Tosho

Développé par **Miyuki CHERBAL**

Titre professionnel visé : **Développeur Web et Web Mobile**

Novembre 2025

Sommaire

1. Introduction

Tosho est une application web conçue pour faciliter la gestion des prêts de livres au sein d'une école japonaise. En japonais, **Tosho** signifie *« bibliothèque »* ou *« livre »*.

Ce projet vient de mon expérience personnelle. Ma fille apprend le japonais dans une école associative pour les enfants d'origine japonaise. Cette école est entièrement gérée par des parents bénévoles, dont je fais partie. Nous avons une petite bibliothèque et nous prêtons régulièrement des livres aux familles afin que les enfants se familiarisent avec la lecture en japonais.

L'application actuellement utilisée pour les prêts de livres manque d'ergonomie et de fonctionnalités essentielles. L'interface administrateur n'est accessible que par le développeur initial, un ancien parent bénévole, et bien qu'elle reste fonctionnelle, elle présente une interface brute sans mise en forme CSS.

Tosho a pour objectif de simplifier la gestion des prêts de livres au quotidien, tout en offrant une expérience utilisateur plus fluide et moderne. Cette solution permet aux parents bénévoles de disposer d'un outil clair et autonome pour centraliser et gérer efficacement la bibliothèque.

Ce projet m'a permis de mettre en pratique les compétences acquises au cours de ma formation de Développeur Web et Web Mobile, de la conception au déploiement. J'ai pu expérimenter l'ensemble du processus de développement : analyse des besoins, architecture logicielle, gestion de la base de données, développement front-end et back-end, ainsi que la sécurisation des accès et la mise en place d'une interface responsive.

Tosho est mon premier projet concret, reflet de mon apprentissage et de mon évolution en tant que développeuse, dont je suis fière de pouvoir présenter aujourd'hui !

2. Compétences du référentiel couvertes par le projet

2.1 Développer la partie front-end d’une application web ou web mobile sécurisée

Installer et configurer son environnement de travail

Pour ce projet, j’ai utilisé **Visual Studio Code (VSCode)** comme environnement de développement, un **IDE** (Integrated Development Environment) que j’utilise depuis le début de ma formation. Je l’ai progressivement configuré selon mes besoins, en installant diverses extensions utiles pour les langages de programmation et leurs frameworks, notamment :

- **PHP Intelephense** : pour bénéficier de l’autocomplétion, de la détection d’erreurs, et d’une meilleure navigation dans le code PHP.
- **PHP Namespace Resolver** : pour faciliter l’importation et la gestion automatique des namespaces dans les fichiers PHP.
- **Twig Language 2** : pour améliorer la coloration syntaxique et l’autocomplétion des fichiers Twig utilisés dans Symfony.

Cette configuration m’a permis de travailler de manière plus efficace et structurée tout au long du projet, en optimisant la lisibilité du code et en réduisant les erreurs de syntaxe.
Le projet est versionné avec **Git** et **GitHub** pour assurer le suivi des modifications et la sauvegarde.

Maquetter des interfaces utilisateur web ou web mobile

J’ai réalisé les wireframes de mon application pour les formats mobile et desktop en utilisant **Figma**. Cette étape m’a permis de définir l’ergonomie et l’organisation des éléments. Ensuite, j’ai transformé ces wireframes en **maquettes**, ce qui m’a aidée à mieux anticiper les besoins visuels. Avoir un rendu concret sous les yeux m’aide à me projeter dans le développement et à rester concentrée sur le développement.

Réaliser des interfaces utilisateur statiques web ou web mobile

J'ai intégré mes maquettes graphiques au fur et à mesure de l'avancement du développement back-end. Pour chaque route définie, j'ai créé un dossier dédié dans le répertoire des `templates`, contenant les fichiers `.twig` nécessaires à l'affichage de la vue correspondante. Cette structure permet de maintenir une séparation claire entre les différentes parties de l'application et de gagner en efficacité lors de l'intégration des interfaces utilisateur. J'ai également veillé à ce que l'interface soit responsive.

Développer la partie dynamique des interfaces utilisateur web ou web mobile

Pour rendre l'interface plus interactive, j'ai utilisé **JavaScript**, notamment pour automatiser certaines actions et améliorer l'expérience utilisateur. Par exemple, lors de la saisie d'un code ISBN dans le formulaire d'ajout d'un livre, un appel est automatiquement envoyé à une **API** externe. Celle-ci retourne les informations liées au livre : titre, auteur, image de couverture, etc. Ces données sont ensuite affichées dynamiquement dans le formulaire, sans rechargement de la page.

2.2 Développer la partie back-end d'une application web ou web mobile sécurisée

Mettre en place une base de données relationnelle

J'ai mis en place une base de données relationnelle avec **MySQL**. J'ai commencé par concevoir un modèle de données sur papier afin de définir les différentes entités et les relations entre elles. Ensuite, j'ai créé ces entités dans **Symfony** en utilisant **Doctrine ORM**. Chaque entité correspond à une table dans la base de données. Grâce à **CLI** (Command Line Interface) de Symfony, j'ai pu générer automatiquement la structure de la base de données, sans avoir à créer manuellement chaque table. Doctrine simplifie également la gestion des relations entre les entités (OneToMany, ManyToOne, etc.).

Développer des composants d'accès aux données SQL

Pour accéder aux données stockées dans la base, j'ai utilisé les `repositories` fournis par Doctrine. Lorsqu'une requête est envoyée à un contrôleur, celui-ci interagit avec `Entity Manager` qui sert d'intermédiaire entre les contrôleurs et les repositories.
Les repositories permettent de récupérer, filtrer, modifier ou supprimer les données de manière sécurisée. Doctrine gère également les connexions à la base de données et applique automatiquement des protections contre les injections SQL.

Développer des composants métier coté serveur

J’ai structuré mon projet selon **l’architecture MVC (Modèle – Vue – Contrôleur)** de Symfony, afin de séparer clairement la logique métier, l’affichage et le traitement des requêtes. Chaque URL est associée à une route, dirigée vers un contrôleur dédié à une fonctionnalité précise (par exemple : prêts des livres, gestion des inventaires etc.).
Les contrôleurs interagissent avec les entités pour récupérer ou modifier les données, puis transmettent les résultats aux vues. Ce mécanisme est utilisé notamment pour la gestion complète du CRUD des livres.

Documentner le déploiement d’une application dynamique web ou web mobile

Pour le déploiement, j’ai utilisé **Docker**, ce qui permet de standardiser l’environnement de développement et de production. J’ai rédigé un `dockerfile` qui décrit toutes les étapes nécessaires pour construire l’image de l’application : installation des dépendances, configuration, copie des fichiers, etc.
Toutes les commandes nécessaires à l’exécution du projet avec Docker sont documentées dans le fichier `README.md` . Cela permet à n’importe quel utilisateur de cloner le dépôt et de lancer l’application en quelques lignes de commande, sans avoir à configurer manuellement l’environnement.

3. Cahier des charges

3.1 Contexte et objectifs

Contexte

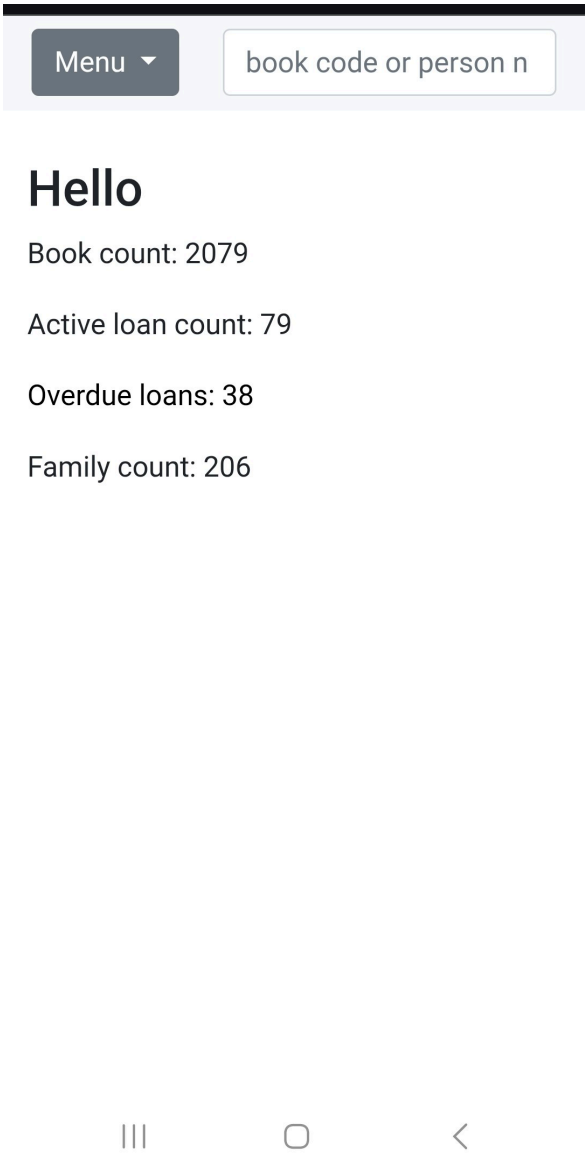
Le projet **Tosho** est inspiré d'une application web actuellement utilisée au sein d'une école japonaise associative qui propose des cours de japonais aux enfants d'origine japonaise. L'école est entièrement gérée par des parents bénévoles, dont je fais partie, et met à disposition une petite bibliothèque afin d'encourager la lecture en japonais auprès des enfants.

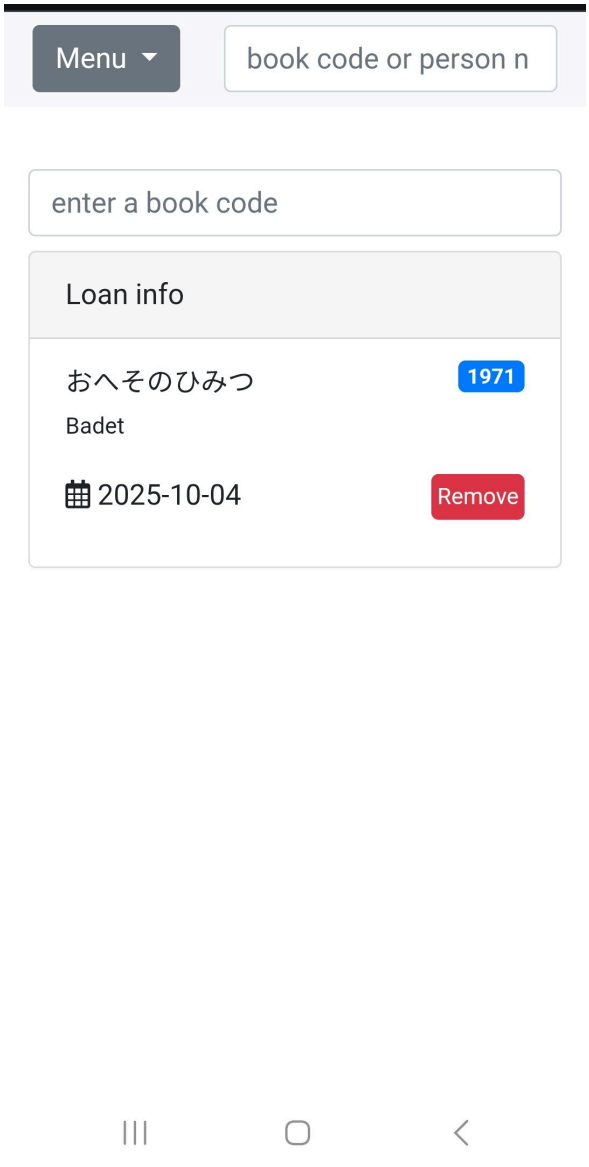
Chaque semaine, les familles adhérentes peuvent emprunter des livres pour leurs enfants. Jusqu'à présent, ces prêts sont gérés via une application web existante, développée il y a plusieurs années par un ancien parent bénévole.
Le fonctionnement actuel est le suivant :

1. La famille choisit les livres à emprunter.
2. Chaque livre possède une étiquette avec un code unique (différent de l'ISBN) généré par l'association.
3. Le parent bibliothécaire saisit le nom de famille de l'emprunteur.
4. La liste des livres déjà empruntés par cette famille s'affiche.
5. Si la famille rapporte des livres, le bibliothécaire enregistre le retour de chaque livre.
6. Pour enregistrer un nouveau prêt, il saisit le code du livre correspondant.

Bien que cette application soit fonctionnelle, elle présente plusieurs limitations importantes :

- La partie administrateur (gestion des livres, des familles adhérentes, des bibliothécaires et des inventaires annuels) n'est pas accessible aux bénévoles actuels et nécessite toujours l'intervention du développeur d'origine.
- La recherche de familles n'est possible qu'en alphabet latin, ce qui rend la recherche en japonais (hiragana) impossible.
- L'interface est peu ergonomique, sans mise en page ni design CSS.





Ces contraintes rendent la gestion quotidienne de la bibliothèque peu flexible pour les parents bénévoles.

C’est dans ce contexte qu’a été conçu Tosho, une nouvelle application web de gestion des prêts de livres, pensée pour offrir une utilisation simple, fluide et autonome, sans dépendance à un intervenant technique extérieur.

Objectifs

Le projet Tosho a pour objectif principal de faciliter et moderniser la gestion de la bibliothéque, tout en offrant aux parents bénévoles un outil simple, efficace et autonome.

Plus précisément, le projet vise à :

- Centraliser la gestion de la bibliothèque, en regroupant toutes les informations relatives aux livres, aux familles et aux bénévoles dans une interface administrateur claire et accessible.
- Améliorer l'ergonomie et l'expérience utilisateur, avec une interface intuitive, responsive et agréable à utiliser, adaptée aux besoins des bénévoles non techniques.
- Permettre un inventaire fiable et autonome, avec la possibilité de signaler facilement les livres manquants, mal rangés ou abîmés.
- Offrir une solution évolutive, qui pourra être enrichie ultérieurement de fonctionnalités supplémentaires (réservations, rappels automatiques, multilingue, etc.).

Ainsi, Tosho se positionne comme une solution moderne et complète, permettant aux bénévoles de gérer la bibliothèque de manière autonome et efficace, tout en garantissant un suivi fiable des prêts et retours de livres.

3.2 User stories

Échelle de priorité :

- Priorité 0 : Obligatoire
- Priorité 1 : Nécessaire
- Priorité 2 : Secondaire

Page de connexion

En tant que...	Je veux...	Afin de...	Priorité
Utilisateur (Admin ou Bibliothécaire)	Me connecter à l'application	Accéder à mes fonctionnalités selon mon rôle	0
Utilisateur	Récupérer mon mot de passe oublié	Pouvoir accéder à nouveau à mon compte	0

Gestion des prêts

En tant que...	Je veux...	Afin de...	Priorité
Bibliothécaire	Enregistrer un prêt (livre, date, famille emprunteuse)	Suivre les emprunts de livres	0

En tant que...	Je veux...	Afin de...	Priorité
Bibliothécaire	Enregistrer le retour d’un livre	Mettre à jour la disponibilité	0

Inventaire (Côté bibliothécaire)

En tant que...	Je veux...	Afin de...	Priorité
Bibliothécaire	Saisir l’ID d’un livre et valider sa présence lors de l’inventaire	Vérifier que le livre est bien là	1
Bibliothécaire	Signaler une anomalie	Identifier anomalie	1

Gestion des livres (Admin uniquement)

En tant que...	Je veux...	Afin de...	Priorité
Admin	Ajouter un nouveau livre	Enrichir l’inventaire	1
Admin	Consulter les détails d’un livre	Vérifier les informations	1
Admin	Modifier les informations d’un livre	Corriger ou mettre à jour	1
Admin	Supprimer un livre	Retirer un livre obsolète	2

Gestion des familles adhérentes (Admin uniquement)

En tant que...	Je veux...	Afin de...	Priorité
Admin	Ajouter une nouvelle famille	Enregistrer les membres	1
Admin	Consulter les informations d’une famille	Vérifier les données	1

En tant que...	Je veux...	Afin de...	Priorité
Admin	Modifier les informations d’une famille	Mettre à jour	1
Admin	Supprimer une famille	Supprimer des adhérents	2

Gestion des bibliothécaires (Admin uniquement)

En tant que...	Je veux...	Afin de...	Priorité
Admin	Créer un compte bibliothécaire	Leur permettre d'accéder à l'application	1
Admin	Modifier un compte bibliothécaire	Mettre à jour leurs informations	1
Admin	Supprimer un compte bibliothécaire	Retirer l'accès à quelqu'un qui ne fait plus partie	2
Admin	Voir la liste des bibliothécaires	Gérer plus facilement l'équipe de gestion	1
Admin	Activer/désactiver un compte bibliothécaire	Contrôler l'accès à l'application	1

Gestion de l’inventaire (Admin uniquement)

En tant que...	Je veux...	Afin de...	Priorité
Admin	Programmer une session d'inventaire	Planifier quand les bénévoles vont vérifier les livres	1
Admin	Actualiser l'état d'inventaire (session ouverte/fermée/à venir, etc.)	Suivre correctement le statut de chaque session	1
Admin	Voir l'avancement de l'inventaire	Savoir combien de livres ont été vérifiés et combien restent	1

En tant que...	Je veux...	Afin de...	Priorité
Admin	Modifier l'état des livres signalés	Mettre à jour l’état après avoir réglé le problème	1

Interface et sécurité

En tant que...	Je veux...	Afin de...	Priorité
Admin	Passer de l’interface Admin à l’interface Bibliothécaire	Gérer la bibliothèque comme un parent bibliothécaire	2
Bibliothécaire	Modifier mon mot de passe	Sécuriser mon compte ou le mettre à jour	1
Bibliothécaire	Initialiser mon mot de passe	En cas de perte de mot de passe	0

3.3 Contraintes

- L’interface doit être **simple** et **intuitive**, adaptée à des utilisateurs non techniques. Les bibliothécaires et **les administrateurs sont des bénévoles**.
- L’accès aux fonctionnalités doit être restreint selon le rôle de l’utilisateur :
  - **Admin (parent bénévole)** : gère les familles, les livres, les bibliothécaires et l’inventaire.
  - **Bibliothécaire (parent bénévole)** : enregistre les prêts et retours, et participe aux sessions d’inventaire.
- Les données doivent être fiables et mises à jour en temps réel afin d’éviter les erreurs de double prêt ou de livres manquants.

3.4 Arborescence

1. Pages publiques

- `/` → Page d'accueil
- `/privacy` → Politique de confidentialité

- `/mentions` → Mentions légales
- `/cookies` → Gestion des cookies

2. Authentification

- `/login` → Connexion
- `/logout` → Déconnexion
- `/reset-password` → Demande de réinitialisation
- `/reset-password/check-email` → Vérification email
- `/reset-password/reset/{token}` → Réinitialisation du mot de passe

3. Espace Administrateur

- `/admin/` → Home admin
- **Gestion des livres**
  - `/admin/book/` → Chercher ou ajouter
  - `/admin/book/{id}` → Détails d'un livre
  - `/admin/book/edit/{id}` → Modifier un livre
  - `/admin/book/delete/{id}` → Supprimer un livre
- **Gestion des familles**
  - `/admin/family/` → Chercher ou ajouter
  - `/admin/family/{id}` → Détails d'une famille
  - `/admin/family/edit/{id}` → Modifier une famille
  - `/admin/family/delete/{id}` → Supprimer une famille
- **Gestion des bibliothécaires**
  - `/admin/librarien/` → Chercher ou ajouter
  - `/admin/librarien/{id}` → Détails
  - `/admin/librarien/delete/{id}` → Supprimer un bibliothécaire
  - `/admin/librarien/change-status/{id}` → Activer / désactiver
- **Gestion des inventaires**
  - `/admin/inventory/` → Chercher ou ajouter
  - `/admin/inventory/{id}` → Détails d'une session
  - `/admin/inventory/items/{id}/{page}` → Gestion des inventaires (livres à vérifier / déjà vérifiés / restant à vérifier)
  - `/admin/inventory/edit/{id}` → Modifier une session
  - `/admin/inventory/delete/{id}` → Supprimer une session

    - `/admin/inventory/edit-item/{id}` → Modifier le statut d’un livre dans une session d’inventaire
- `/admin/switch` → Passer à l’interface bibliothécaire

4. Espace Bibliothécaire

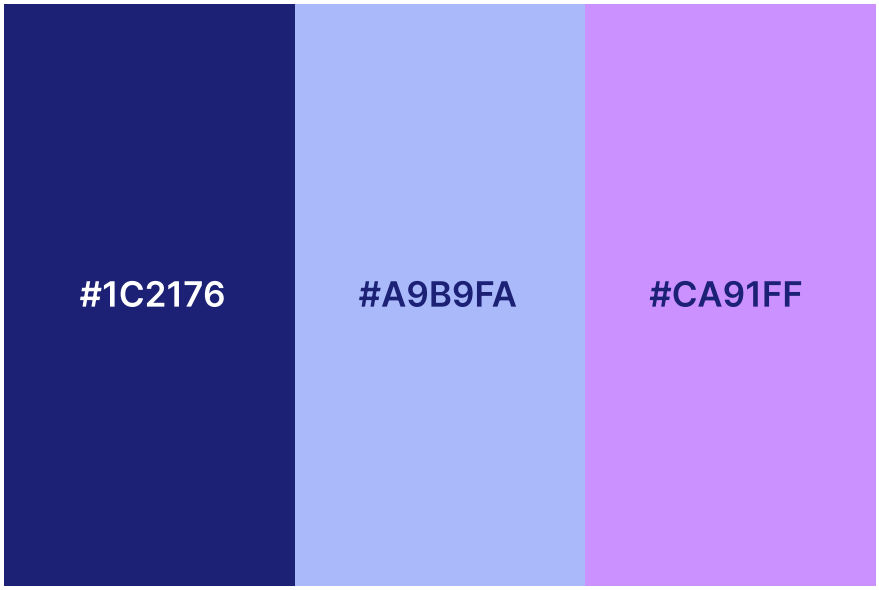
- `/home/` → Home bibliothécaire
- **Prêts et retours**
    - `/loan/` → Prêts et retours
        - `/loan/book/{id}` → Détails d’un livre
        - `/loan/loan-book/{id}` → Prêter un livre à une famille déjà sélectionnée
        - `/loan/loan-book/{id}/{family}` → Prêter un livre en choisissant une famille
        - `/loan/family/{id}` → Voir les prêts d’une famille
        - `/loan/return/{id}` → Retour d’un livre
        - `/loan/overdue` → Livres en retard
- **Inventaire (côté bibliothécaires)**
    - `/inventory/` → Liste des sessions d'inventaire ouvertes
    - `/inventory/{id}` → Détails d’une session
    - `/inventory/{id}/add/{book}` → Ajouter un livre à la session
    - `/inventory/{id}/edit/{item}` → Modifier le statut d’un livre dans une session d’inventaire
    - `/inventory/{id}/{page}` → Liste des livres ajoutés (livres à vérifier / déjà vérifiés / restant à vérifier)
- **Compte utilisateur**
    - `/account/` → Mon compte
    - `/account/edit` → Modifier mon compte
    - `/account/change` → Changer mon mot de passe

4. Conception visuelle

L’identité visuelle de **Tosho** a été pensée pour refléter l’esprit d’une association scolaire : à la fois **ludique**, **conviviale** et **accessible**.
L’objectif est de proposer une interface simple à comprendre, agréable à utiliser et adaptée aux parents bénévoles.

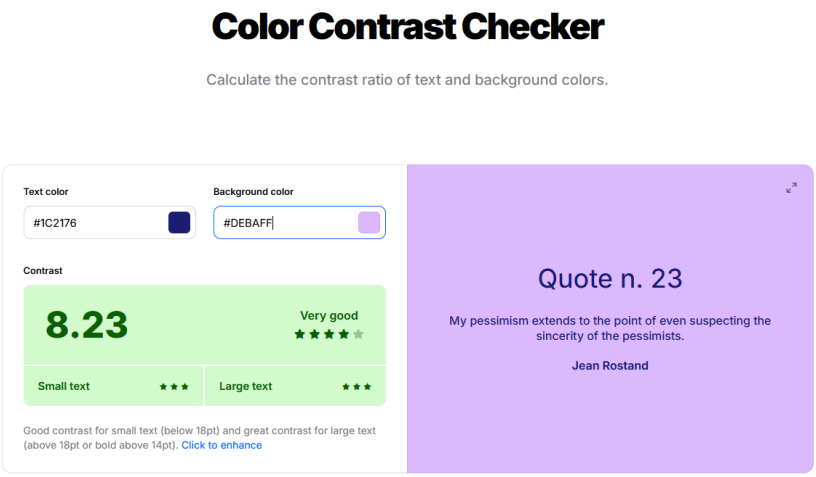
4.1 Charte graphique

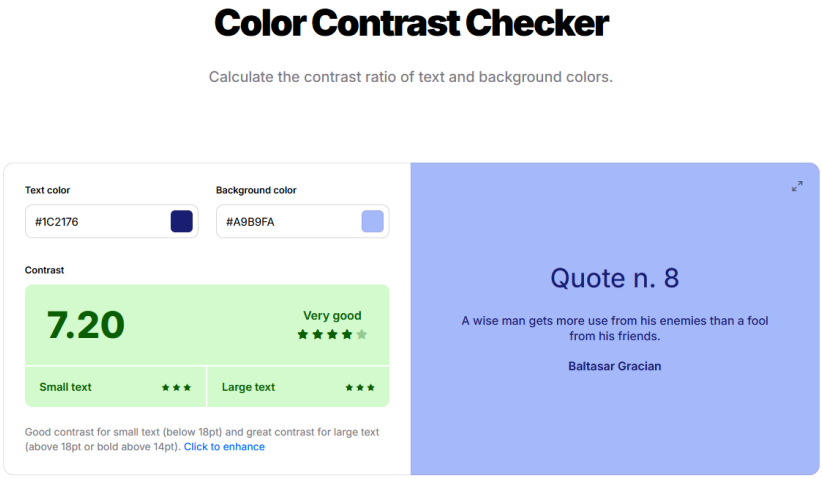
Couleurs principales



- `#1C2176` : pour le texte, les bordures et les icônes
- `#CA91FF` : couleur principale de l'interface **bibliothécaire**
- `#A9B9FA` : couleur principale de l'interface **admin**

Une bonne visibilité des choix de couleurs est vérifiée à l’aide du site https://coolors.co/contrast-checker/





Typographie

Le choix des polices a été fait avec soin pour garantir une lecture claire tout en apportant une touche moderne.

- `"MuseoModerno"` : pour les titres et le menu
- `"Outfit"` ; pour les textes courants

Logo

Le logo ***Tosho*** — qui signifie *livre* ou *bibliothéque* en japonais — a été conçu sur Figma. J'ai chosisi la police `"Climate Crisis"` pour son style rétro, en harmonie avec le style **pixel art** des icônes.

Tosho

Icônes

J'ai choisi les icônes **"Pixel Free Icons"** au style **pixel art** afin d’apporter une touche **ludique** et **conviviale** à l’application.
Elles ont été exportées au format **SVG** depuis Figma.
À partir de ces icônes, j'ai généré un favicon en utilisant l’outil **RealFaviconGenerator.net** pour optimiser la compatibilité sur différents navigateurs et appareils.

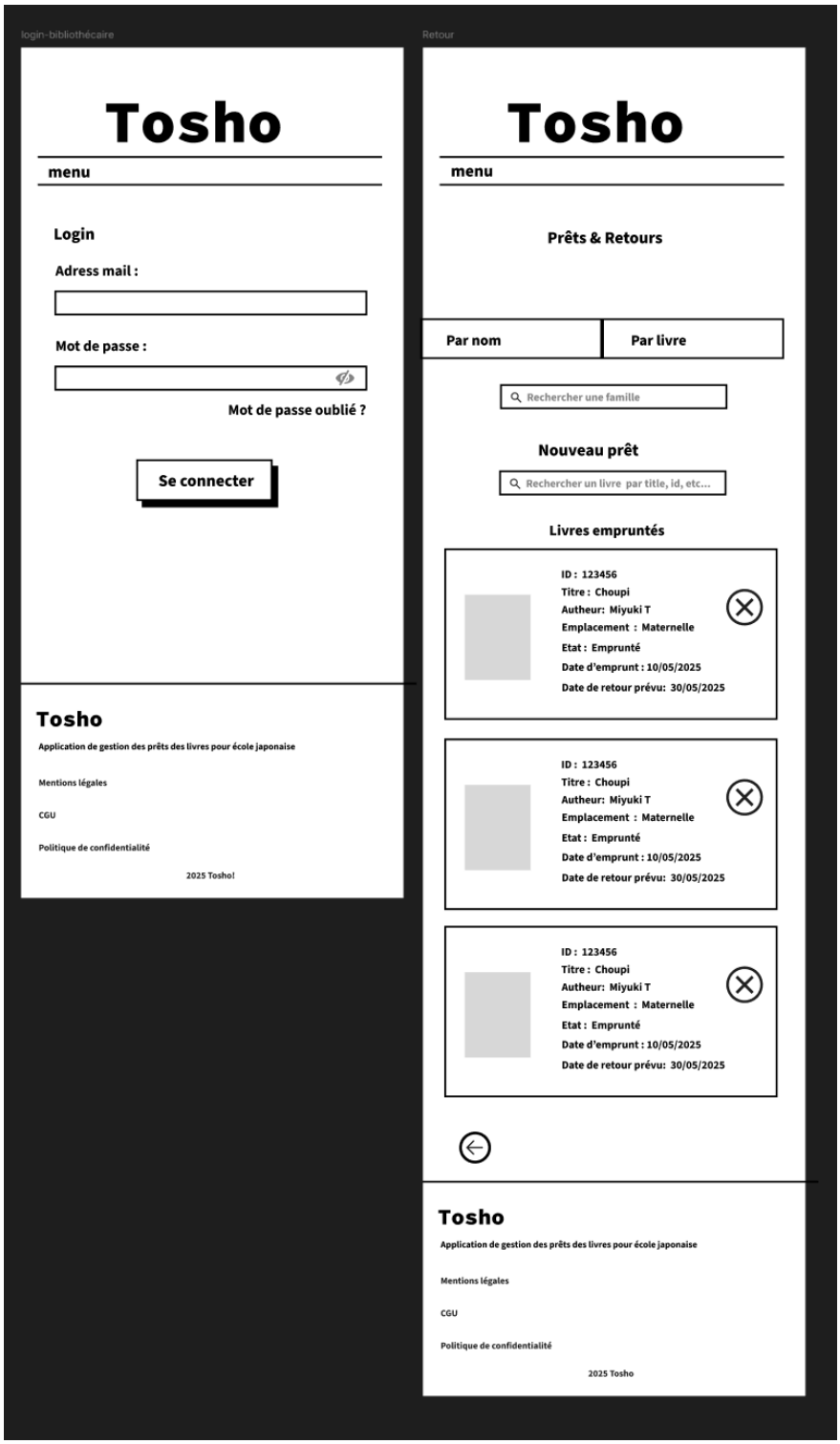


4.2 Wireframes

Les wireframes m'ont permis de **planifier la structure et l’organisation de l’interface** avant de passer à la création des maquettes.
Ils servent à visualiser rapidement la disposition des éléments et le parcours utilisateur sans se soucier du design final.

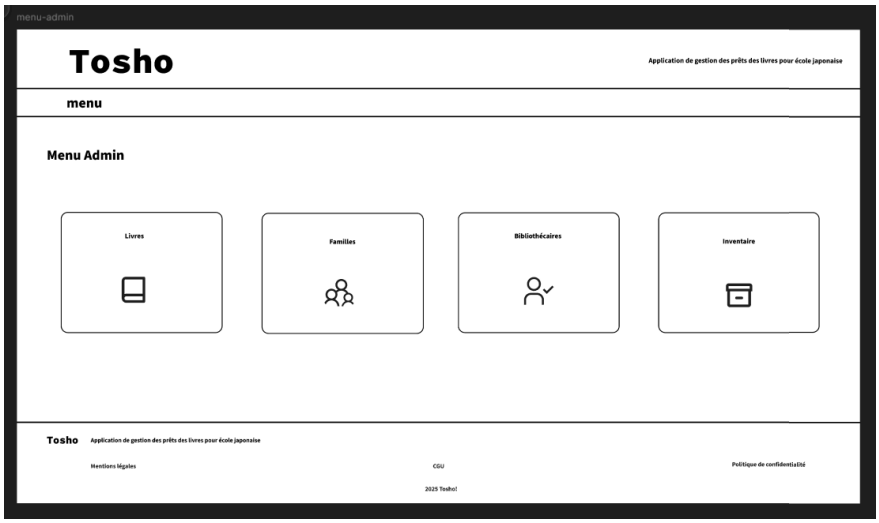
Version mobile

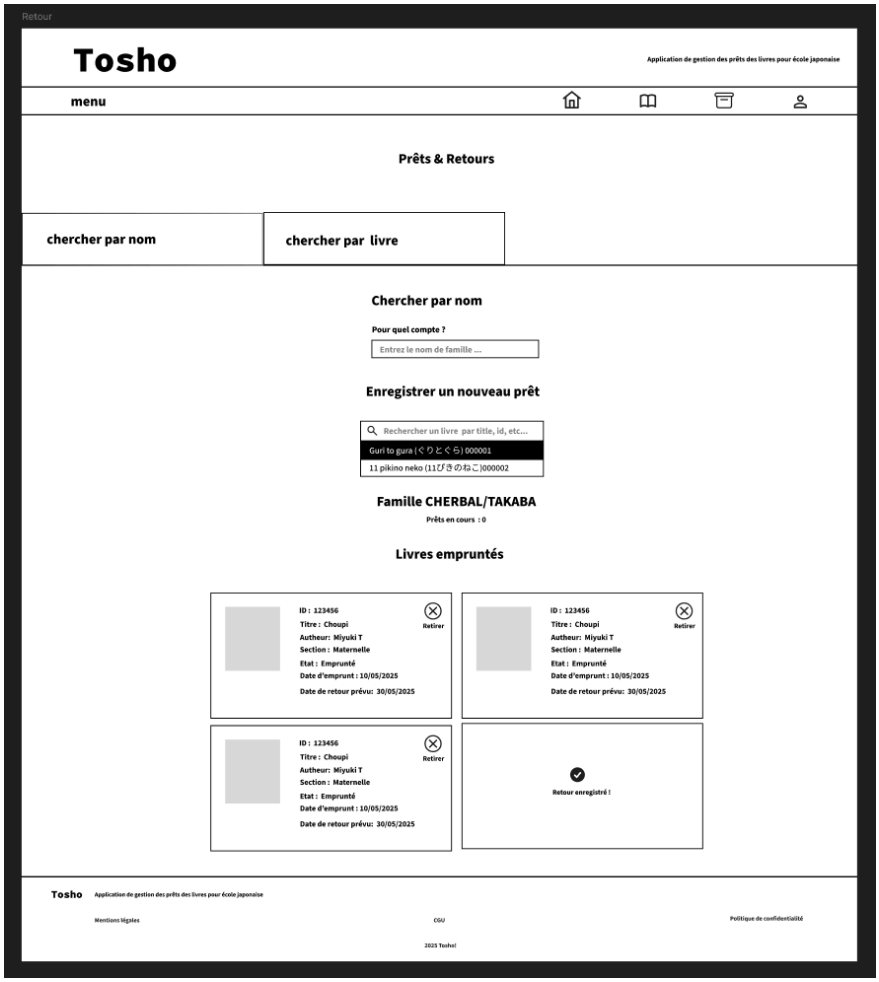
- Les sections principales sont **claires et accessibles**, avec un menu compact pour gagner de la place.
- Les onglets et boutons sont positionnés pour une navigation **facile avec le pouce**.
- Les cartes représentant les livres ou les emprunts sont **empilées verticalement** pour un accès rapide.



Version desktop

- Les éléments sont **bien espacés** et la navigation facile avec les onglets.



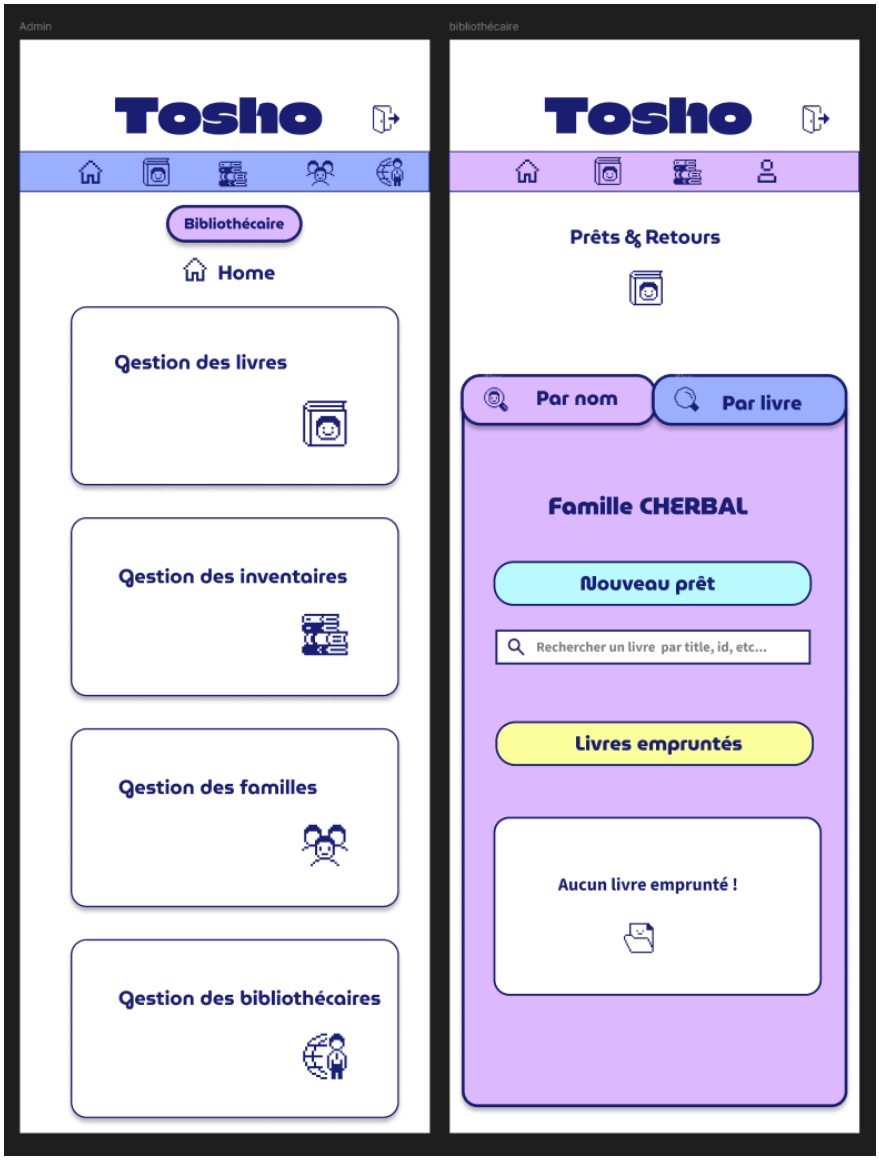


4.3 Maquettes

Les maquettes m'ont permis de visualiser le rendu attendu et de vérifier que l’interface est adaptée aux utilisateurs.
Pour assurer la meilleure expérience utilisateur (**UX**) sur mobile comme sur desktop, j’ai ajusté l’emplacement et la disposition des différents éléments.

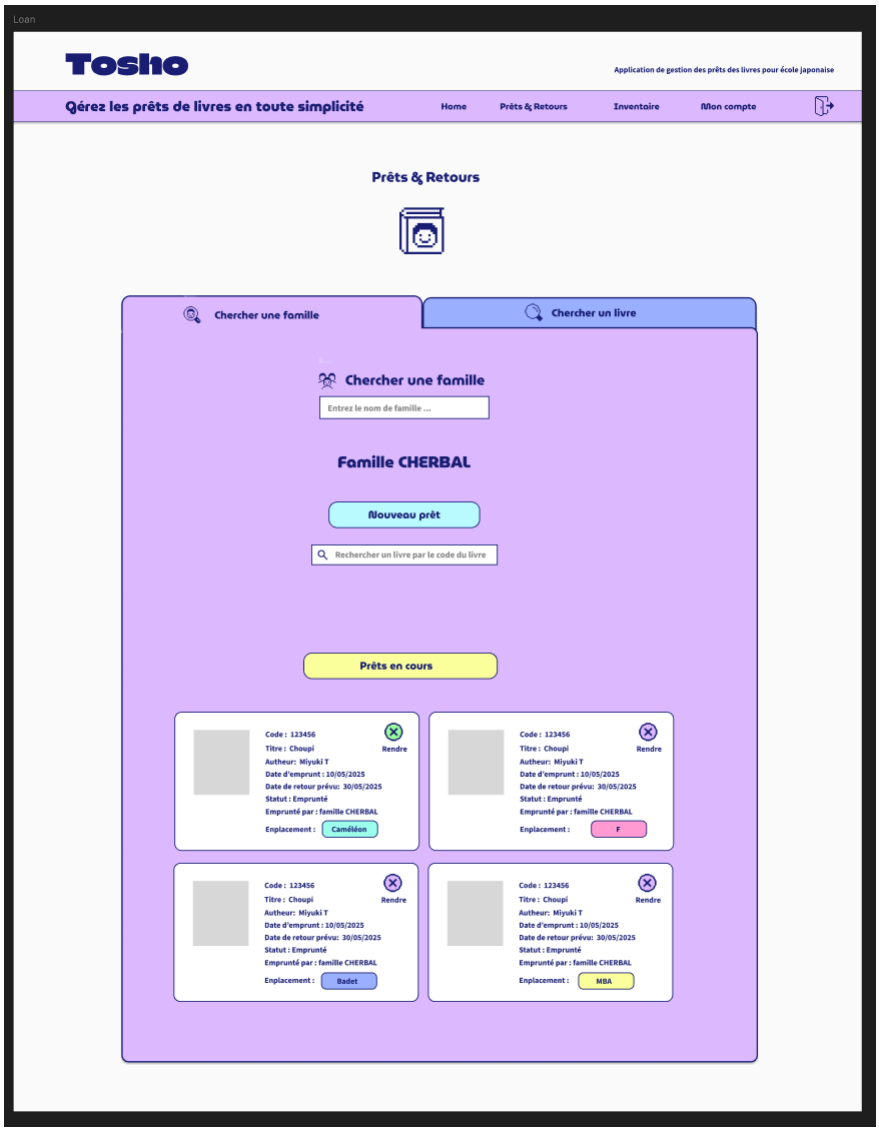
Version mobile

- Le menu utilise des **icônes** pour gagner de la place.
- Les onglets sont adaptés à la **taille de l'écran**.
- Les cartes de livres ou d'emprunts sont affichées les unes au-dessus des autres pour faciliter la lecture et rendre la navigation plus fluide.



Version desktop

- Des **effets hover** sont ajoutés sur le menu et les cartes pour améliorer l'interactivité et guider l'utilisateur.
- Les onglets et sections restent bien visibles et accessibles pour une navigation intuitive.



5. Conception technique

5.1 Technologies utilisées

Front-end

- **HTML** : J'ai structuré le code avec des balises sémantiques comme `<header>`, `<nav>`, `<main>` et `<footer>` afin d'assurer une bonne organisation du contenu. Pour rendre l'application **responsive**, j'ai ajouté la balise suivante :

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

indispensable pour adapter l'affichage aux différentes tailles d'écran.

- **CSS** : Les fichiers CSS sont séparés par composants pour une meilleure organisation. J'ai également créé un fichier spécifique pour les **variables CSS** (couleurs, tailles, polices) afin d'assurer une **cohérence visuelle** et de pouvoir modifier facilement le style global du site. J'ai utilisé `@media screen` pour adapter le design aux différentes tailles d'écran.
- **Twig** : J'ai utilisé Twig, le moteur de template de Symfony, pour créer des pages dynamiques. Il permet de séparer le code PHP de l'affichage et de réutiliser facilement des éléments comme le `<header>`, le `<footer>` ou les onglets (tabs) sur toutes les pages.
- **JavaScript** : J'ai utilisé JavaScript pour ajouter des interactions dynamiques à l'application, notamment la saisie automatique (préremplissage) du formulaire d'ajoute des livres, ainsi que l'activation et la désactivation des comptes en temps réel, sans recharger la page.

Back-end

- Language : **PHP 8.2**
- Framework : **Symfony 6.4**
- SGBD : **MySQL**

```
PS C:\Users\99MFOVA-PP019-W\Documents\Mon projet\Tosho\Tosho> php -v
PHP 8.2.12 (cli) (built: Oct 24 2023 21:15:15) (ZTS Visual C++ 2019 x64)
Copyright (c) The PHP Group
Zend Engine v4.2.12, Copyright (c) Zend Technologies
PS C:\Users\99MFOVA-PP019-W\Documents\Mon projet\Tosho\Tosho> symfony console --version
Symfony 6.4.22 (env: dev, debug: true)
```

Le choix de **Symfony 6.4** permet de bénéficier du **Long-Term Support (LTS)**. Cela signifie qu'elle bénéficie de mises à jour de sécurité et de corrections de bugs pendant plusieurs années, ce qui rend

le projet plus stable et durable.

Symfony facilite la **gestion du back-end** grâce à ses nombreux outils intégrés :

- Gestion de la base de données relationnelle et des entités avec **Doctrine ORM (Object Relational Mapping)**
- création et validation des formulaires,
- gestion de la sécurité et des rôles utilisateurs.

Bundles et composants Symfony

Symfony repose sur des **composants** et des **bundles**, qui permettent de structurer et de réutiliser facilement le code dans l'application.

- Les **composants Symfony** sont des **bibliothèques PHP indépendantes**. Chaque composant remplit une tâche technique précise.
  Exemples :
  - **HttpFoundation** : gère les requêtes et réponses HTTP
  - **Routing** : gère le système de routes
  - **Form** : gère la création et la validation des formulaires
  - **Security** : gère la sécurité et l'authentification
- Les **bundles** sont des ensembles de composants configurés pour ajouter une fonctionnalité complète à une application Symfony. Certains bundles sont intégrés dans le framework et prêts à l'emploi.
  Exemples :
  - **TwigBundle** : permet de gérer les vues avec Twig,
  - **DoctrineBundle** : gère la base de données via Doctrine ORM,
  - **SecurityBundle** : s'occupe de l'authentification et des rôles utilisateurs,
  - **SymfonyCastsResetPasswordBundle** : facilite la mise en place de la fonctionnalité de réinitialisation de mot de passe des utilisateurs.

5.2 Versionning

La sauvegarde et le suivi du code sont assurés par **Git**, avec un dépôt distant sur **GitHub**.

J'ai organisé le développement avec plusieurs branches :

- `dev` : utilisée pour le développement
- `cybercite` : utilisée pendant la période de stage depuis l'ordinateur fourni par l'entreprise

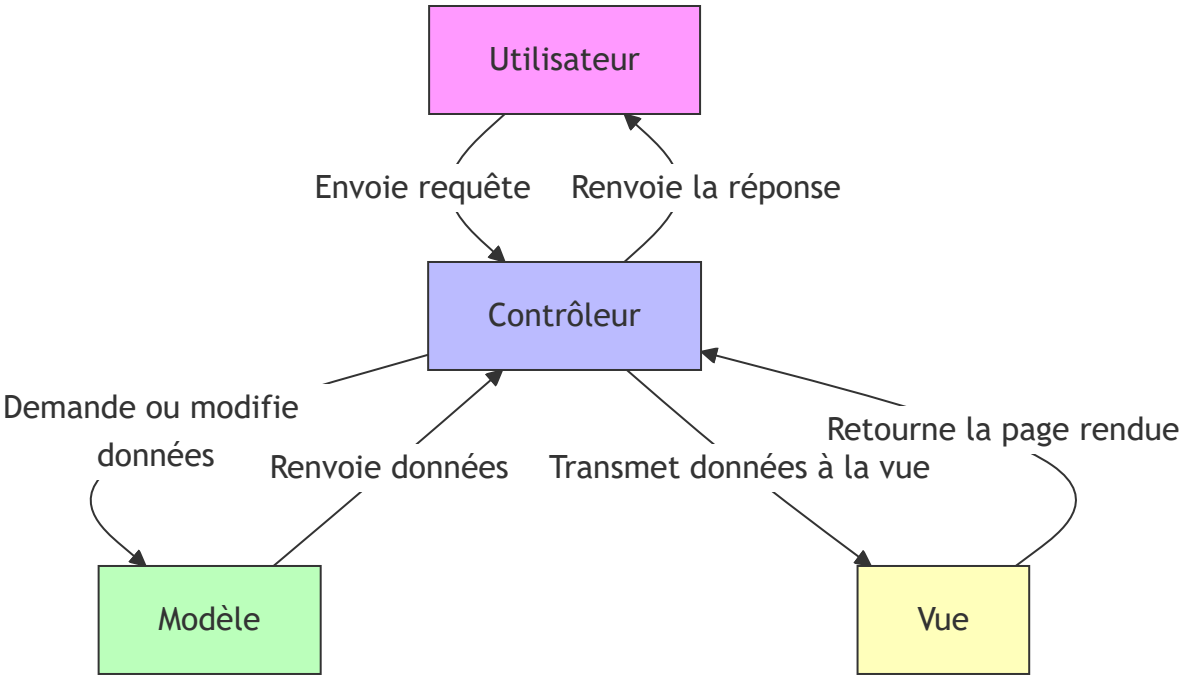
- `docker-deploy` : dédiée au déploiement via **Docker**, contenant les fichiers et configurations de production.

Cette organisation permet de séparer clairement le travail de développement local des configurations et fichiers liés au déploiement.

5.3 Architecture MVC

L'application suit le modèle **MVC (Model – View – Controller)** propre à Symfony, qui sépare clairement les responsabilités :

- **Controller (Contrôleur)** : reçoit les requêtes de l'utilisateur, exécute la logique métier et envoie les données vers la vue correspondante.
- **Model (Modèle)** : gère les entités et communique avec la base de données via **Doctrine ORM**, puis renvoie les données au contrôleur.
- **View (Vue)** : reçoit les données du contrôleur et génère l'affichage des pages avec **Twig**.



5.4 Base de données relationnelle

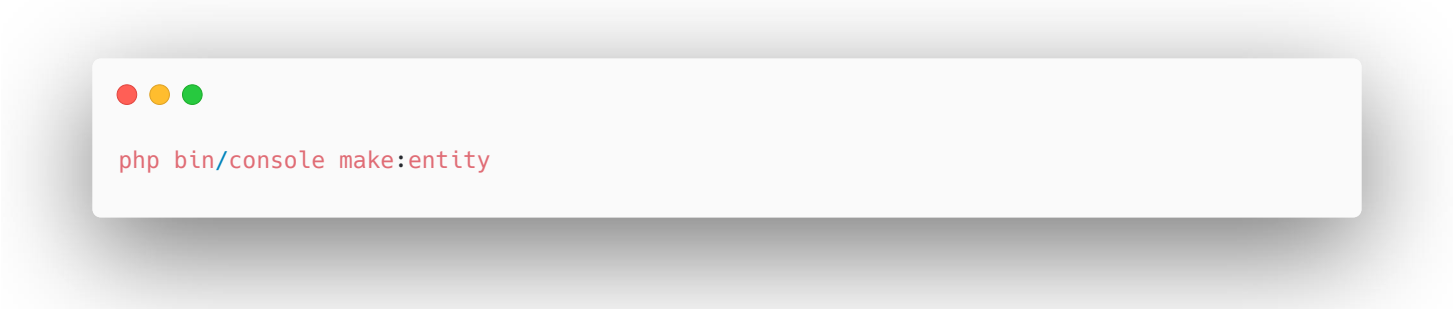
Conception

J'ai d'abord conçu ma base de données sur papier, en définissant les tables et leurs relations afin de répondre aux besoins fonctionnels de l'application.

Ensuite, j'ai présenté le schéma à mon formateur, puis j'ai ajusté la structure en applicant ses retours afin d'améliorer la cohérence et la logique du modèle.
Cette étape m'a permis de comprendre le concept de relations entre les tables (OneToMany, ManyToOne, etc.) et de définir précisément les tables ainsi que leurs relations, avant de passer à la modélisation dans Symfony.

Modélisation et relations

Après avoir conçu le schéma de la base de données sur papier et validé sa cohérence avec mon formateur, j'ai traduit ce modèle en **entités Symfony**.
Cette étape permet de transformer les tables et leurs relations en **classes PHP**, avec Doctrine qui gère automatiquement la création des tables et des clés étrangères.
J'ai créé ces entités grâce à la commande CLI de Symfony :



Cette commande permet de définir :

- le nom de l'entité
- les **propriétés** (les champs de la table)
- les **relations** avec d'autres entités.

Dans **Field type**, on peut définir la relation entre les entités. **Doctrine** s'occupe ensuite d'ajuster les champs pour gérer correctement les relations.

Entités créées

Entité	Description
`User`	Contient les informations des utilisateurs et leurs **rôles** pour gérer les droits d'accès (admin, bibliothécaire).
`Family`	Représente les familles adhérentes à l'association. Stocke les informations de contact et les liens avec `Loan` .

Entité	Description
`Book`	Contient les informations des livres (titre, auteur, statut, code, localisation, etc.) et les relations avec `Loan` et `InventoryItem`.
`Loan`	Représente un prêt de livre : lien entre un livre et une famille.
`Inventory`	Représente une session d'inventaire.
`InventoryItem`	Relie un livre à une session d'inventaire et permet de signaler les anomalies (perdu, mal rangé, non trouvé, autre).

Un inventoryItem correspond à un livre précis dans une session d'inventaire.
Chaque inventoryItem contient :

- le livre (`Book`)
- la session d'inventaire (`Inventory`)
- son statut : par exemple « vérifié », « mal rangé », « non trouvé »
- une note éventuelle pour préciser le problème
- la date de création et de modification
- la ou les personnes qui ont validé ou contrôlé ce livre dans la session

En clair, chaque ligne d'inventoryItem = un livre vérifié dans une session, avec son état et les informations associées.

Cardinalités

Entité source	Type de relation	Entité cible	Description
`Family`	**OneToMany**	`Loan`	Une famille peut avoir plusieurs prêts, mais chaque prêt correspond à une seule famille.
`Loan`	**ManyToOne**	`Book`	Un prêt correspond à un seul livre, mais un livre peut être emprunté plusieurs fois.
`User`	**OneToMany**	`Loan`	Un utilisateur (bibliothécaire) peut gérer plusieurs prêts.
`InventoryItem`	**ManyToOne**	`Book`	Chaque item d'inventaire est lié à un seul livre.

Entité source	Type de relation	Entité cible	Description
`InventoryItem`	**ManyToOne**	`Inventory`	Chaque item d’inventaire appartient à une seule session d’inventaire.

Générer la base de données relationnelle

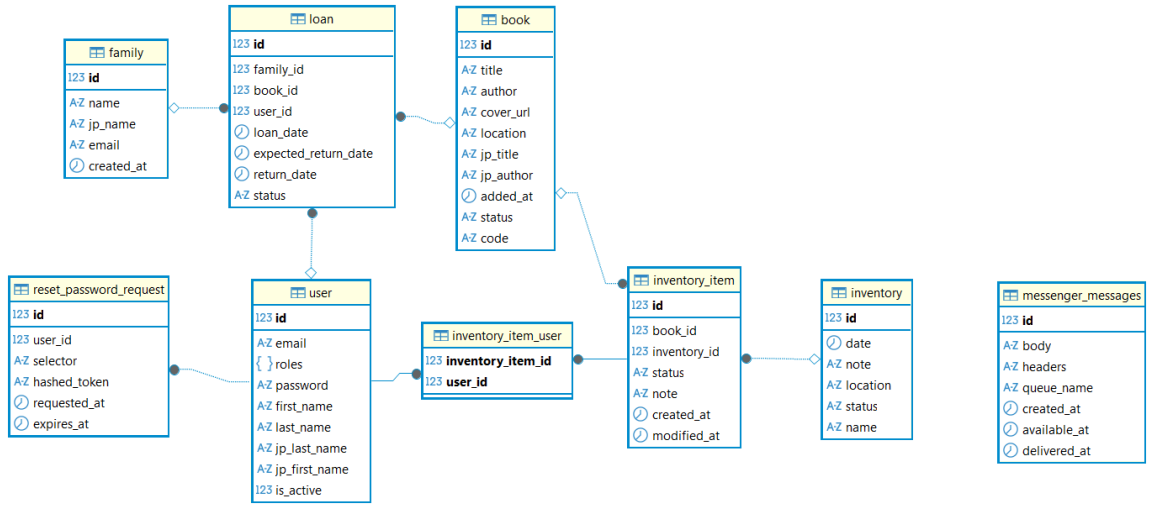
Une fois les entités créées, Symfony et Doctrine permettent de générer automatiquement la base de données.
Pour créer la base de données :

```
php bin/console doctrine:database:create
```

Pour créer ou mettre à jour les tables selon les entités :

```
php bin/console doctrine:schema:update --force
```

Ces commandes synchronisent la base de données avec le modèle défini dans le code.



5.5 Sécurité

Symfony intègre un système de sécurité complet et modulaire qui permet de protéger les données, de contrôler les accès et de garantir la fiabilité de Tosho.

Gestion des rôles et autorisations

Les accès sont gérés à travers des rôles utilisateurs définis dans le fichier `security.yaml` . L'application distingue notamment :

- `ROLE_ADMIN` : accès complet à l'ensemble des fonctionnalités (gestion des bibliothécaires, inventaires, et livres etc.)
- `ROLE_LIBRARIAN` : accès restreint à la gestion des prêts et à la consultation du catalogue.

Ces rôles permettent d'adapter les permissions selon le profil et les responsabilités de chaque utilisateur.

Une fois que l'utilisateur s'est connecté, avec `AuthenticationSuccessHandler` , dirige vers la page d'accueil selon le rôle.

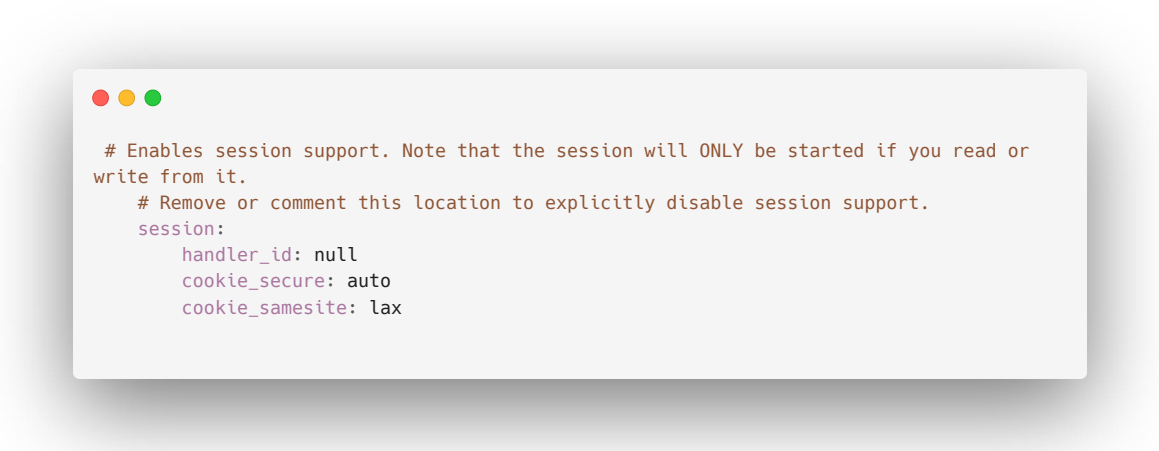
```php
class LoginSuccessHandler implements AuthenticationSuccessHandlerInterface
{
    private RouterInterface $router;

    public function __construct(RouterInterface $router)
    {
        $this->router = $router;
    }

    public function onAuthenticationSuccess(Request $request, TokenInterface $token):
RedirectResponse
    {
        $roles = $token->getRoleNames();

        if (in_array('ROLE_ADMIN', $roles, true)) {
            return new RedirectResponse($this->router->generate('admin_home'));
        }

        if (in_array('ROLE_LIBRARIEN', $roles, true)) {
            return new RedirectResponse($this->router->generate('home'));
        }

        return new RedirectResponse($this->router->generate('login'));
    }
}
```

En complément, un fichier `UserChecker.php` vérifie, avant la connexion, si le compte utilisateur est toujours actif.

Si le compte d'un bibliothécaire a été désactivé par un administrateur, le `UserChecker` empêche la connexion et bloque l'accès à l'application.
Cela permet d'éviter qu'un ancien bénévol puisse encore se connecter et renforce ainsi la sécurité du système.

Authentification et sessions sécurisées

L'authentification est gérée automatiquement par Symfony à l'aide de son composant Security.
Une fois connecté, l'utilisateur est identifié par une session sécurisée stockée côté serveur.
Cela évite d'avoir à se reconnecter à chaque requête, tout en garantissant que les informations d'accès restent protégées.



framework.yaml

Protection CSRF et sécurité des requêtes

Symfony protège automatiquement les formulaires grâce à un **token CSRF (Cross-Site Request Forgery)**.
Ce jeton est généré et vérifié à chaque soumission de formulaire pour s'assurer que la requête vient bien d'un utilisateur authentifié du site, et non d'une attaque externe.

Dans mon projet, j'ai créé les formulaires en utilisant `AbstractType` et en définissant des `FormType` .
Symfony ajoute automatiquement un `token CSRF` à chaque formulaire, ce qui empêche qu'une action soit effectuée par un utilisateur non autorisé. Cela garantit la sécurité des formulaires sans effort supplémentaire.

```php
<?php

namespace App\Form\Family;

use App\Entity\Family;
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\Extension\Core\Type\EmailType;
use Symfony\Component\Form\Extension\Core\Type\CollectionType;

class FamilyForm extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('name', TextType::class, [
                'label' => 'Nom de famille : '
            ])
            ->add('jpName', TextType::class, [
                'label' => 'Nom de famille en japonais : '
            ])
            ->add('email', EmailType::class, [
                'label' => 'Email : '
            ]);
    }

    public function configureOptions(OptionsResolver $resolver): void
    {
        $resolver->setDefaults([
            'data_class' => Family::class,
        ]);
    }
}
```

En complément, Symfony et Doctrine utilisent des **requêtes préparées** pour communiquer avec la base de données.
Cela signifie que les données saisies par les utilisateurs ne sont jamais injectées directement dans les requêtes SQL, ce qui protège efficacement contre les injections SQL malveillantes.

Par exemple, dans le FamilyRepository :

```php
<?php

namespace App\Repository;

use App\Entity\Family;
use Doctrine\Bundle\DoctrineBundle\Repository\ServiceEntityRepository;
use Doctrine\Persistence\ManagerRegistry;
use Knp\Component\Pager\PaginatorInterface;

/**
 * @extends ServiceEntityRepository<Family>
 */
class FamilyRepository extends ServiceEntityRepository
{
    public function __construct(ManagerRegistry $registry)
    {
        parent::__construct($registry, Family::class);
    }
    public function findAllByName(string $name)
    {
        $qb = $this->createQueryBuilder('f');
        $qb
            ->andWhere('f.name LIKE :name')
            ->orWhere('f.jpName LIKE :name')
            ->setParameter('name', "%" . $name . "%");
        return $qb->getQuery()->getResult();
    }

    public function findOneById(int $familyId)
    {
        $qb = $this->createQueryBuilder('family');
        $qb
            ->andWhere('family.id = :familyId')
            ->setParameter('familyId', $familyId);
        return $qb->getQuery()->getOneOrNullResult();
    }
}
```

Ici, l'utilisation de `createQueryBuilder` avec `setParameter` garantit que les données saisies par l'utilisateur sont sécurisées. Les valeurs ne sont pas injectées directement dans la requête SQL : **la requête est préparée séparément** et `setParameter` permet d'y lier les valeurs de manière sécurisée. Cela protège efficacement **contre les injections SQL**.

Sécurisation des mots de passe

Les mots de passe ne sont jamais stockés en clair dans la base de données.
Symfony utilise un algorithme de hachage robuste afin de rendre les mots de passe illisibles.
Lorsqu'un utilisateur se connecte, le mot de passe saisi est haché et comparé à celui enregistré, sans jamais révéler sa valeur réelle.

Filtrage des accès

L’accès à certaines pages ou fonctionnalités est restreint selon le rôle de l’utilisateur :

Dans le code, la méthode `isGranted()` est utilisée pour limiter les actions selon le rôle.

Dans les vues Twig, `{% if is_granted('ROLE_ADMIN') %}` permet d’afficher certains éléments uniquement aux administrateucd cs.

Ce contrôle garantit que chaque utilisateur n’a accès qu’aux informations et fonctionnalités qui le concernent.

6. Développement

6.1 Front-end

Mise en place des onglets

Pour améliorer la fluidité de la navigation, j’ai mis en place un **système d’onglets** permettant de basculer facilement entre deux fonctionnalités, sans recharger la page entière.

Sur l’interface administrateur, chaque module (livres, familles, inventaires, etc.) dispose de deux formulaires distincts :

- **Formulaire de recherche**
- **Formulaire d’ajout**

De même, sur l’interface bibliothécaire, la page **Prêts & Retours** propose deux modes de recherche :

- **Par famille**
- **Par livre**

La page **Inventaire** utilise également ce principe :

- **Onglet suivi de l’inventaire**
- **Onglet ajout d’un livre à la session**

Pour rendre cette navigation fluide, j’ai utilisé des **onglets dynamiques** avec Twig et CSS.

L’extrait suivant montre la structure principale de ce système d’onglet :

```twig
{% extends 'base.html.twig' %}

{% block body %}
    <div class="page-title">
        <h2>Prêts & Retours</h2>
        <img src="{{ asset('img/child-book.svg') }}" alt="book">
    </div>

    <div class="tab-container">
        <div class="tabs">
            <a href="{{ path('loan', {tab: 'family'}) }}" class="tab librarien-tab left-tab family-tab {{ tab == 'family' ? 'active' : '' }}"><div class="tab-icon"></div><span class="tab-title">Par famille</span></a>
            <a href="{{ path('loan', {tab: 'book'}) }}" class="tab librarien-tab right-tab book-tab {{ tab == 'book' ? 'active' : '' }}"><div class="tab-icon"></div><span class="tab-title">Par livre</span></a>
        </div>

        <div class="tab-content librarien-tab left-tab family-tab {{ tab == 'family' ? 'active' : '' }}">
            {% include 'loan/search-family.html.twig'  %}
        </div>

        <div class="tab-content librarien-tab right-tab book-tab {{ tab == 'book' ? 'active' : '' }}">
            {% include 'loan/search-book.html.twig' %}
        </div>
    </div>
{% endblock %}
```

Lorsque l’utilisateur clique sur un onglet, une **classe CSS** `active` est appliquée à l’élément sélectionné, ce qui affiche automatiquement le contenu correspondant (le formulaire de recherche par famille ou par livre).
Un effet **hover** a également été ajouté.

```css
/* --- Contenu des onglets --- */
.tab-content {
  display: none; /* caché par défaut */
  padding: 3rem;
  border: 2px solid;
  border-top: none;
  border-radius: 0 0 12px 12px;
  background: #fff;
}

/* --- Quand un onglet est actif --- */
.tab.active {
  border-bottom: none; /* supprime la bordure inférieure pour se fondre avec le contenu
*/
}

.tab-content.active {
  display: block; /* affiche le contenu lié à l'onglet actif */
}

/* --- Effet au survol --- */
.tab.librarien-tab:hover, .tab.admin-tab:hover {
  background-color: var(--color-btn-hover);
}

/* --- Empêche le changement de couleur au survol si l'onglet est déjà actif --- */
.tab.librarien-tab.left-tab.active:hover {
  background-color: var(--color-primary);
}
.tab.librarien-tab.right-tab.active:hover {
  background-color: var(--color-admin);
```

Côté **contrôleur**, le paramètre `tab` est récupéré dans la requête pour déterminer quel contenu afficher :

```php
$tab = $request->query->get('tab', 'family');
```

Cela permet d'afficher dynamiquement le bon onglet après une recherche ou un rafraîchissement de page.

Interaction front-end : saisie automatique via ISBN

L'objectif de cette fonctionnalité est de simplifier l'ajout d'un nouveau livre dans le catalogue. Lorsqu'un utilisateur saisit le code ISBN (International Standard Book Number), le formulaire se

préremplit automatiquement avec les informations correspondantes (titre, auteur, etc.). Cela permet de gagner du temps et d'éviter les erreurs de saisie.

Installation de `StimulusBundle`

Stimulus est un framework JavaScript léger qui permet d'ajouter des comportements interactifs sans transformer tout mon appli en **SPA** (Single Page Application).

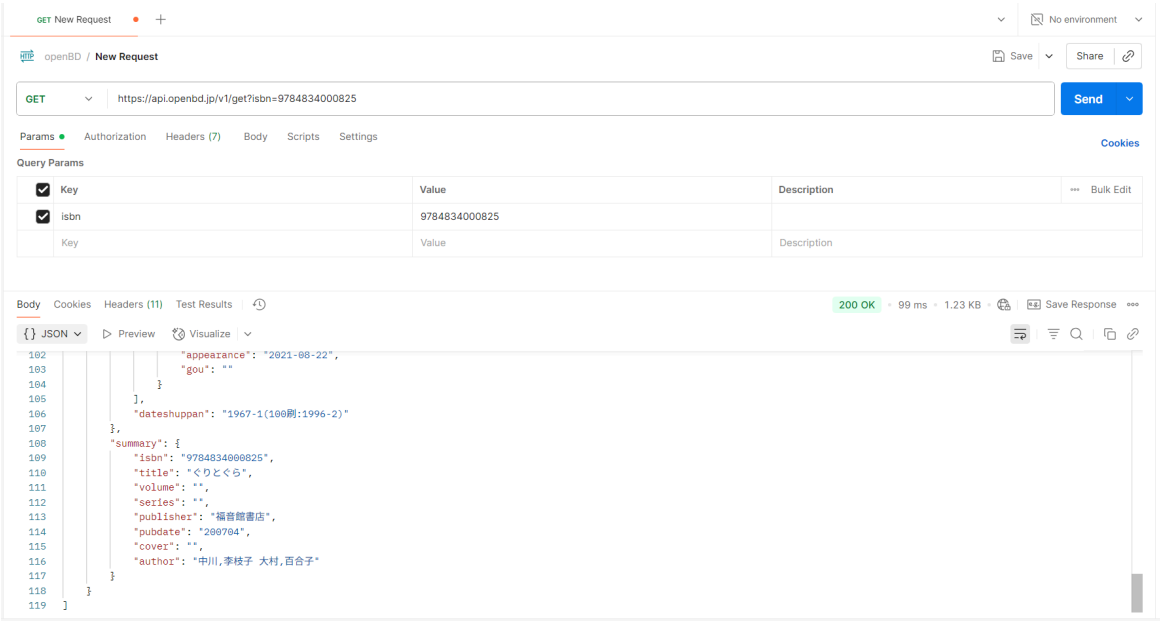
Sur les conseils de mon formateur, je me suis documenté sur le `StimulusBundle` de Symfony, qui facilite son intégration dans une application web Symfony.
J'ai ensuite installé le bundle via `Composer`, ce qui a généré un fichier `hello_controller.js` dans le dossier assets, que j'ai ensuite renommé en `isbn_controller.js`.

API : OpenBD et OpenLibrary

Tous les livres de la bibliothèque sont en japonais. L'affichage de certaines informations, comme le titre et les auteurs, doit donc être disponible à la fois en japonais et en romaji (transcription en alphabet latin) afin de faciliter la lecture pour tous les utilisateurs.

Pour récupérer les informations en japonais, j'ai utilisé l'API gratuite ***OpenBD***. Cette API japonaise permet d'obtenir les données des livres publiés au Japon via ISBN.

Avant de l'intégrer, j'ai testé la réponse de l'API via **Postman** :

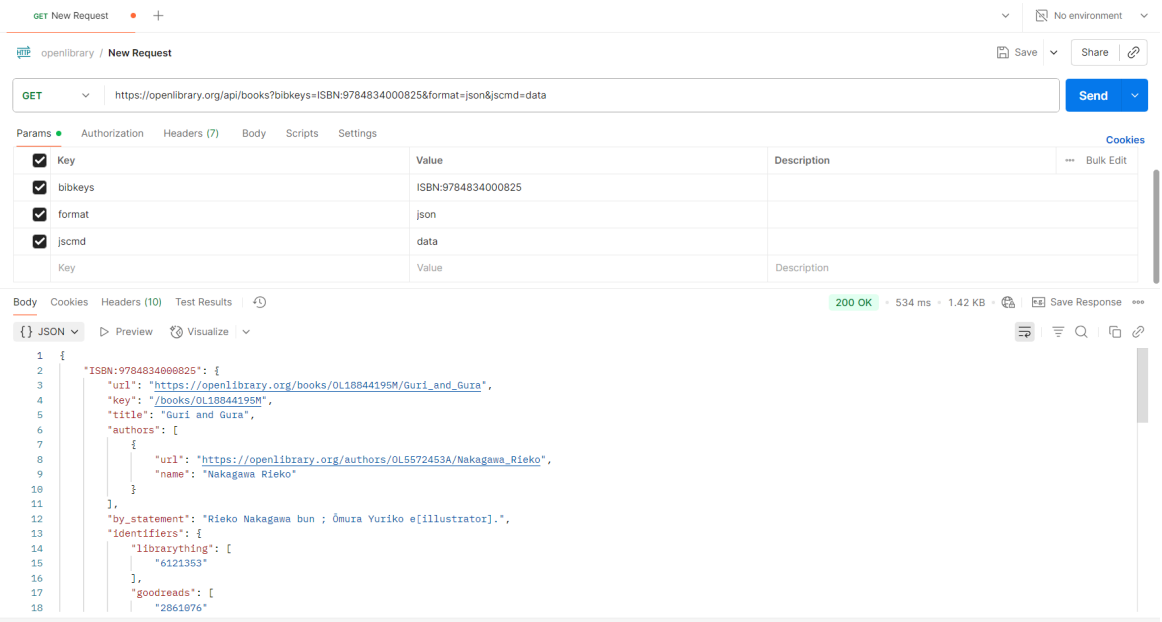


Voici la partie du contrôleur Stimulus dédiée à la récupération des données en japonais :

```
export default class extends Controller {
  async getBookInfoJp(e) {
    e.preventDefault();
    const isbn = document.querySelector("#isbn_search").value.trim();
    if (!isbn) {
      alert("Veuillez saisir un ISBN");
      return;
    }

    const openBdUrl = `https://api.openbd.jp/v1/get?isbn=${isbn}`;
    try {
      const response = await fetch(openBdUrl);

      if (!response.ok) {
        throw new Error(`Erreur HTTP: ${response.status}`);
      }
      const data = await response.json();
      const jpBook = data[0].summary;

      document.querySelector("#book_form_jpTitle").value = jpBook.title;
      document.querySelector("#book_form_jpAuthor").value = jpBook.author;
    } catch (error) {
      console.error(
        "Erreur lors de la récupération des informations du livre :",
        error
      );
    }
  }
```

Pour obtenir les informations en romaji ainsi que l'URL de la couverture du livre, j'ai utilisé l'API **OpenLibrary**.

J'ai également vérifié les données accessibles via **Postman** :



Une fois les tests validés, j'ai implémenté la fonction correspondante dans le même contrôleur :

```
async getBookInfoByISBN(e) {
    e.preventDefault();
    const isbn = document.querySelector("#isbn_search").value.trim();
    const url = `https://openlibrary.org/api/books?
bibkeys=ISBN:${isbn}&format=json&jscmd=data`;

    try {
      const response = await fetch(url);
      if (!response.ok) {
        throw new Error(`Erreur HTTP: ${response.status}`);
      }
      const data = await response.json();

      // On récupère les données du livre à partir de l'ISBN
      const bookKey = `ISBN:${isbn}`;
      const bookInfo = data[bookKey];

      if (!bookInfo) {
        console.log("Aucun livre trouvé avec cet ISBN.");
        return;
      }

      // Extraction des informations utiles
      const title = bookInfo.title || "";
      const authors = bookInfo.authors
        ? bookInfo.authors.map((author) => author.name).join(", ")
        : "";
      const coverUrl = bookInfo.cover.small

      // Affichage des informations
      document.querySelector("#book_form_title").value = title;
      document.querySelector("#book_form_author").value = authors;
      document.querySelector("#book_form_coverUrl").value = coverUrl;
      document.querySelector("img.book-cover").src = coverUrl;
      document.querySelector("img.book-cover").alt = title;

      return {
        title,
        authors,
      };
    } catch (error) {
      console.error(
        "Erreur lors de la récupération des informations du livre :",
        error
      );
    }
  }
```

Ce contrôleur est ensuite lié au formulaire d'ajout de livre à l'aide des attributs `data-controller` et `data-action` dans le fichier `Twig`, ce qui permet de déclencher automatiquement la méthode JavaScript.

```
<div data-controller="isbn">
    <form>
        <label class="isbn-label" for="isbn">Pré-remplir le formulaire par un code
ISBN</label>
        <input type="text" placeholder="ISBN" id="isbn_search" name="isbn" class="search-
input">
        <button data-action="click->isbn#getBookInfoJp isbn#getBookInfoByISBN"
type="button" id="btn_search">Chercher</button>
    </form>
</div>
```

Ainsi, la méthode est déclenchée automatiquement lorsqu'un utilisateur saisit un ISBN et valide la recherche.

6.2 Back-end

Logique métier de la fonctionnalité « prêt de livres »

Pour la gestion du prêt de livres, les règles suivantes sont appliquées :

- Un livre ne peut être prêté que s'il est disponible.
- Lorsqu'un livre est prêté, son **statut** passe à « emprunté » ; lorsqu'il est retourné, il redevient « disponible ».
- **Chaque prêt est lié au livre et à la famille** emprunteuse, et le statut du prêt est mis à jour en conséquence.

Ces règles garantissent que les informations sur les livres et les prêts restent cohérentes dans l'application.

Mettre en place des Enum

Pour gérer les statuts des livres et des prêts, j'ai mis en place des **Enums**.

`BookStatusEnum : available , borrowed`

`LoanStatusEnum : inProgress , returned , overdue`

L'utilisation des Enums permet de minimiser les fautes de frappe lors de l'implémentation et d'assurer la cohérence des statuts dans toute l'application.

LoanController

Le `LoanController` gère la logique principale des prêts.

Sur la route `'loan'`, si la requête est de type **POST**, le contrôleur détermine l'action à effectuer selon les données envoyées par l'utilisateur :

- Si `family_name` ou `keyword` est fourni, on utilise les méthodes personnalisées `findAllByName()` ou `findAllWithFilterQuery()` pour récupérer les résultats.
- Une fois qu'une famille ou un livre est sélectionné, l'utilisateur est redirigé vers la route `'show-book'` ou `'loan-by-family'` avec l'ID correspondant. (**Read**)
- Si `book_code` est fourni, on cherche le livre correspondant avec la méthode personnalisée `findOneByCode()`. Si le livre existe, l'utilisateur est redirigé vers la route `'show-book'` avec l'ID du livre.

```php
#[Route('/', name: 'loan')]
    public function index(
        Request $request,
        FamilyRepository $familyRepository,
        BookRepository $bookRepository
    ): Response {

        $tab = $request->query->get('tab', 'family');
        $books = null;
        $results = null;

        if (!$request->isMethod('POST')) {
            return $this->render('loan/index.html.twig', [
                'tab' => $tab,
                'books' => $books,
                'families' => $results
            ]);
        }

        // chercher par famille
        if ($request->request->has('family_name')) {
            $name = $request->request->get('family_name');
            $results = $familyRepository->findAllByName($name);
            if ($results) {
                return $this->render('loan/index.html.twig', [
                    'families' => $results,
                    'tab' => 'family'
                ]);
            }
            return $this->render('loan/index.html.twig', [
                'noResult' => $name,
                'tab' => 'family'
            ]);
        }

        // chercher par livre avec code
        if ($request->request->has('book_code')) {
            $code = $request->request->get('book_code');
            $book = $bookRepository->findOneByCode($code);
            if ($book) {
                return $this->redirectToRoute('show-book', [
                    'id' => $book->getId()
                ]);
            }
            return $this->render('loan/index.html.twig', [
                'tab' => 'book',
                'notFoundCode' => $code
            ]);
        }

        // chercher par livre avec mot-clé
        if ($request->request->has('keyword')) {
            $keyword = $request->request->get('keyword');
            $books = $bookRepository->findAllWithFilterQuery($keyword);
            if ($books) {
                return $this->render('loan/index.html.twig', [
                    'books' => $books,
                    'tab' => 'book'
                ]);
            }
            return $this->render('loan/index.html.twig', [
                'tab' => 'book',
                'notFound' => $keyword
            ]);
        }

        return new Response('500 Internal Server Error',
Response::HTTP_INTERNAL_SERVER_ERROR);
    }
```

Cette organisation permet de gérer toutes les recherches et redirections depuis la même route, tout en gardant le code clair et maintenable.

Méthodes de recherche dans le Repository

Dans l'exemple ci-dessous, la méthode `findAllWithFilterQuery()` effectue une recherche sur plusieurs champs (title, author, jpTitle, jpAuthor).
L'opérateur `LIKE` permet de récupérer tous les livres dont le titre ou l'auteur contient le mot-clé saisi par l'utilisateur :

```php
public function findAllWithFilterQuery(
        string $keyword
    ) {
        $qb = $this->createQueryBuilder('b');
        $qb
            ->andWhere('b.title LIKE :keyword')
            ->orWhere('b.author LIKE :keyword')
            ->orWhere('b.jpTitle LIKE :keyword')
            ->orWhere('b.jpAuthor LIKE :keyword')
            ->setParameter('keyword', "%" . $keyword . "%")
        ;
        return $qb->getQuery()->getResult();
    }
```

Cette approche permet d'effectuer une recherche partielle et souple sur les titres et auteurs, aussi bien en français qu'en japonais.

ParamConverter

Le ParamConverter de Symfony permet de récupérer directement une entité depuis l'URL.

Par exemple, pour la route `'/loan/book/{id}'`, Symfony injecte automatiquement l'objet Book correspondant, ce qui évite d'écrire :

```php
$book = $bookRepository->find($id);
```

Prêter un livre à une famille

Sur la route `'loan-by-family'`, lorsque l'utilisateur saisit le code du livre à prêter, une vérification du statut est effectuée.

Si le livre est disponible, son statut passe à « emprunté » (**Update**) et un nouveau prêt est enregistré en base de données (**Create**) grâce à l' `EntityManager` .

```php
if ($book->getStatus() !== BookStatusEnum::borrowed) {
            $loan = new Loan();
            $loan->setFamily($family);
            $loan->setBook($book);
            $loan->setStatus(LoanStatusEnum::inProgress);
            $loan->setLoanDate(new \DateTime());
            $loan->setUser($this->getUser());
            $book->setStatus(BookStatusEnum::borrowed);
            $em->persist($loan);
            $em->persist($book);
            $em->flush();

            return $this->redirectToRoute('loan-by-family', [
                'id' => $family->getId(),
            ]);
        }
```

Si le livre est déjà emprunté, un message d'erreur s'affiche.

Retour d'un livre

La méthode `returnBook` gère le retour d'un livre :

```php
    #[Route(path: '/return/{id}', name: 'return-book')]
     public function returnBook(
         Loan $loan,
         EntityManagerInterface $em
     ): Response {
         $book = $loan->getBook();

         if (
             $loan->getStatus() != LoanStatusEnum::returned
             && $book->getStatus() != BookStatusEnum::available
         ) {
             $loan->setStatus(LoanStatusEnum::returned);
             $book->setStatus(BookStatusEnum::available);
             $loan->setReturnDate(new \DateTime());
             $loan->setUser($this->getUser());
             $em->persist($loan);
             $em->persist($book);
             $em->flush();
             return $this->redirectToRoute('loan-by-family', [
                 'id' => $loan->getFamily()->getId()
             ]);
         }
         return new Response('500 Internal Server Error',
Response::HTTP_INTERNAL_SERVER_ERROR);
     }
}
```

Lors du retour d'un livre, seuls les statuts du livre et du prêt sont mis à jour (**Update**) afin de conserver l'historique des emprunts. Aucune donnée n'est supprimée (pas de **Delete** ici).

7. Jeu d'essai - Prêter et rendre un livre

Pour tester le bon fonctionnement du module de prêt, j'ai simulé plusieurs scénarios.

Depuis la page **Prêts & Retours**, le bibliothécaire peut rechercher des prêts :

- **Par famille** : recherche d'une famille par son nom, depuis l'onglet "Par famille".
- **Par livre** : recherche d'un livre par son code ou un mot-clé, depuis l'onglet "Par livre".

Prêter depuis l'onglet "Par famille"

Dans l'onglet Recherche par famille, une fois la famille sélectionnée, **la liste des prêts en cours** s'affiche.

Lorsqu’un bibliothécaire souhaite prêter un nouveau livre à la famille acuelle, il saisit le code du livre dans le champ prévu, puis clique sur **“Prêter”**.

Au niveau de la base de données :

- le statut du livre passe à **emprunté (** `BookStatusEnum::borrowed` **)**
- **un nouveau prêt est créé avec l’ID du livre et celui de la famille**
- le statut du prêt est défini sur **"en cours" (** `LoanStatusEnum::inProgress` **)**.
- **la date de retour prévu est enregistrée automatiquement**.

Si le code saisi correspond à un livre déjà prêté à une autre famille, un **message d’erreur** est affiché.

Au niveau de l’affichage :

Le prêt est automatiquement ajouté à la liste des prêts de cette famille.
L’interface redirige ensuite vers la page de la famille, où **la liste actualisée des prêts s’affiche**.

Rendre depuis l’onglet “Par famille”

Pour enregistrer un retour, le bibliothécaire clique sur **“Rendre”** dans la liste des prêts en cours.

Au niveau de la base de données :

- le statut du livre repasse à **"disponible" (** `BookStatusEnum::available` **)**
- le statut du prêt devient **"rendu" (** `LoanStatusEnum::returned` **)**
- la date de retour est enregistrée automatiquement.

Au niveau de l’affichage :

Les prêts ayant le statut **"rendu"** ne sont plus affichés dans la liste active de la famille, afin de faciliter la gestion des prêts.

Prêter depuis l’onglet “Par livre”

Dans cet onglet, le bibliothécaire peut rechercher un livre par code ou mot-clé.
Une fois le livre trouvé, sa fiche d'information s’affiche avec son statut actuel.

Si le livre est disponible, un bouton **“Prêter ce livre”** apparaît.
Le bibliothécaire choisit alors la famille à qui le livre sera prêté.

Une fois la famille sélectionnée, le prêt est enregistré et l'utilisateur est redirigé automatiquement vers la page de la famille, où **la liste des prêts est mise à jour**.

Rendre depuis l'onglet "Par livre"

Si le livre est actuellement prêté, un bouton **"Rendre"** est visible dans l'en-tête de la fiche du livre. En cliquant dessus, le bibliothécaire enregistre le retour du livre et l'utilisateur est automatiquement redirigé vers la page de la famille concernée avec **la liste des prêts mise à jour**.

8. Déploiement

8.1 Choix de l'environnement et mise en place de Docker

La majeure partie du développement de mon projet s'est déroulée dans un environnement **Windows** avec **XAMPP** comme serveur local. Cependant, au fil de l'avancement, j'ai constaté que cette configuration manquait de performance : le chargement des pages était particulièrement **lent** et XAMPP **manquait de stabilité**.

Durant mon stage, j'ai travaillé dans un environnement **Linux**, et j'ai eu l'occasion d'assister à la mise en production d'un projet avec **Docker**.
Cette expérience m'a motivé à faire évoluer mon propre projet vers un environnement **Ubuntu**, en utilisant **Docker** pour exécuter mon application dans des conteneurs.

Cette nouvelle configuration s'est révélée beaucoup plus rapide, stable et proche d'un environnement de production réel.

L'utilisation de Docker présente plusieurs avantages :

- **Reproductibilité** : tous les développeurs ont le même environnement, indépendant du système hôte.
- **Facilité de déploiement et de maintenance**.

Le passage de **Windows + XAMPP** à **Ubuntu + Docker** a permis d'obtenir un environnement de développement plus fiable, performant et plus proche d'une configuration de production.

8.2 Configuration de Docker

Pour **Docker**, j'ai configuré :

`docker-compose.yml` : définit les services et conteneurs (PHP, MySQL, Nginx…), ainsi que le réseau pour qu'ils puissent communiquer entre eux. Chaque conteneur est réutilisable et peut être reconstruit facilement avec `docker compose up --build`

`Dockerfile` : script exécuté lors de la création du conteneur. Il installe les dépendances, Composer, Symfony CLI, définit le répertoire de travail, copie les fichiers du projet et expose le port.

Cette configuration permet de lancer l'application sur n'importe quelle machine avec Docker.

8.3 Mise en production

Une fois l'environnement Dockerisé testé localement, j'ai déployé l'application sur un **VPS Ubuntu** loué chez RackNerd et associé à un nom de domaine.

J'ai installé **Nginx** comme serveur web pour :

- **Recevoir les requêtes HTTP et HTTPS**
- **Rediriger les requêtes vers les conteneurs Docker qui exécutent l'application Symfony**
- Gérer le **HTTPS avec certificat SSL et redirection automatique vers HTTPS**

Les variables d'environnement sont configurées dans un fichier `.env` , ce qui permet de stocker les informations sensibles (identifiants de base de données, clés API…) hors du code source. Ce fichier est **exclu du dépôt Git** via `.gitignore` pour garantir la sécurité.

8.4 Documentation et prise en main

Pour faciliter la prise en main du projet, j'ai rédigé un `README.md` qui contient :

- Présentation du projet
- Pré-requis pour l'installation
- Étapes pour **exécuter le projet avec Docker**
- **Commandes utiles**

Pour simplifier l'utilisation de Docker, j'ai ajouté un `Makefile` , qui permet de transformer des commandes longues et répétitives en **commandes simples à exécuter**.

9. Difficultés rencontrées

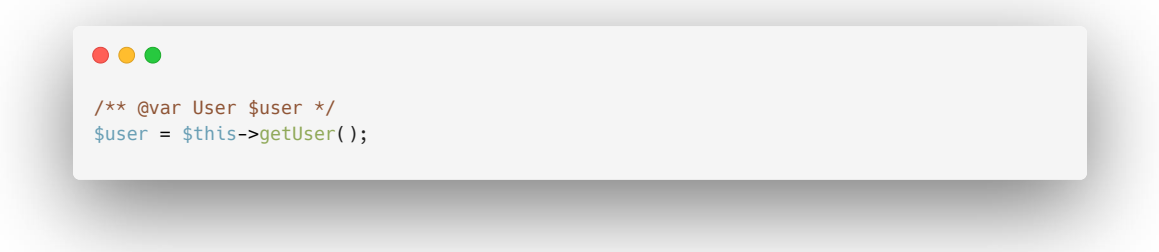
Utilisation de `$this->getUser()`

Lors du développement de la fonctionnalité de changement de mot de passe, j'ai rencontré une difficulté avec la méthode `$this->getUser()` .
Cette méthode permet normalement de récupérer l'utilisateur actuellement connecté dans un contrôleur Symfony.

Cependant, lors de son utilisation, une erreur de typage est apparue : Symfony ne reconnaissait pas automatiquement que l'objet retourné était une instance de ma classe `User` .

Après quelques recherches sur Internet, j'ai trouvé la solution sur *Stack Overflow*.

Il suffisait d'ajouter une annotation de typage explicite avant l'utilisation de la variable :



Lors de ma période de stage, la bonne pratique consistant à typer les variables et les retours de méthode m'a également été expliquée, ce qui m'a aidé à mieux comprendre la logique derrière cette correction.

Activation / Désactivation du compte bibliothécaire

Lors de l'ajout de la fonctionnalité permettant d'activer ou de désactiver le compte d'un bibliothécaire, j'ai rencontré une difficulté liée à **l'interaction entre le front-end et le back-end en temps réel**.

Dans l'interface d'admin, chaque bibliothécaire dispose d'un bouton permettant de changer son statut de compte (*Activé* ou *Désactivé*).
L'objectif était que le changement s'affiche tout de suite à l'écran, sans recharger la page, et qu'il soit bien enregistré dans la base de données.

```javascript
const activeButtons = document.querySelectorAll(".active-btn");
  // console.log(activeButtons);
  activeButtons.forEach((button) => {
    button.addEventListener("click", async () => {
      const route = button.dataset.href;
        // console.log(route)
      try {
        const response = await fetch(route);
        if (!response.ok) {
          throw new Error(`Response status: ${response.status}`);
        }
        const json = await response.json();
        console.log(json);

        const text = button.textContent;
        if (json.isActive) {
          button.textContent = "Activé";
          button.classList.remove("btn-red");
          button.classList.add("btn-green");
        } else {
          button.textContent = "Désactivé";
          button.classList.remove("btn-green");
          button.classList.add("btn-red");
        }
      } catch (error) {
        console.error(error.message);
      }
    });
  });
```

La solution mise en place consistait à envoyer une **requête asynchrone (fetch) vers le contrôleur Symfony**, qui modifie ensuite le statut de l'utilisateur dans la base de données et renvoie une réponse **JSON**.

```php
#[Route('/change-status/{id}', name: 'change-status')]
    public function change(
        User $user
        EntityManagerInterface $em
    ): JsonResponse {
        $user->setIsActive(!$user->isActive());
        $em->flush();

        return $this->json([
            'isActive' => $user->isActive()
        ]);
    }
```

Le front-end met à jour l'affichage du bouton en fonction du nouveau statut.

Une fois la mise à jour effectuée côté serveur, la réponse est utilisée pour mettre à jour l'affichage du bouton en temps réel.

Lazy loading

Contexte

Dans la page de gestion des inventaires, les administrateurs peuvent consulter l'avancement d'une session d'inventaire en cours. Ils doivent pouvoir visualiser la liste des livres vérifiés ainsi que ceux signalés(mal rangé, livre non trouvé, etc.).

Difficulté

Lorsque l'inventaire est récupéré via le **ParamConverter**, les informations des livres associés aux `inventoryItems` ne sont pas automatiquement chargées.

```php
#[Route(
        '/{id}/items/{page}',
        name: 'admin-items',
        requirements: ['page' => '^(checked|not-ok)$']
    )]
    public function items(
        Inventory $inventory,
        string $page,
        InventoryRepository $inventoryRepository,
        InventoryItemRepository $inventoryItemRepository
    ): Response {
      // besoin de récupérer avec inventoryItems
        $currentInventory = $inventoryRepository->findWithItems($inventory->getId());
        $items = null;
        $notOkItems = null;

        if ($page === 'checked') {
            $items = $inventoryItemRepository->findAllByInventory($inventory);
        }
        if ($page === 'not-ok') {
            $notOkItems = $inventoryItemRepository->
            findAllByInventoryAndNotOkStatus($inventory);
        }
        return $this->render('admin/inventory/index.html.twig', [
            'items' => $items,
            'notOkItems' => $notOkItems,
            'currentInventory' => $currentInventory,
            'tab' => 'search',
            'page' => $page
        ]);
    }
```

Par défaut, Doctrine utilise le lazy loading, ce qui signifie que les relations ne sont chargées que lorsqu'elles sont explicitement utilisées. Ainsi, si l'on accède aux inventoryItems dans Twig sans les avoir préchargés, leur collection reste vide. Ici,
en faisant `dd($inventory);` , le résultat montre que inventoryItems est vide :

```
InventoryController.php on line 105:
App\Entity\Inventory {#714 ▼
  -id: 11
  -status: App\Enum\InventoryStatusEnum {#700 ▶}
  -date: DateTime @1764547200 {#708 ▶}
  -note: null
  -location: App\Enum\LocationEnum {#711 ▶}
  -inventoryItems: Doctrine\ORM\PersistentCollection {#754 ▼
    #collection: Doctrine\Common\Collections\ArrayCollection {#753 ▼
      -elements: []
    }
    #initialized: false
    -snapshot: []
    -owner: App\Entity\Inventory {#714}
    -association: Doctrine\ORM\Mapping\OneToManyAssociationMapping {#637 …}
    -backRefFieldName: "inventory"
    -isDirty: false
    -em: Container6ruIr5J\EntityManagerGhostEbeb667 {#239 …12}
    -typeClass: Doctrine\ORM\Mapping\ClassMetadata {#716 …}
  }
  -name: "Inventaire badet 2025"
}
```

Solution

Pour résoudre ce problème, j'ai créé une requête personnalisée dans le InventoryRepository afin de récupérer l'inventaire avec tous ses items et les livres associés en une seule requête :

```php
public function findWithItems($id)
{
    $qb = $this->createQueryBuilder('i');
    $qb
        ->addSelect('ii')
        ->addSelect('book')
        ->leftJoin('i.inventoryItems', 'ii')
        ->leftJoin('ii.book', 'book')
        ->where('i.id = :id')
        ->setParameter('id', $id)
    ;
    return $qb->getQuery()->getOneOrNullResult();
}
```

Après cette modification, inventoryItems contient bien tous les éléments et les livres associés :

```
InventoryController.php on line 106:
App\Entity\Inventory {#714 ▼
  -id: 11
  -status: App\Enum\InventoryStatusEnum {#700 ▶}
  -date: DateTime @1764547200 {#708 ▶}
  -note: null
  -location: App\Enum\LocationEnum {#711 ▶}
  -inventoryItems: Doctrine\ORM\PersistentCollection {#754 ▼
    #collection: Doctrine\Common\Collections\ArrayCollection {#753 ▼
      -elements: array:1 [▼
        0 => App\Entity\InventoryItem {#1045 ▼
          -id: 32
          -book: App\Entity\Book {#1039 ▼
            #id: 41
            -title: "Shirokuma-chan no Birthday"
            -author: "Ken Wakayama"
            -coverUrl: null
            -location: App\Enum\LocationEnum {#711 ▶}
            -loans: Doctrine\ORM\PersistentCollection {#1037 ▶}
            -jpTitle: "しろくまちゃんの誕生日"
            -jpAuthor: "ワカヤマ ケン"
            -addedAt: DateTimeImmutable @1762858200 {#1050 ▶}
            -inventoryItems: Doctrine\ORM\PersistentCollection {#1034 ▶}
            -status: App\Enum\BookStatusEnum {#1048 ▶}
            -code: "0011"
          }
          -inventory: App\Entity\Inventory {#714}
          -status: App\Enum\InventoryItemStatusEnum {#947 ▼
            +name: "badLocation"
            +value: "Mal rangé"
          }
          -note: null
          -createdAt: DateTimeImmutable @1762951815 {#1051 ▶}
          -modifiedAt: null
          -user: Doctrine\ORM\PersistentCollection {#1042 ▶}
        }
      ]
    }
    ...
}
```

10. Veille technologique

Tout au long de ma formation, je me suis documenté et informé pour progresser, résoudre des problèmes techniques et me tenir à jour sur les évolutions dans le domaine du développement web.

La documentation officielle de PHP a été une ressource que j'ai beaucoup consultée. Celle de **Symfony**, très bien structurée et accompagnée d'exemples concrets, m'a également été d'une grande aide, notamment pour la mise en place des formulaires et la configuration des routes.

Lors de bugs ou de difficultés techniques, j'ai effectué des recherches approfondies sur le web. **Stack Overflow** a été l'une de mes principales ressources : je faisais toujours **attention à la date des réponses** pour m'assurer de leur pertinence avec les versions récentes des outils que j'utilisais.

Côté front-end, le site **MDN Web Docs** a été ma principale ressource, très utile pour approfondir ma compréhension de HTML, CSS et JavaScript.
Pour la conception visuelle de mon application, j'ai souvent consulté le site **Dribbble**, qui m'a permis de m'inspirer de designs modernes et de me tenir informé des tendances actuelles en UI/UX.

Quand un bug persistait malgré mes recherches, j'utilisais ChatGPT comme solution de dernier recours. Cela m'a permis de gagner du temps et de débloquer des situations complexes, grâce à des explications claires et des exemples de code adaptés à mon problème.

J'ai également regardé de nombreuses vidéos sur YouTube pour approfondir certains sujets, notamment l'utilisation de Git, ainsi que pour enrichir ma culture générale dans le domaine du développement.

Ces ressources m'ont également permis de m'habituer à lire et comprendre **la documentation en anglais**, qui est souvent plus complète et mise à jour.

11. Documentation en anglais

11.1 Contexte

Lors de ma période de stage, j'ai eu l'occasion de observer des **revues de code via GitLab**. Cela m'a permis de comprendre l'importance d'écrire un **code propre et lisible** (clean code).

J'ai reçu des retours sur mon projet Tosho, et mon tuteur m'a parlé de la pratique de **"Early Return"**. Dans mon code initial, j'avais imbriqué plusieurs conditions `if` et `else` , ce qui rendait le code difficile à lire.

`LoanController` avant :

```php
#[Route(path: '/loan-book/{id}', name: 'loan-book')]
    public function loanBook(
        Book $book,
        Request $request,
        FamilyRepository $familyRepository
    ): Response {
        $results = null;
        if ($request->isMethod('POST')) {
            // chercher par famille
            if ($request->request->has('loan_family')) {
                $name = $request->request->get('loan_family');
                $results = $familyRepository->findAllByName($name);
                if ($results) {
                    return $this->render('loan/index.html.twig', [
                        'searchedFamilies' => $results,
                        'bookToLoan' => $book,
                        'tab' => 'book'
                    ]);
                } else {
                    echo('aucune famille trouvé !');
                }
            }
        }
        return $this->render('loan/index.html.twig', [
            'bookToLoan' => $book,
            'searchedFamilies' => $results,
            'tab' => 'book',
       ]);
    }
```

`LoanController` après :

```php
#[Route(path: '/loan-book/{id}', name: 'loan-book')]
    public function loanBook(
        Book $book,
        Request $request,
        FamilyRepository $familyRepository
    ): Response {
        $results = null;

        if (!$request->isMethod('POST')) {
            return $this->render('loan/index.html.twig', [
                'bookToLoan' => $book,
                'searchedFamilies' => $results,
                'tab' => 'book',
            ]);
        }

        $name = $request->request->get('loan_family');
        $results = $familyRepository->findAllByName($name);

        if (!$results) {
            return $this->render('loan/index.html.twig', [
                'noResult' => $name,
                'bookToLoan' => $book,
                'tab' => 'book'
            ]);
        }

        if ($results) {
            return $this->render('loan/index.html.twig', [
                'searchedFamilies' => $results,
                'bookToLoan' => $book,
                'tab' => 'book'
            ]);
        }
        return new Response('500 Internal Server Error',
 Response::HTTP_INTERNAL_SERVER_ERROR);
    }
```

La pratique de **“Early Return”** consiste à **quitter une fonction dès qu'une condition est remplie**, afin de réduire l'imbrication. Après cette explication, je me suis documenté sur ce sujet pour mieux l'appliquer dans mon projet.

11.2 Early Return vs. Classic If-Else: A Universal Pattern for Writing Cleaner Code

Writing conditional logic is something every developer does—no matter the language. But how you structure those conditions affects how readable, testable, and maintainable your code becomes.

What Is Early Return or Guard Clause?

Early return means exiting a function as soon as a certain condition is met—usually to handle an edge case or invalid input.

A guard clause is a specific use of early return at the top of the function, to prevent deeper logic from running if key conditions aren’t met.

This avoids unnecessary nesting and keeps your core logic flat and easy to follow.

Example 1: Classic if...else (Nested Logic)

```
function sendWelcomeEmail(user) {
    if (user) {
        console.log(`Sending welcome email to ${user.email}`);
    } else {
        return;
    }
}
```

The real logic is wrapped inside an if block, which can become messy as the function grows.

Example 2: Early Return / Guard Clause

```
function sendWelcomeEmail(user) {
    if (!user) {
        return;
    }

    console.log(`Sending welcome email to ${user.email}`);
}
```

This structure handles the invalid case immediately, then continues with the main logic. It’s easier to read and requires less indentation.

Benefits of Using Guard Clauses :

- Reduces code nesting and cognitive load
- Keeps the core logic visually prioritized
- Handles edge cases early and clearly
- Makes the function easier to modify and extend

This approach works well in any language, because it's a logic structuring choice — not a language feature.

source : Eddie Goldman / Early Return vs. Classic If-Else: A Universal Pattern for Writing Cleaner Code https://dev.to/eddiegoldman/early-return-vs-classic-if-else-a-universal-pattern-for-writing-cleaner-code-1083

11.3 Retour anticipé contre l'If-Else classique : Un modèle universel pour écrire du code plus propre

Ecrire logique conditionnelle est quelquechose que tous les développeurs font dans n'importe quelle language. Cependant, comment structurer ces conditions impacte comment votre code devient lisible, testable, et maintenable.

Qu'est ce que l'Early Return (Retour anticipé) ou la Guard Clause (Clause de garde) ?

L'Early return consiste à sortir d'une fonction dès que certain condition est remplie pour traiter un cas particulier ou une entrée invalide.

Une Guard Clause est une utilisation spécifique de l'Early return au début d'une fonction, pour empêcher une logique plus profonde si une condition clé n'est pas remplie.

Cela évite les imbrications non nécessaires et garder votre logique principale simple et facile à suivre.

Example 1 : if...else classique (logique imbriquée)

> idem à la version originale

La logique réele est enveloppé à l'intérieur d'un bloc de `if` , ce qui peut rapidement alourdir le code quand la fonction devient plus complexe.

Example 2 : Retour anticipé / Clause de garde

> idem à la version originale

Cette structure gère les cas invalides immediatement, en suite continue avec la logique principale. C'est plus facile à lire et demande moins d'indentation.

Avantage d'utiliser la Clause de garde :

- Réduire les code imbriqués et la charge mentale
- Garder la logique principale priorisé visuellement
- Gère les cas particuliers plus tôt et plus claire

- Rendre les fonctions plus facile à modifier et à faire évoluer

Cet approche marche bien dans n'importe quelle language, car c'est une choix de structure logique, et non d'une fonctionnaliré propre à un langage.

12. Conclusion

12.1 Bilan global

Ce projet m’a permis de mettre en pratique mes compétences en développement web et d’apprendre à résoudre les problèmes rencontrés au cours du développement.
Dans ce domaine, il est essentiel de continuer à apprendre, de se tenir à jour et de s’adapter en permanence aux nouvelles technologies.

La période de stage en entreprise m’a également beaucoup apporté : elle m’a permis de découvrir la gestion de projets réels et d’adopter de bonnes pratiques professionnelles, notamment en matière d’utilisation de Git et GitLab, ainsi que de respect des principes du clean code.
J’ai également participé à plusieurs tâches techniques variées, telles que :

- la création d'un composant de **pagination avec React**,
- l’envoi de données issues d’un **fichier CSV vers une API**
- la personnalisation d’un **template Keycloak** (avec lecture de documentation en anglais)
- la création d’un **middleware avec Laravel**
- le développement d’un **composant React** permettant de trier et de modifier l’ordre d’affichage d’un tableau

Enfin, ce parcours de formation m’a offert une vision complète du cycle de développement web, de la conception à la mise en ligne, et m’a permis de renforcer à la fois mes compétences techniques et mon autonomie.

12.2 Roadmap

Évolutions futures de projet Tosho

La responsable actuelle du service IT de l’association japonaise quittera son poste, et je me suis engagé à reprendre cette fonction.

Je prévois donc de proposer officiellement cette application à l’association afin qu’elle soit utilisée pour la gestion réelle de la bibliothèque.

Plusieurs pistes d’évolution sont envisagées pour faire progresser l’application :

- Recherche interactive avec **AJAX** : affichage automatique des suggestions lors de la saisie
- Interface multilingue (français / japonais)
- Mise en place d’un planning pour les parents bibliothécaires
- Envoi d’e-mails automatiques de rappel pour les retours en retard
- Système de réservation de livres en ligne

Projet professionnel

Après cette formation, j’envisage de poursuivre mes études en alternance.
Je souhaite approfondir mes compétences en langages de programmation, en frameworks modernes et découvrir davantage le domaine du DevOps.
La recherche d’une entreprise d’accueil est actuellement en cours.

Au sein de l’association, une refonte du site vitrine est prévue. Le site actuel a été développé avec Vue.js. Une fois que j'aurai repris le poste de responsable IT, je prévois de réaliser ce projet en autonomie.