

DOSSIER PROJET

BookNest – Gérez votre
bibliothèque personnelle

**HAMMOUCHE
MOHAMMED
2025**



SOMMAIRE

01

INTRODUCTION

02

L'AGENCE

03

MA PLACE

04

MES MISSIONS

05

BILAN

06

CONCLUSION

01

À PROPOS DE MOI

À propos de moi

Je m'appelle Mohammed Hammouche et je vis à Marseille. Mon goût pour la technologie ne date pas d'hier : dès 2019 je lançais une micro-entreprise e-commerce et réalisais des créations graphiques en freelance sur Fiverr. Entre 2020 et 2023, j'ai enchaîné plusieurs expériences terrain – vendeur high-tech, réparateur de smartphones, qui m'ont appris la rigueur, le contact client et la résolution de problèmes rapides.

Curieux de nature, j'ai découvert le développement web en autodidacte avant de rejoindre, en avril 2024, la formation Développeur Web & Web Mobile (DWWM) à La Plateforme_ Marseille. L'alternance m'a permis de passer des prototypes personnels à des projets en production : sites en Symfony, front-end React, APIs Node/Express, bases MySQL et PostgreSQL.

Aujourd'hui, j'affectionne particulièrement le stack JavaScript full-stack (React, Node, Prisma) et les workflows conteneurisés avec Docker. Je prends plaisir à coder, à tester de nouvelles technos (Next.js, GSAP) et à pousser mes projets jusqu'au déploiement (Railway, Netlify).

Mon objectif avec BookNest : démontrer mes compétences full-stack, de l'UX aux pipelines CI/CD, et valider officiellement le titre DWWM pour embrayer sur des missions encore plus ambitieuses.



02

INTRODUCTION
GÉNÉRALE

2. Introduction générale

Dans le cadre de la formation Développeur Web et Web Mobile (DWWM) à La Plateforme_ Marseille, un projet personnel nous a été demandé afin de mettre en pratique les compétences acquises tout au long du parcours. J'ai choisi de créer BookNest, une application web de gestion de bibliothèque personnelle. Elle permet à un utilisateur de rechercher, consulter, organiser et noter ses livres préférés grâce à l'intégration de l'API Google Books.

Le choix de ce sujet répond à une double motivation : passionné par la lecture et intéressé par les technologies web modernes, j'ai souhaité allier ces deux centres d'intérêt dans un projet utile, intuitif et esthétique. Le développement de BookNest m'a permis de travailler sur un cas concret simulant un contexte professionnel : expression de besoins, architecture REST, consommation d'API tierce, responsive design, authentification sécurisée, et déploiement en environnement conteneurisé.

BookNest repose sur une stack JavaScript moderne : React, Vite et TailwindCSS pour le front-end, Node.js/Express pour l'API, Prisma et PostgreSQL pour la base de données, le tout orchestré avec Docker. J'ai également intégré Nodemailer pour les emails de notification et JWT pour la gestion des sessions utilisateur.

Ce dossier a pour objectif de présenter les étapes clés de la réalisation de BookNest, les choix techniques effectués, les fonctionnalités implémentées, ainsi que les difficultés rencontrées et les solutions apportées pour aboutir à une application web fonctionnelle, sécurisée et évolutive.

03

LISTE DES COMPÉTENCES MISES EN ŒUVRE

3. Liste des compétences mises en œuvre

Dans le cadre du projet BookNest, les compétences du référentiel DWWM suivantes ont été mobilisées :

1. Installer et configurer l'environnement de travail

- Mise en place de l'environnement Node.js/Express, PostgreSQL, Prisma ORM, Vite, Docker et gestion des variables d'environnement (.env).
- Connexion à l'API Google Books, configuration de Nodemailer pour les emails.

2. Maquetter des interfaces utilisateur web ou web mobile

- Création de wireframes et d'un flow de navigation desktop/mobile.
- Définition d'une charte graphique minimaliste avec TailwindCSS (couleurs, typographies, espacements).

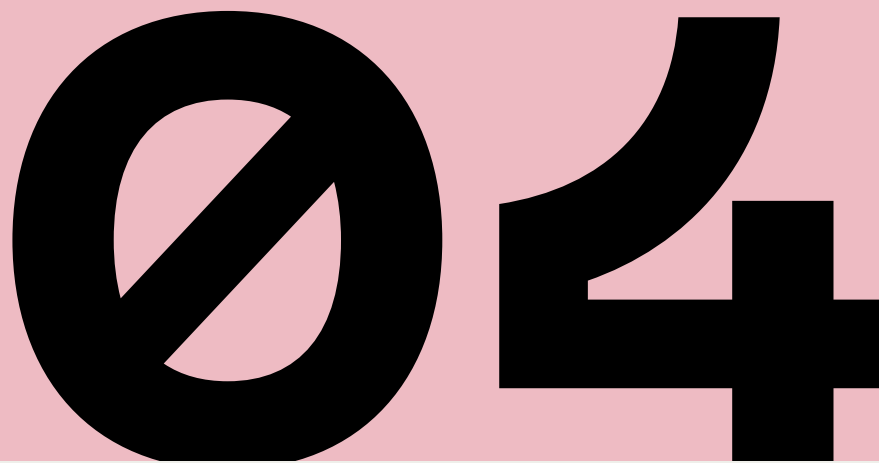
3. Réaliser des interfaces utilisateur statiques web ou web mobile

- Intégration HTML5 via JSX avec React.
- Mise en page responsive à l'aide de TailwindCSS.

4. Développer la partie dynamique des interfaces

- Interaction utilisateur via React (état, événements, formulaires dynamiques).
- Utilisation de React Query et Axios pour la récupération des données API.

1. Développer des composants métier côté serveur
 - Création d'une API REST sécurisée avec Express (routes, middlewares).
 - Utilisation de JWT pour l'authentification et gestion des rôles utilisateurs.
2. Mettre en place une base de données relationnelle
 - Modélisation via Prisma ORM (schéma utilisateurs, livres, statuts).
 - Migrations automatiques, Seed de données de test, Prisma Studio.
3. Développer des composants d'accès aux données
 - Requêtes Prisma pour manipuler les données (CRUD utilisateurs/livres).
 - Intégration avec l'API Google Books pour enrichir les données automatiquement.
4. Documenter le déploiement d'une application dynamique
 - Conteneurisation avec Docker et Docker Compose.
 - Déploiement du backend sur Railway et frontend sur Netlify.
5. Mettre en œuvre une démarche de résolution de problème
 - Debugging avec console.log, Postman, Prisma Studio.
 - Tests manuels sur chaque feature, itérations par composants.
6. Apprendre en continu (veille technologique)
Suivi des évolutions de React, Prisma, Docker, Google Books API via GitHub et documentation officielle.



EXPRESSION DES BESOINS

4. Expression des besoins

1. Présentation du besoin général

Avec la multiplication des plateformes de lecture (Kindle, Kobo, Google Books...), il devient difficile pour les lecteurs de suivre, retrouver et organiser leurs livres. Les applications existantes sont souvent centrées sur l'achat ou la lecture directe, sans offrir un véritable outil de gestion de collection personnelle.

BookNest répond à ce besoin. Il s'agit d'une application web centralisée qui permet à l'utilisateur de rechercher, sauvegarder et annoter ses livres préférés via une interface simple, responsive et accessible sur tous les supports. L'application exploite l'API Google Books pour enrichir les fiches livres automatiquement.

2. Objectifs du projet

- Permettre aux utilisateurs de rechercher des livres via l'API Google Books
- Offrir un tableau de bord personnalisé avec la gestion des statuts de lecture (à lire, en cours, lu)
- Intégrer un système de notation, d'annotations personnelles et d'évaluation
- Mettre en place une interface claire, intuitive et responsive

Garantir une authentification sécurisée avec gestion des rôles

3. Fonctionnalités attendues

- Recherche de livres enrichie (titre, auteur) avec suggestions API Google Books
- Consultation complète de la fiche d'un livre : image, auteur, description, lien vers Google Books
- Authentification complète : inscription, connexion, vérification d'email, réinitialisation de mot de passe
- Tableau de bord avec tri par statut (à lire, en cours, lu)
- Page d'administration avec gestion des utilisateurs et de leurs collections
- Interface responsive pensée pour desktop et mobile

4. User stories

En tant que...	Je veux...	Afin de...
Utilisateur anonyme	rechercher un livre via Google Books	obtenir des informations rapidement
Utilisateur authentifié	ajouter un livre à ma collection	suivre ma progression de lecture
Utilisateur authentifié	noter et annoter un livre	garder une trace de mes avis personnels
Admin	gérer les utilisateurs et leurs collections	modérer l'utilisation de la plateforme

5. Limites du projet

Bien que BookNest couvre les fonctionnalités clés d'une application de gestion de bibliothèque, certaines limites ont été identifiées en raison des choix techniques, du périmètre initial défini et du temps imparti.

- Pas de recherche multicritère avancée
- La recherche actuelle se base sur l'API Google Books avec un simple champ texte. Un filtrage par catégories, genres ou langues aurait amélioré la pertinence des résultats.
- Pas de fonctionnalité de partage de collection
- Les collections sont strictement privées. Il n'est pas encore possible de partager ses listes ou avis avec d'autres utilisateurs. Cette fonctionnalité nécessiterait une couche sociale supplémentaire (profils publics, permissions, etc.).
- Pas de recommandations personnalisées
- Aucun système de recommandation basé sur les habitudes de lecture ou les genres favoris n'est encore en place. Cela nécessiterait une logique de scoring, voire du machine learning.
- Application disponible uniquement en français
- Aucune gestion multilingue n'a été mise en place. L'ensemble de l'application et des données textuelles est pensé pour un usage francophone uniquement.

Ces limitations sont assumées et pourront constituer des axes d'évolution si le projet est amené à évoluer.

5. Limites du projet

Bien que BookNest couvre les fonctionnalités clés d'une application de gestion de bibliothèque, certaines limites ont été identifiées en raison des choix techniques, du périmètre initial défini et du temps imparti.

- Pas de recherche multicritère avancée
- La recherche actuelle se base sur l'API Google Books avec un simple champ texte. Un filtrage par catégories, genres ou langues aurait amélioré la pertinence des résultats.
- Pas de fonctionnalité de partage de collection
- Les collections sont strictement privées. Il n'est pas encore possible de partager ses listes ou avis avec d'autres utilisateurs. Cette fonctionnalité nécessiterait une couche sociale supplémentaire (profils publics, permissions, etc.).
- Pas de recommandations personnalisées
- Aucun système de recommandation basé sur les habitudes de lecture ou les genres favoris n'est encore en place. Cela nécessiterait une logique de scoring, voire du machine learning.
- Application disponible uniquement en français
- Aucune gestion multilingue n'a été mise en place. L'ensemble de l'application et des données textuelles est pensé pour un usage francophone uniquement.

Ces limitations sont assumées et pourront constituer des axes d'évolution si le projet est amené à évoluer.



6. Contraintes du projet

- Technologiques : stack imposée en lien avec la formation (React, Node.js, Docker, PostgreSQL, Tailwind)
- Pédagogiques : travail individuel avec deadline imposée
- Fonctionnelles : portée limitée à la gestion de collection, sans fonctions sociales avancées

7. Public cible

- Lecteurs actifs souhaitant organiser leurs lectures personnelles
- Utilisateurs déçus par les applications de gestion trop commerciales
- Jeunes adultes connectés, amateurs de lecture et d'interfaces modernes

05

**ENVIRONNEMENT
TECHNIQUE**

1. Outils de développement et configuration

- IDE utilisé : Visual Studio Code
- Environnement de développement : Docker + Docker Compose (multi-conteneurs : frontend, backend, base de données)
- Lancement du projet :
 - docker-compose up pour démarrer les services
 - Frontend : npm run dev via Vite (React)
 - Backend : node index.js (Express)
- Versioning : Git en local, GitHub pour l'hébergement distant
- Dépendances :
 - Backend : npm + Express, Prisma, JWT, Nodemailer
 - Frontend : npm + React, TailwindCSS, React Query, Axios
- Build : Vite (frontend)
- Outils complémentaires : Postman (tests API), Prisma Studio (DB visualisation)

2. Technologies back-end

- Langage : JavaScript (Node.js)
- Framework : Express
- ORM : Prisma (PostgreSQL)
- Authentification : JWT + rôles utilisateurs
- Services externes : API Google Books
- Sécurité : validation des données, middleware d'authentification, gestion des tokens

3. Base de données

- Système : PostgreSQL
- Modélisation : Prisma (modèles User, Book)
- Migrations : automatiques via Prisma
- Seed de données : script intégré au démarrage
- Accès admin : via Prisma Studio
(<http://localhost:5566>)

4. Technologies front-end

- Langage : JavaScript (React)
- Framework CSS : Tailwind CSS
- Librairies :
 - React Query (gestion serveur state)
 - React Hook Form (formulaires)
 - Axios (requêtes HTTP)
 - React Icons (icônes)
- UI : Responsive design, layout modulaire, composants réutilisables

5. Services tiers

- Google Books API : récupération dynamique des données livres
- Nodemailer : envoi d'emails (vérification, réinitialisation, notification)
- Ethereal.email : test des emails en dev

6. Sécurité

- Authentification via tokens JWT
- Middleware de contrôle d'accès
- Validation des entrées (frontend et backend)
- Rôles utilisateur (USER / ADMIN)
- Expiration et renouvellement des tokens

7. Responsive & accessibilité

- Interface responsive (mobile, tablette, desktop)
- Contraste, lisibilité, navigation clavier et messages d'erreurs accessibles

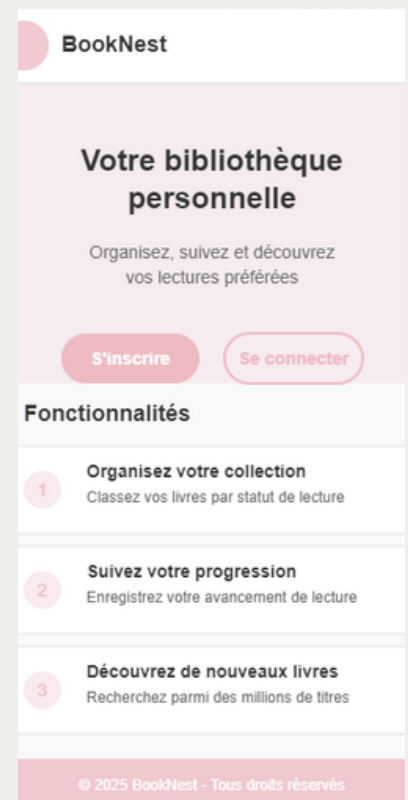
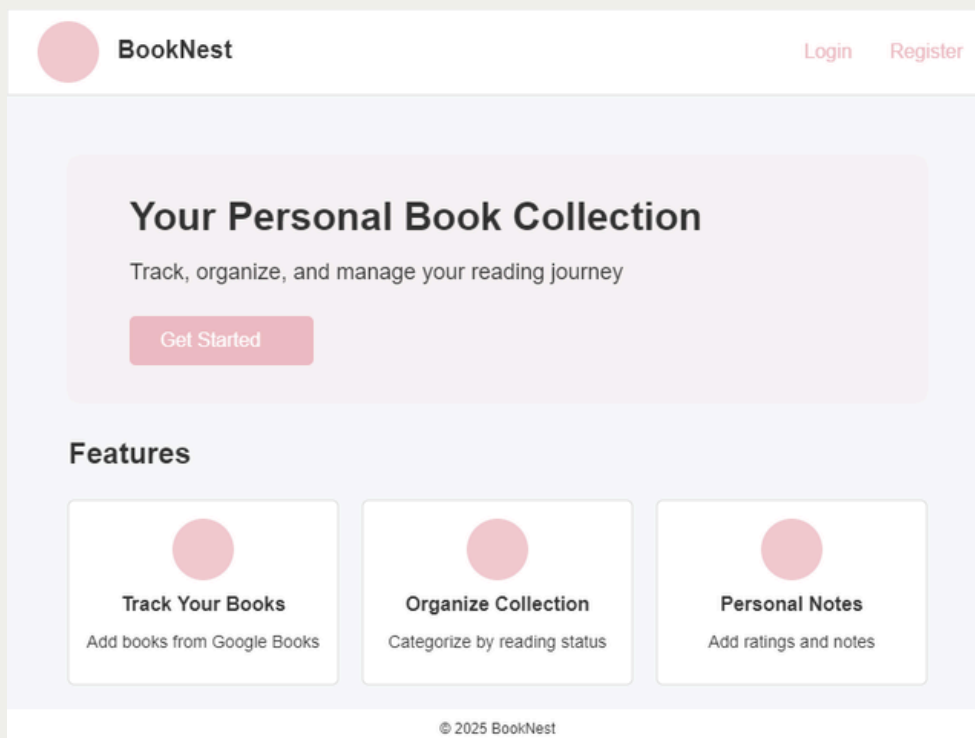
06

**RÉALISATIONS CÔTÉ
FRONT-END**

6.1. Maquettes & enchaînement


- 6.1. Maquettes & enchaînement
- Avant de commencer le développement de l'interface utilisateur, j'ai réalisé des maquettes haute fidélité sur Figma pour planifier le design et les interactions. L'objectif était de concevoir une interface moderne, fluide, ergonomique et mobile-first.
- Ces maquettes illustrent les principales pages :

Page d'accueil (accès public)





Page de inscription

 BookNest

Create an Account

Full Name

Email Address


Password

Password strength: Medium

Register

Already have an account? [Login](#)

u00a9 2025 BookNest

 BookNest

Create an Account

Full Name

Email Address

Password

Password strength: Medium


☐ I agree to the [Terms and Conditions](#)

Register

Already have an account? [Login](#)

© 2025 BookNest - Tous droits réservés

Page de connexion

 BookNest

Login

Email Address


Password

[Forgot Password?](#)

Login

Don't have an account? [Register](#)

© 2025 BookNest

 BookNest

Login

Email Address

Password

[Forgot Password?](#)

Login

Don't have an account? [Register](#)


Or login with


G

f

u00a9 2025 BookNest - Tous droits r serv s

Tableau de bord

 BookNest



All Books

Want to Read

Reading

Read

Search

Profile

Total Books

24

Want to Read

10

Reading

3

Read

11

Book Title 1

Author Name

Reading

Book Title 2

Author Name

Want to Read

Book Title 3

Author Name

Read

Book Title 4

Author Name


Want to Read


Book Title 5

Book Title 6

Book Title 7

Book Title 8

 BookNest



All Books

Reading

Read

Search

Total Books

24

Want to Read

10

Reading

3

Book Title 1

Author Name

Reading

Book Title 2

Author Name

Want to Read

Book Title 3

Book Title 4

Home

Search

Profile

Menu

Fiche d'un livre

BookNest

JD

All Books

Want to Read

Reading

Read

Search

Profile

Back

To Kill a Mockingbird

Harper Lee

Fiction

Classic

Rating

Status

Reading

Description

The unforgettable novel of a childhood in a sleepy Southern town and the crisis of conscience

My Notes

Started reading on May 15th. Really enjoying the character development...

Update Status

Remove

u2190

Book Details

To Kill a Mockingbird

Harper Lee

Fiction, Classic

Your Rating:

u2605 u2605 u2605 u2605 u2605

Reading Status:

Currently Reading

u25bc

Progress:

50%

Description:

The unforgettable novel of a childhood in a sleepy Southern town and the crisis of conscience

Your Notes:

Add your notes here...

Home

Search

Profile

Menu

Page de inscription

BookNest

Create an Account

Full Name

Email Address

Password

Password strength: Medium

Register

Already have an account? Login

BookNest

Create an Account

Full Name

Email Address

Password

Password strength: Medium

I agree to the Terms and Conditions

Register

Already have an account? Login

Page de connexion

BookNest

Login

Email Address

Password

Forgot Password?

Login

Don't have an account? Register

BookNest

Login

Email Address

Password

Forgot Password?

Login

Don't have an account? Register

Or login with

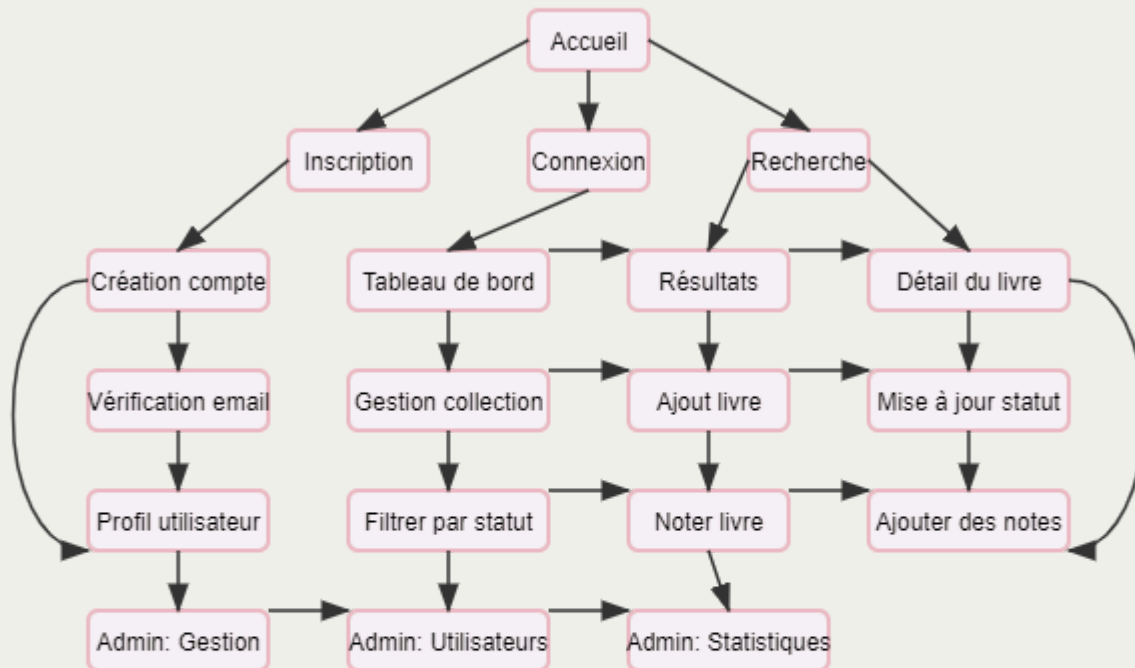
G

f

u00a9 2025 BookNest - Tous droits r  serv  s

Chaque maquette a été pensée dans une logique responsive et accessible : contraste élevé, navigation intuitive, bonnes pratiques RGAA (balises ARIA, focus clavier, structure lisible).

Schéma du parcours utilisateur



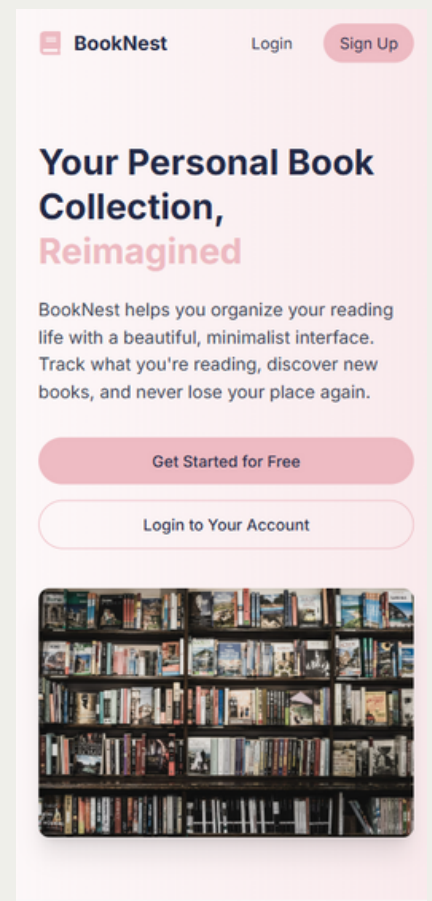
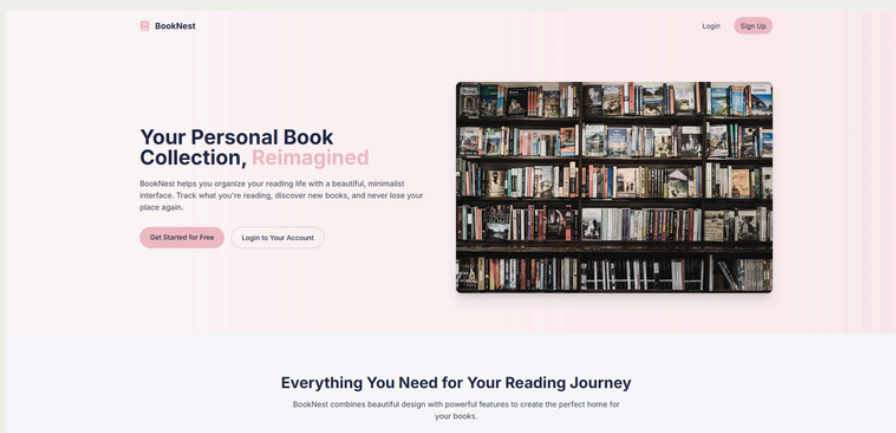
6.2. Interfaces statiques

- 6.1. Maquettes & enchaînement
- Avant de commencer le développement de l'interface utilisateur, j'ai réalisé des maquettes haute fidélité sur Figma pour planifier le design et les interactions. L'objectif était de concevoir une interface moderne, fluide, ergonomique et mobile-first.
- Ces maquettes illustrent les principales pages :

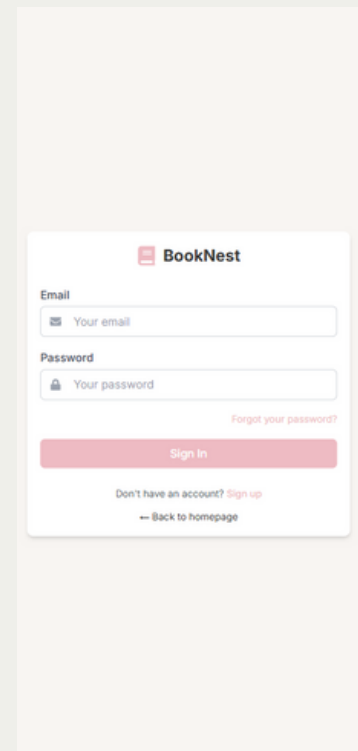
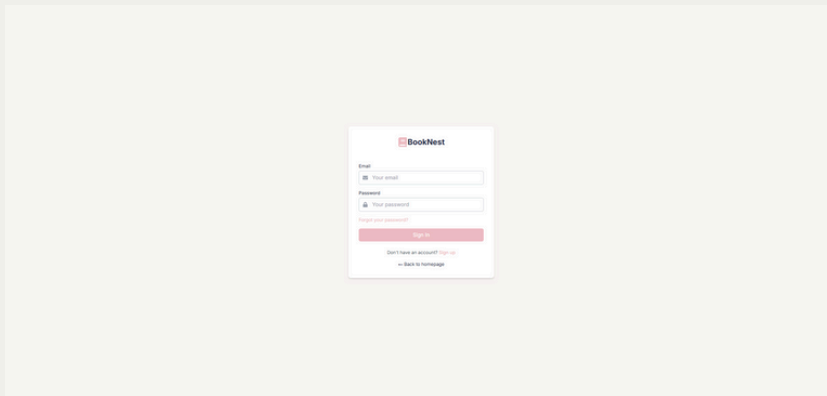
6.2. Interfaces statiques

- Le front-end de BookNest a été développé avec React (via Vite) et TailwindCSS. J'ai structuré l'interface en composants modulaires pour faciliter la réutilisabilité et la maintenance du code.
- Composants clés :
- BookCard : carte visuelle d'un livre avec actions (ajout, suppression, statut)
- BookGrid : affichage en grille de la collection
- Formulaire : pages login/register, mise à jour de profil
- Header/Footer : navigation persistante responsive
- Les interfaces ont été codées dans une logique mobile-first, avec gestion des points de rupture Tailwind (sm, md, lg, etc.), flexbox, grid, et classes d'état (hover, focus).
- Les pages statiques incluent :
- Page d'accueil
- Pages de connexion / inscription
- Pages de profil
- Interface admin (liste des utilisateurs)

Capture de la page d'accueil du site



Capture de la page connexion du site



Extrait de code du fichier layout.jsx (layout principal)

```
const Layout = () => {  
  // Get user information and logout function  
  const { user, logout } = useAuth()  
  // Navigation hook for redirects  
  const navigate = useNavigate()  
  
  // Logout function that redirects to the login page  
  const handleLogout = () => {  
    logout()  
    navigate('/login')  
  }  
}
```

6.3. Interfaces dynamiques

- Recherche dynamique avec autocomplétion
- Fonctionnalité UX majeure : dès que l'utilisateur tape au moins deux lettres dans la barre de recherche, une requête est envoyée à l'API Google Books. Les résultats sont affichés en direct dans une dropdown interactive.
- **Étapes techniques :**
 - L'utilisateur tape un mot-clé dans la barre de recherche
 - Requête HTTP envoyée en temps réel (Axios)
 - Résultats affichés sous forme de cartes avec couverture, titre, auteur
 - Navigation clavier et souris active (↑ ↓ Entrée)
 - Redirection vers la page détaillée du livre choisi

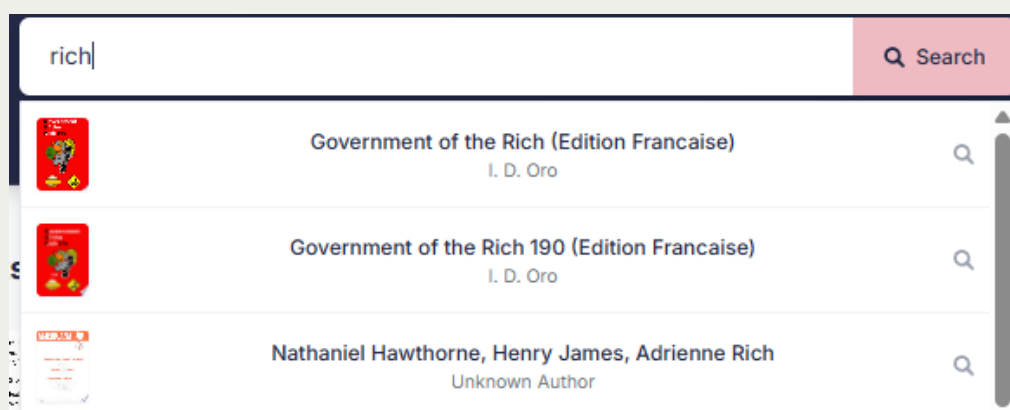
- **Feedback visuel et UX :**
- Chargement asynchrone avec spinner
- Message "Aucun résultat" si la recherche échoue
- Navigation accessible au clavier (focus visible, touche Escape pour fermer la dropdown)
- Dropdown fermée automatiquement après sélection

```

1  /* Autocomplete suggestions dropdown */
2  (showSuggestions && suggestions.length > 0 && (
3    <div className="absolute top-full left-0 right-0 bg-white shadow-lg rounded-b-lg z-50 mt-1 border border-gray-200 max-h-80 overflow-y-auto">
4      {isLoadingSuggestions ? (
5        <div className="flex items-center justify-center p-4">
6          <svg className="animate-spin h-5 w-5 text-[#E6B8C3]" xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0 24 24">
7            <circle className="opacity-25" cx="12" cy="12" r="10" stroke="currentColor" strokeWidth="4"></circle>
8            <path className="opacity-75" fill="currentColor" d="M4 12a8 8 0 018-8V0C5.373 0 0 5.373 0 12h4zm2 5.291A7.962 7.962 0 010 12h0c0 3.042 1.135 5.824 3 7.938l3-2.647z"></path>
9          </svg>
10         <span className="ml-2 text-sm text-gray-600">Loading suggestions...</span>
11       </div>
12     ) : (
13       <ul>
14         {suggestions.map(suggestion => (
15           <li
16             key={suggestion.id}
17             className="border-b border-gray-100 last:border-b-0 hover:bg-gray-50 cursor-pointer transition-colors duration-150"
18             onClick={() => {
19               setSearchTerm(suggestion.title);
20               setShowSuggestions(false);
21               searchBooks(suggestion.title);
22             }}
23           >
24             <div className="flex items-center p-3">
25               {suggestion.thumbnail ? (
26                 <img
27                   src={suggestion.thumbnail}
28                   alt={suggestion.title}
29                   className="w-10 h-14 object-cover mr-3 rounded shadow-sm"
30                 />
31               ) : (
32                 <div className="w-10 h-14 bg-gray-100 flex items-center justify-center mr-3 rounded">
33                   <FaBook className="text-[#E6B8C3] opacity-50" />
34                 </div>
35               )}
36               <div className="flex-1 min-w-0">
37                 <p className="font-medium text-[#232946] truncate">{suggestion.title}</p>
38                 <p className="text-sm text-gray-500 truncate">{suggestion.author}</p>
39               </div>
40               <FaSearch className="text-gray-400 ml-2" />
41             </div>
42           </li>
43         ))}
44       </ul>
45     )}
46   </div>
47 )}
48
49 <button
50   type="submit"
51   className="bg-[#E6B8C3] hover:bg-[#E0A1AC] text-[#232946] font-medium px-6 py-4 flex items-center justify-center transition-colors duration-200"
52   disabled={!(searchTerm.trim() || isLoading)}
53 >
54   {isLoading ? (
55     <span className="flex items-center">
56       <svg className="animate-spin ml-1 mr-2 h-5 w-5 text-white" xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0 24 24">
57         <circle className="opacity-25" cx="12" cy="12" r="10" stroke="currentColor" strokeWidth="4"></circle>
58         <path className="opacity-75" fill="currentColor" d="M4 12a8 8 0 018-8V0C5.373 0 0 5.373 0 12h4zm2 5.291A7.962 7.962 0 010 12h0c0 3.042 1.135 5.824 3 7.938l3-2.647z"></path>
59       </svg>
60       Searching...
61     </span>
62   ) : (
63     <span className="flex items-center">
64       <FaSearch className="mr-2" />
65       Search
66     </span>
67   )}
68 </button>
69 </div>
70 </form>

```

Affichage des résultats dynamique



07

**RÉALISATIONS CÔTÉ
BACK-END**

7.1 Modélisation de la base de données

- L'application BookNest utilise PostgreSQL comme base relationnelle. La modélisation repose sur deux entités principales : User et Book, décrites via Prisma ORM. L'objectif était d'avoir un schéma normalisé, évolutif et sécurisé.
- Tables principales
- Table users :
 - Stocke les informations de chaque utilisateur (nom, email, mot de passe, rôle).
 - Gère les statuts de vérification d'email, les tokens de réinitialisation, les avatars, etc.
- Table books :
 - Représente les livres ajoutés à la collection personnelle d'un utilisateur.
 - Contient le titre, les auteurs, le statut de lecture, l'image, les notes personnelles et la note utilisateur.
- Contraintes et index
- Clés étrangères : books.userId → users.id (onDelete: Cascade).
- Index sur users.email (unique) pour optimiser les connexions.
- Index sur books.status pour accélérer le tri par lecture.

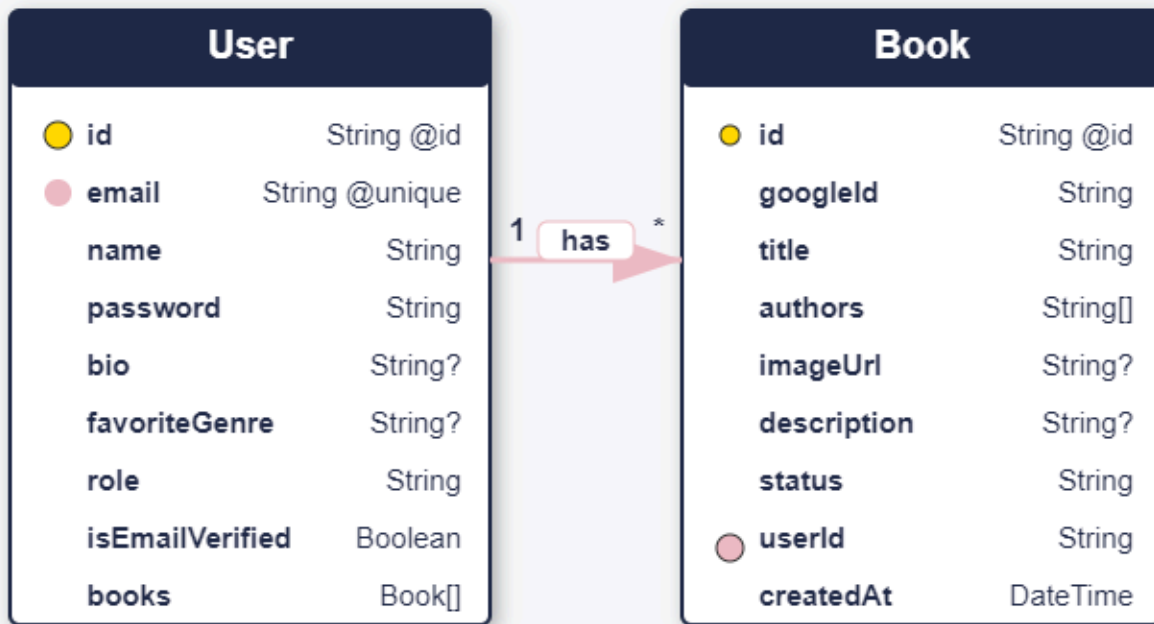
Exemple de structure (extrait Prisma)

```
1 model User {
2   id      String   @id @default(cuid())
3   email   String   @unique
4   name    String
5   password String
6   bio     String?  @db.Text
7   favoriteGenre String?
8   picture String?  @default("https://ui-avatars.com/api/?name=$name")
9   role    String   @default("USER")
10  isEmailVerified Boolean @default(false)
11  verificationToken String?
12  verificationExpiry DateTime?
13  resetToken String?
14  resetTokenExpiry DateTime?
15  books    Book[]
16  createdAt DateTime @default(now())
17  updatedAt DateTime @default(now())
18 }
```

```
20 model Book {
21   id      String   @id @default(uuid())
22   googleId String
23   title   String
24   authors String[]
25   imageUrl String?
26   description String? @db.Text
27   notes   String?  @db.Text
28   rating  Int?
29   status  String   @default("WANT_TO_READ")
30   addedBy User      @relation(fields: [userId], references: [id], onDelete: Cascade)
31   userId  String
32   createdAt DateTime @default(now())
33 }
34
```

7.1 Base de données

- Le schéma de la base de données a été modélisé avec Prisma ORM et repose sur deux entités principales : User et Book, liées par une relation un-à-plusieurs. Chaque utilisateur peut avoir plusieurs livres.



7.1.1 Description des tables principales

- **Table User**
- **id** : identifiant unique (PK)
- **email** : adresse email unique (avec validation)
- **password** : mot de passe haché avec bcrypt
- **isEmailVerified** : statut de vérification d'email
- **role** : USER ou ADMIN
- **autres champs** : bio, genre préféré, image de profil, timestamps

- **Table Book**

- id : identifiant unique (PK)
- googleId : ID du livre dans l'API Google Books
- title, authors, description, imageUrl : métadonnées du livre
- status : statut de lecture (à lire, en cours, lu)
- notes, rating : informations personnalisées utilisateur
- userId : clé étrangère vers User (avec suppression en cascade)

```
1  model User {
2    id          String   @id @default(cuid())
3    email       String   @unique
4    name        String
5    password    String
6    bio         String?  @db.Text
7    favoriteGenre String?
8    picture     String?  @default("https://ui-avatars.com/api/?name=$name")
9    role        String   @default("USER")
10   isEmailVerified Boolean @default(false)
11   verificationToken String?
12   verificationExpiry DateTime?
13   resetToken    String?
14   resetTokenExpiry DateTime?
15   books         Book[]
16   createdAt     DateTime @default(now())
17   updatedAt     DateTime @default(now())
18 }

20 model Book {
21   id          String   @id @default(uuid())
22   googleId    String
23   title       String
24   authors     String[]
25   imageUrl    String?
26   description String?  @db.Text
27   notes       String?  @db.Text
28   rating      Int?
29   status      String   @default("WANT_TO_READ")
30   addedBy     User     @relation(fields: [userId], references: [id], onDelete: Cascade)
31   userId      String
32   createdAt   DateTime @default(now())
33 }
34
```

7.1.2 Contraintes d'intégrité et index

- Clés étrangères :
- books.userId → users.id (ON DELETE CASCADE)
- Index :
- Index UNIQUE sur users.email
- Index sur books.status pour les filtrages rapides
- Justification : améliore les performances, garantit l'intégrité référentielle, et permet une recherche efficace dans les collections utilisateur.

7.1.3 Conclusion

- Base conforme aux formes normales (1NF, 2NF, 3NF)
- Séparation claire entre données utilisateur et livres
- Évolutivité assurée : ajout futur de collections, tags, recommandations
- Sauvegarde et restauration simples via Prisma CLI

7.2 Composants d'accès aux données

- L'ORM Prisma facilite l'accès aux données avec un typage strict et une syntaxe fluide. Quelques exemples :
- **`const books = await prisma.book.findMany({ where: { userId } });`**
- **`const user = await prisma.user.findUnique({ where: { id }, include: { books: true } });`**

7.3 Logique métier

- La logique serveur est répartie entre des services modulaires :
- Service Auth : gestion de l'inscription, connexion, vérification email, réinitialisation de mot de passe
- Service Livre : ajout, modification, suppression, filtrage des livres, statut de lecture, évaluation
- Service Email : envoi d'emails de vérification, bienvenue, réinitialisation, changement de mot de passe (via Nodemailer)
- Service API Google Books :
- Appels à <https://www.googleapis.com/books/v1/volumes?q=> pour rechercher un livre
- Parsing des résultats (titre, auteurs, description, image)
- Ajout aux collections utilisateur sans stockage de contenu externe en base

8. Déploiement de l'application

BookNest implémente plusieurs niveaux de sécurité pour protéger les données et garantir une expérience utilisateur fiable.

a. Authentification

- Système basé sur **JWT** avec signature serveur via `JWT_SECRET`.
- Expiration du token : 7 jours.
- Stockage sécurisé côté client (`localStorage`) et en-tête `Authorization`.

b. Validation et filtrage

- Toutes les données envoyées depuis le frontend sont validées serveur.
- Les mots de passe sont hashés avec `bcryptjs` avant stockage.
- L'adresse email est obligatoirement vérifiée pour accéder aux fonctionnalités.

c. Appels API sécurisés

- Les appels à l'API Google Books sont effectués uniquement côté serveur, la clé d'API étant protégée dans le fichier `.env`.

d. Protection contre les attaques

- **CSRF** : les appels frontend utilisent `fetch/axios` avec token **JWT** ; pas de formulaire pur.
- **XSS** : aucun contenu utilisateur n'est affiché sans `sanitize`.
- **SQL Injection** : évité via `Prisma` (ORM sécurisé).
- **Rate limiting** : activable pour les routes sensibles (`auth`, `password reset`).

e. Fichiers et exemples associés

- `middleware/authenticateToken.js` : protège les routes utilisateurs
- `middleware/isAdmin.js` : filtre les droits admin
- `controllers/books.js` : vérifie que l'utilisateur est propriétaire du livre
- `services/email.js` : encapsule l'envoi des emails de vérification

7.4 Sécurité

BookNest implémente plusieurs niveaux de sécurité pour protéger les données et garantir une expérience utilisateur fiable.

a. **Authentification**

- Système basé sur **JWT** avec signature serveur via `JWT_SECRET`.
- Expiration du token : 7 jours.
- Stockage sécurisé côté client (`localStorage`) et en-tête `Authorization`.

b. **Validation et filtrage**

- Toutes les données envoyées depuis le frontend sont validées serveur.
- Les mots de passe sont hashés avec `bcryptjs` avant stockage.
- L'adresse email est obligatoirement vérifiée pour accéder aux fonctionnalités.

c. **Appels API sécurisés**

- Les appels à l'API Google Books sont effectués uniquement côté serveur, la clé d'API étant protégée dans le fichier `.env`.

d. **Protection contre les attaques**

- **CSRF** : les appels frontend utilisent `fetch/axios` avec token **JWT** ; pas de formulaire pur.
- **XSS** : aucun contenu utilisateur n'est affiché sans `sanitize`.
- **SQL Injection** : évité via `Prisma` (ORM sécurisé).
- **Rate limiting** : activable pour les routes sensibles (`auth`, `password reset`).

e. **Fichiers et exemples associés**

- `middleware/authenticateToken.js` : protège les routes utilisateurs
- `middleware/isAdmin.js` : filtre les droits admin
- `controllers/books.js` : vérifie que l'utilisateur est propriétaire du livre
- `services/email.js` : encapsule l'envoi des emails de vérification

MOHAMMED HAMMOUCHE

DWWM

BookNest

juin 2025