BOOK STORE

Rapport de la réalisation d'une application web en React et NodeJS.

Réalisé par: Sara BEN CHIKH
Safae HALLI
Malak SENHAJI

Encadrement: Prof. Abdelmajid Bousselham

Année académique: 2022-2023

contentu

01 Introduction

Notre projet est une application web de Book Store permettant aux utilisateurs de rechercher et d'acheter des livres en ligne. Nous avons utilisé React pour le développement de l'interface utilisateur et Node.JS pour la partie backend.

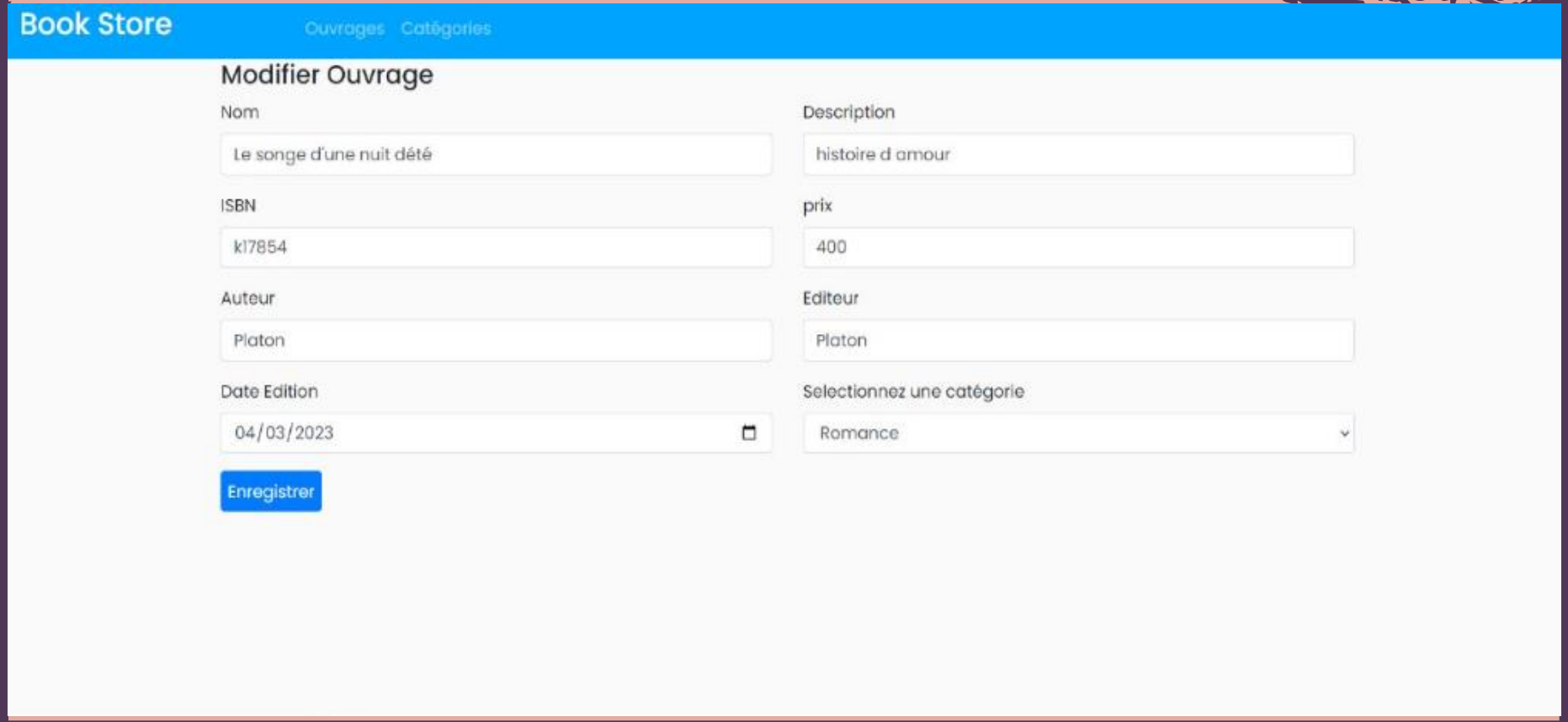
02 Architecture de l'application

L'application suit une architecture client-serveur. Le serveur est développé en Node.js et utilise une base de données MongoDB pour stocker les informations sur les livres, les utilisateurs et les commandes. Le client est développé en React et communique avec le serveur via des requêtes HTTP.

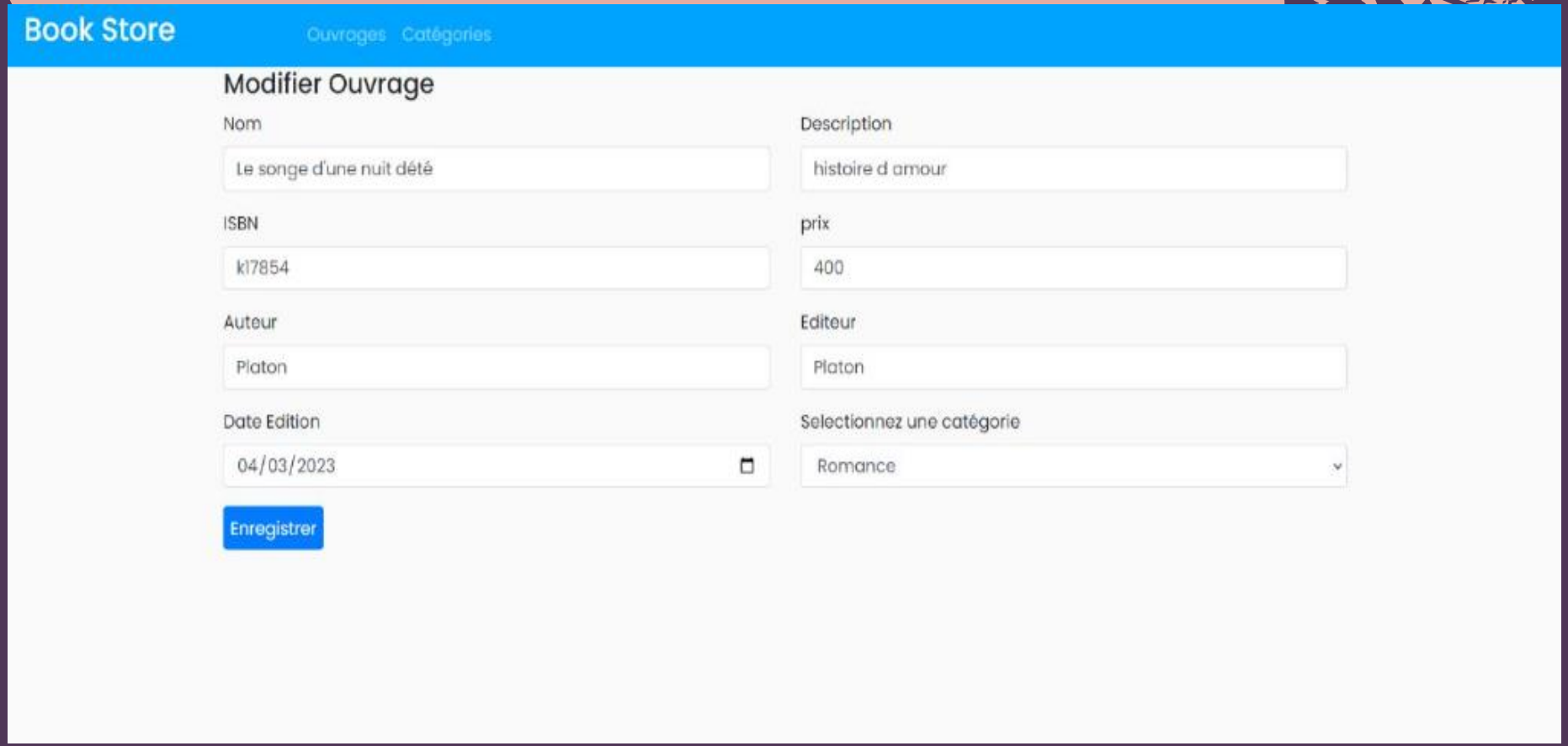
Exemples

Liste des ouvrages

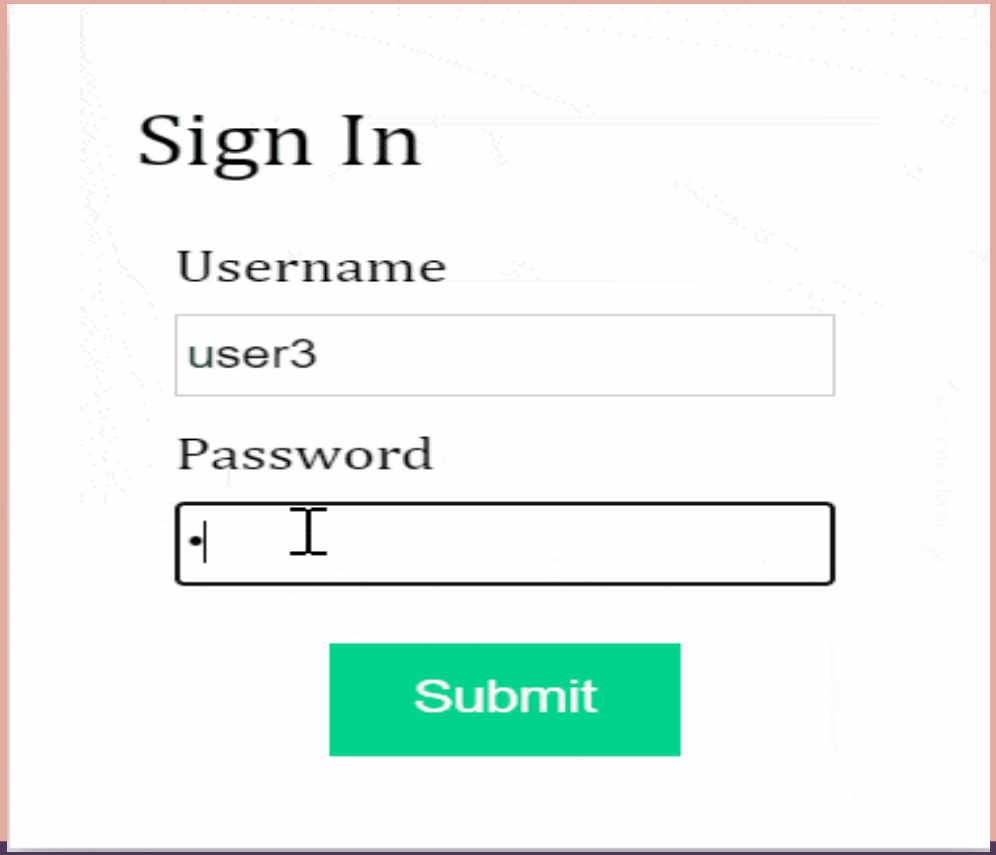
Liste des catégories



Modifier ouvrage



Formulaire d'authentification



03 Fonctionnalités de l'application

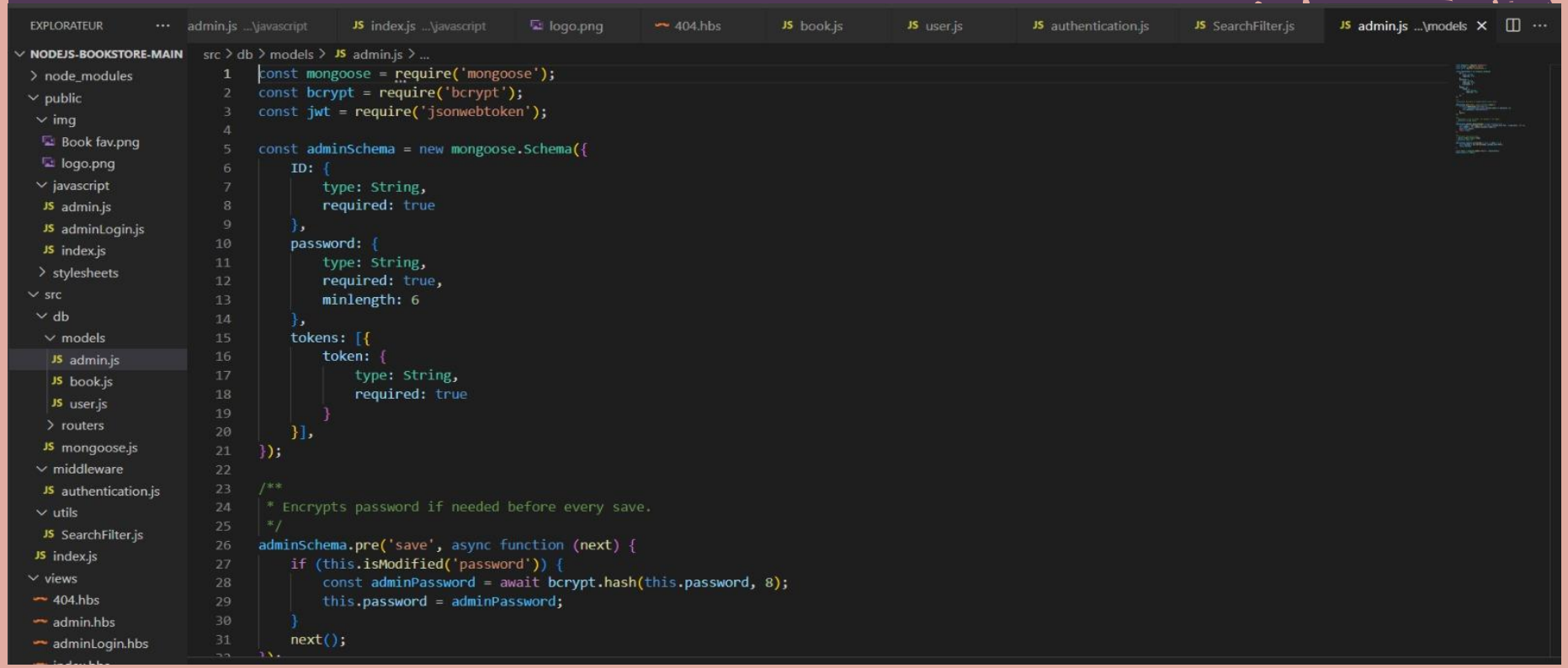
L'application permet aux utilisateurs de :

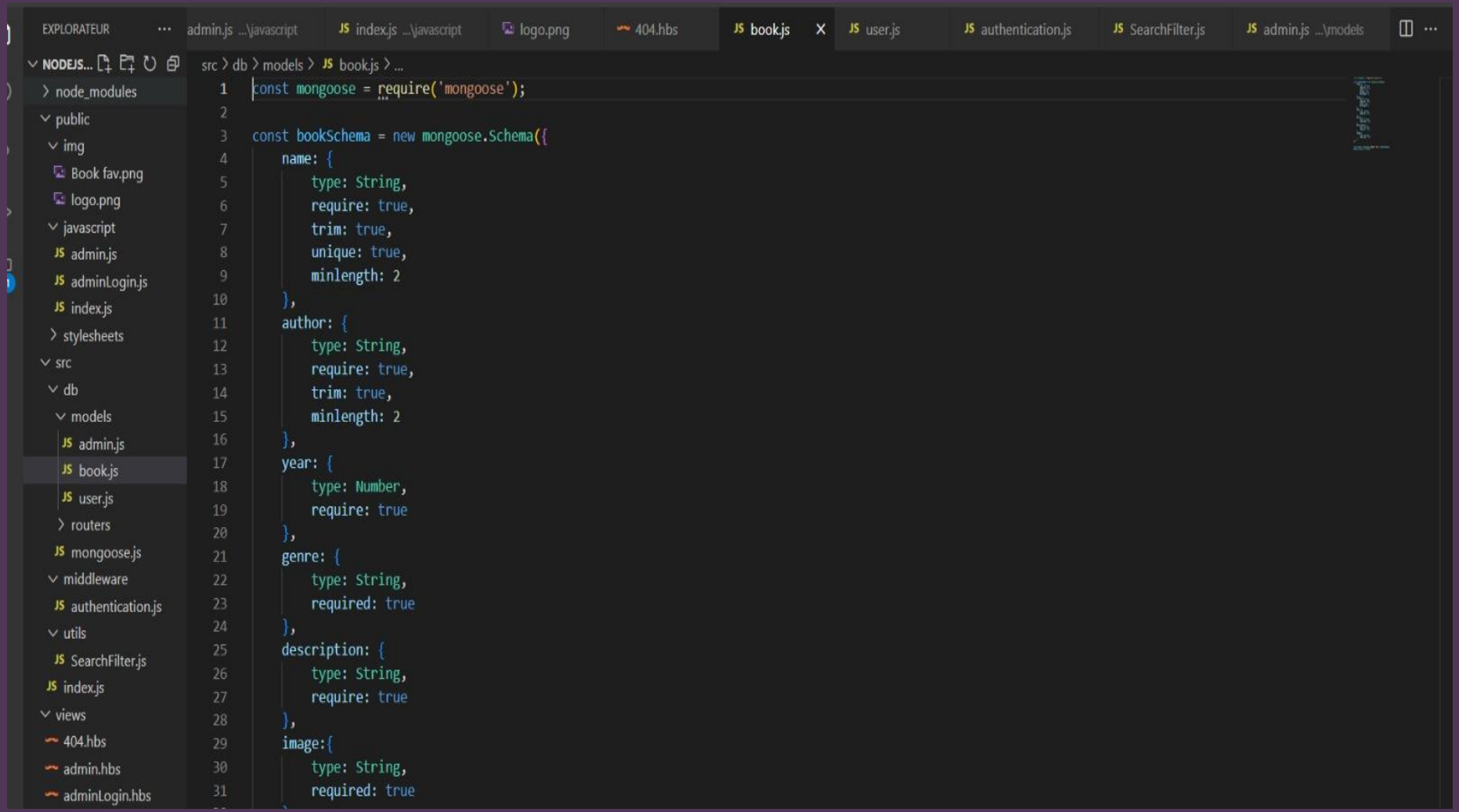
*rechercher des livres par titre, auteur ou catégorie
*consulter les détails des livres, y compris les avis des clients
*ajouter des livres à leur panier d'achat
*passer des commandes et suivre leur état
*consulter leur historique de commande
*gérer leur profil utilisateur

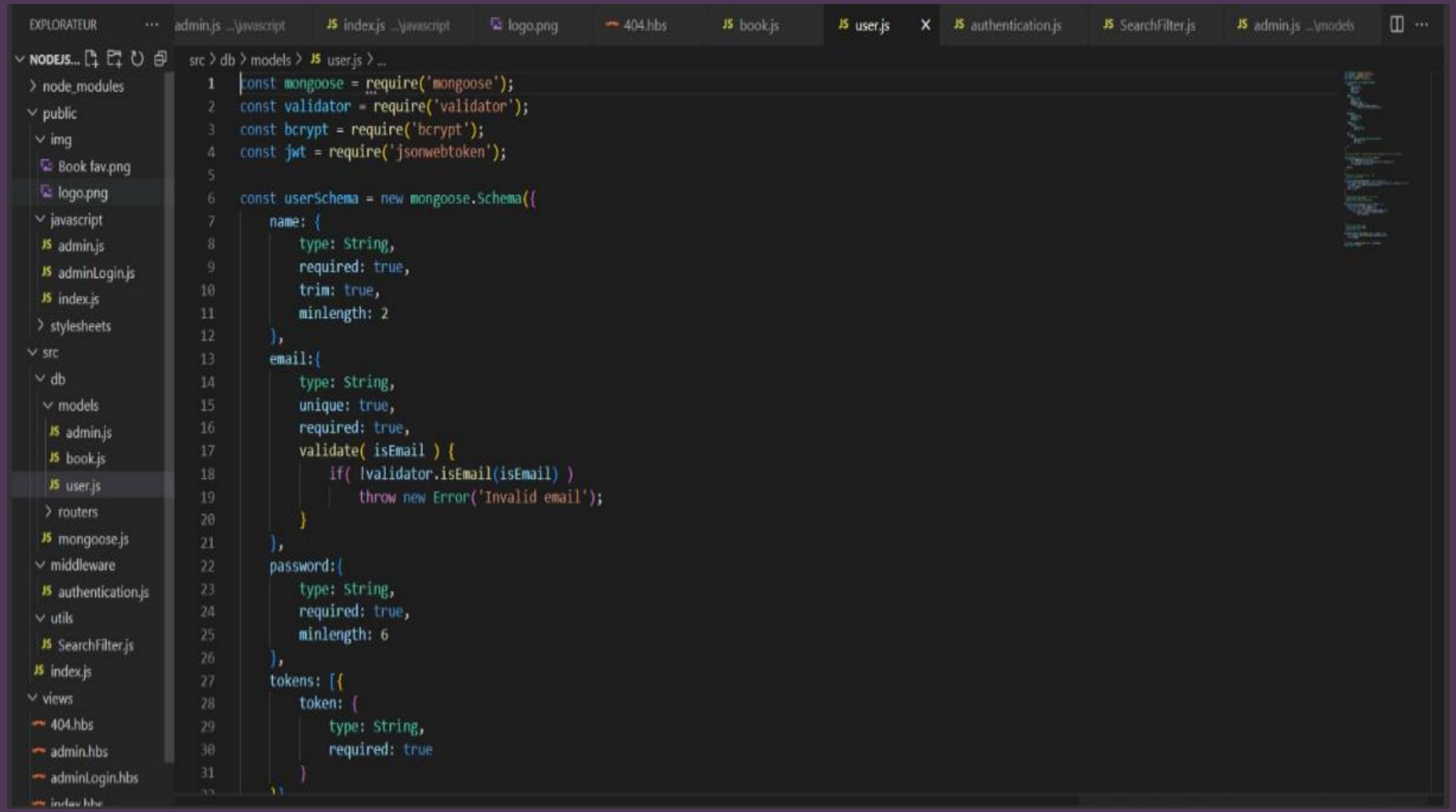
04 Implémentation

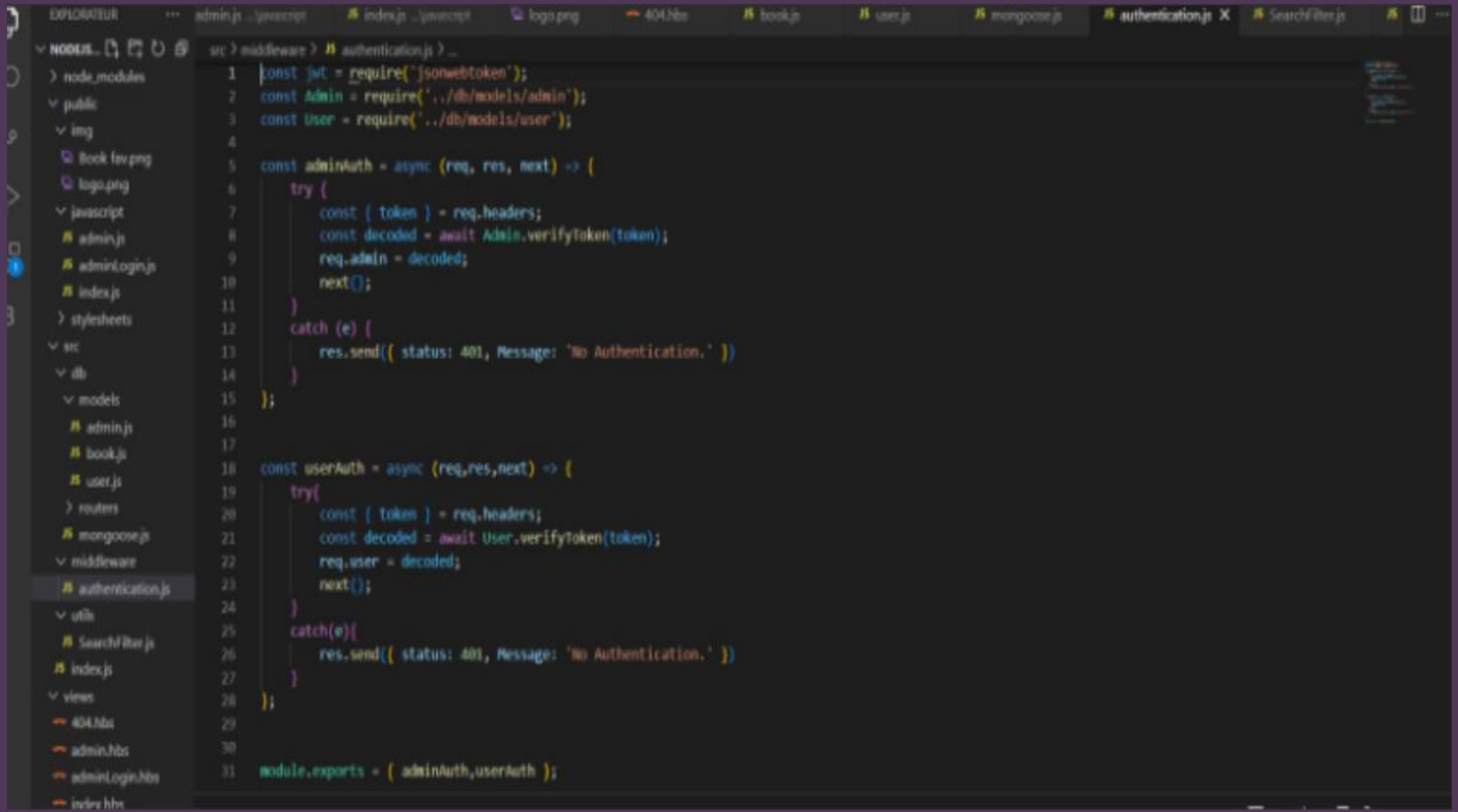
Nous avons utilisé React pour construire l'interface utilisateur. Nous avons créé des composants réutilisables pour afficher les livres, les commandes, les profils utilisateur, etc. Côté serveur, nous avons développé avec Node.js et Express.js + React . Nous avons utilisé MongoDB pour stocker les données des utilisateurs, des livres et des commandes. Nous avons également mis en place une authentification pour sécuriser l'accès aux fonctionnalités protégées.

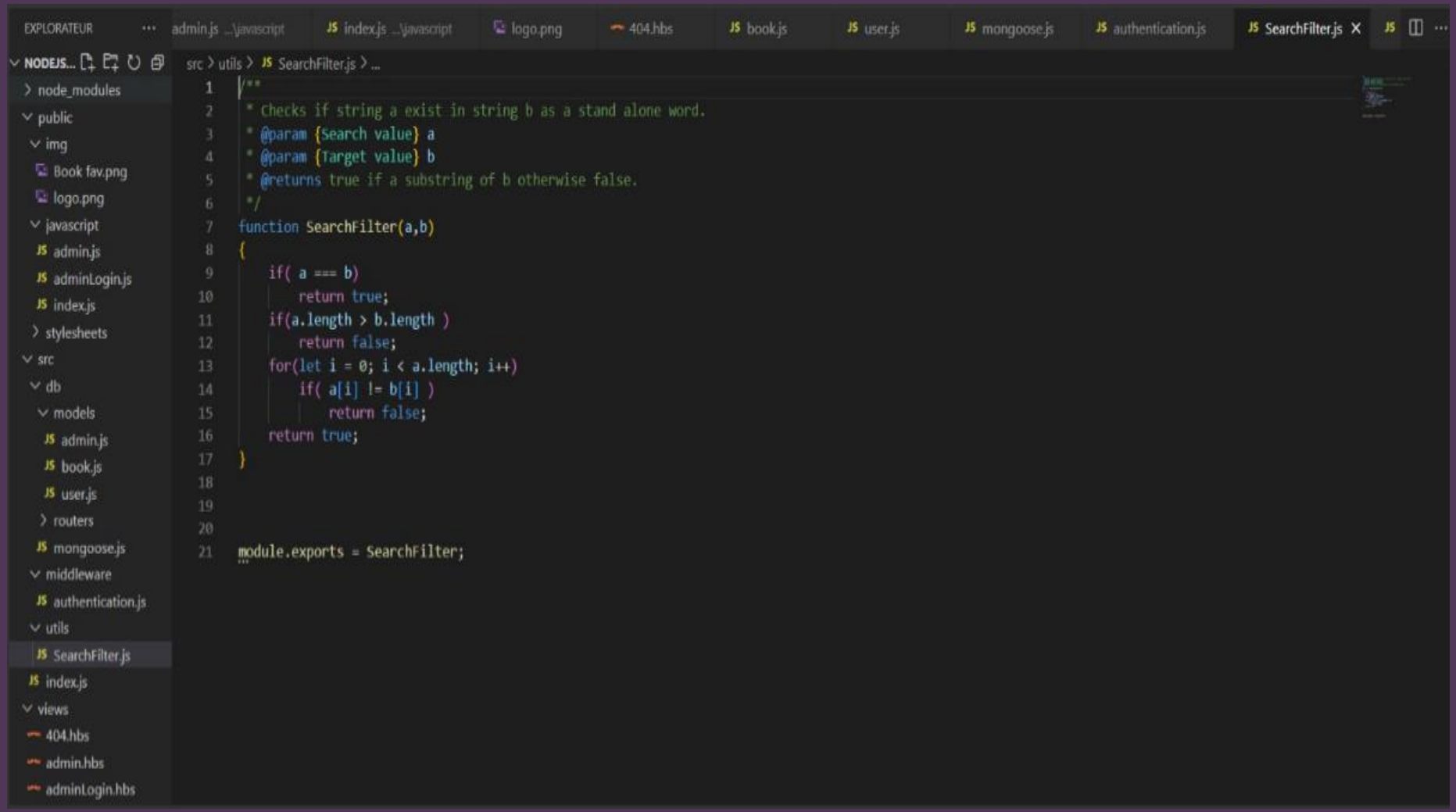
05 Code


EXPLORATEUR
admin.js ...\javascript
index.js ...\javascript
logo.png
404.hbs
book.js
user.js
authentication.js
SearchFilter.js
admin.js ...\models
NODEJS-BOOKSTORE-MAIN
node_modules
public
img
Book fav.png
logo.png
javascript
admin.js
adminLogin.js
index.js
stylesheets
src
db
models
admin.js
book.js
user.js
routers
mongoose.js
middleware
authentication.js
utils
SearchFilter.js
index.js
views
404.hbs
admin.hbs
adminLogin.hbs
src > db > models > admin.js > ...
const mongoose = require('mongoose');
const bcrypt = require('bcrypt');
const jwt = require('jsonwebtoken');
const adminSchema = new mongoose.Schema({
ID: {
type: String,
required: true
},
password: {
type: String,
required: true,
minlength: 6
},
tokens: [{
token: {
type: String,
required: true
}
}],
});
/**
* Encrypts password if needed before every save.
*/
adminSchema.pre('save', async function (next) {
if (this.isModified('password')) {
const adminPassword = await bcrypt.hash(this.password, 8);
this.password = adminPassword;
}
next();


EXPLORATEUR
admin.js ...\javascript
index.js ...\javascript
logo.png
404.hbs
book.js
user.js
authentication.js
SearchFilter.js
admin.js ...\models
NODEJS...
node_modules
public
img
Book fav.png
logo.png
javascript
admin.js
adminLogin.js
index.js
stylesheets
src
db
models
admin.js
book.js
user.js
routers
mongoose.js
middleware
authentication.js
utils
SearchFilter.js
index.js
views
404.hbs
admin.hbs
adminLogin.hbs
src > db > models > book.js > ...
1 const mongoose = require('mongoose');
2
3 const bookSchema = new mongoose.Schema({
4 name: {
5 type: String,
6 require: true,
7 trim: true,
8 unique: true,
9 minlength: 2
10 },
11 author: {
12 type: String,
13 require: true,
14 trim: true,
15 minlength: 2
16 },
17 year: {
18 type: Number,
19 require: true
20 },
21 genre: {
22 type: String,
23 required: true
24 },
25 description: {
26 type: String,
27 require: true
28 },
29 image:{
30 type: String,
31 required: true


EXPLORATEUR
admin.js ...\javascript
index.js ...\javascript
logo.png
404.hbs
book.js
user.js
authentication.js
SearchFilter.js
admin.js ...\models
NODEJS...
node_modules
public
img
Book fav.png
logo.png
javascript
admin.js
adminLogin.js
index.js
stylesheets
src
db
models
admin.js
book.js
user.js
routers
mongoose.js
middleware
authentication.js
utils
SearchFilter.js
index.js
views
404.hbs
admin.hbs
adminLogin.hbs
src > db > models > user.js > ...
const mongoose = require('mongoose');
const validator = require('validator');
const bcrypt = require('bcrypt');
const jwt = require('jsonwebtoken');
const userSchema = new mongoose.Schema({
name: {
type: String,
required: true,
trim: true,
minlength: 2
},
email:{
type: String,
unique: true,
required: true,
validate(isEmail) {
if(!validator.isEmail(isEmail))
throw new Error('Invalid email');
}
},
password:{
type: String,
required: true,
minlength: 6
},
tokens: [{
token: {
type: String,
required: true
}


EXPLORATEUR
admin.js ..\javascript
index.js ..\javascript
logo.png
404.hbs
book.js
user.js
mongoose.js
authentication.js
SearchFilter.js
src > middleware > authentication.js > ...
node_modules
public
img
Book fav.png
logo.png
javascript
admin.js
adminLogin.js
index.js
stylesheets
src
db
models
admin.js
book.js
user.js
routers
mongoose.js
middleware
authentication.js
utils
SearchFilter.js
index.js
views
404.hbs
admin.hbs
adminLogin.hbs
index.hbs
const jwt = require('jsonwebtoken');
const Admin = require('../db/models/admin');
const User = require('../db/models/user');
const adminAuth = async (req, res, next) => {
try {
const { token } = req.headers;
const decoded = await Admin.verifyToken(token);
req.admin = decoded;
next();
}
catch (e) {
res.send({ status: 401, Message: 'No Authentication.' })
}
};
const userAuth = async (req,res,next) => {
try{
const { token } = req.headers;
const decoded = await User.verifyToken(token);
req.user = decoded;
next();
}
catch(e){
res.send({ status: 401, Message: 'No Authentication.' })
}
};
module.exports = { adminAuth,userAuth };


EXPLORATEUR
admin.js ...\javascript
index.js ...\javascript
logo.png
404.hbs
book.js
user.js
mongoose.js
authentication.js
SearchFilter.js
NODEJS...
src > utils > SearchFilter.js > ...
node_modules
public
img
Book fav.png
logo.png
javascript
admin.js
adminLogin.js
index.js
stylesheets
src
db
models
admin.js
book.js
user.js
routers
mongoose.js
middleware
authentication.js
utils
SearchFilter.js
index.js
views
404.hbs
admin.hbs
adminLogin.hbs
/**
* Checks if string a exist in string b as a stand alone word.
* @param {Search value} a
* @param {Target value} b
* @returns true if a substring of b otherwise false.
*/
function SearchFilter(a,b)
{
if(a === b)
return true;
if(a.length > b.length)
return false;
for(let i = 0; i < a.length; i++)
if(a[i] != b[i])
return false;
return true;
}
module.exports = SearchFilter;

06 Conclusion

En conclusion, notre application Book Store est une plateforme en ligne conviviale pour les utilisateurs qui cherchent à acheter des livres. Nous avons utilisé React et Node.js pour offrir une expérience utilisateur optimale, et avons mis en place des fonctionnalités robustes telles que la recherche de livres, la gestion du panier d'achat, la passation de commandes et la gestion du profil utilisateur.