identifiants et crée une session en stockant l'utilisateur courant sous la clé 'bibliotheca_current_user'.
- **Persistance de session** : Au chargement, index.html lit localStorage via initializeApp() pour décider d'afficher la page principale ou les formulaires d'authentification, sans redirection vers une autre page.
- **Déconnexion** : Supprime la session courante et redirige vers la page de connexion.

4.4.2 Affichage utilisateur et horloge

La barre de navigation supérieure affiche dynamiquement le nom de l'utilisateur connecté (#currentUsername). Une horloge numérique mise à jour chaque seconde (#clock) apporte un aspect dynamique et professionnel à l'interface.

</> Module 3 : Dashboard & Intégration d'API

4.5.1 Mise à jour des indicateurs KPI

Le Dashboard calcule et affiche en temps réel quatre indicateurs clés de performance (KPI) :

1. **Total Livres** : Nombre total de livres dans la bibliothèque.
2. **Total Auteurs** : Nombre total d'auteurs enregistrés.
3. **API Status** : Indicateur textuel ("OK") confirmant la connectivité à l'API externe.
4. **Dernier Ajout** : Titre du dernier livre ajouté à la collection.

Ces KPI sont mis à jour à chaque modification des données via la fonction updateKPIs().

4.5.2 Visualisation avec Chart.js

Un graphique à barres est généré avec la librairie **Chart.js**. Il présente la répartition des livres par genre littéraire. Le graphique est entièrement dynamique : il est recréé (destroy() puis new Chart()) à chaque mise à jour des données pour refléter l'état actuel de la collection.

4.5.3 Intégration asynchrone avec une API publique

L'application effectue un appel asynchrone à l'API **OpenLibrary** via la fonction fetchExternalInfo(). Cet appel :

- Utilise fetch() avec gestion des promesses (async/await).
- Récupère les données JSON d'un livre spécifique (œuvre référence : OL26301W).
- Affiche dans le Dashboard le titre et un extrait de la description du livre.
- Gère les erreurs de réseau ou d'API avec un bloc try...catch et affiche un message approprié à l'utilisateur.

Cette intégration démontre la capacité de l'application à consommer et à afficher des données externes de manière réactive.

Dashboard et visualisation des données

KPI de la bibliothèque

Le Dashboard présente quatre **indicateurs clés de performance (KPI)** visant à offrir une vision synthétique et immédiate de l'état de la bibliothèque.

5.1.1 Nombre total de livres

- **Calcul** : Longueur du tableau books stocké en mémoire et dans le LocalStorage.
- **Affichage** : Valeur numérique mise en avant dans une carte colorée (bleue) avec l'icône fa-book.
- **Signification** : Indicateur principal de la taille de la collection.

5.1.2 Nombre d'auteurs

- **Calcul** : Longueur du tableau authors.
- **Affichage** : Carte verte avec l'icône fa-user-tie.
- **Signification** : Mesure de la diversité des auteurs représentés dans la bibliothèque.

5.1.3 Statut de l'API

- **Affichage** : Carte info (bleu clair) affichant le texte "OK" lorsque l'appel à OpenLibrary réussit.
- **Fonction** : Indicateur de santé et de connectivité externe de l'application.

5.1.4 Dernier ajout

- **Calcul** : Titre du dernier élément ajouté au tableau books (index books.length - 1).
- **Affichage** : Carte orange (bg-warning) présentant le titre tronqué si nécessaire.
- **Signification** : Permet de visualiser rapidement la dernière activité d'enrichissement de la collection.

Ces KPI sont **dynamiquement mis à jour** après chaque opération CRUD via la fonction updateKPIs(), garantissant un aperçu en temps réel.

Graphiques et statistiques

5.2.1 Présentation des graphiques Chart.js

Un graphique à barres est généré avec la bibliothèque **Chart.js** pour visualiser la répartition des livres par genre littéraire.

Implémentation technique :

- Le contexte du canvas est récupéré via getElementById('booksChart').getContext('2d').
- Les données sont préparées dynamiquement :

```
const genres = [...new Set(books.map(b => b.genre))];
const counts = genres.map(g => books.filter(b => b.genre === g).length);
```

- Le graphique est instancié avec type: 'bar', des couleurs cohérentes (#4e73df) et des options de responsive design.
- Avant chaque nouvelle génération, le graphique existant est détruit (myChart.destroy()) pour éviter les conflits de rendu.

5.2.2 Analyse des résultats affichés

Le graphique permet une **analyse visuelle rapide** :

- Identification des **genres les plus représentés** dans la collection (barres les plus hautes).
- Visualisation de l'**équilibre ou du déséquilibre** entre les catégories.
- Capacité à **suivre l'évolution** de la répartition après chaque ajout ou suppression de livre.

Cette visualisation transforme des données tabulaires en informations **immédiatement actionnables**, répondant à un besoin classique des dashboards professionnels.

Intégration de l'API externe (Asynchrone)

Présentation de l'API OpenLibrary

L'application intègre l'**API OpenLibrary**, un projet ouvert offrant un accès gratuit à des métadonnées bibliographiques.

L'endpoint utilisé est :

`https://openlibrary.org/works/OL26301W.json`

Il retourne des informations structurées sur l'œuvre *"À la recherche du temps perdu"* de Marcel Proust, sous forme d'objet JSON contenant le titre, la description, les auteurs, etc.

Utilisation de fetch()

L'appel asynchrone est réalisé avec l'API moderne fetch() :

```
const response = await fetch('https://openlibrary.org/works/OL26301W.json');
const data = await response.json();
```

- **async** : Déclare une fonction asynchrone
- **await** : Suspend l'exécution jusqu'à résolution de la promesse
- **Flux séquentiel** : Lecture naturelle du code
- **Gestion d'erreurs** : Utilisation de `try...catch`
- **Interface utilisateur** : Affichage après traitement complet

Traitement des données JSON

Les données retournées sont parsées en objet JavaScript via response.json().

Des informations pertinentes sont extraites :

- data.title : titre de l'œuvre.
- data.description.value : description complète (tronquée à 100 caractères pour l'affichage).

Exploitation des données dans le dashboard

Les données de l'API sont injectées dans la section dédiée du Dashboard (#api-data) :

```
apiContainer.innerHTML = `
    <p><strong>Livre du jour (API) :</strong> ${data.title}</p>
    <p><strong>Description :</strong> ${data.description?.value?.substring(0, 100) || "Pas de
        description"}...</p>
`;
```

Cela **enrichit l'expérience utilisateur** en proposant un contenu externe dynamique et pertinent, tout en démontrant la capacité d'intégration de sources de données variées.

Gestion des erreurs (try / catch)

Une gestion robuste des erreurs est implémentée pour assurer la stabilité de l'application :

```
try {
    const response = await fetch('https://openlibrary.org/works/OL26301W.json');
    const data = await response.json();
    apiContainer.innerHTML = `
        <p><strong>Livre du jour (API) :</strong> ${data.title}</p>
        <p><strong>Description :</strong> ${data.description?.value?.substring(0, 100) || "Pas de
            description"}...</p>
    `;
} catch (error) {
    apiContainer.innerHTML = "Impossible de charger les données API.";
    console.error("Erreur API:", error);
}
```

Tests et validation

La phase de tests a permis de valider le bon fonctionnement de l'application et sa robustesse. Les tests fonctionnels ont couvert l'ensemble des opérations CRUD : l'ajout, la modification, la suppression et l'affichage des livres et des auteurs se sont déroulés sans erreur, avec des validations côté client et des retours visuels immédiats.

La persistance des données a été vérifiée via le LocalStorage : après un rechargement de la page ou une fermeture du navigateur, les données restent disponibles et cohérentes.

Difficultés rencontrées et solutions apportées

Problèmes techniques	Choix techniques	Solutions mises en place
Gestion du graphique dynamique	Chart.js	Destruction et recréation du graphique à chaque mise à jour des données
Persistance des données	LocalStorage	Sauvegarde automatique en JSON après chaque opération CRUD
Appels API fiables	Async/await avec gestion d'erreurs	Implémentation de try/catch avec messages d'erreur utilisateur
Navigation SPA sans framework	JavaScript Vanilla	Gestion manuelle du DOM et des événements de navigation

Résultats et captures d'écran

Présentation des interfaces finales

- Interface Dashboard avec KPI et graphiques

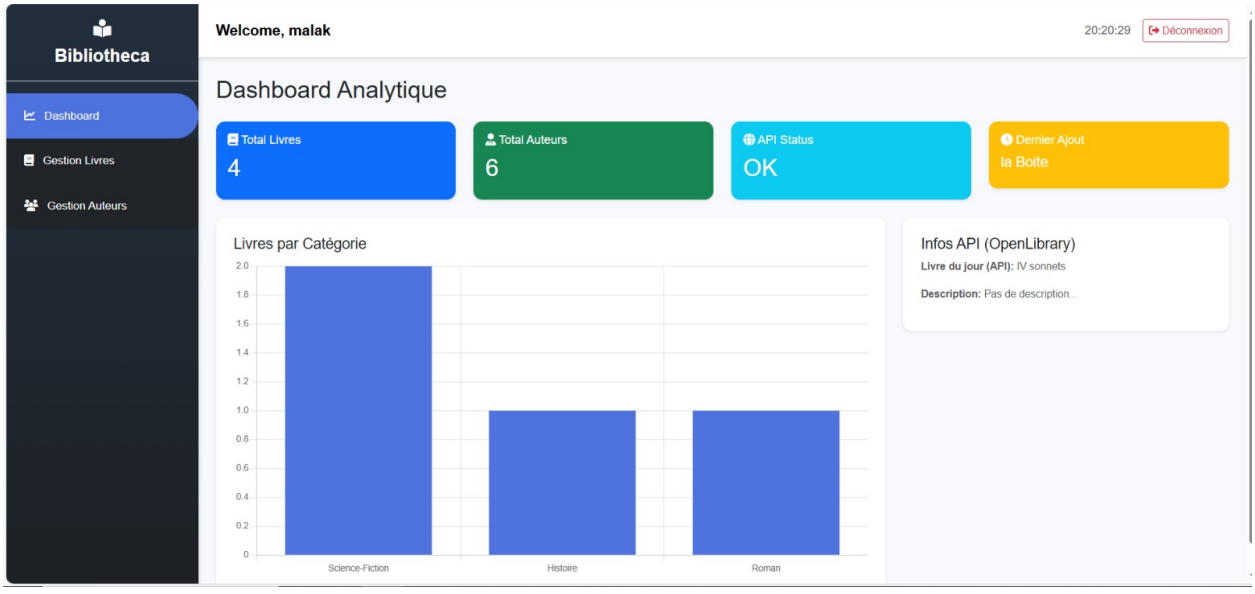

Figure 1: Dashboard

- Module de gestion des livres avec CRUD complet

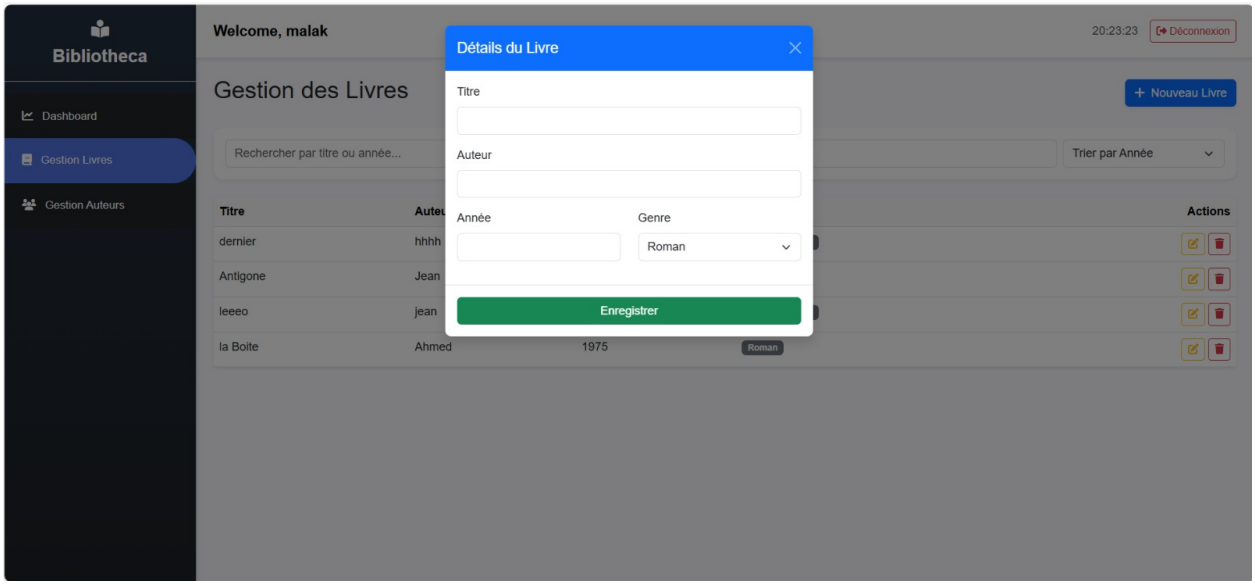

Figure 2: Ajouter livre

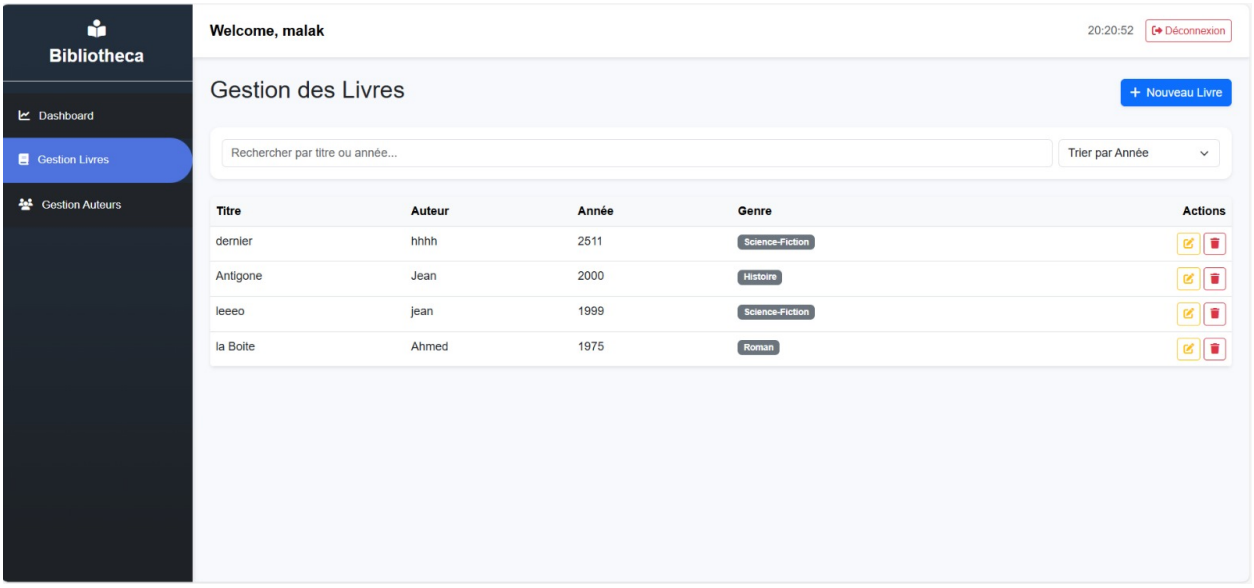

Figure 3: Afficher livre

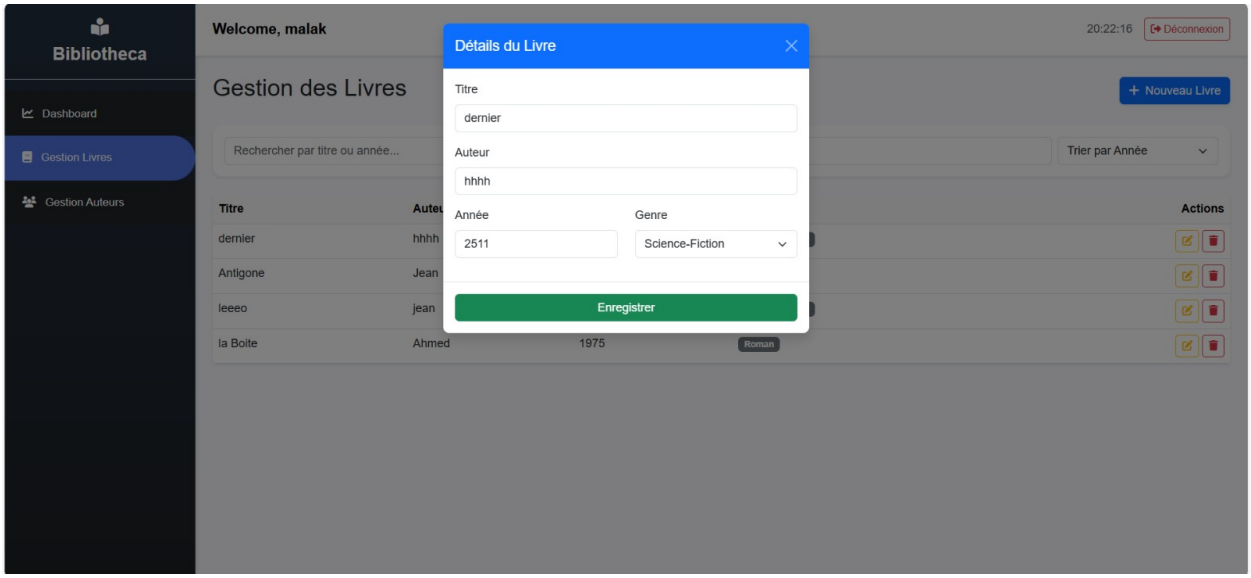

Figure 4: Modifier livre

- Module de gestion des auteurs simplifié

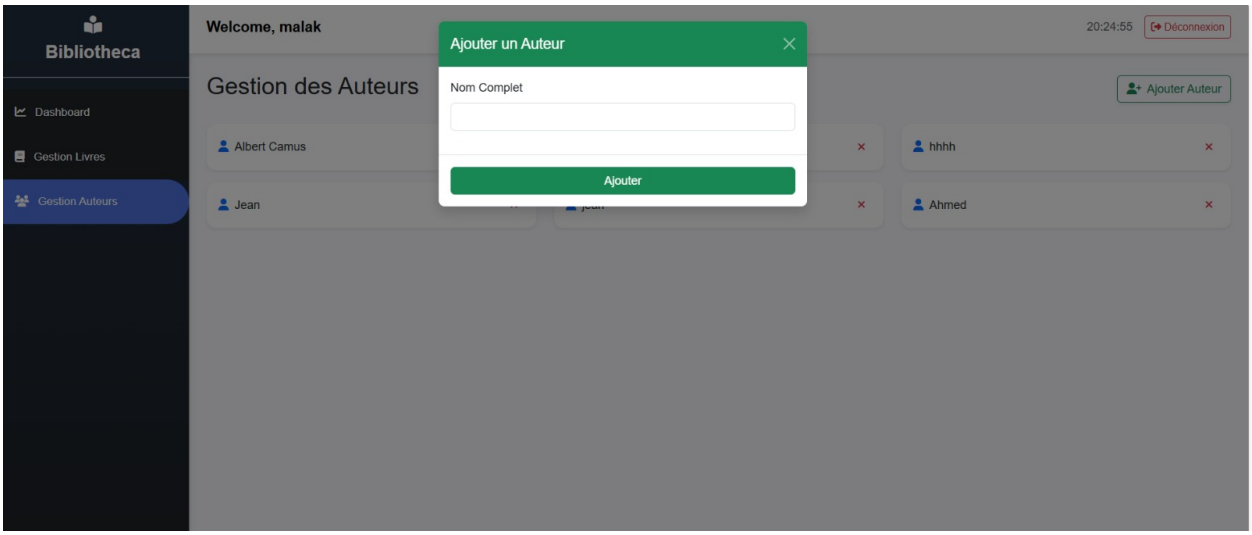

Figure 5: Ajouter auteur

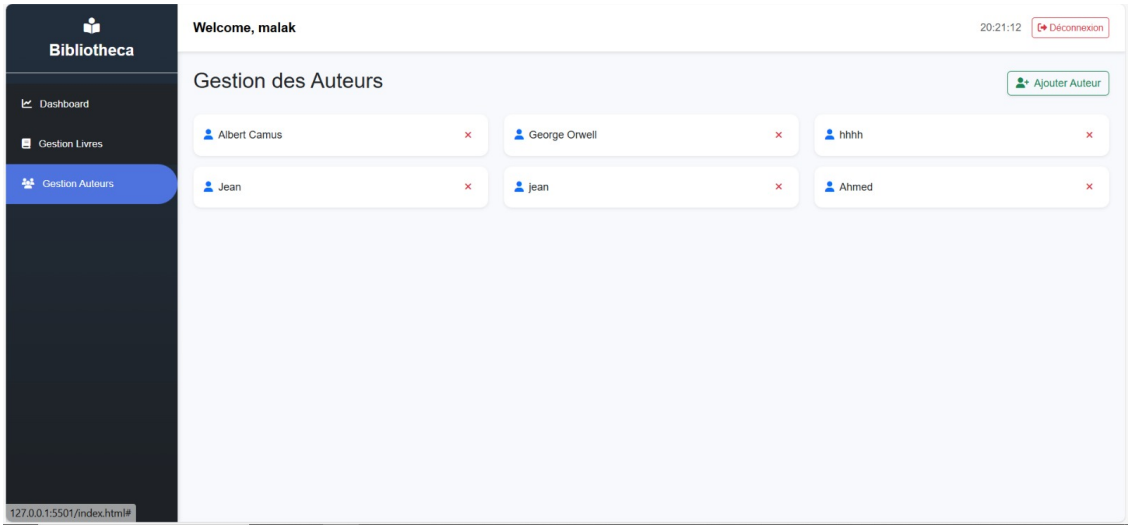

Figure 6: Afficher auteur

- Interface d'authentification moderne

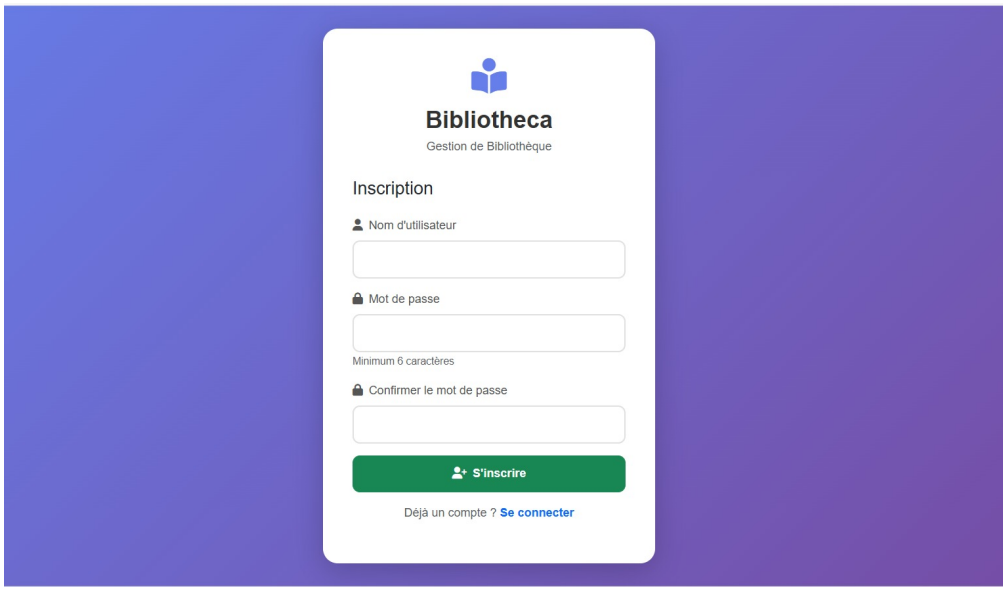

Figure 7: Page d'inscription

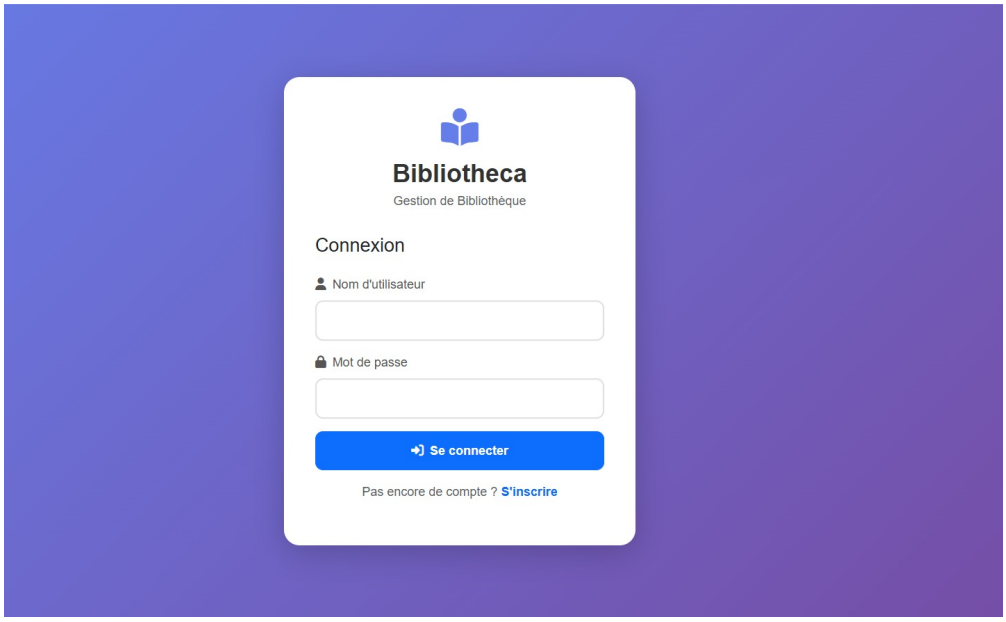

Figure 8: Page de connexion

- Le stockage des données locales

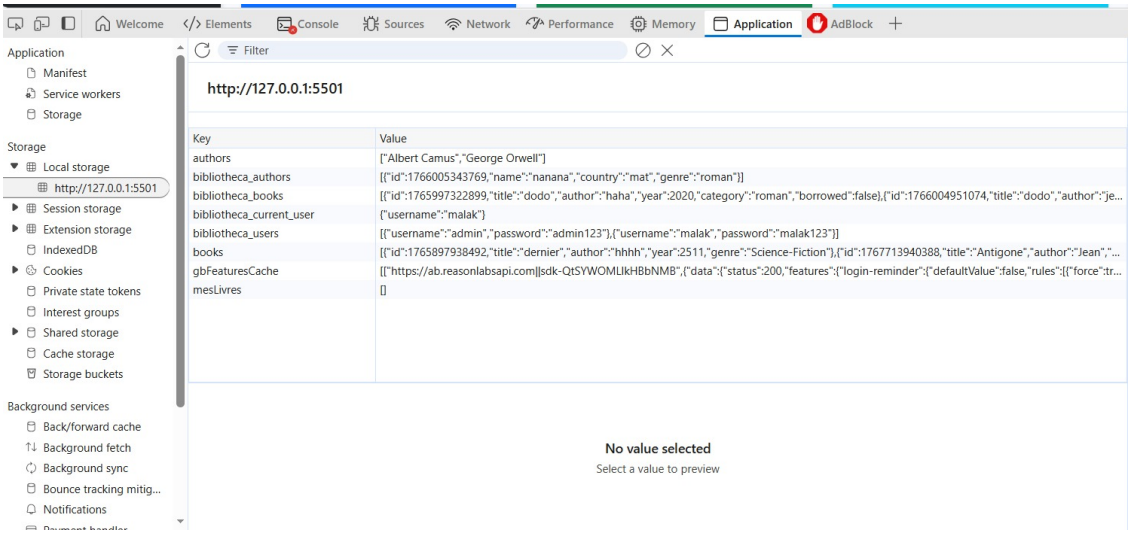

Figure 9: Local storage

Conclusion

Ce projet **Bibliotheca** a représenté bien plus qu'un simple exercice académique : c'est la concrétisation d'une **application web fonctionnelle, moderne et professionnelle**, entièrement développée avec les technologies de base du Web.

À travers cette réalisation, nous avons transformé un cahier des charges exigeant en une **interface opérationnelle et intuitive** qui répond aux besoins de gestion d'une bibliothèque numérique. Chaque composant — depuis l'**authentification sécurisée** jusqu'au **dashboard analytique** en passant par les modules CRUD complets — a été pensé, conçu et intégré pour offrir une **expérience utilisateur fluide et professionnelle**.

Le défi de créer une **SPA sans framework** nous a poussés à **approfondir notre compréhension fondamentale de JavaScript** et à maîtriser des concepts essentiels comme la manipulation fine du DOM, la gestion d'état, l'asynchronisme et l'architecture modulaire. L'intégration réussie d'**API externes** et de **visualisations dynamiques** démontre notre capacité à connecter une application à des écosystèmes de données plus larges.

Sur le plan visuel, l'interface **soignée et responsive**, inspirée des standards professionnels, prouve qu'on peut allier **esthétique et fonctionnalité** même avec des outils de base. L'utilisation judicieuse de **Bootstrap** et de **Chart.js** nous a permis de nous concentrer sur la logique métier tout en garantissant une expérience utilisateur de qualité.

Ce projet nous a également appris à **travailler de manière structurée** : organisation en sprints hebdomadaires, documentation progressive, tests systématiques et gestion des difficultés techniques.