

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

contenu

01

Introduction

02

Architecture de
l'application

03

Fonctionnalités
de l'application

04

Implémentation

05

Code

06

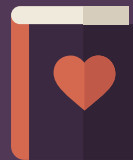
Conclusion

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.

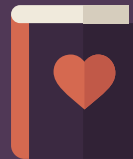


02

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

Book Store

Ouvrages Catégories

Liste ouvrages

#	Nom	ISBN	Auteur	Editeur	prix	Categorie	Action
1	Le songe d'une nuit d'été	k17854	Platon	Platon	400	Romance	Delete Edit
2	La chartreuse de parme	l7854	Sheakspear	Sheakspear	250	Romance	Delete Edit
#	Nom	ISBN	Auteur	Editeur	prix	Categorie	Action

Ajouter

Liste des catégories

Book Store

Ouvrages Catégories

Modifier Ouvrage

Nom

Le songe d'une nuit d'été

ISBN

k17854

Auteur

Platon

Date Edition

04/03/2023



Enregistrer

Description

histoire d amour

prix

400

Editeur

Platon

Sélectionnez une catégorie

Romance



Modifier ouvrage

Book Store

[Ouvrages](#) [Catégories](#)

Modifier Ouvrage

Nom

Description

ISBN

prix

Auteur

Editeur

Date Edition



Selectionnez une catégorie



Enregistrer

Formulaire d'authentification

Sign In

Username

Password

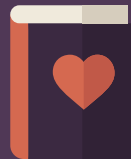
Submit

03

Fonctionnalités de l'application

L'application permet aux utilisateurs de :

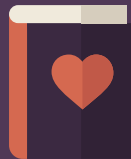
- *rechercher des livres par titre, auteur ou catégorie
- *consulter les détails des livres, y compris les avis des clients
- *ajouter des livres à leur panier d'achat
- *passer des commandes et suivre leur état
- *consulter leur historique de commande
- *gérer leur profil utilisateur



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 JS index.js ...\javascript logo.png 404.hbs JS book.js JS user.js JS authentication.js JS SearchFilter.js JS admin.js ...\models

▼ NODEJS-BOOKSTORE-MAIN
 > node_modules
 > public
 > img
 Book fav.png
 logo.png
 > javascript
 JS admin.js
 JS adminLogin.js
 JS index.js
 > stylesheets
 > src
 > db
 > models
 JS admin.js
 JS book.js
 JS user.js
 > routers
 JS mongoose.js
 > middleware
 JS authentication.js
 > utils
 JS SearchFilter.js
 > JS index.js
 > views
 404.hbs
 admin.hbs
 adminLogin.hbs
 index.hbs

src > db > models > JS admin.js > ...

```
1  const mongoose = require('mongoose');
2  const bcrypt = require('bcrypt');
3  const jwt = require('jsonwebtoken');
4
5  const adminSchema = new mongoose.Schema({
6    ID: {
7      type: String,
8      required: true
9    },
10   password: {
11     type: String,
12     required: true,
13     minlength: 6
14   },
15   tokens: [{
16     token: {
17       type: String,
18       required: true
19     }
20   }],
21 });
22
23 /**
24  * Encrypts password if needed before every save.
25  */
26 adminSchema.pre('save', async function (next) {
27   if (this.isModified('password')) {
28     const adminPassword = await bcrypt.hash(this.password, 8);
29     this.password = adminPassword;
30   }
31   next();
32 });
```

EXPLORATEUR... admin.js ...\javascript JS index.js ...\javascript logo.png 404.hbs JS book.js X JS user.js JS authentication.js JS SearchFilter.js JS admin.js ...\models

▼ NODEJS...
node_modules
public
img
Book fav.png
logo.png
javascript
JS admin.js
JS adminLogin.js
JS index.js
stylesheets
src
db
models
JS admin.js
JS book.js
JS user.js
routers
JS mongoose.js
middleware
JS authentication.js
utils
JS SearchFilter.js
JS index.js
views
404.hbs
admin.hbs
adminLogin.hbs

src > db > models > JS 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
32 }
33 });

EXPLORERadmin.js ...\javascriptJS index.js ...\javascriptlogo.png404.hbsJS book.jsJS user.jsXJS authentication.jsJS SearchFilter.jsJS admin.js ...\models

NODEJS...node_modulespublicimgBook fav.pnglogo.pngjavascriptJS admin.jsJS adminLogin.jsJS index.jsstylesheetssrcdbmodelsadmin.jsbook.jsuser.jsroutersJS mongoose.jsmiddlewareJS authentication.jsutilsJS SearchFilter.jsJS index.jsviews404.hbsadmin.hbsadminLogin.hbsindex.hbs

src > db > models > JS user.js > ...

```
1 const mongoose = require('mongoose');
2 const validator = require('validator');
3 const bcrypt = require('bcrypt');
4 const jwt = require('jsonwebtoken');
5
6 const userSchema = new mongoose.Schema({
7   name: {
8     type: String,
9     required: true,
10    trim: true,
11    minlength: 2
12  },
13  email: {
14    type: String,
15    unique: true,
16    required: true,
17    validate( isEmail ) {
18      if( !validator.isEmail(isEmail) )
19        throw new Error('Invalid email');
20    }
21  },
22  password: {
23    type: String,
24    required: true,
25    minlength: 6
26  },
27  tokens: [{
28    token: {
29      type: String,
30      required: true
31    }
32  }
33 ]
34 }
```

EXPLORERadmin.js ... javascriptindex.js ... javascriptlogo.png404.hbsbook.jsuser.jsmongoose.jsauthentication.jsXSearchFilter.js

node_modulespublicimgBook fav.pnglogo.pngjavascriptadmin.jsadminLogin.jsindex.jsstylesheetssrcdbmodelsadmin.jsbook.jsuser.jsroutersmongoose.jsmiddlewareauthentication.jsutilsSearchFilter.jsindex.jsviews404.hbsadmin.hbsadminLogin.hbsindex.hbs

src > middleware > authentication.js > ...

```
1 const jwt = require('jsonwebtoken');
2 const Admin = require('../db/models/admin');
3 const User = require('../db/models/user');
4
5 const adminAuth = async (req, res, next) => {
6   try {
7     const { token } = req.headers;
8     const decoded = await Admin.verifyToken(token);
9     req.admin = decoded;
10    next();
11  }
12  catch (e) {
13    res.send({ status: 401, Message: 'No Authentication.' });
14  }
15 };
16
17
18 const userAuth = async (req, res, next) => {
19   try{
20     const { token } = req.headers;
21     const decoded = await User.verifyToken(token);
22     req.user = decoded;
23     next();
24   }
25   catch(e){
26     res.send({ status: 401, Message: 'No Authentication.' });
27   }
28 };
29
30
31 module.exports = { adminAuth, userAuth };
```

EXPLORERadmin.js ... javascriptJS index.js ... javascriptlogo.png404.hbsJS book.jsJS user.jsJS mongoose.jsJS authentication.jsJS SearchFilter.js XJS ...

NODES...src > utils > JS SearchFilter.js > ...

> node_modules

> public

> img

Book fav.png

logo.png

> javascript

JS admin.js

JS adminLogin.js

JS index.js

> stylesheets

> src

> db

> models

JS admin.js

JS book.js

JS user.js

> routers

JS mongoose.js

> middleware

JS authentication.js

> utils

JS SearchFilter.js

JS index.js

> views

404.hbs

admin.hbs

adminLogin.hbs

```
1  /**
2   * Checks if string a exist in string b as a stand alone word.
3   * @param {Search value} a
4   * @param {Target value} b
5   * @returns true if a substring of b otherwise false.
6   */
7  function SearchFilter(a,b)
8  {
9      if( a === b)
10         return true;
11         if(a.length > b.length )
12             return false;
13         for(let i = 0; i < a.length; i++)
14             if( a[i] != b[i] )
15                 return false;
16         return true;
17     }
18
19
20
21     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.

