

Front End DEVELOPMENT



Projet FrontEnd 294

Dario Chasi – Nima Zarrabi – Thibaud Racine – William Trelles

ETML – Vennes – Classe CID2A

18/03/2024 – 27/05/2024

Client - Grégory Charmier

Introduction

Réalisation du FrontEnd d'une application permettant de partager sa passion pour la lecture.

Le BackEnd de l'application à déjà été effectuer dans un projet passé, il doit être adapté aux demandes et mis en lien avec le FrontEnd.

Analyse

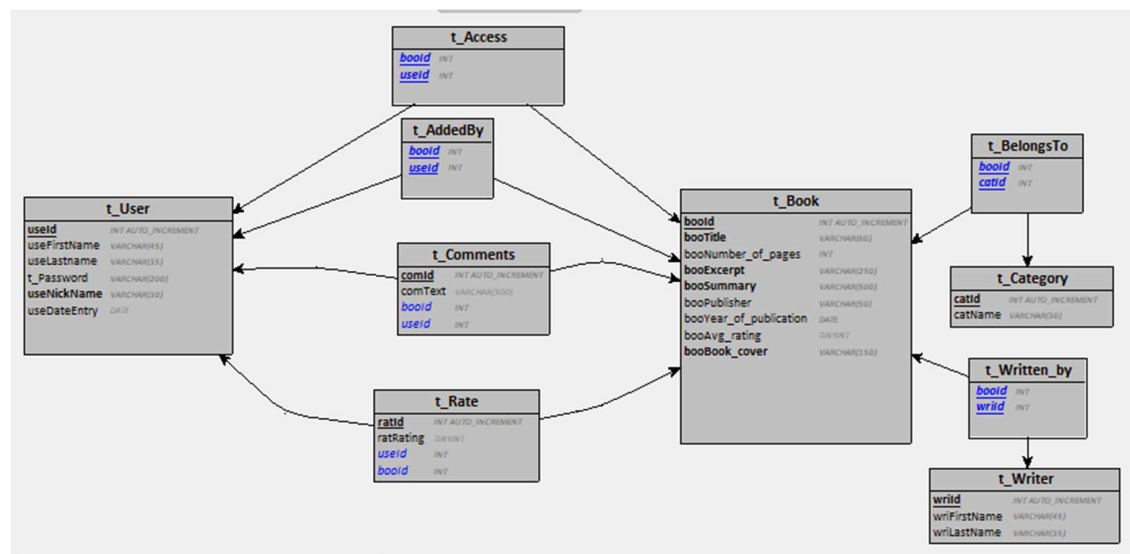
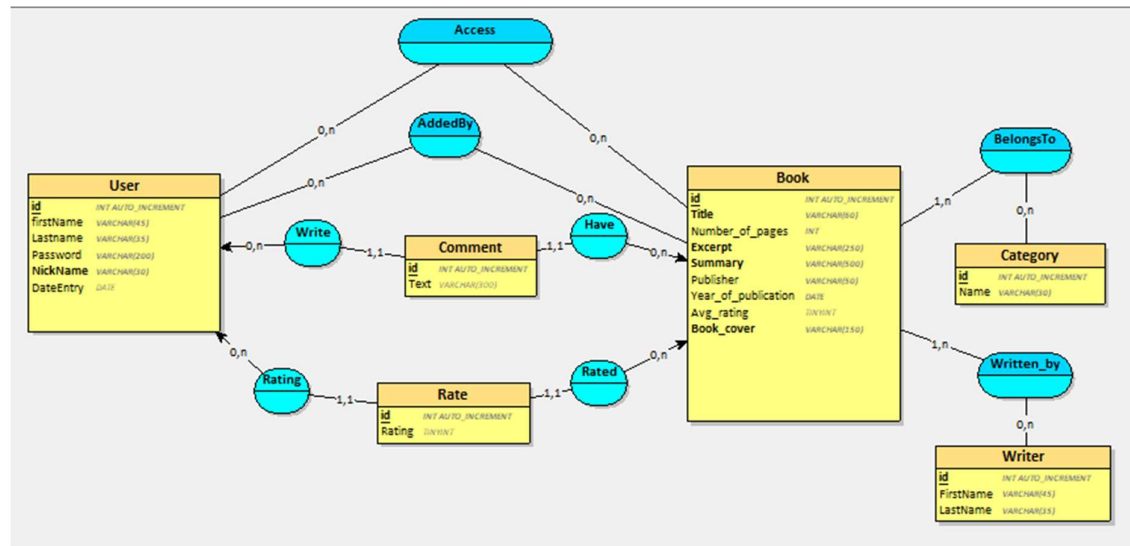
Le FrontEnd doit être effectué en Node + VueJS, et doit permettre aux utilisateurs de visualiser une liste d'ouvrages stockés dans une base de données, ainsi de pouvoir ajouter, modifier et supprimer de ces ouvrages.

Les pages requises sont les suivantes : une page d'accueil, une page où seront affichés les livres, une page pour l'ajout d'un livre et une page pour la suppression des livres, une page qui affiche les détails d'un livre et une page pour "se logger".

Planification

Nos taches ont été planifiées sur Trello [lien au Trello](#). Cet outil nous a permis de créer des tâches que l'on s'est assigner, puis les valider une fois terminer.

Analyse de de la base de données



L'analyse de la base de données présentée montre une structure conçue pour gérer les informations concernant à des utilisateurs, des écrivains, des catégories d'ouvrages, des ouvrages, ainsi que des commentaires et appréciations des utilisateurs sur ces ouvrages. Voici une analyse des différentes tables et de leur rôle au sein de la base de données :

Table t_User : stocke les informations sur les utilisateurs.

Champs : Comprend l'id de l'utilisateur, le nom, le prénom, un pseudonyme unique, la date d'entrée, le nombre de commentaires, et le nombre d'appréciations postés par l'utilisateur.

Analyse : La contrainte d'unicité sur le pseudonyme garantit que chaque utilisateur est unique dans le système. Ainsi, le nom et le prénom sont des champs qui ne

peuvent pas être de valeur nulle pour éviter des doublons involontaires ou des confusions lors de l'identification des utilisateurs.

Table t_Writer : contient les informations sur les écrivains.

Champs : Inclut un id écrivain auto-incrémenté, ainsi que le nom et le prénom de l'écrivain.

Analyse : Cette structure est simple et efficace pour identifier de manière unique chaque écrivain.

Table t_Category : elle gère les différentes catégories d'ouvrages.

Champs : Comprend un id de catégorie auto-incrémenté et le nom de la catégorie.

Analyse : Permet de classer les ouvrages en catégories pour une recherche et une organisation facile.

Table t_Book : cette table détaille les informations sur chaque ouvrage.

Champs : Contient un id ouvrage, le titre qui est unique, un lien vers un extrait, le nombre de pages, un synopsis, le nom de l'éditeur, la date d'édition, une moyenne d'appréciations, un lien vers la couverture, et les clés étrangères pour l'écrivain et la catégorie.

Analyse : La table est structurée pour offrir une vue détaillée de chaque ouvrage, incluant des relations vers l'écrivain et la catégorie. La contrainte d'unicité sur le titre garantit qu'aucun ouvrage ne soit enregistré plus d'une fois.

Table t_Access : trace quels utilisateurs ont accédé à quels ouvrages.

Analyse : Crée une relation entre les utilisateurs et les ouvrages, permettant de suivre les interactions des utilisateurs avec différents ouvrages.

Table t_Comments : cette table stock les commentaires des utilisateurs sur les ouvrages.

Champs : Associe chaque commentaire à un utilisateur et un ouvrage spécifique.

Analyse : Permet de recueillir des retours détaillés des utilisateurs sur les ouvrages, enrichissant ainsi la base de données avec des opinions et critiques constructives.

Table t_Rate : elle enregistre les appréciations des utilisateurs pour chaque ouvrage.

Champs : Lie chaque appréciation à un utilisateur et un ouvrage, avec une valeur d'appréciation.

Analyse : Offre un moyen d'évaluer la réception des ouvrages par les utilisateurs, ce qui permet de calculer la moyenne d'appréciations stockée dans t_ouvrage.

Table t_Written_by : trace quels utilisateurs ont ajouté quels ouvrages.

Analyse : Crée une relation entre les utilisateurs et les ouvrages, permettant de suivre les interactions des utilisateurs avec différents ouvrages.

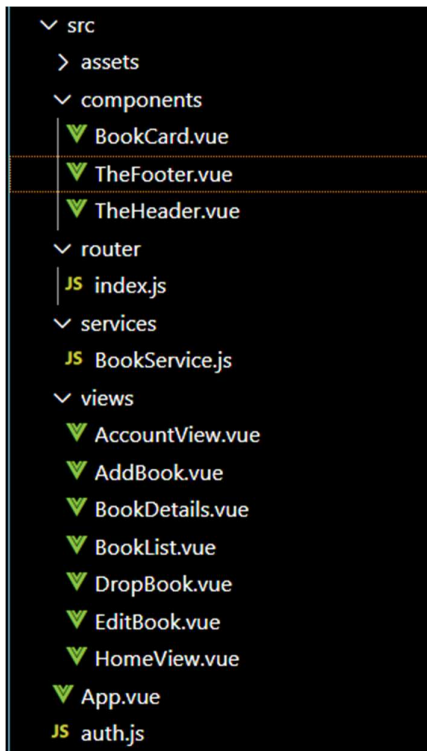
Table t_BelongsTo : trace la catégorie de l'ouvrage.

Analyse : Crée une relation entre les catégories et les ouvrages, permettant de suivre les interactions des catégories avec différents ouvrages.

Table t_AddedBy : trace quels écrivains ont ajouté quels ouvrages.

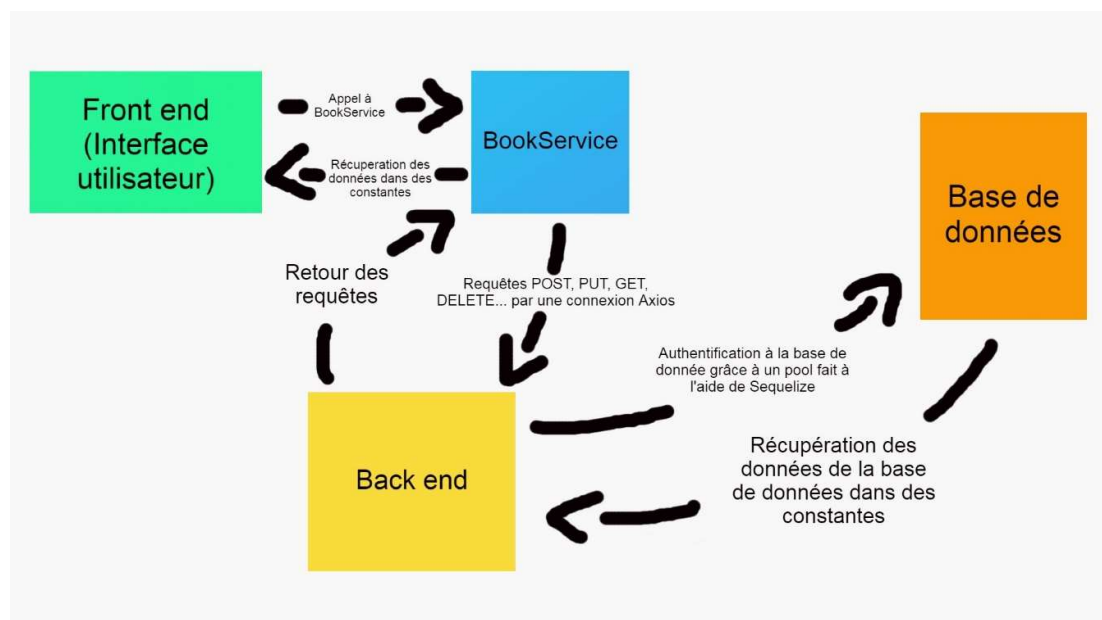
Analyse : Crée une relation entre les écrivains et les ouvrages, permettant de suivre les interactions des écrivains avec différents ouvrages.

Analyse de la structure du code



Nous avons séparé les différents fichiers dans leurs respective endroit par rapport à son fonctionnement dans le code.

Schéma de l'architecture représentant les interactions entre le frontend et le backend



Cors

Pour permettre une connexion entre notre frontend et notre backend, nous avons dû utiliser CORS, ce qui nous a permis d'accéder correctement à notre site web.

```
import cors from "cors";
var corsOptions = {
  origin: `http://localhost:5173`,
  optionsSuccessStatus: 200, // For legacy browser support
};

app.use(express.json());
app.use(cors(corsOptions));
import { initDb, sequelize } from "../db/sequelize.mjs";
```

Multer

Upload de l'image dans le frontend

```
const uploadImage = async (imageFile) => {
  const formData = new FormData()
  formData.append('image', imageFile)

  try {
    const response = await apiClient.post('/upload', formData, {
      headers: {
        'Content-Type': 'multipart/form-data'
      }
    })

    console.log('Image correctement chargée:', response.data)
    return response.data.url
  } catch (error) {
    console.error("Erreur lors du chargement de l'image:", error)
    throw error
  }
}
```

Routes

Routes du backend, reliés au frontend

```
export default {
  // get tous les books
  getBooks() {
    | return apiClient.get('/books')
  },
  getCategorie(id) {
    | return apiClient.get('/categories/' + id)
  },
  getBook(id) {
    | return apiClient.get('/books/' + id, {
    |   headers: {
    |     'Content-Type': 'multipart/form-data'
    |   }
    | })
  },
  getBookRates(id) {
    | return apiClient.get('/books/${id}/rates')
  },
  getBookComments(id) {
    | return apiClient.get('/books/${id}/comments')
  },
  insertCover(image) {
    | const formData = new FormData()
    | formData.append('image', image)

    | return apiClient.post('/books/upload', formData, {
    |   headers: {
    |     'Content-Type': 'multipart/form-data'
    |   }
    | })
  },
  createBook(newBook) {
    | return apiClient.post('/books/', newBook, {
    |   headers: {
    |     'Content-Type': 'multipart/form-data'
    |   }
    | })
  },
}
```

```
editBook(id, newBook) {
  | return apiClient.put('/books/' + id, newBook, {
  |   headers: {
  |     'Content-Type': 'multipart/form-data'
  |   }
  | })
},
deleteBook(id) {
  | return apiClient.delete('/books/' + id)
},
getCategories() {
  | return apiClient.get('/categories')
},
getCategory(id) {
  | return apiClient.get('/categories/' + id)
},
getCategoryByName(name) {
  | return apiClient.get('/categories/name/${name}')
},
getWriters() {
  | return apiClient.get('/authors')
},
getWriter(id) {
  | return apiClient.get('/authors/' + id)
},
getWriterByFirstname(firstname) {
  | return apiClient.get('/authors/firstname/${firstname}')
},
getUser(id) {
  | return apiClient.get('/users/' + id)
},
login(credentials) {
  | return apiClient.post('/login', credentials)
}
```


Réalisation

Sécurité

Token

Notre site nécessite des token JWT afin d'accéder à ses informations

```
const token = jwt.sign(  
  { userId: user.id, isAdmin: user.isAdmin },  
  privateKey,  
  {  
    expiresIn: "1y",  
  }  
);
```

Authentification

Différents moyens d'authentification ont été mis en place pour vérifier l'identité de l'utilisateur :

```
JS authAdmin.mjs  
JS authBooks.mjs  
JS authComments.mjs  
JS authRates.mjs  
JS authVer.mjs
```

Nos différents niveaux d'authentification nous permettent de vérifier si l'utilisateur possède son token, que nous recevons lors de la connexion. Nous avons également une vérification pour les administrateurs qui nous permet de vérifier le rôle de l'utilisateur. Enfin, nous effectuons des vérifications pour nous assurer que l'utilisateur est bien l'auteur du livre ou la personne qui a posté l'appréciation ou le commentaire, tout en attribuant tous les droits aux administrateurs.

Connexion

Pour la connexion nous avons rajouter un login qui stocke le token de l'utilisateur grâce à localStorage que nous avons dû implémenter dans notre code

```
import axios from 'axios'

const apiClient = axios.create({
  baseURL: 'http://localhost:3901/api',
  withCredentials: false,
  headers: {
    Accept: 'application/json',
    'Content-Type': 'application/json; charset=utf-8'
  }
})

apiClient.interceptors.request.use((config) => {
  const token = localStorage.getItem('jwt')
  if (token) {
    config.headers.Authorization = `Bearer ${token}`
  }
  return config
})
```

Validation des données

Comme mesure de sécurité, nous avons ajouté cette vérification pour tous les champs de type "string" :

```
title: {
  type: DataTypes.STRING,
  allowNull: false,
  unique: {
    msg: "Ce titre est déjà utilisé.",
  },
  validate: {
    is: {
      args: /^[A-Za-z0-9\s]/,
      msg: "Seules les lettres, les chiffres et les espaces sont autorisées.",
    },
    notEmpty: {
      msg: "Le titre du livre ne peut pas être vide",
    },
    notNull: {
      msg: "Le titre du livre est une propriété obligatoire.",
    },
    len: {
      args: [0, 60],
      msg: "Le titre du livre doit contenir au maximum 60 caractères.",
    },
  },
},
```

L'argument `/^[A-Za-z0-9\s]/` sert à s'assurer que seul des lettres de A à Z majuscule et minuscules sont autorisés, ainsi que les chiffres de 0 à 9.

La partie finale `"\s"` signifie tout caractère d'espacement, eux aussi sont donc autorisés.

Empêcher l'envoi de données vides, nulles ou limiter le nombre de caractères nous permettra d'éviter des problèmes potentiels.

Cette validation ajoute une couche supplémentaire de protection à notre base de données contre les injections SQL dans les formulaires.

Afin de modifier ou supprimer un livre, l'utilisateur doit être authentifié avec le compte de l'utilisateur qui a ajouté ce livre. [\[06\]](#)

Hashage

Hashage de mot passe :

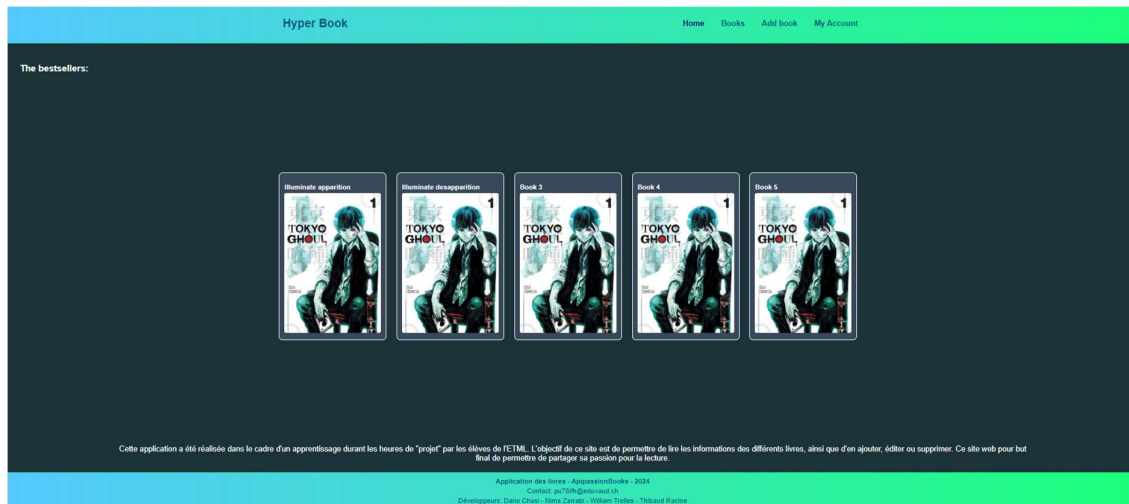
```
const importUsers = async () => {
  for (const user of users) {
    try {
      const hash = await bcrypt.hash(user.password, 10); // temps pour hasher = du sel
      const createdUser = await User.create({
        firstName: user.firstName,
        lastName: user.lastName,
        nickName: user.nickName,
        email: user.email,
        password: hash,
        dateEntry: user.dateEntry,
        isAdmin: user.isAdmin,
      });
      console.log("Utilisateur créé:", createdUser.toJSON());
    } catch (error) {
      console.error("Problème lors de la création de l'utilisateur:", error);
    }
  }
};
```

Pour maintenir une meilleure sécurité des mots de passe, avant de les insérer dans notre base de données, nous les hashons avec du sel pour renforcer cette étape de hashage.

Fonctionnalités techniques

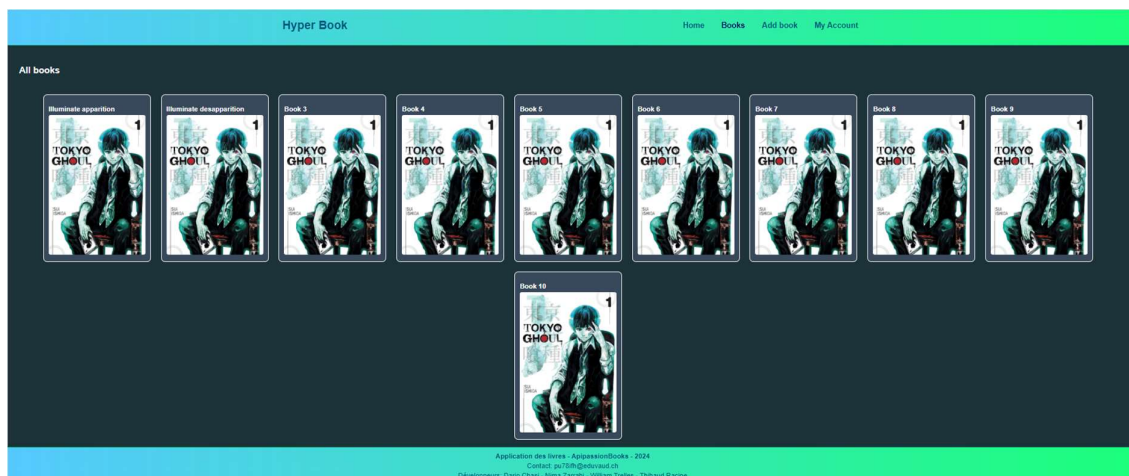
Page d'accueil

La page d'accueil fourni aux utilisateurs une compréhension de l'utilité du site. Elle inclut une section explicative qui décrit les objectifs et les fonctionnalités principales du site, permettant aux visiteurs de savoir comment ils peuvent en bénéficier. De plus, la page d'accueil affiche les cinq derniers ouvrages ajoutés, accessibles à tout public.



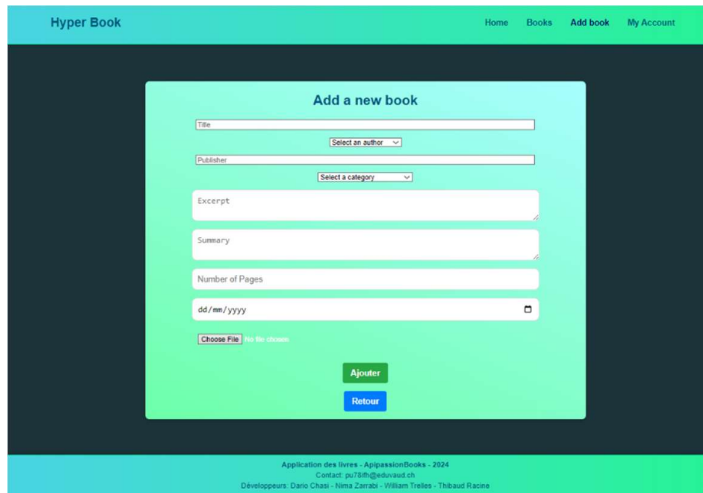
Page avec la liste de livres

Une page contenant la liste de livres est présente dans le site avec leur cover et une disposition visuellement agréable pour l'utilisateur.



Page d'ajout de livre

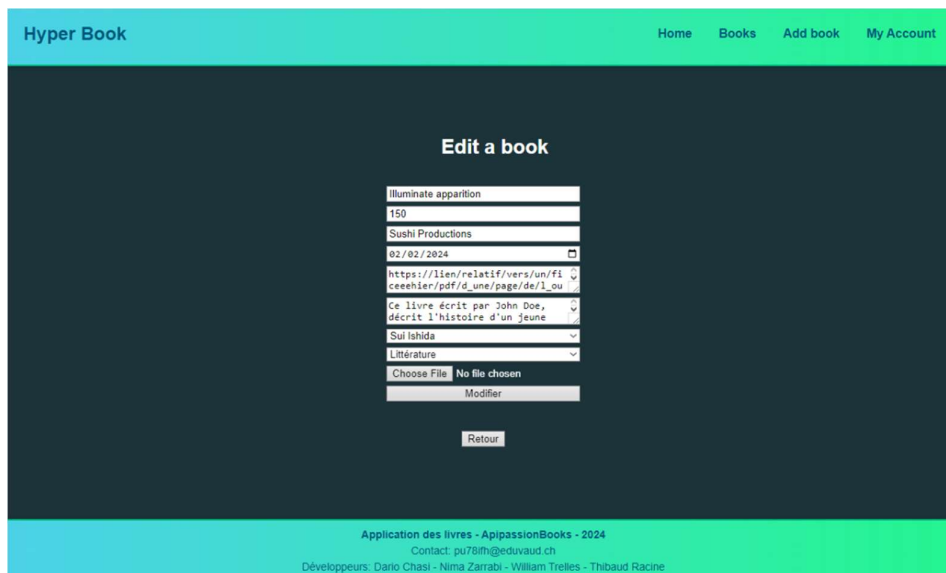
Cette fonctionnalité permet aux utilisateurs autorisés d'ajouter de nouveaux livres à la base de données du site. Elle comprend un formulaire où l'on peut entrer des détails tels que le titre, l'auteur, la catégorie, un résumé, la date d'ajout, le nombre de pages et télécharger une couverture.



The screenshot shows the 'Add a new book' form in the Hyper Book application. The form is titled 'Add a new book' and is located in the center of the page. It contains several input fields: 'Title', 'Publisher', 'Select an author' (a dropdown menu), 'Select a category' (a dropdown menu), 'Excerpt', 'Summary', 'Number of Pages', and 'dd/mm/yyyy' (a date field). There is also a 'Choose File' button for uploading a cover image. At the bottom of the form are two buttons: 'Ajouter' (Add) and 'Retour' (Return). The page has a dark blue header with the 'Hyper Book' logo and navigation links: 'Home', 'Books', 'Add book', and 'My Account'. The footer contains copyright information: 'Application des livres - ApipassionBooks - 2024', 'Contact: pu78fh@eduvaud.ch', and 'Développeurs: Dario Chasi - Nima Zarrabi - William Trelles - Thibaud Racine'.

Page de modification

Une page de modification de livre est accessible depuis la page des détails d'un livre, elle contient un formulaire qui permet à un utilisateur de modifier un livre existant.



The screenshot shows the 'Edit a book' form in the Hyper Book application. The form is titled 'Edit a book' and is located in the center of the page. It contains several input fields: 'Illuminate apparition', '150', 'Sushi Productions', '02/02/2024' (a date field), 'https://lien/relatif/vers/un/fi...' (a URL field), 'Ce livre écrit par John Doe, décrit l'histoire d'un jeune', 'Sui Ishida', 'Littérature', and 'Choose File' (a file upload button). At the bottom of the form are two buttons: 'Modifier' (Modify) and 'Retour' (Return). The page has a dark blue header with the 'Hyper Book' logo and navigation links: 'Home', 'Books', 'Add book', and 'My Account'. The footer contains copyright information: 'Application des livres - ApipassionBooks - 2024', 'Contact: pu78fh@eduvaud.ch', and 'Développeurs: Dario Chasi - Nima Zarrabi - William Trelles - Thibaud Racine'.

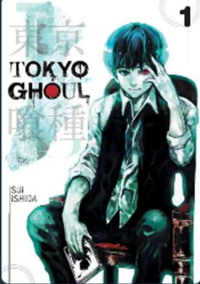
Page vue détail

Une page de détail d'un livre est présente quand on clique sur un livre, elle contient les différentes informations du livre, ainsi que ses Évaluations et leurs moyennes.

Cette page contient aussi un bouton pour modifier le livre en question, ainsi qu'un pour le supprimer.

Hyper BookHomeBooksAdd bookMy Account

Illuminate apparition



Résumé: Ce livre écrit par John Doe, décrit l'histoire d'un jeune homme...

Catégorie: Littérature

Auteur: Sui Ishida
Éditeur: Sushi Productions
Pages: 150
Posté par: etml5

Évaluations:

Moyenne des évaluations: 0
Utilisateur 1: 4 étoiles
Utilisateur 2: 5 étoiles

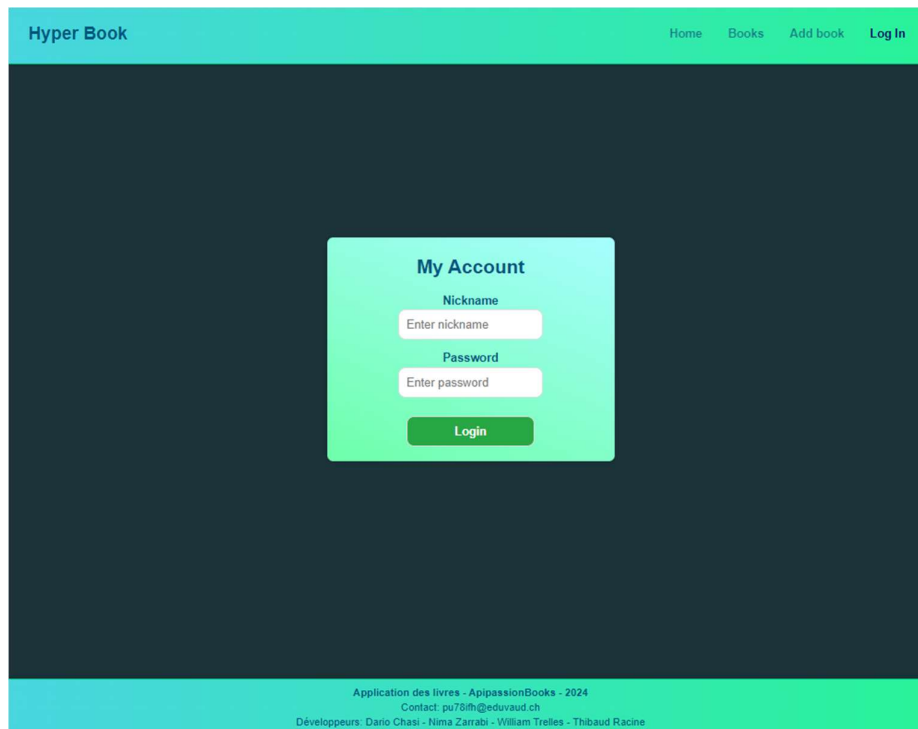
Commentaires:

Aucun commentaire pour ce livre.

[Modifier ce livre](#)
[Supprimer](#)
[Retour](#)

Application des livres - ApipassionBooks - 2024
Contact: pu78fh@eduvaud.ch
Développeurs: Dario Chasi - Nima Zarrabi - William Trelles - Thibaud Racine

Page vue login



Hyper Book

Home Books Add book Log In

My Account

Nickname

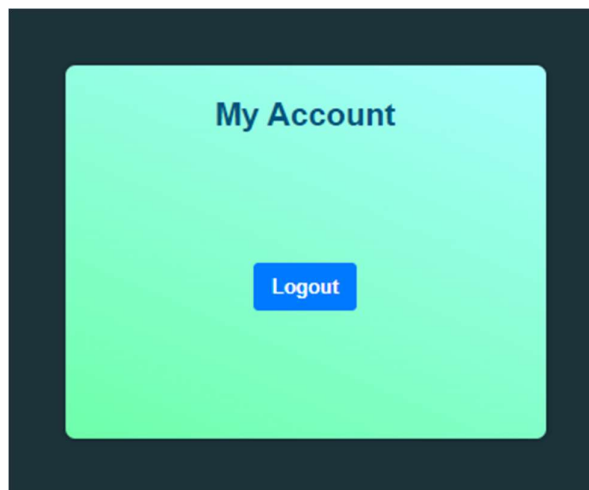
Enter nickname

Password

Enter password

Login

Application des livres - ApipassionBooks - 2024
Contact: pu78fh@eduvaud.ch
Développeurs: Dario Chasi - Nima Zarrabi - William Trelles - Thibaud Racine



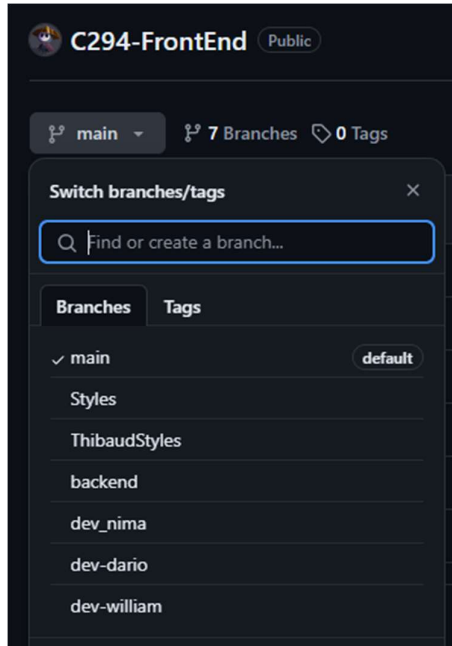
My Account

Logout

Les utilisateurs qui disposent du rôle admin peuvent éditer, modifier et créer tous les livres. Cela est possible grâce à comment le rôle sont réalisés dans le backend.

Gestion avec GitHub

Lien au GitHub : <https://github.com/NimaETML/C294-FrontEnd>



Nous avons travaillé sur GitHub en suivant une logique "Trunk-Based", chaque membre du groupe travail sur sa propre branche et une fois une fonctionnalité ajoutée et totalement fonctionnel, le membre merge sa branche dans le main.

Si un autre membre à merge entretemps, cela peut poser des conflits de merge, ces conflits sont facilement réglés si la fonctionnalité ajoutée n'est que sur une page avec une version clairement plus à jour, ce qui n'était pas toujours le cas.

Malgré quelques conflits qui ont pris un moment à régler, nous avons pu travailler avec GitHub

sans trop de problèmes.

Branches présentes au début du dernier jour.

Tests

Pour tester l'application nous avons utilisé des tests fonctionnels, de cette manière nous pouvons tester les différentes fonctionnalités que notre site offre comme l'affichage des livres qui sont stockés dans une base de données, l'ajout et la suppression des livres ainsi comme la possibilité de se logger pour avoir une expérience plus personnalisée.

Test 1

But : afficher les livres de la base de données.

Étapes :

1. Accéder à l'application.
2. Accéder à la section "Books" grâce à la barre de navigation.

Résultat : les livres sont affichés avec leur catégorie, auteur et cover.

Test 2

But : Accéder aux détails d'un livre

Étapes :

1. Accéder à la liste des livres.
2. Cliquer sur le livre dont nous voulons obtenir les détails.

Résultat : les détails du livre sont affichés et lisibles, les détails affichés sont les suivants : Titre, nombre de pages, l'auteur, la catégorie et la cover.

Test 3

But : Supprimer un livre

Étapes :

1. Accéder à la liste des livres.
2. Cliquer sur le livre comme on fait pour obtenir les détails.
3. Cliquer sur le bouton "Delete".

Résultat : Le livre est supprimé.

En faisant ces tests nous pouvons tester que les fonctionnalités marchent et nous assurer que l'utilisateur n'aura pas aucun problème quand il interagisse avec le site.

En réalisant ces tests, nous pouvons vérifier que les fonctionnalités du site web fonctionnent correctement et nous assurer que l'utilisateur n'aura pas de problèmes lors de son interaction avec le site. Les tests permettent de valider que chaque fonctionnalité répond aux attentes. Cela garantit une expérience utilisateur fluide et sans accroc, en identifiant et corrigeant les éventuels bugs ou problèmes avant la mise en production du site.

Ecoconception

Le point éco-conception choisit est "Utiliser le chargement paresseux" (Lazy loading).

C'est le point n° 50 dans la liste de 115 bonnes pratiques d'écoconception Web.

Grâce au "Lazy-Loading" nous pouvons différer le chargement des ressources, dans notre cas les différentes pages, jusqu'à ce qu'elles soient nécessaires c'est à dire lorsque l'utilisateur accède à ces pages.

Cette pratique nous permet de réduire le temps de chargement initial de données et la consommation de la bande passante, cela va améliorer l'expérience utilisateur et réduire l'empreinte énergétique du site web.

De plus, en ne chargeant que ce qui est nécessaire, on réduit également le nombre de requêtes HTTP, ce qui allège le trafic réseau et diminue l'impact environnemental global de l'application.

Conclusions

Conclusion générale

Ce projet nous a permis de mettre en œuvre nos connaissances apprises durant le module FrontEnd, mais aussi des nouvelles connaissances, car nous n'avions pas encore gérer la connexion entre le frontend et l'API (sauf durant l'exercice donné numéros 3, mais nous ne l'avons pas tous fait).

Faire un FrontEnd nous a appris l'utilisation d'un backend, beaucoup mieux représenté avec une bonne interface que juste Insomnia ou Postman.

Ce projet nous a aussi permis d'approfondir nos compétences en Vue avec des formulaire plus compliqué et l'introduction d'authentification avec Tokens JWT, ainsi qu'introduire de vrais styles CSS.

Petit Bémol par rapport au projet est d'avoir perdu un collaborateur en plein milieu du projet. Heureusement nous avons reçu du renfort avec un nouveau collaborateur mais il a eu besoin de temps pour s'intégrer au projet.

Conclusion personnelle

Dario:

Ce projet nous a permis de voir beaucoup de choses. Personnellement, la partie que j'ai le plus aimée dans ce projet a été l'intégration de notre ancien code Backend dans ce projet Frontend. Cela nous a permis de voir directement certaines interactions entre le front et le back, ce que j'ai adoré.

Nima:

J'ai bien aimé ce projet, j'y ai appris beaucoup sur Vue. Je pense qu'utiliser un backend effectué dans un projet fait auparavant est très, mais j'aurais bien aimé pousser le site encore plus loin.

Thibaud:

Malgré avoir dû changer de groupe en milieu de projet, je pense bien m'être intégré au groupe et avoir fait mon possible pour réussir le projet.

William :

Ce projet d'un site web pour la gestion de livres présente plusieurs fonctionnalités techniques clés qui assurent une expérience utilisateur optimale et une administration efficace. Cela nous a aidé à appliquer toutes les connaissances vu pendant ce module et le précédente.

Critique sur la planification du projet

Nous n'avons pas planifié le projet en avance, mais nous avons, dans le [Trello](#), inscrit toutes les tâches nécessaires au début du projet, ce qui nous a permis d'avoir une vue de l'ensemble des tâches nécessaire.

Nous avons ensuite commencé à travailler sur les tâches en commençant par les tâches nécessaires à la compétition d'autres tâches.

Cette stratégie n'a pas été très efficace, car nous n'avons pas toujours mis à jour le Trello, de plus, nos tâches étaient souvent trop générales et ne correspondait pas complètement au travail à faire.

Une planification avec IceScrum aurait été plus pratique, mais elle aurait duré beaucoup plus longtemps à mettre en place, et nous n'avons pas assez de temps.

Un simple Trello est donc adapté pour ce type de projet, mais nous aurions du mieux l'utiliser.

Webographie

Doc officiel VueJS : <https://vuejs.org/guide/introduction>

Multer: [multer - npm \(npmjs.com\)](https://www.npmjs.com/package/multer)

Sequelize: [Models & Tables | Sequelize](#)

W3 School: https://www.w3schools.com/js/js_dates.asp

Utilisation d'Intelligence artificielle

Correction de l'orthographe, création des données pour les mock, exemples d'une correcte utilisation des modèles et du multer.