

Dossier de projet professionnel

# Système informatique de gestion des bibliothèques



Présentée par Morgane Maréchal

## Sommaire

<b>Introduction :</b>	3
<b>Liste des compétences du référentiel couvertes :</b>	3
<b>Spécificités du projet</b>	4
Contexte de création.....	4
Périmètre et cible.....	4
Méthode de travail.....	5
Descriptif fonctionnel.....	8
Arborescence du projet.....	10
<b>Spécification technique et conception :</b>	11
Charte graphique.....	11
Maquette.....	14
Les langages utilisés.....	19
La base de données.....	25
<b>Réalisation</b>	28
Architecture MVC.....	28
Le back-end avec AdonisJS.....	23
Le front-end avec React.....	26
Les tests.....	26
CI/CD.....	42
Sécurisation.....	45
<b>Conclusion</b>	48
<b>ANNEXES</b>	31

# Introduction :

Le projet de système informatique de gestion des bibliothèques est un projet personnel que j'ai réalisé dans le cadre de la formation Concepteur développeur d'application de la formation LaPlateforme. C'est un titre RNCP de niveau VI.

J'ai appelé cette application Khaba harta car en langue ancienne (araméen) cela veut dire "écrit ouvert".

## Liste des compétences du référentiel couvertes :

Ce projet couvre les compétences nécessaires à l'obtention du titre répartie en deux activités:

### Activité 1, "Développer une application sécurisée":

- Installer et configurer son environnement de travail en fonction du projet
- Développer des interfaces utilisateur
- Développer des composants métiers
- Contribuer à la gestion d'un projet informatique

### Activité 2, "Concevoir et développer une application sécurisée organisée en couches":

- Ce projet couvre les compétences :
- Analyser les besoins et maquetter une applications
- Définir l'architecture logicielle d'une applications
- Concevoir et mettre en place une base de données relationnelle
- Développer des composants d'accès aux données SQL

### Activité 3, "Préparer le déploiement d'une application sécurisée"

- Préparer et exécuter les plans de test d'une application
- Préparer et documenter le déploiement d'une application
- Contribuer à la mise en production dans une démarche devops

# Spécificités du projet

## Contexte de création

Ce projet de système informatique de gestion des bibliothèques est un projet personnel. J'en ai eu l'idée car j'aime beaucoup les livres et l'accès à l'information. Il existe assez peu d'application de ce type et j'avais envie de créer la mienne avec des fonctionnalités et un design spécifique.

## Périmètre et cible

Le site est une application de gestion de bibliothèques. La cible regroupe des utilisateurs sur un panel d'âge et de situation sociale assez large. Cela comprend aussi l'utilisation, de la part des différents types d'utilisateurs, de supports différents (téléphones mobiles, ordinateurs, tablettes) qu'il fallait prendre en compte pour le responsive de la solution. Les utilisateurs peuvent aussi bien surfer sur le site et réserver des livres sur leur ordinateur ou sur leur smartphone sans perdre de l'expérience qualité. Ce qui implique une version mobile aussi orientée expérience utilisateur que la version laptop. Il existe deux types de profils, les usagers de bibliothèques classiques qui consultent les ressources (livres disponibles, ...) de la bibliothèque via son portail et les administrateurs qui gèrent les livres, les emprunts, les notices et les comptes utilisateurs. Plus généralement, cette application est destinée à différents organismes (bibliothèques publiques, centre de documentation et d'information scolaire, fonds documentaire d'associations, ...) et à ceux qui fréquentent et s'intéressent aux fonds documentaires de ces organismes.

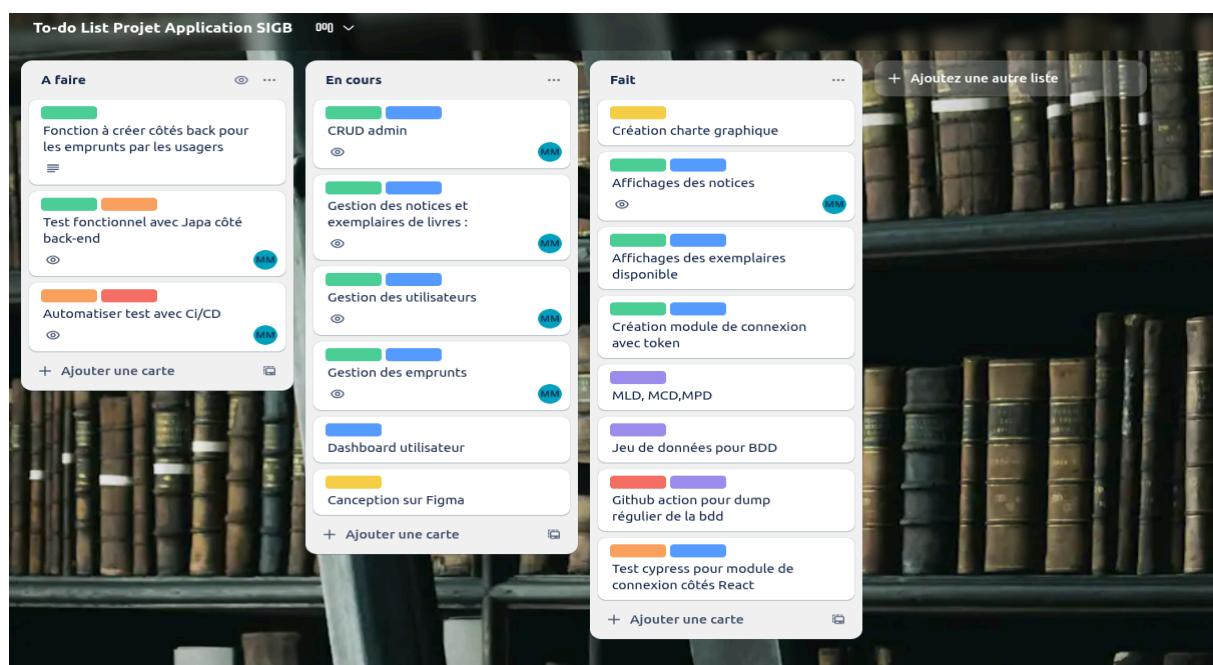
Pour rendre cela possible, j'ai utilisé le concept "mobile first". Mobile First est un concept de Web Design optimisé pour le mobile qui va au-delà du Responsive Web Design. Il consiste à concevoir un site en mettant la priorité sur la version mobile et en adaptant progressivement le web design pour les écrans plus large.

J'ai aussi utilisé une charte graphique et une bibliothèque de composants pour un design agréable.

## Méthode de travail

### Trello pour l'organisation des tâches

J'ai utilisé Trello et son système de tickets pour m'organiser. Trello est un outil de gestion de projet en ligne. Il repose sur une organisation des projets en colonnes avec chacune dedans des étiquettes qui détails des tâches. On peut assigner ces tâches à un développeur dans le cas d'un travail de groupe, et elles sont mobiles d'une planche à l'autre. Je me suis également fixé des objectifs journaliers afin d'avancer sur mon projet et de développer des nouvelles fonctionnalités en continu..



The screenshot shows a digital Kanban board interface. A card titled "Création charte graphique" is selected. The card has a yellow label under "Etiquettes". The "Description" section contains the text "Création de la charte graphique avec choix des couleurs, typo, logo". At the bottom of the card, there are buttons for "Sauvegarder" (Save) and "Annuler" (Cancel). To the right of the card, a sidebar displays a comment from "Morgane MARÉCHAL" with the timestamp "il y a 1 minute". The sidebar also includes a text input field for "Commentaires et activité" (Comments and activity) and a button "Afficher les détails" (View details).

Les tâches sont réparties en trois colonnes, celles qui sont à faire, celles qui sont en cours et celles qui sont terminées. Je peux rédiger des étiquettes pour chaque fonctionnalité dont j'ai besoin, j'ai choisis des codes de couleurs pour visualiser plus généralement le domaine de chaque tâche (front-end, BDD, design, ...).

Je me suis inspirée de la méthode Kanban, une méthode Agile même si cela peut sembler un peu contre-intuitif dans un projet fait seul, étant donné qu'elles sont souvent utilisées dans des équipes. Cependant voici quelques avantages d'appliquer Kanban à un projet solo :

- Gestion visuelle du travail : Kanban est basé sur des cartes placées sur des colonnes, représentant différentes étapes du travail avec une personnalisation possible des couleurs.
- Priorisation des tâches : je dois constamment décider quelles sont les tâches les plus importantes qui sont à la racine de la création de différentes fonctionnalité (comme mettre d'abord à jour la bdd, puis le controller et les routes avant d'attaquer le front-end) et les traiter en premier
- Suivi de la productivité de manière visuelle. Je peux voir combien de tâches sont terminées, en cours, en attente, changer les étiquettes de colonne peut être motivant...
- Flexibilité Si une tâche prend plus de temps que prévu ou si de nouvelles priorités surgissent on peut ajuster le tableau
- Amélioration de la concentration : en découplant le travail en petites tâches gérables et en ayant des objectifs clairs et un tableau de bord visuel, je peux

mieux me concentrer sur une tâche à la fois et ne pas m' éparpiller en pensant à plusieurs tâches en même temps.

- Intégration plus simple de futurs collaborateurs. En effet, si je souhaite travailler avec d'autres personnes, cela permettra d'assigner des tâches à chacun en limitant les risques de conflits entre développements et redondances de fonctionnalités.

Préparation aux développements futurs : avec une approche agile dès le début d'un projet solo permet de se préparer à une gestion de projet qui permet de livrer une version fonctionnelle de votre projet, cela me permet de tester des idées.

### **Figma pour le travail commun sur le design**



Figma est un éditeur de graphiques vectoriels et un outil de prototypage. C'est un outils collaboratif qui permet à plusieurs personnes de travailler sur un même projet

Cela permet de créer des composants et de les organiser en pages.

C'est un outil basé sur le cloud, je peux y accéder depuis n'importe quel appareil, ce qui me permet de travailler où que je sois. J'utilise aussi souvent des composants réutilisables, ce qui me permet de maintenir une cohérence dans mes designs et me permet de travailler de manière fluide et précise. Je peux aussi utiliser, si je le souhaite, des plugins et des bibliothèques de composants (il existe par exemple des menus, des cartes, et des formulaires tout prêts) pour aller plus vite ,même si en général je préfère créer mes propres composants de A à Z.

### **Git et Github pour le versionning**



Github est une plateforme de versionning qui permet de centraliser un projet et d'y participer à plusieurs. C'est un outil puissant qui me permet de faire des branche et de revenir à des versions antérieures de mon projet si besoin.

Le projet a nécessité deux répertoires (back et front clairement séparés):

opac\_server

opac\_web



Git est un logiciel de gestion de version décentralisé. C'est le plus populaire parmi les développeurs. Il permet de communiquer entre la version local de l'application et la version remote qui est centralisée sur github.

Pour chaque feature du projet j'ai créé des branches afin que je puisse travailler sur des éléments différents sans risque de régression en cas d'erreurs. Je réalise régulièrement des commits pour ne pas perdre mon travail en cours. Quand une feature est terminée, j'envoie une pull request sur la branche en remote qui est ensuite validée. Je mets également en place des processus de test et de vérification de code avec Github Action quand je push une branche en remote afin de vérifier qu'il y a bien eu progression dans les features et non pas régression.



#### Github actions pour l'intégration continu

Github Actions , service CI/CD, permet aux développeurs d'automatiser des tâches au sein de leur dépôt GitHub. Par exemple des tests sur différentes parties du code. Les "jobs" effectués par Github action sont fait sur une simulation (virtualisation) du projet. Les GitHub Actions sont basées sur des processus automatisés constitués d'une série d'étapes appelés "Workflow" que l'on décrit dans des fichiers ajoutés dans le projet et qui sont lu par Github Actions.

## Descriptif fonctionnel

Le projet permet de gérer une bibliothèque.

Le design est contemporain et respecte la charte graphique. Le site est responsive.

Le site doit comprendre une barre de recherche avec autocomplétion, ainsi que des filtres par catégorie / sous-catégories sans recharge de page.

Un clic sur chaque notice de livre renvoie à une page d'information complète générée dynamiquement les informations sur le livre et les exemplaires disponibles. L'utilisateur peut se créer un compte avec un module Inscription / Connexion.

Un usager de bibliothèque inscrit peut aussi réserver des livres, faire des commentaires et ajouter des favoris.

L'administrateur possède son propre tableau de bord ce qui permet un suivi des emprunts. Il peut contacter par email les utilisateurs en retard dans les retours de livre et enregistrer les retours de livres. Il peut également assurer un rôle de modération en cas de commentaires problématiques des utilisateurs.

L'administrateur du site peut également créer des notices et enregistrer des exemplaires. Il peut aussi créer d'autres administrateurs.

# Arborescence du projet

## Pour tous les utilisateurs:

- Page d'accueil (home)
- Page de connexion
- Page d'inscription
- Page pour chaque notice

## Pour les personnes ayant un compte:

- Page dashboard (informations de prêt)
- Page mon compte (information de connexion)

Optionnel :

- Page historique
- Module sur les favoris
- Module sur les commentaires postés et le suivi des réponses

## Pour les administrateurs:

- Page administrative - gestion des notices et exemplaires
- Page administrative - gestion des prêts
- Page administrative - gestion des utilisateurs

## Spécification technique et conception :

### Charte graphique

Une charte graphique web est un ensemble de règles et de normes graphiques qui constituent l'identité visuelle d'un site web. La charte graphique web contient des règles d'utilisation de tous les éléments graphiques, et représente donc un élément essentiel du site.

La charte graphique est explicitée ici.

Elle comprend plusieurs éléments :



**Logo** : la charte doit contenir le logo de l'application ainsi que ses variantes, ses références couleurs et ses différentes tailles. Les couleurs du logo reprennent celles existantes dans la charte graphique pour une meilleure harmonie dans l'application.

Le favicon :



### Typographie

Utilisation de la Google font Marriweather  
Titres : Merriweather (serif) → Élégant et lisible.

Bold 700

**Whereas recognition of the inherent dignity**

Bold 700 italic

***Whereas recognition of the inherent dignity***

font-family:  
'Merriweather', serif;  
font-weight: 700;

Corps de texte : Open Sans (sans-serif) → Moderne et accessible.

Éléments UI (créé avec Material UI)

Boutons :

Bordure arrondie (border-radius: 8px).

Ombre portée légère (box-shadow: 0 2px 4px rgba(0,0,0,0.1)).

Cartes de livres empruntés :  
Fond blanc, ombre subtile, bordure 1px solid #E0E0E0.



Carte de gestions des emprunts  
On utilise le même type d'ombre



### Les couleurs : la charte doit contenir les références des couleurs utilisées

Bleu : Évoque la confiance, la connaissance, le sérieux et la concentration (parfait pour une bibliothèque).

Vert : Apporte de la sérénité, lien avec la nature (livres anciens/bois), couleur douce et calme.

Or : Rappelle les tranches de livres dorées, les enluminures, ajoute un peu de luxe.



### La prise en compte de l'accessibilité

Certaines personnes ont une mauvaise perception des couleurs (daltonisme, ...), il est donc important de prendre en compte les contrastes.

Site ressource pour vérifier l'accessibilité des couleurs par rapport au contraste :  
<https://www.skynettetechnologies.com/color-contrast-checker>



## WCAG Compliance Results

**Swap Colors**

SMALL sample text: 14pt (18.5px)  
LARGE sample text: 18pt (24px)

Color Contrast	AA	AAA
<b>6.97</b>		
Pass		
Small Text	Large Text	UI Components

Les boutons et les icônes :

Les boutons et les icônes sont créés en utilisant les éléments déjà existants de React et material UI.

Ci-dessous voici une partie des boutons utilisés.

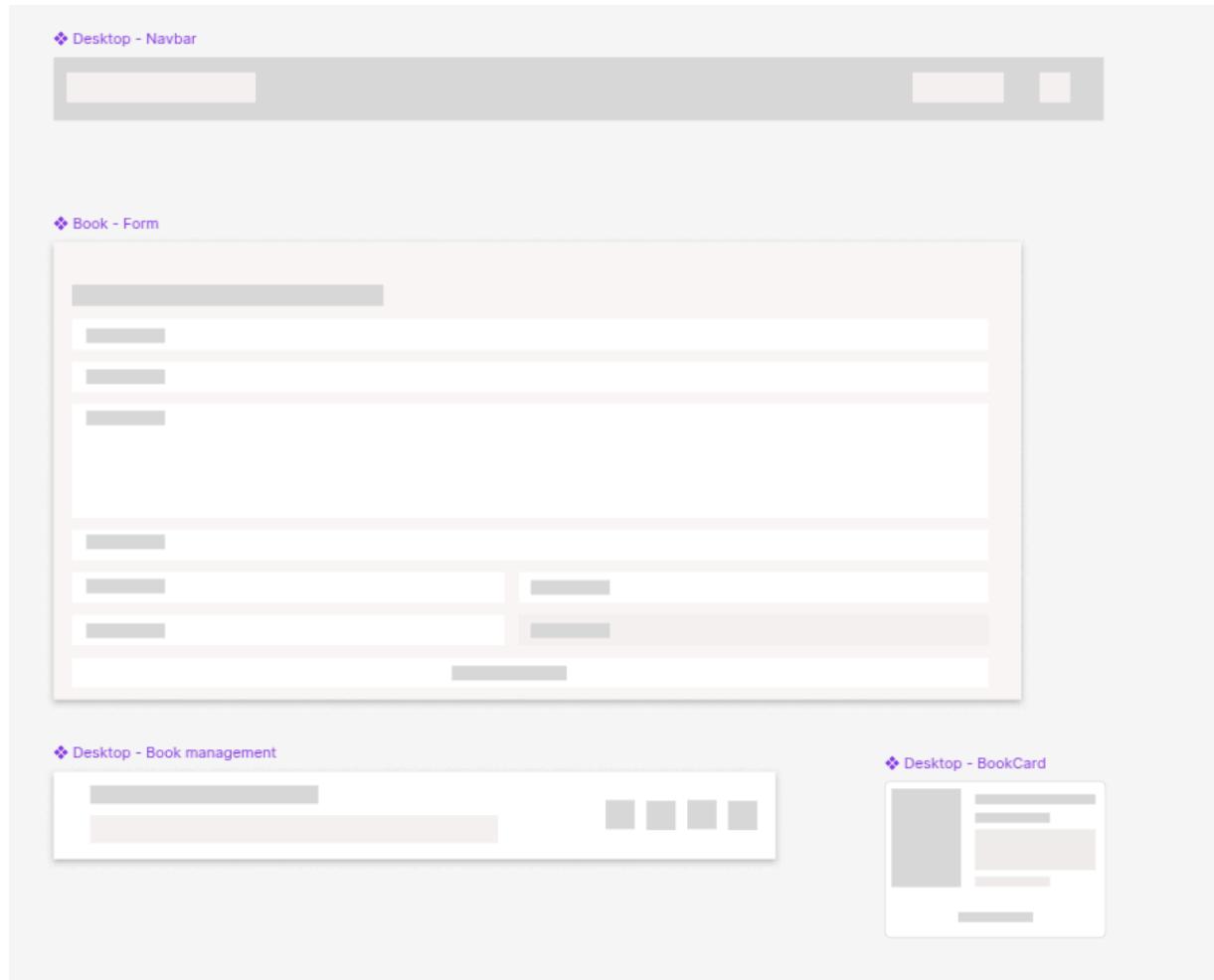
Ce sont des icône de la library React-icon qui sont au couleurs de la charte graphique.



# Maquettage

Maquettes basse fidélité du projet :

Dans un premier temps j'ai fait les composants en basse fidélité



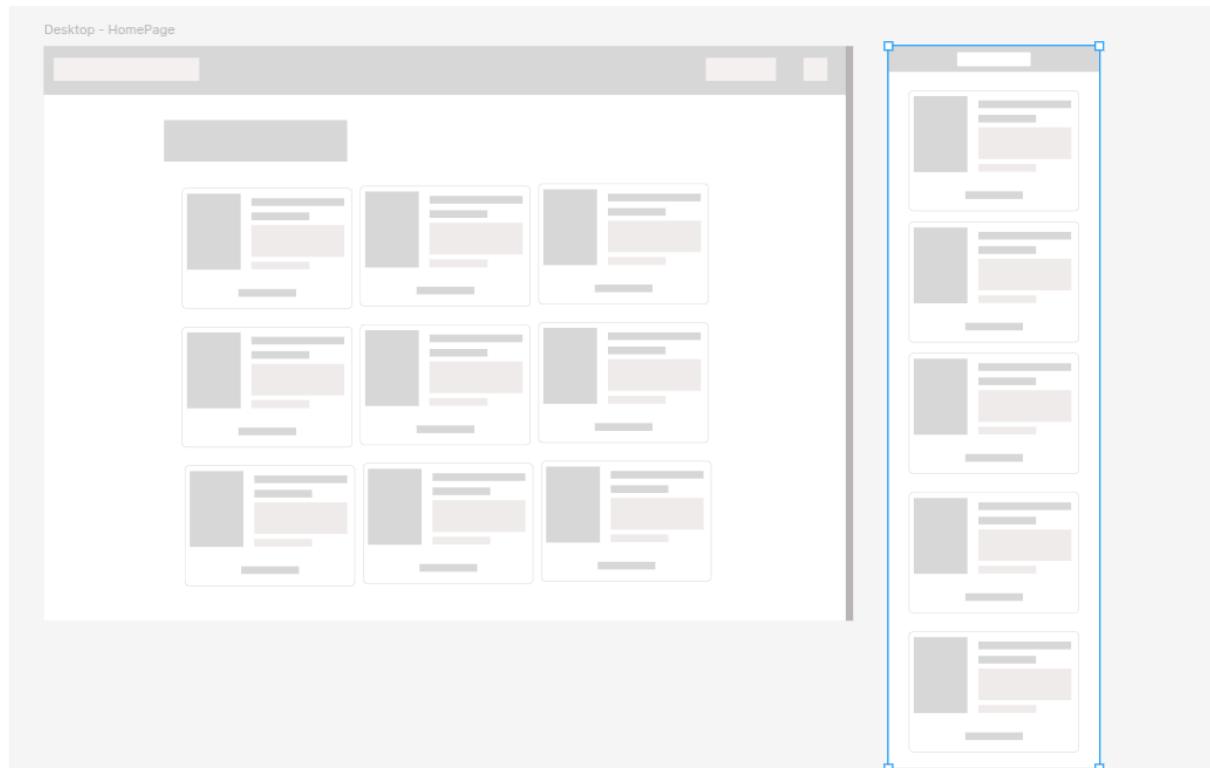
Ensuite je les ai assemblés pour faire le rendu visuel général de l'interface en prenant en compte le fait que React (la library utilisée pour le front-end) utilise une architecture basée sur l'assemblage de composants réutilisables.

Figma et [React.js](#) sont deux outils qui fonctionnent très bien ensemble.

Maquetter des composants sur Figma avant de les développer en React permet de visualiser, tester et valider l'interface avant de coder, ce qui réduit les erreurs de conception avec une vue d'ensemble qui peut être ajuster sans avoir à modifier des

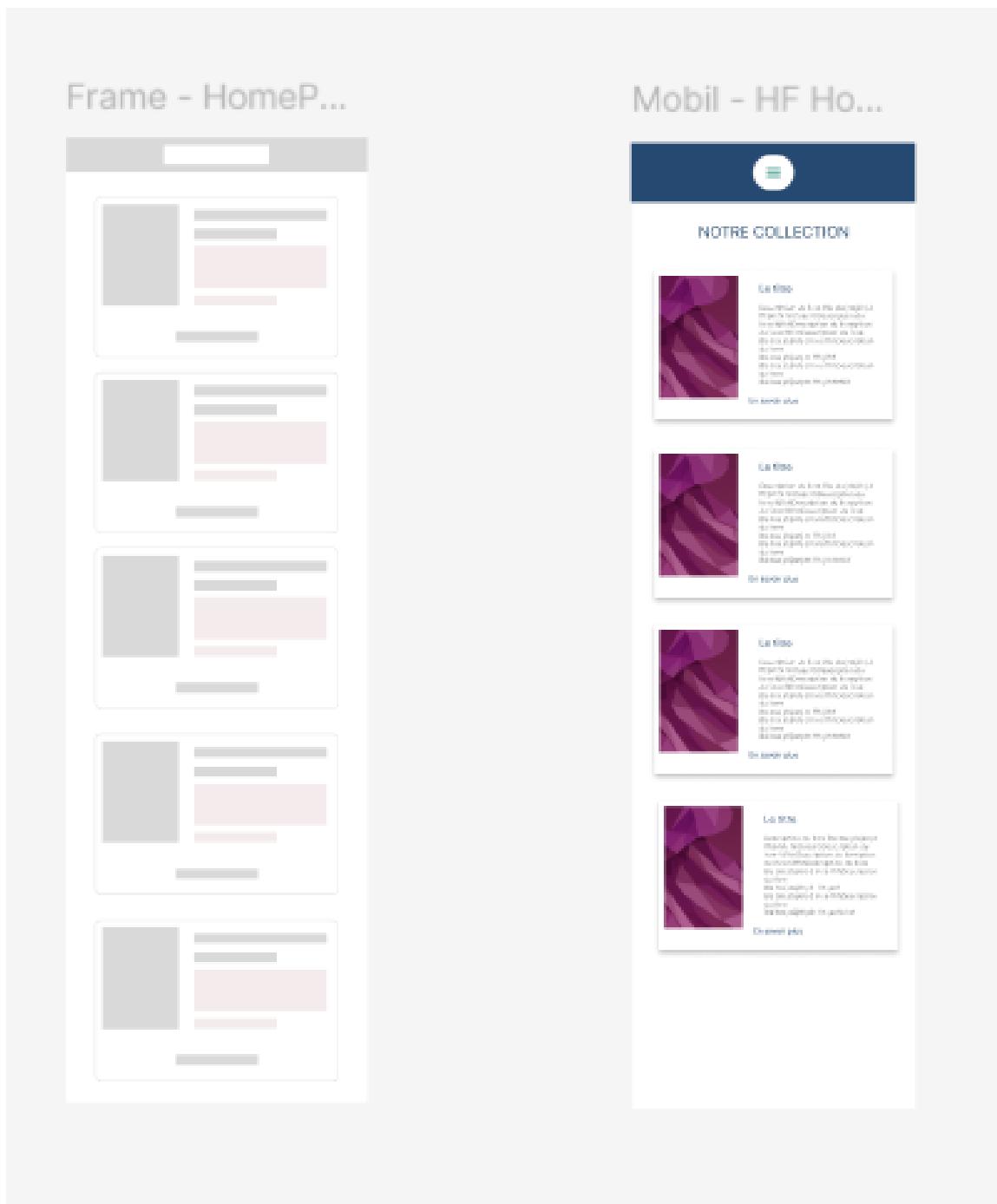
composants.. Il est aussi possible de créer des prototypes interactifs, ce qui permet de simuler un parcours utilisateur et de tester l'UX. Cela facilite (dans le cadre d'une équipe) la communication avec les développeurs et les parties prenantes. Cela permet un gain de temps dans le développement, car les composants sont bien définis visuellement et fonctionnellement. Il y a moins de retours en arrière.

Figma et React peuvent être étroitement liés via des systèmes de design. On peut créer une bibliothèque de composants partagée dans Figma (couleurs, tailles) qui alimente les composants React. Certains plugins ou outils comme Figma-to-Code facilitent même la transition entre le design et le code. Ainsi, Figma sert de source de vérité visuelle, tandis que React implémente le comportement dynamique. Ce lien améliore la cohérence UI/UX. Respecter les étapes même si c'est fastidieux au début permet sur le temps plus long de gagner du temps et de mieux se situer dans l'avancement du projet.





## Passage de basse à haute fidélité



Maquettes haute fidélité du projet :

Ici les composants Navbar et Card (qui affiche le résumé d'un livre)

The screenshot shows a dark blue header bar with the text "NOTRE COLLECTION" on the left and "GESTION" on the right, accompanied by a user icon. Below the header is a white card for a book titled "Le titre". The card features a purple abstract geometric background image on the left. The title "Le titre" is centered above a large block of placeholder text: "Description du livre Bla bla jdskjh d fffhjnfk hnDescriptio...". Below this is the editor information "Edition : Tralala" and a "En savoir plus" button at the bottom.

## Les langages utilisés

Pour ce projet, j'ai utilisé un stack technique qui repose sur Javascript et Typescript. Les données sont en PostgresSql et en Redis (NoSql). Dans les paragraphes suivant je vais expliquer pourquoi j'ai choisi ce stack technique avec ces langages.

### Le front-end

La partie front-end est gérée par la library React et la library graphique MaterialUI. Utiliser React avec Material UI (MUI) m'offre de nombreux avantages pour développer des interfaces web modernes. MUI est une bibliothèque de composants React fondée sur les principes de Material Design de Google, ce qui me permet de construire rapidement des interfaces adaptatives, sans devoir tout coder moi-même. Il suffit d'importer le composant existant qui peut être personnalisé avec du style ()sx.

MUI propose des composants prêts à l'emploi comme des boutons, menus, champs de formulaire ou dialogues, avec les comportements interactifs propres et soignés déjà intégrés (gestion du focus, accessibilité, transitions, etc.). Cela me fait gagner un temps précieux sur ce qui aurait nécessité de nombreuses lignes de codes en css. Même si react possède déjà bon nombre de composants, ceux de MUI me plaisaient un peu plus esthétiquement.

En effet, pour le style et le CSS, MUI me permet d'utiliser la prop sx, les composants stylés ou même le ThemeProvider pour centraliser mes choix de couleurs et typographies.

Voici un thème général à toute l'application grâce aux import ThemeProvider et createTheme de MUI

```
import { createTheme } from '@mui/material/styles';

const theme = createTheme({
  palette: {
    primary: {
      light: '#2A5C8D',
      main: '#2A5C8D', // Bleu Profond
      dark: '#ba000d', //rouge
      contrastText: '#000',
      gold: '#A98E36'
    },
    secondary: {
      main: '#4B8F8C', // Vert sage
    },
    tertiary: {
      main: '#A98E36'
    },
    grey: [
      50: '#fafafa',
      100: '#f5f5f5',
      200: '#e0e0e0'
    ]
  }
});
```

Enfin, grâce à React, je peux écrire le HTML en JSX, un format qui ressemble au HTML mais intégré au JavaScript.

En combinant React et MUI, je crée plus rapidement des applications cohérentes visuellement, tout en gardant un code bien structuré que je peux faire évoluer.

## Le back-end

En back-end j'ai choisi d'utiliser AdonisJS. Utiliser AdonisJS avec TypeScript représente un vrai atout pour développer des applications backend modernes, sécurisées et maintenables. Le typage fort de Typescript permet d'éviter de nombreuses erreurs et force à un code plus rigoureux. C'est également plus claire lors de relectures tardives. AdonisJS est un framework Node.js complet inspiré de Laravel (PHP), et il est pensé dès le départ pour fonctionner avec TypeScript.

Avec AdonisJS, je bénéficie d'une structure de projet claire. Tout est bien organisé : routes, contrôleurs, modèles, middlewares, validateurs, etc. Cela m'aide à garder un code propre et évolutif, ce qui est essentiel dans un projet dont je veux pouvoir étendre les fonctionnalités sans risque de régression.

L'un des gros avantages d'AdonisJS est qu'il intègre de nombreuses fonctionnalités par défaut : système d'authentification, ORM (Lucid), validation, gestion des requêtes HTTP, et bien plus. Je n'ai pas besoin d'ajouter des dizaines de packages externes comme je le ferais avec d'autre framework [node.js](#) comme Express. Il y a déjà une préselection de packages essentiels pour ne pas se perdre dans trop de dépendances et modules node.

Avec TypeScript, je bénéficie en plus d'une meilleure détection d'erreurs à la compilation. L'ORM Lucid est aussi typé, ce qui me permet de manipuler mes modèles avec sécurité. Par exemple, si je change une colonne dans ma base de données, TypeScript m'aide à identifier les parties du code à adapter.

Un autre atout, c'est le CLI d'AdonisJS. Il me permet de générer des fichiers (contrôleurs, modèles, routes...) très rapidement avec les bonnes conventions. Cela me fait gagner du temps et réduit les erreurs humaines.

En résumé, AdonisJS avec TypeScript me permet de développer plus vite une fois bien pris en main, plus proprement, avec un meilleur contrôle sur la qualité du code.

### **Les bases de données :**

J'utilise PostgreSQL avec Adminer, et Redis avec Commander. Voici pourquoi je combine ces deux types de langages de gestion de données Sql et NoSql pour mon projet.

#### **PostgreSQL & Adminer**

PostgreSQL est un système de gestion de base de données relationnelle robuste, open-source, et très utilisé depuis longtemps. Il gère des relations complexes, et propose des fonctionnalités avancées.

Je l'utilise pour stocker des données structurées et critiques : utilisateurs, notices, réservations, etc.

Adminer est un outil web minimaliste pour gérer les bases de données SQL. Plus léger que phpMyAdmin, il me permet d'interroger, modifier, et visualiser facilement mes tables PostgreSQL, directement depuis le navigateur.

Pourquoi je les utilise ensemble : PostgreSQL me garantit performance, fiabilité et flexibilité pour la gestion des données métier, et Adminer me fait gagner du temps pour les consultations rapides, les modifications manuelles, ou les tests SQL, sans me noyer dans des lignes de commande.

### Redis & Commander

Redis est une base de données en mémoire extrêmement rapide, orientée clé-valeur. Je l'utilise pour gérer le cache et les notifications. Redis est idéal quand la rapidité et la légèreté prime sur la persistance.

Redis Commander est une interface web pour visualiser et manipuler les données stockées dans Redis. Il m'aide à naviguer dans les clés, inspecter les valeurs, et tester rapidement mes flux de données.

Pourquoi je les combine : Redis me permet d'optimiser les performances de mon application, et Redis Commander me donne une vue claire et rapide sur les clés stockées, ce qui m'est essentiel pour déboguer, tester ou surveiller le comportement de mon API en temps réel.

**En résumé, je choisis PostgreSQL + Adminer pour la gestion structurée et visuelle des données, et Redis + Commander pour la rapidité et le contrôle sur les données volatiles. Ces couples d'outils me permettent de travailler efficacement en alliant performance, stabilité et ergonomie.**

Autre point important, les éléments liés à mes données sont conteneurisés avec Docker.



Docker est une plateforme qui me permet de créer, déployer et exécuter des applications dans des conteneurs. Un conteneur est un mini-environnement isolé, et léger. Contrairement à une machine virtuelle, un conteneur partage le noyau de l'OS, ce qui le rend très rapide et moins gourmand en ressources.

Quand je travaille avec des bases de données SQL (PostgreSQL) ou NoSQL (Redis), je choisis de les conteneuriser avec Docker pour plusieurs raisons :

L'installation est simplifiée : en une seule commande (docker-compose up), je lance PostgreSQL, Redis, Adminer ou Redis Commander, sans rien configurer manuellement sur ma machine. Il existent de nombreuses images à jours de langages, outils, frameworks, etc. Cela simplifie grandement le travail et l'installation d'outils qui ne seraient utilisés que dans un cadre précis.

Voici un extrait de fichier docker-compose.yml :

```
▷ Run Service
redis:
  image: redis:7-alpine
  container_name: redis
  ports:
    - "6379:6379"
  volumes:
    - redis_data:/data
  restart: always

▷ Run Service
redis-commander:
  image: rediscommander/redis-commander:latest
  container_name: redis-commander
  environment:
    - REDIS_HOSTS=local:redis:6379
  ports:
    - "8081:8081"
  depends_on:
    - redis
  restart: always
```

L'environnement est reproductible : peu importe où je déploie (local, CI/CD, serveur), mes conteneurs fonctionnent de la même façon.

L'isolation : chaque service est indépendant. Si je casse Redis en testant un script, ma base PostgreSQL reste intacte. L'utilisation de plusieurs environnements (comme par exemple MariaDb et Mysql peuvent parfois le faire) ne rentrent pas en conflits les uns avec les autres car il est simple d'arrêter un conteneur, les commandes CLI de docker servent à simplifier ce genre d'actions.

Portabilité : Si une personne veut se joindre au projet, elle n'a qu'à utiliser le même fichier docker-compose.yml que moi pour avoir les mêmes environnements (les mêmes versions, ...). Elle peut lancer tout l'environnement sans rien installer d'autre que Docker.

Gestion simplifiée des interfaces graphiques : Adminer et Redis Commander tournent aussi dans des conteneurs. Je peux y accéder via mon navigateur sans installer d'outil local, ce qui me permet de visualiser et manipuler les données facilement et rapidement. Sur un temps plus long, avec l'accumulation de projets de développement cela m'évite d'avoir une multitude d'outils dont je ne me sers plus installés sur mon ordinateur.

En résumé, je conteneurise mes bases SQL et NoSQL pour gagner en flexibilité, stabilité et rapidité de déploiement. Docker me permet de gérer mon environnement de manière portable et évolutive pour un projet amené à s'étendre.

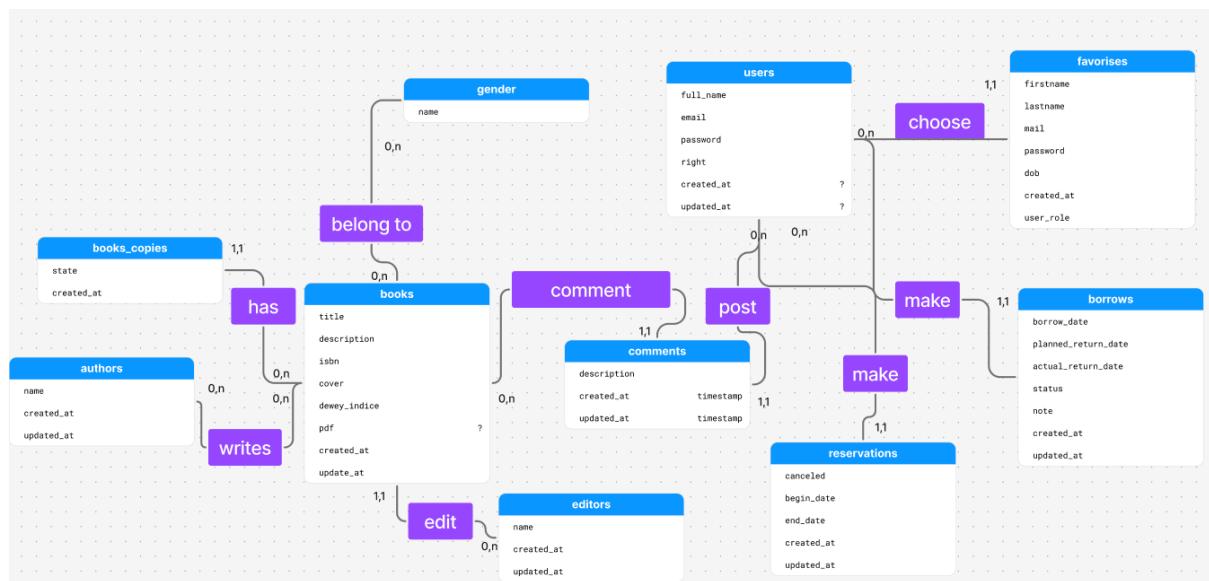
## La base de données

## Base de données : MCD, MLD, MPD

Le MCD (modèle conceptuel de données) fournit une description graphique pour représenter des modèles de données sous la forme de diagrammes pouvant contenir des entités ou des associations. Il peut être utilisé pour décrire les besoins en information ou par exemple le genre d'information nécessaire à l'élaboration du cahier des charges.

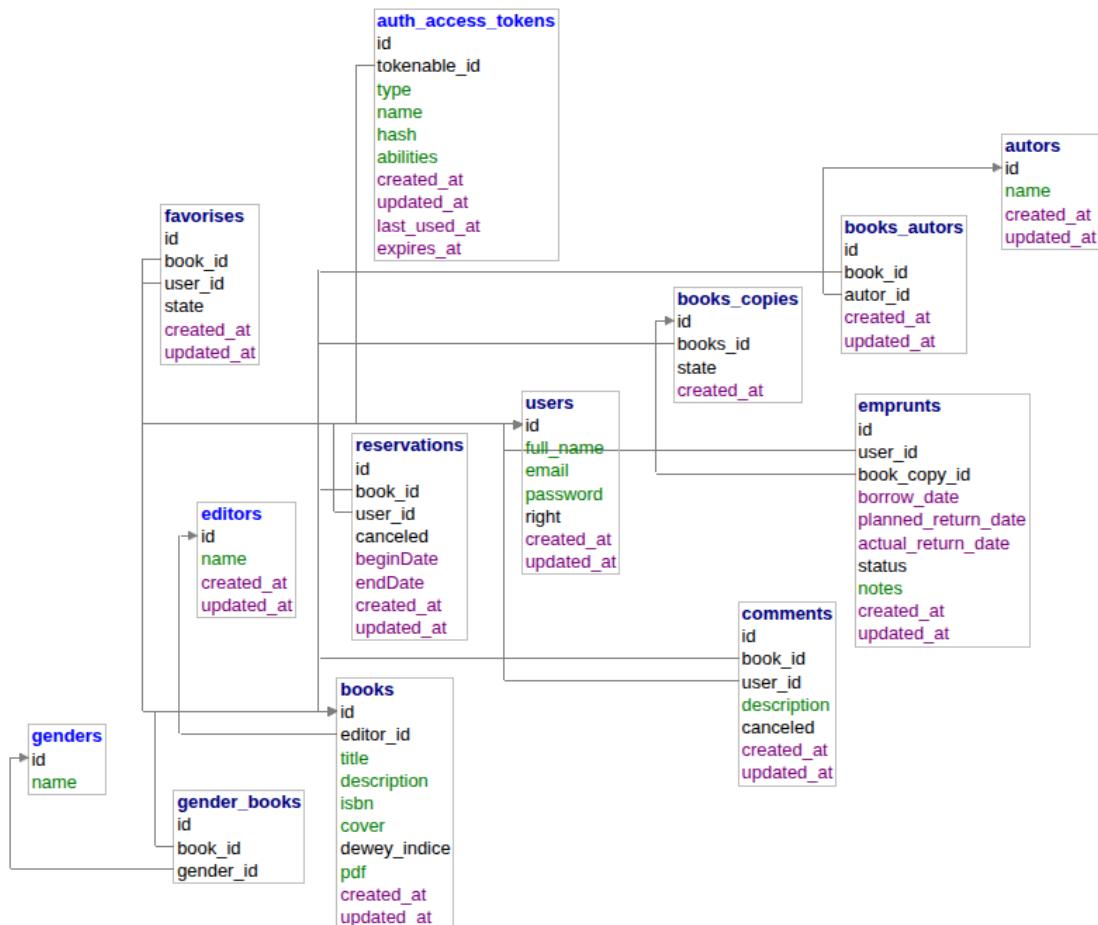
Le MCD comprend les tables des entités dont on aura besoin, les relations entre les tables, les cardinalités et les attributs.

Le modèle conceptuel de données (MCD) ci-dessous :

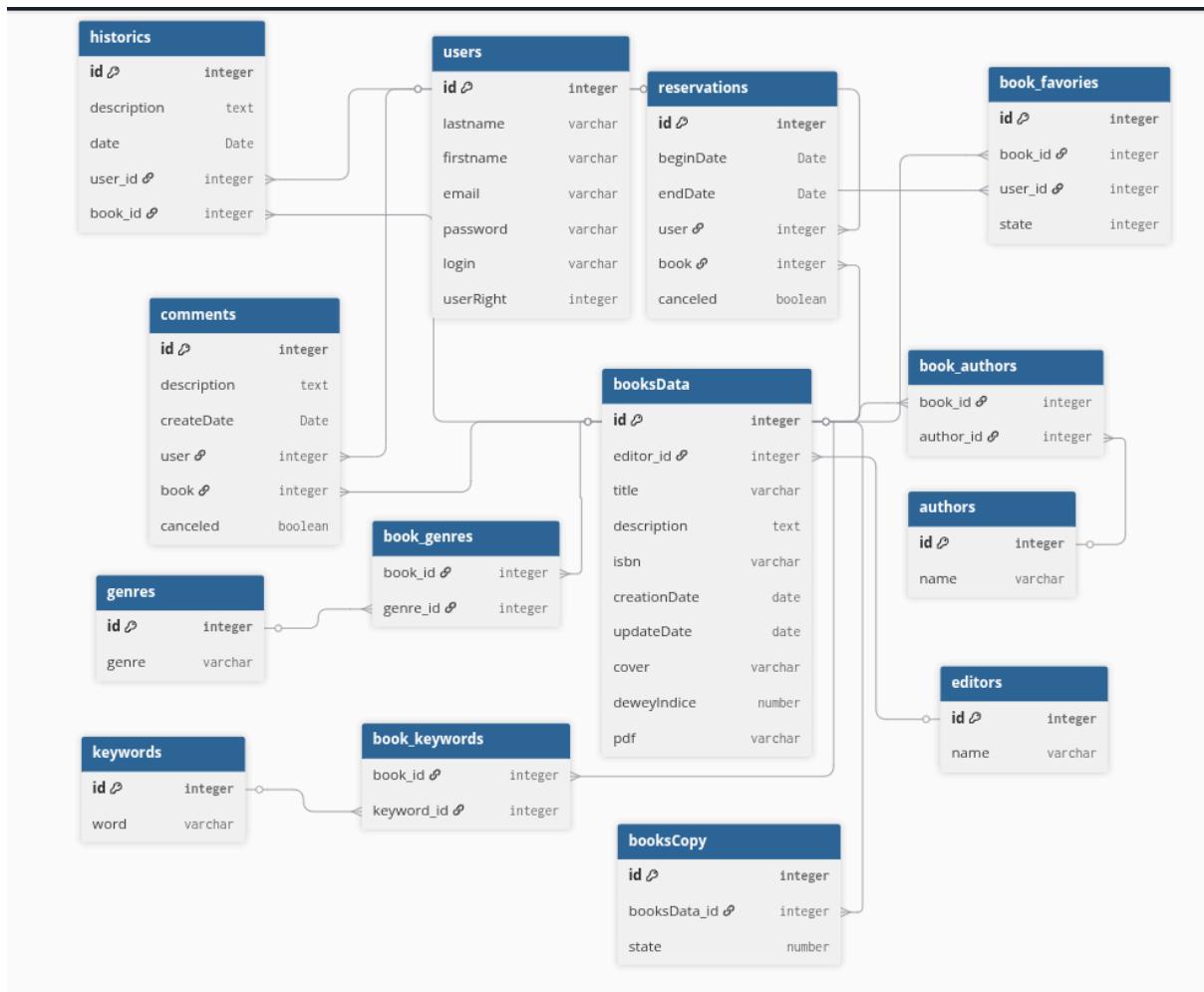


Le modèle logique de données (MLD) ci-dessous :

Pour le modèle logique de données, on ajoute les clés primaires et les clés étrangères. Si la relation entre deux table sont ‘many to many’, il faut rajouter une table intermédiaire.



## Le modèle physique de données (MPD)



# Réalisation

## Architecture MVC

L'architecture dans [Adonis.js](#) favorise le design pattern MVC. Le MVC est un design pattern qui signifie Modèle - Vue - Contrôleur. Ainsi chaque fichier créé a un rôle bien défini. Le but de MVC est justement de séparer la logique du code en trois parties que l'on retrouve dans des fichiers distincts.



Modèle : cette partie gère ce qu'on appelle la logique métier du site. Elle comprend notamment la gestion des données qui sont stockées, mais aussi tout le code qui prend des décisions autour de ces données. Son objectif est de fournir une interface d'action la plus simple possible au contrôleur

Vue : cette partie se concentre sur l'affichage. Elle récupère la variable qui contient les données qu'on doit afficher.

Contrôleur : cette partie gère les échanges avec l'utilisateur. C'est en quelque sorte l'intermédiaire entre l'utilisateur, le modèle et la vue. Le contrôleur va recevoir des requêtes de l'utilisateur.

```
▽ app
  ▷ controllers
  ▷ exceptions
  ▷ middleware
  ▷ models
  ▷ validators
```

AdonisJS favorise une architecture MVC (Model-View-Controller) stricte, avec un système d'injection de dépendances efficace, ce qui le rend très adapté aux API REST bien structurées. Voici comment est découpé le MVC dans [Adonis.js](#).

### M (Model) :

Utilisation de Lucid ORM pour gérer les modèles de données, comme User, Post, Book, etc. Chaque modèle représente une table de la base de données, avec ses méthodes et relations.

Exemple de Model (qui représente la table authors):

```
import { DateTime } from 'luxon'
import { BaseModel, column, manyToMany } from '@adonisjs/lucid/orm'
import Book from './book.js'

export default class Autor extends BaseModel {
  @column({ isPrimary: true })
  declare id: number

  @column()
  declare name: string

  @column.dateTime({ autoCreate: true })
  declare createdAt: DateTime

  @column.dateTime({ autoCreate: true, autoUpdate: true })
  declare updatedAt: DateTime

  @manyToMany(() => Book, {
    pivotTable: 'books_utors',
    pivotForeignKey: 'autor_id',
    pivotRelatedForeignKey: 'book_id',
  })

  public books: any

  @manyToMany(() => Autor, {
    localKey: 'id',
    pivotForeignKey: 'book_id',
    relatedKey: 'id',
    pivotRelatedForeignKey: 'autor_id',
  })
  public Autor: any
}
```

## V (View) :

Même si une API REST n'a pas de vues HTML, la couche "Vue" est remplacée par la représentation JSON des données. Contrôle sur le renvoie des réponses (return user.toJSON() ou response.json()), donc il y a une logique de "vue".

## C (Controller) :

Définition des contrôleurs (ex. UsersController, BooksController) pour gérer la logique métier, récupérer des données depuis les modèles et les retourner via des réponses HTTP.

Exemple de méthodes dans le controller :

```
public async getGenresByBook({ params }: HttpContext) {
    const book = await Book.findOrFail(params.id) // Trouver le livre par son ID
    const genres = await book.related('genders').query() // Récupérer tous les genres associés
    return genres
}

public async getBooksByGender({ params }: HttpContext) {
    const gender = await Gender.findOrFail(params.id) // Trouver le genre par son ID
    const books = await gender.related('books').query() // Récupérer tous les livres associés
    return books
}
```

## Une structure Orienté Objet :

Cette architecture repose sur le programmation orienté objet (POO). La programmation orientée objet est très présente dans AdonisJS, car le framework est entièrement construit autour de ce paradigme. Les modèles avec chacun un controller associé (comme User, Book, Borrows, etc.) sont des classes qui étendent BaseModel, avec des attributs (champs) et des méthodes qui sont directement associés aux classes parentes..

Les contrôleurs sont aussi des classes, chaque méthode représentant en général une requête HTTP liée aux données d'une table. Le système de validation, les middlewares, et les services suivent également une logique de POO. La relation entre les entités et tables de la bdd est exprimée via des méthodes (ex. @hasMany, @belongsTo). L'injection de dépendances dans les contrôleurs est aussi un principe issu de la POO.

La structure d'AdonisJS respecte en grande partie les principes SOLID, surtout si je développe en respectant les bonnes pratiques du framework.



Voici les détails du principe SOLID :

**S**ingle Responsibility Principle (Responsabilité unique) :Les contrôleurs, modèles, middlewares et services ont des rôles bien définis. Ex. : un contrôleur gère la logique HTTP, un modèle les données, un middleware la sécurité.

**O**pen/Closed Principle (Ouvert/fermé): Je peux étendre les fonctionnalités (par exemple via des services ou hooks) sans modifier les classes existantes, ce qui permet d'ajouter du comportement sans casser le code.

**L** – Liskov Substitution Principle:Les classes dérivées (ex. modèles personnalisés ou services hérités) peuvent remplacer les classes parents sans comportement inattendu si je respecte les types et conventions du framework.

**I** – Interface Segregation Principle: Ce principe est moins explicite ici.

**D** – Dependency Inversion Principle : AdonisJS supporte l'injection de dépendances via IoC (Inversion of Control). Je peux injecter des services ou repositories dans mes classes sans dépendre directement de leur implémentation.

## Le Router

Le router AdonisJS permet de définir les routes de l'application en associant des URL à des actions spécifiques (contrôleurs, fonctions, etc.). Il utilise des méthodes HTTP comme GET, POST, PUT, DELETE pour gérer les requêtes.

Par exemple, pour une route (la première) qui renvoie les informations d'un utilisateur (endpoint : users) qui vient de s'authentifier avec un token. La route en dessous représente une actions get qui permet de récupérer les données de tous les livres

```
// pour l'authentification
//genere un token pour un utilisateur existant => login
router.post('users/:id/tokens', async ({ params }) => {
    const user = await User.findOrFail(params.id)
    const token = await User.accessTokens.create(user)

    return {
        type: 'bearer',
        value: token.value!.release(),
    }
})

//pour les livres
const BooksController = () => import('#controllers/books_controller')
router.get('getBooks', [BooksController, 'getBooks'])
router.get('getBooksAutors', [BooksController, 'getBooksAutors'])
```

## L'API REST (Representational State Transfer)

Une API REST est un ensemble de règles permettant à des systèmes de communiquer via HTTP. Elle repose sur des méthodes HTTP comme GET, POST, PUT, DELETE pour effectuer des opérations vers un serveur.

Le CRUD (Create, Read, Update, Delete) représente les quatre opérations de base pour manipuler des données :

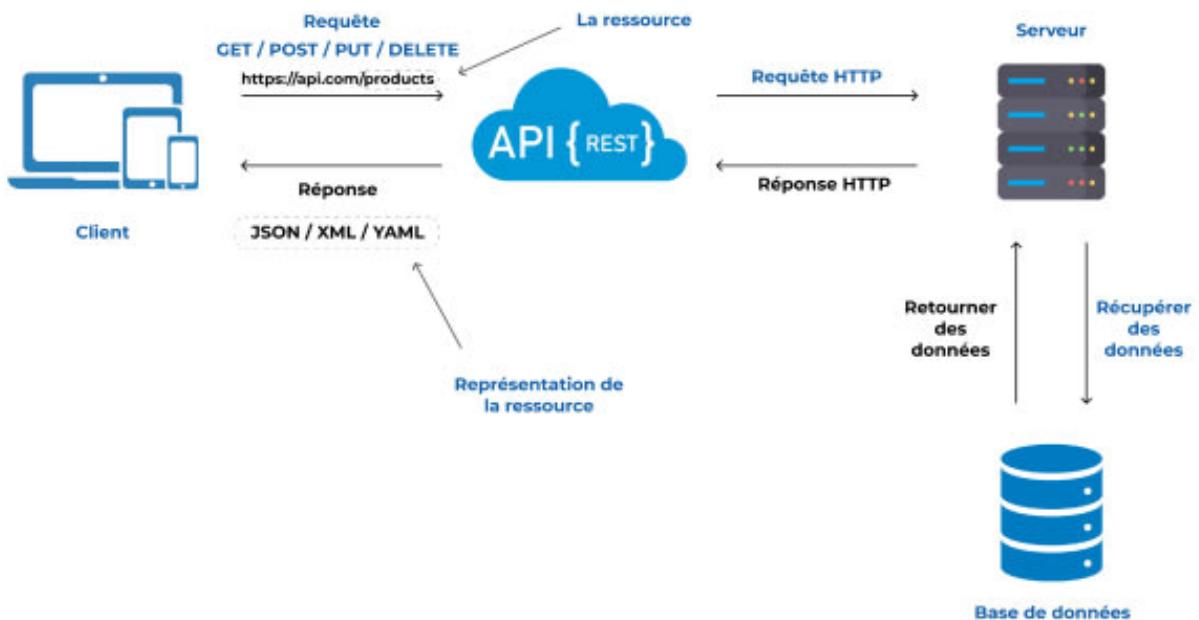
- Create : Créer de nouvelles données. C'est la méthode POST.
- Read : Lire des données, c'est la méthode GET.
- Update : Mettre à jour une ressource existante avec PUT ou PATCH.
- Delete : Supprimer une donnée. C'est la méthode DELETE.

L'API REST simplifie la gestion des données en séparant les opérations. Elle repose sur des standards HTTP bien définis (GET, POST, PUT, DELETE). Cela me permet de structurer les opérations CRUD de manière claire. En utilisant les méthodes HTTP, je peux facilement créer, lire, mettre à jour et supprimer des notices de livres, des emprunts, ... C'est simple à maintenir et permet de mieux structurer le code, avec des endpoints bien définis.

Une API REST utilise des endpoints spécifiques. Par exemple, un endpoint comme /utilisateurs permet de gérer les utilisateurs en appliquant les méthodes du CRUD (GET pour lire, POST pour créer, PUT ou PATCH pour modifier, DELETE pour supprimer), /books permet de gérer les livres, /borrows permet de gérer les emprunts. En résumé, une API REST permet d'effectuer les opérations CRUD sur des données via HTTP.

Je trouve aussi qu'elle facilite l'intégration avec d'autres systèmes, car elle repose sur des standards largement utilisés. Je peux par exemple utiliser cette API REST pour envoyer des données vers une application mobile si je le désire et cela en utilisant le même back-end avec les mêmes endpoint. Comme les données sont envoyées au format JSON, elles peuvent être traitées par de nombreux langages (Kotlin ou Swift par exemple).

De plus, en suivant ce modèle, je peux rapidement tester et déboguer mes services, tout en assurant une bonne évolutivité. C'est plus simple de savoir à quel niveau se situe l'erreur.



## L'interface front avec React

ReactJS est un des frameworks les plus populaires pour le développement d'interfaces utilisateur, il est très connu dans la communauté des développeurs.

Voici quelques avantages pour son utilisation :

- Un écosystème riche : React bénéficie d'une vaste communauté et d'un large choix de bibliothèques et d'outils pour étendre ses fonctionnalités.
- Facilité d'intégration : Peut être intégré dans des projets existants, même dans des applications non React.
- Facilité pour utiliser React Native (dans le prolongement éventuel du projet en application mobile) : Permet de créer des applications mobiles natives pour iOS et Android avec un même code de base.
- Les composants sont réutilisables : React permet de créer des composants modulaires et réutilisables, ce qui rend le code plus propre et maintenable.
- Le Virtual DOM : Il optimise les performances en mettant à jour uniquement les parties du DOM qui changent, ce qui accélère le rendu.
- Support de TypeScript : De plus en plus populaire, React supporte TypeScript pour un typage plus strict et une meilleure gestion des erreurs.
- Débogage et outils de développement : Les extensions comme React DevTools simplifient le débogage et l'analyse des performances.

En prenant en compte ces avantages, j'ai un rendu des données et une expérience utilisateurs qui sont meilleurs. En effet, les nombreux composants de ReactJS et des library associées sont connues pour leur design et le confort utilisateur qui sont testés au préalable.

Maintenant on va s'intéresser davantage au spécificités techniques de ReactJS. Avec [React.js](#) j'ai créé les composants de mes pages visibles dans le navigateur. Les composants utilisés vont fonctionner avec les données qui sont traitées par [Adonis.js](#). Pour cela j'utilise la fonction fetch dans React pour pouvoir récupérer et envoyer des données utiles à l'application.

Cette fonction fetch() qui récupère les données sur une route sera utilisée dans un hook. Dans React, les hooks sont des fonctions qui permettent d'ajouter des

fonctionnalités comme l'état (useState), les effets secondaires (useEffect), ou d'autres comportements à des composants fonctionnels, sans avoir besoin de classes.

useEffect, par exemple, est un hook utilisé pour gérer des effets secondaires dans un composant, comme les appels API ou les modifications du DOM. C'est pour cela qu'on fetch() dans useEffect. Il se déclenche après chaque rendu du composant. Par défaut, il s'exécute après chaque mise à jour du DOM. Si on lui passe un tableau vide [], il ne s'exécutera qu'une seule fois après le premier rendu.

```
useEffect(() => {
  const fetchDataForPosts = async () => {
    try {
      const response = await fetch(`http://localhost:3333/getBooksAutors`);
      if (!response.ok) {
        throw new Error(`HTTP error: Status ${response.status}`);
      }
      let postsData = await response.json();
      console.log(postsData);
      setData(postsData);
      setError(null);
    } catch (err) {
      setError(err.message);
      setData([]);
    } finally {
      setLoading(false);
    }
  };
  fetchDataForPosts();
}, []);
```

## Les tests

### Cahiers des recettes de tests:

Le cahier de recette (cahier de test), est un document essentiel pour assurer le bon fonctionnement des applications web et mobile. C'est un tableau des différents tests à réaliser afin de s'assurer que les nouvelles fonctionnalités sont opérationnelles et de détecter d'éventuelles régressions dans l'ensemble du code. On peut faire des cahiers des recettes aussi bien pour le back testé avec [JapaJS](#) et le front testé avec CypressJS

Voici ses différents avantages :

- Détection des bugs en amont en testant de manière systématique, on identifie rapidement les erreurs ou oubli.
- Gain de temps en maintenance, lors de l'évolution de l'application, on peut rejouer les tests pour vérifier qu'il n'y a pas de régressions (tests de non-régression).
- Permet de vérifier que chaque fonctionnalité est conforme aux spécifications.
- Traçabilité des tests : On peut suivre précisément ce qui a été testé, quand, et par qui. Cela permet de conserver un historique des campagnes de test.
- Communication claire sert de référence dans une équipe (par exemple entre développeurs, testeurs et chefs d'équipe)

Exemple de tests possibles pour un module de connexion (login)

### Cas de tests fonctionnels:

- Connexion réussie avec un email et mot de passe valides.
- Connexion échouée avec un mot de passe invalide.
- Connexion échouée avec un email inexistant.

Tests de validation des champs (frontend ou backend) :

- Refuser un email mal formé (sans @ par exemple).
- Refuser un mot de passe trop court (ex : pas de chiffre et < à 8 caractères).

- Vérifier l'affichage des bons messages d'erreur.

Exemple de cahier des recettes pour le module de connexion :

Composant testé	Description	Etapes du test	Données testée	Résultat reçu	Succès ou échec
<b>Formulaire de Connexion</b>	Connexion avec de bons identifiants	Cliquer sur le bouton se connecter. Entrer les données et Cliquer sur Valider		Redirection vers Dashboard	succès
<b>Formulaire de Connexion</b>	Connexion avec un mauvais mot de passe	Cliquer sur le bouton se connecter. Entrer les données et Cliquer sur Valider		Inscription en rouge mauvais identifiants	échec
<b>Formulaire de Connexion</b>	Connexion avec un mauvais email	Cliquer sur le bouton se connecter. Entrer les données et Cliquer sur Valider		Inscription en rouge Mauvais identifiants	échec
<b>Formulaire d'enregistrement</b>	Enregistrement avec des données valides	Cliquer sur le bouton s'enregistrer. Entrer les données et Cliquer sur Valider		Inscription réussie!	succès
<b>Formulaire d'enregistrement</b>	Enregistrement avec un email qui existe déjà	Cliquer sur le bouton s'enregistrer. Entrer les données et Cliquer sur Valider			échec
<b>Formulaire d'enregistrement</b>	Enregistrement avec un mot de passe nom valide.	Cliquer sur le bouton s'enregistrer. Entrer les données et Cliquer sur Valider		Afficher en rouge conditions du MP	succès
<b>Formulaire d'enregistrement</b>	Format email non valide	Cliquer sur le bouton s'enregistrer. Entrer les données et Cliquer sur Valider			

## Utilisation de Japa côté Adonis.js

Japa est un framework de tests pour AdonisJS, conçu pour faciliter l'écriture de tests unitaires et fonctionnels. Il est inspiré de frameworks comme Jest et Mocha, et permet d'écrire des tests avec une syntaxe claire et expressive. Il s'intègre facilement avec le système de tests d'AdonisJS, utilisant des bases de données et des contextes de requêtes spécifiques à chaque test. L'un de ses grands avantages est sa rapidité et sa simplicité d'utilisation.

## Utilisation de Cypress côté [React.js](#)

Comment fonctionne Cypress

Cypress est un outil open-source permettant d'écrire les tests, il fonctionne très bien avec React.js et propose une interface graphique.

Cypress permet d'effectuer une grande variété de tests, mais la librairie a été développée pour permettre d'effectuer des tests de composants et des tests end-to-end.

The screenshot shows the Cypress application interface. At the top, there is a navigation bar with a 'cy' icon, the repository name 'sigb (main)', and links for 'v14.5.1 • Upgrade', 'Docs', and 'Log in'. Below the navigation, the title 'Welcome to Cypress!' is displayed, followed by a link 'Review the differences between each testing type →'. Two main sections are shown: 'E2E Testing' (represented by a monitor icon) and 'Component Testing' (represented by a grid icon). Both sections contain descriptive text and a 'Configured' status indicator. The 'E2E Testing' section states: 'Build and test the entire experience of your application from end-to-end to ensure each flow matches your expectations.' The 'Component Testing' section states: 'Build and test your components from your design system in isolation in order to ensure each state matches your expectations.'

Les tests de composant testent les composants un à un pour s'assurer du bon fonctionnement de chacun, quelles que soient les données reçues.

Les tests de composants sont idéalement développés en parallèle du composant lui-même. A l'inverse, il vaudra mieux attendre une certaine stabilité pour les tests end-to-end pour ne pas modifier les tests quand un composant est modifié.

Les tests End to End sont une technique utilisée pour vérifier si une application comporte comme prévu, du début à la fin d'un comportement qui ressemble à un comportement utilisateur.. Ils consistent à vérifier que l'utilisateur final puisse achever les principaux scénarios d'utilisation de l'application. Ce sont des scénarios de test prédéfinis que pourrait suivre un utilisateur sur l'application. Par exemple, l'inscription et la connexion qui demande de naviguer sur plusieurs pages et au travers de plusieurs composants..

Exemple de tests composants :

The screenshot shows the Cypress Component Testing interface. On the left, the component tree displays a test for 'form\_registerRegisterForm.cy.js' under the 'RegisterForm' component. The test tree shows a 'TEST BODY' section with a 'mount' command. On the right, a registration form titled 'S'inscrire' is displayed. It has four input fields: 'Nom d'utilisateur' (nono), 'Email' (nono@net.fr), 'Mot de passe' (\*\*\*\*\*), and 'Confirmer le mot de passe' (\*\*\*\*\*). Below the form is a blue 'S'INSCRIRE' button. At the bottom of the form, a green success message says 'Inscription réussie !'. At the bottom of the interface, there are two buttons: 'E2E specs' and 'Component specs', with 'Component specs' being highlighted. Below the interface, a file tree shows files in the 'src' directory: 'NavBarNavBar.cy.js', 'form\_connectionLoginForm.cy.js', and 'form\_registerRegisterForm.cy.js'.

Exemple de test end-to-end :

```
✓ describe('Inscription et connexion', () => {
  const user = {
    full_name: 'Jean Dupont',
    email: 'lebossdu02@example.com',
    password: 'MotDePasse123',
  };

  it('permet de s'inscrire puis de se connecter', () => [
    // Incription
    cy.visit('/register');

    cy.get('input[name="full_name"]').type(user.full_name);
    cy.get('input[name="email"]').type(user.email);
    cy.get('input[name="password"]').type(user.password);
    cy.get('input[name="confirmPassword"]').type(user.password);

    cy.get('button[type="submit"]').click();

    cy.wait(2000);
    cy.contains('Inscription réussie !', { timeout: 10000 }).should('be.visible');

    cy.visit('/login');

    cy.get('input[name="email"]').type(user.email);
    cy.get('input[name="password"]').type(user.password);

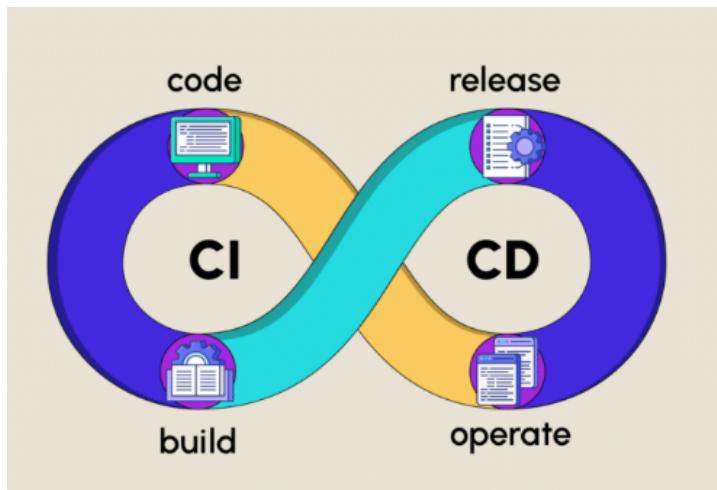
    cy.get('button[type="submit"]').click();

    cy.url().should('include', '/dashboard');
    cy.contains('Bienvenue').should('be.visible');
  ]);
});
```

## CI/CD

CI : L'intégration continue, est un ensemble de pratiques utilisées en génie logiciel consistant à vérifier à chaque modification de code source que le résultat des modifications ne produit pas de régression dans l'application développée.

CD : La Livraison Continue automatise le » push » des applications vers les environnements de livraison (mise en prod)



### Github au service de l'intégration continue

GitHub Actions est un système d'intégration et de déploiement continu (CI/CD) qui permet d'automatiser des workflows, comme des tests et des builds. Il fonctionne avec des runners qui exécutent des jobs dans des environnements virtuels (virtual environments) qui simulent le fonctionnement de l'application dans des systèmes d'exploitation comme Ubuntu, macOS ou Windows.

L'un des principaux avantages d'utiliser GitHub Actions, c'est pour les tests. En effet plus le développement du projet avance, plus il faut être sûr de ne pas subir de régression. Il est donc important de tester les points clés régulièrement. Certaines méthodes sont utilisées un peu partout dans l'application et un problème dans l'une

d'elle rendrait l'application non fonctionnel. Avec des tests Cypress ou Japa, on peut exécuter ces tests dans un environnement contrôlé à chaque push ou pull request. Grâce aux virtual environments, les tests sont effectués sur dans des environnements propres, stables et identiques à chaque exécution et qu'ils ne polluent pas la base de données originelles (par exemple si on teste des formulaires d'enregistrement).

Cela évite les problèmes liés à des différences entre les environnements de développement local et de production. De plus, l'intégration avec GitHub Actions permet de déclencher automatiquement les tests Cypress à chaque changement de code, lors de push de branch en remote par exemple, garantissant une détection rapide des erreurs et une mise à jour continue de l'application avec un feedback immédiat sur la qualité du code. On peut par exemple savoir rapidement quel test a échoué et quelle partie de l'application doit être réparée.

Exemple de workflows :

Faire des tests cypress lors des push pour s'assurer qu'il n'y ait pas de régression :

On peut voir que Github Actions fait tourner virtuellement Node, react et ses dépendances pour effectuer les tests de Cypress.

```
name: Cypress E2E Tests

on:
  push:
    branches:
      - '**'

jobs:
  cypress-run:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Setup Node.js
        uses: actions/setup-node@v4
        with:
          node-version: 18

      - name: Install dependencies
        run: npm install

      - name: Start the app (en background)
        run:
          npm run dev &
          npx wait-on http://localhost:3333

      - name: Run Cypress tests
        uses: cypress-io/github-action@v6
        with:
          start: npm run dev
          wait-on: 'http://localhost:3333'
          browser: chrome
```

Le suivi des différents jobs se fait dans github, dans l'onglet Actions. Le détails des opérations est inscrit et s'il y a des erreurs, Github Actions en donne les causes.

## Exemple de jobs qui échoue

```
Run Cypress tests

226
227 | Tests:      1
228 | Passing:    0
229 | Failing:   1
230 | Pending:    0
231 | Skipped:   0
232 | Screenshots: 1
233 | Video:     false
234 | Duration:  5 seconds
235 | Spec Ran: register.cy.js
236
237
238
239 (Screenshots)
240
241 - /home/runner/work/opac_web/opac_web/cypress/screenshots/register.cy.js/Inscripti
242       on et connexion -- permet de s'inscrire puis de se connecter (failed).png
243
244
245 =====
246
247 (Run Finished)
248
249
250 Spec                               Tests  Passing  Failing  Pending  Skipped
251
252 | ✘ register.cy.js                00:05      1        -        1        -        -
253 |
```

Jobs réussis :

All workflows				Filter workflow runs	
Showing runs from all workflows				Event	Status
51 workflow runs				Branch	Actor
✓ Merge pull request #37 from morgane-marechal/upgradeDb	Upload Existing Backup #81: Commit d1261c0 pushed by morgane-marechal	master	2 days ago	...	
✓ add redis containers	Upload Existing Backup #80: Commit 4c8e8a9 pushed by morgane-marechal	upgradeDb	2 days ago	...	
✓ Merge pull request #36 from morgane-marechal/upgradeDb	Upload Existing Backup #79: Commit a40d28a pushed by morgane-marechal	master	2 days ago	...	
✓ clean code	Upload Existing Backup #78: Commit df7d417 pushed by morgane-marechal	upgradeDb	2 days ago	...	

# Sécurisation

Le côté back-end est sécurisé directement à l'installation de [Adonis.js](#). Il est conçu pour offrir une structure sécurisée et maintenable. La sécurité dans AdonisJS repose sur plusieurs couches, telles que la validation des données, les middlewares, et la cryptographie.

## Avec AdonisJS

### Vine.js

Vine.js, l'outil de validation intégré à AdonisJS, permet de définir des schémas stricts pour valider les entrées utilisateur. Cela garantit que seules les données conformes sont traitées, ce qui protège contre les attaques comme les injections SQL ou XSS.

### Hachage de mot de passe

Pour sécuriser les mots de passe, Adonis utilise la méthode de hachage (comme Argon2 ou Bcrypt), ce qui rend impossible la récupération du mot de passe en clair même si la base de données est compromise.

### JWT (JSON Web Token)

Adonis utilise des tokens pour identifier les utilisateurs. Il est possible de travailler avec des JWT ou des tokens basés sur des sessions. Ces tokens sont signés, uniques et peuvent avoir une durée de validité limitée. Côté client, les tokens sont généralement envoyés dans les en-têtes des requêtes HTTP. Dans mon application j'ai préféré l'envoi de JWT plutôt que l'utilisation de session. Cela permet une utilisation limitée dans le temps que je peux gérer.

### Les middlewares

Les middlewares sont des fonctions qui s'exécutent entre la réception de la requête et la réponse. Adonis permet d'utiliser des middlewares pour effectuer des vérifications de sécurité. Par exemple, le middleware auth() protège les routes en vérifiant si l'utilisateur est authentifié. Les middlewares peuvent aussi être

personnalisés pour des besoins spécifiques, comme l'autorisation d'accès basée sur des rôles.

## CORS

La configuration des headers de sécurité et des paramètres CORS (Cross-Origin Resource Sharing) est également facilement personnalisable, cela évite des requêtes multiples et non naturelles de la part d'une IP.

## CSRF

Enfin, Adonis gère aussi la protection contre les attaques CSRF (Cross-Site Request Forgery)

Dans l'ensemble, AdonisJS met en place une sécurité complète dès son installation, en combinant Vine.js pour la validation des données, l'utilisation de tokens pour l'authentification, et des middlewares pour protéger les routes critiques.

## Avec ReactJS

Dans mon application, je fais les rendu graphique et les intéractions avec les utilisateurs côté ReactJS, du coup les attaques XSS vont être plutôt gérées par ReactJS.

## Reflected XSS

C'est le type de faille XSS où un attaquant injecte un script malveillant dans l'URL ou dans un formulaire, du coup ce script est directement exécuté par le navigateur dès qu'il est renvoyé par le serveur. Cela peut arriver lorsqu'une application prend les données envoyées par l'utilisateur (comme les paramètres d'URL) et les réinjecte sans les filtrer. Exemple avec un filtre search :

[http://example.com/?search=<script>alert\('Affichage d'une alerte inutile'\);</script>](http://example.com/?search=<script>alert('Affichage d'une alerte inutile');</script>)

Pour gérer ce type de faille, ReactJS échappe automatiquement toutes les données envoyées depuis le DOM. Cela signifie que si un attaquant essaie d'injecter un script, React le traitera comme une simple chaîne de texte (et non comme un script), ce qui empêche l'exécution de ce code malveillant. Material-UI s'appuie sur React et échappe de même les script et le html de ses composants

## Stored XSS

Dans ce type de faille, le script malveillant est stocké sur le serveur (par exemple dans une base de données) et est exécuté chaque fois que la page est chargée par un utilisateur. Il peut toucher plusieurs utilisateurs (par exemple dans le cas où c'est stocké dans une table qui affiche automatiquement des données quand on arrive sur le site. Dans ce cas, si un `<script>alert('Affichage d'une alerte inutile');</script>` est stocké, une alerte s'affichera à chaque fois qu'un utilisateur fera un read de la base de données. Cela peut rendre le site impraticable et le départ des usagers. Là aussi le contenu est échappé automatiquement par ReactJS.

## DOM-based XSS

Cela arrive lorsque qu'un usager modifie le DOM (Document Object Model) directement sur le côté client en injectant des scripts malveillants quand il n'y a pas assez de contrôle. Le DOM virtuel limite beaucoup ce genre d'attaque si l'application est codé selon les standards de React.

**ReactJS évite donc les vulnérabilités XSS, car il échappe automatiquement les données utilisateur et gère le DOM de manière sécurisée et les composants Material-UI, qui manipulent des données, respectent les mêmes règles de sécurité. Cela crée un bon combo avec AdonisJS en back qui ajoute d'autres type de sécurité.**

Du coup, ReactJS et MUI offrent une protection efficace contre les failles XSS en évitant les manipulations directes du DOM et en échappant automatiquement les données provenant des utilisateurs et en proposant des composants sécurisés.

## Conclusion

Ce projet permet de mettre en avant de nombreuses compétences que j'ai apprises durant mes années de formation. Il m'a permis de prendre en main des framework sécurisés reposant sur Typescript et Javascript et de travailler la POO.

Avec la séparation des répertoires back-end et front-end, je peux aussi plus facilement étendre les fonctionnalités de mon application. Par exemple, on peut imaginer un troisième répertoire pour développer une application mobile en react Native avec des fonctionnalités spécifique au téléphone comme la lecture de code ISBN.

Des processus de tests et la mise en place d'un CI/CD efficace me permettent de rendre possible l'extension du projet en évitant les régressions. Cela permet également l'ajout de collaborateurs qui voudraient contribuer au projet et qui pourraient plus facilement s'intégrer dans le processus de développement.

# ANNEXES

Sitographie :

Méthode Agile

[https://fr.wikipedia.org/wiki/M%C3%A9thode\\_agile](https://fr.wikipedia.org/wiki/M%C3%A9thode_agile)

Maquettes et prototypes pour application mobile : méthodologie

<https://www.snoweb.io/fr/web-design/maquette-application-mobile/#wireframe>

Tutoriel sur les wireframes Figma pour les débutants (2025)

[https://www.youtube.com/watch?v=iyrEStiTZh0&ab\\_channel=AlienaCai](https://www.youtube.com/watch?v=iyrEStiTZh0&ab_channel=AlienaCai)

La documentation officielle de AdonisJS (2025)

<https://docs.adonisjs.com/guides/preface/introduction>

La documentation officielle de ReactJS (2025)

<https://react.dev/>

La documentation officielle de MUI

<https://mui.com/material-ui/>

La documentation officielle de Redis

<https://redis.io/fr/redis-enterprise/structures-de-donnees/>

Cross Site Scripting (XSS) / OWASP Foundation. (s. d.).

<https://owasp.org/www-community/attacks/xss/>

Cross-site scripting (37/07/205)

[https://developer.mozilla.org/fr/docs/Glossary/Cross-site\\_scripting](https://developer.mozilla.org/fr/docs/Glossary/Cross-site_scripting)