




Projet de NOSQL

Master en Science de données et système intelligent

Application pour gestion des comptes
bancaires(MongoDB,MapReduce,Nodejs)



Réalisé par :

- RABAB KAF
- SALWA KBIRI ALAQUI

Encadré par :

- Mr . Y OUBENAALLA

Introduction générale

Ce rapport présente une application de gestion bancaire intuitive et robuste, conçue pour offrir une expérience utilisateur optimale tout en garantissant la sécurité et la fiabilité des opérations.

Cette application vise à fournir une plateforme complète permettant à ses utilisateurs de gérer leurs comptes bancaires et de gérer les informations personnelles associées à ces comptes. Le rapport détaille les fonctionnalités principales de l'application, son architecture, ainsi que les outils et technologies utilisés pour son développement.

Chapitre 1 : Outils et Objectifs

Introduction

Ce premier chapitre éclaire les outils et les objectifs centraux de l'application de gestion bancaire. Il explore les outils utilisés pour concevoir et déployer cette plateforme.

1 Objectifs de l'Application

BankRS, notre application de gestion bancaire, est conçue pour les employés de la banque, leur offrant des fonctionnalités complètes pour gérer les utilisateurs. Ces employés peuvent ajouter de nouveaux utilisateurs, consulter des listes spécifiques d'utilisateurs pour chaque type de compte (Jeunesse, Épargne, Chèques), supprimer des utilisateurs et mettre à jour leurs informations ainsi que celles de leurs comptes associés.

Les trois types de comptes disponibles, adaptés aux besoins variés des clients, comprennent:

- **le Compte Jeunesse** : orienté vers les jeunes clients avec des fonctionnalités éducatives et des limites de dépenses adaptées .
- **le Compte Épargne** : axé sur l'accumulation d'économies avec des taux d'intérêt plus élevés .
- **le Compte Chèques** : destiné aux transactions courantes avec des retraits fréquents et des paiements par chèque.

L'objectif principal de **BankRS** est d'offrir aux employés bancaires la capacité de gérer efficacement les utilisateurs et leurs comptes, en fournissant des fonctionnalités CRUD étendues, telles que la création, la consultation, la mise à jour et la suppression des informations associées aux comptes bancaires.

2 Technologies et Base de données

2.1 Base de donnée:

Le modèle de données pour les utilisateurs dans **BankRS** est conçu pour capturer des informations détaillées sur chaque utilisateur, avec un accent particulier sur la gestion de leurs comptes bancaires. Chaque utilisateur est défini par un certain nombre de champs clés.

- *CIN* (Code d'Identification Nationale) : Identifiant unique de l'utilisateur, assurant l'unicité et l'identification fiable.

- **Nom Complet** : Enregistre le nom complet de l'utilisateur.
- **Email et Téléphone** : Informations de contact essentielles de l'utilisateur.
- **Image** : Stocke le chemin de l'image de profil de l'utilisateur.
- **Date de création** : Enregistre la date de création du compte utilisateur, facilitant la gestion chronologique des données.

Les comptes bancaires sont gérés comme une liste pour chaque utilisateur. Chaque compte est défini par :

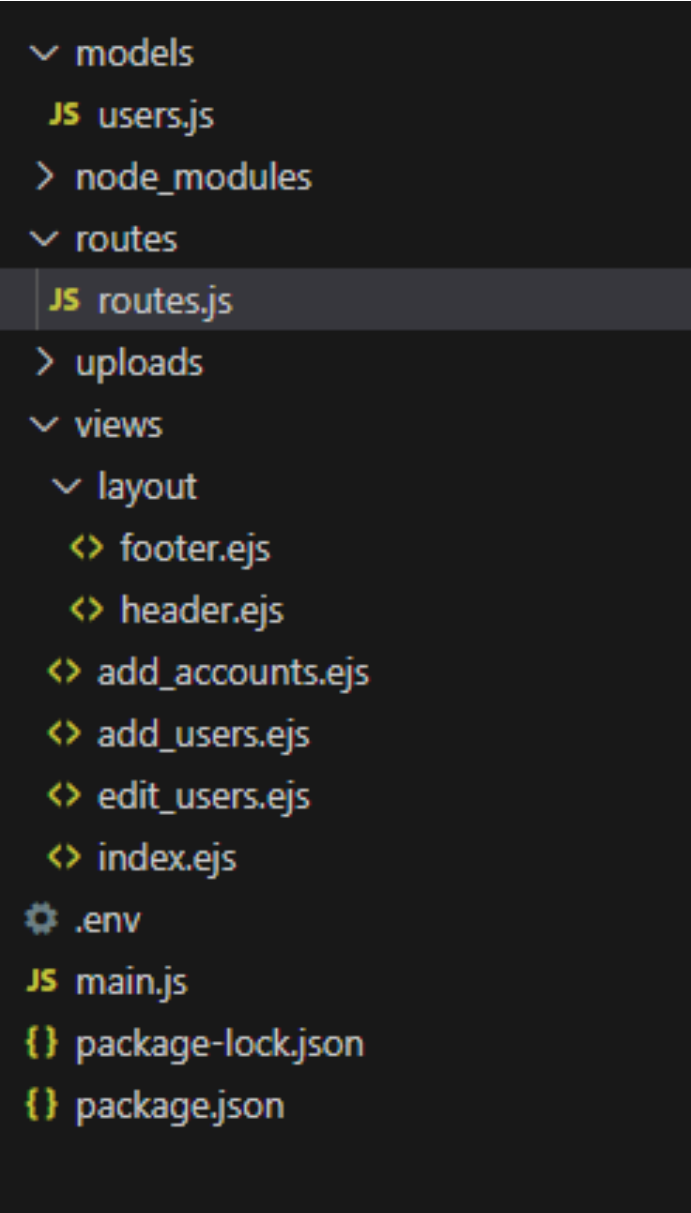
- **Numéro de Compte** : Un identifiant unique pour chaque compte, assurant qu'aucun numéro de compte ne soit en double.
- **Type de Compte** : Catégorise le compte en épargne, chèques ou jeunesse.
- **Solde du Compte** : Enregistre le solde actuel du compte, initialisé à zéro par défaut.

L'imbrication des données dans le modèle de l'utilisateur de **BankRS** est réalisée pour gérer les comptes bancaires associés à chaque utilisateur de manière organisée et pratique. Cette approche imbriquée permet de structurer les informations, créant une relation directe et logique entre l'utilisateur et ses différents comptes.

La définition d'une liste de comptes bancaires à l'intérieur du schéma de l'utilisateur offre une représentation claire et hiérarchisée des données. Chaque utilisateur peut posséder plusieurs comptes, et ceux-ci sont stockés sous la forme d'un tableau au sein du document utilisateur.

2.2 Architecture d'application:

L'architecture de ce projet est soigneusement conçue pour offrir une organisation claire et modulaire, avec une séparation nette des responsabilités entre les différentes composantes telles que les modèles de données, les routes pour la logique applicative, les vues pour l'interface utilisateur, ainsi que les fichiers de configuration et de dépendances. Cette architecture suit une structure typique d'un projet web basé sur **Node.js** et **Express** avec approche **MapReduce**. Voici une explication de chaque répertoire et de son rôle



```

  models
    JS users.js
  > node_modules
  routes
    JS routes.js
  > uploads
  views
    layout
      <> footer.ejs
      <> header.ejs
      <> add_accounts.ejs
      <> add_users.ejs
      <> edit_users.ejs
      <> index.ejs
  .env
  JS main.js
  {} package-lock.json
  {} package.json

```

model

Ce répertoire contient les fichiers de modèles de données de l'application. Dans ce cas précis, le fichier `users.js` définit le schéma de la base de données utilisé pour représenter les utilisateurs et leurs comptes bancaires.

node_modules

Ce répertoire est généré lorsque on installe les dépendances Node.js à l'aide de npm (Node Package Manager). Il stocke toutes les dépendances et bibliothèques tierces nécessaires à votre application.

routes

Ce répertoire contient le fichier routes.js, où sont définies les routes et les actions associées à chaque route de votre application. Il contient la logique de contrôle des routes, faisant le lien entre les requêtes HTTP et les opérations à effectuer sur les données.

uploads

Ce répertoire est utilisé pour stocker les images téléchargés par les utilisateurs.

views

Ce répertoire contient les fichiers de vues, qui sont des fichiers ejs . Les vues représentent la partie visible de notre application web. Chaque fichier ejs dans ce répertoire correspond à une page ou à une partie de notre application. Les fichiers footer.ejs et header.ejs sont des éléments réutilisables qui sont inclus dans l'autres vues pour maintenir une structure cohérente.

.env

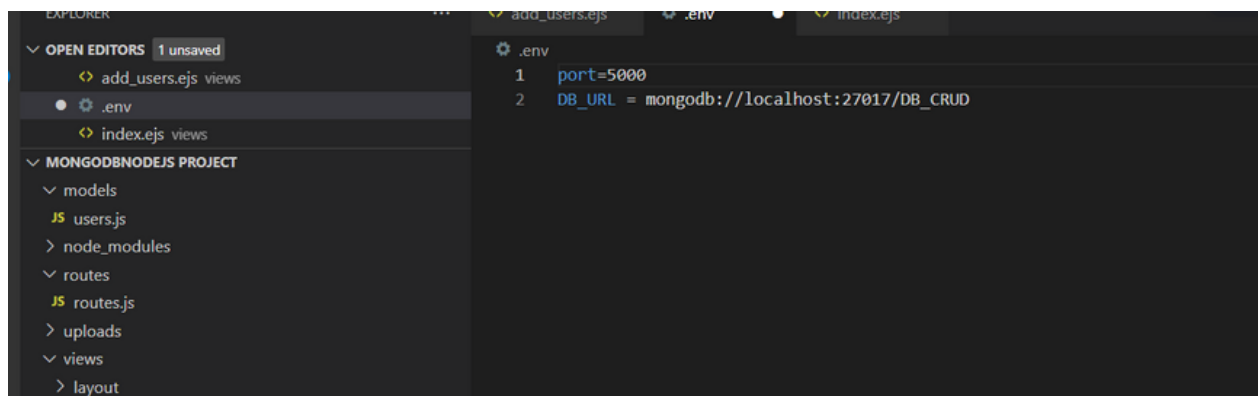
Fichier de configuration pour stocker des variables d'environnement ,le port de l'application et l'URL de la base de données.

main.js

Ce fichier principal établit la configuration initiale de l'application Express, incluant la connexion à la base de données MongoDB via Mongoose, la configuration des middlewares pour gérer les sessions et les requêtes, et l'utilisation du moteur de template EJS pour la gestion des vues. De plus, il redirige les requêtes vers le fichier de routes spécifié.

package-lock.json et package.json

Ces fichiers sont utilisés pour gérer les dépendances de notre application. package.json contient des métadonnées sur le projet et la liste des dépendances, tandis que package-lock.json verrouille les versions exactes des dépendances installées pour assurer la reproductibilité des installations.



Utilisateur de MapReduce:

L'application suit une structure classique basée sur Node.js et Express avec une base de données MongoDB. Elle utilise MapReduce pour effectuer des recherches dans la base de données en fonction du type de compte bancaire spécifié dans le formulaire de recherche.

- **Route de recherche (/search)** : Cette route gère la soumission du formulaire de recherche pour rechercher les utilisateurs en fonction du type de compte bancaire.

```
app.get("/search", (req, res) => {
  const accountType = req.query.accountType;

  const mapFunction = function () {
    emit(this.bankAccount.accountType, this);
  };

  const reduceFunction = function (key, values) {
    return values;
  };

  const finalizeFunction = function (key, reducedValue) {
    return reducedValue;
  };

  const mapReduceOptions = {
    map: mapFunction,
    reduce: reduceFunction,
    finalize: finalizeFunction,
    out: { inline: 1 },
    query: {
      "bankAccount.accountType": {
        $regex: new RegExp(accountType, "i"),
      },
    },
  },
};
```

- **Extraction des données** : Lorsqu'un utilisateur soumet le formulaire de recherche avec un type de compte spécifique (accountType), la route utilise la fonction mapReduce de MongoDB.
- **Fonction de mapping** : La fonction mapFunction est utilisée pour mapper les données en émettant la clé bankAccount.accountType associée à chaque utilisateur.

```

User.mapReduce(mapReduceOptions, function (err, results) {
  if (err) {
    res.render("index", {
      title: "Page Accueil",
      users: [],
      message: { type: "danger", message: err.message },
    });
  } else {
    const users = results ? results.map((result) => result.value) : [];
    res.render("index", {
      title: "Page Accueil",
      users: users,
      message: null,
    });
  }
});
});

```

- **Fonction de réduction** : La fonction `reduceFunction` est utilisée pour réduire les valeurs associées à chaque clé (dans ce cas, le type de compte) en retournant ces valeurs.
- **Options de MapReduce** : Les options de MapReduce sont configurées pour spécifier la fonction de mapping, la fonction de réduction, la requête (query) pour filtrer les données, et la sortie (out) sous forme de tableau pour les résultats en ligne.
- **Traitement des résultats** : En fonction des résultats de MapReduce, les utilisateurs correspondants sont extraits et renvoyés à la vue `index.ejs`. En cas d'erreur, un message d'erreur est rendu sur la page.

2.2 Technologies utilisées

L'architecture technologique de notre application repose sur un ensemble de technologies modernes, chacune apportant ses avantages distincts. Node.js forme le socle dynamique, permettant une exécution côté serveur en utilisant JavaScript. Associé à Express, un framework web minimaliste et robuste, Node.js facilite le développement d'applications web réactives. MongoDB, en tant que base de données NoSQL, offre une approche souple et évolutive pour gérer nos données. Dans cette exploration détaillée, plongeons dans le fonctionnement et les avantages de chacune de ces technologies clés.

Nodejs

Node.js est un environnement d'exécution JavaScript côté serveur. Son principal rôle dans une application est d'exécuter du code JavaScript côté serveur, offrant ainsi la possibilité de créer des applications web dynamiques et des serveurs. Dans notre application, Node.js agit comme le serveur, gérant les requêtes des clients, interagissant avec la base de données MongoDB, et orchestrant la logique



métier de l'application. Il permet une exécution rapide et efficace du code JavaScript, en utilisant un modèle asynchrone pour gérer les opérations de manière non bloquante, ce qui est particulièrement utile pour les applications nécessitant des mises à jour en temps réel ou des interactions fréquentes avec la base de données.

Express

Express est un framework web minimaliste pour Node.js, conçu pour simplifier la création d'applications web et d'API. Il fournit un ensemble de fonctionnalités robustes pour gérer les requêtes HTTP, définir des routes, gérer les middlewares, et bien plus encore. Dans notre application, Express agit comme une couche intermédiaire entre le serveur Node.js et les routes définies, offrant une structure et des fonctionnalités pour gérer les requêtes entrantes et sortantes. Cela permet de structurer efficacement le code, de gérer les requêtes et les réponses de manière organisée, et de rendre la création d'applications web plus rapide et plus fluide.

MongoDB

MongoDB est une base de données NoSQL, orientée document, qui stocke les données sous forme de documents JSON flexibles. Contrairement aux bases de données relationnelles, MongoDB n'utilise pas de schémas prédéfinis, ce qui offre une grande souplesse pour gérer des données de structures différentes au sein de la même collection. Il permet de stocker des données complexes, des schémas évolutifs et de facilement s'adapter à des besoins changeants. Dans notre application, MongoDB est utilisé pour stocker et gérer les données des utilisateurs, des comptes bancaires et d'autres informations, offrant une solution de stockage flexible et évolutive.



Notre base de données spécifique pour cette application stocke les informations des utilisateurs ainsi que leurs comptes bancaires. La structure de cette base est adaptée à notre modèle de données, organisée autour de deux collections principales : la collection "users" et la sous-collection "bankAccounts" imbriquée dans chaque utilisateur.

```
{
  "_id": {
    "$oid": "657c72d1195454210032c058"
  },
  "name": "HACHIMI Fatima",
  "email": "hachimifatima@gmail.com",
  "phone": "0987654321",
  "image": "image-1702654673158_Sabine Dumont.jfif",
  "CIN": "N1234",
  "bankAccounts": [
    {
      "balance": 1000,
      "_id": { },
      "accountNumber": "1234",
      "accountType": "Jeunesse"
    }
  ],
  "created": {
    "$date": "2023-12-15T15:37:53.206Z"
  },
}
```

Exemple d'un document de notre base de données

La collection "users" contient les données principales des utilisateurs telles que leur nom, email, numéro de téléphone et leur image de profil. Chaque document de cette collection peut également inclure un ou plusieurs comptes bancaires, stockés dans la sous-collection "bankAccounts". Ces comptes sont associés à chaque utilisateur et comprennent des informations telles que le numéro de compte, le type de compte (épargne, chèques, jeunesse) et le solde actuel.

Chapitre 2 : Réalisation de l'application

1 Accueil

La page d'accueil constitue l'interface centrale pour l'interaction des utilisateurs avec le système de gestion bancaire. Elle est méthodiquement structurée, tirant parti de Bootstrap pour offrir une présentation à la fois esthétique et réactive. La barre de navigation ainsi que le tableau de données semblent être des éléments clés pour une gestion fluide des utilisateurs et de leurs comptes. Cette analyse détaillée vous permettra de mieux appréhender chaque fonctionnalité de cette interface cruciale.

BankRS

Accueil Add Users About Us Contact

Rechercher les utilisateurs de ce compte :

Épargne

Rechercher

Show 10 entries Search:

CIN	Image	Nom Complet	E-Mail	Téléphone	Action
S223		ALAOUI Mohammed	mohammedalaoui@gmail.com	0607352048	
N1234		HACHIMI Fatima	hachimifatima@gmail.com	0987654321	
K84		KAF Rabab	rababkaf@gmail.com	0866788676	
H43		ALAOUI Salwa	salwa.kbirialaoui1@usmba.ac.ma	0607814614	

Showing 1 to 4 of 4 entries

Previous 1 Next

Page d'accueil

1.Barre de navigation supérieure : Elle offre une navigation fluide entre les différentes sections de notre application, telles que l'accueil et l'ajout d'utilisateurs.

2. Formulaire de recherche : Une boîte de sélection permet de rechercher des utilisateurs en fonction du type de compte bancaire, en sélectionnant le type souhaité.

Rechercher les utilisateurs de ce compte :

Épargne

Rechercher

Show 10 entries

Search:

CIN	Image	Nom Complet	E-Mail	Téléphonbe	Action
N1234		HACHIMI Fatima	hachimifatima@gmail.com	0987654321	  
H43		ALAOUI Salwa	salwa.kbirialaoui1@usmba.ac.ma	0607814614	  

Showing 1 to 2 of 2 entries

Previous 1 Next

Recherche des utilisateurs ayant un compte de type 'jeunesse'

3. la barre de recherche : les fonctionnalités sont polyvalentes, permettant une recherche précise et efficace des utilisateurs. On peut retrouver des fonctionnalités de recherche par nom, CIN, numéro de téléphone et adresse e-mail. Ces critères multiples offrent une souplesse d'utilisation pour localiser rapidement un utilisateur spécifique









Rechercher les utilisateurs de ce compte :

Épargne

Rechercher

Show 10 entries

Search: alaoui

CIN	Image	Nom Complet	E-Mail	Téléphonbe	Action
S223		ALAOUI Mohammed	mohammedalaoui@gmail.com	0607352048	  
H43		ALAOUI Salwa	salwa.kbirialaoui1@usmba.ac.ma	0607814614	  

Showing 1 to 2 of 2 entries (filtered from 4 total entries)

Previous 1 Next

Exemple de recherche par nom

4. Tableau des utilisateurs : Affiche les utilisateurs avec des détails tels que le CIN, l'image de profil, le nom complet, l'e-mail et le téléphone. Pour chaque utilisateur, des actions sont disponibles, comme l'édition, l'ajout de comptes supplémentaires et la suppression.

2 Ajouter user

La section d'ajout d'un nouvel utilisateur représente la porte d'entrée de notre système. Elle offre une interface soigneusement conçue, permettant de saisir et d'intégrer de nouvelles données utilisateur essentielles. À travers cette interface, on peut fournir des informations clés telles que le nom complet, l'email, le numéro de téléphone, et même des détails sur le compte bancaire. Cette étape cruciale nous permet d'enrichir notre base de données avec de nouveaux utilisateurs

Ajouter Nouveau Utilisateur

CIN:

K99

Nom Complet:

soufiane nassif

Email:

soufianenassif@gmail.com

Image:

Choisir un fichier

black boys_ Photo.jfif

Phone:

0876345890

Informations du Compte Bancaire

Numéro de Compte:

99

Type de Compte:

Épargne

Solde du Compte:

6900

Ajouter

Formule d'insertion

Ce bloc de code gère l'ajout d'un nouvel utilisateur et de ses informations associées dans la base de données.

```
// Insert user into database
router.post("/add", upload, async (req, res) => {
  try {
    const existingUser = await User.findOne({ CIN: req.body.CIN }).exec();

    if (existingUser) {
      req.session.message = {
        type: "danger",
        message: "Un utilisateur avec le même CIN existe déjà.",
      };
      return res.redirect('/');
    }

    const newAccount = {
      accountNumber: req.body.accountNumber,
      accountType: req.body.accountType,
      balance: req.body.balance
    };

    const user = new User({
      name: req.body.nomCompleet,
      email: req.body.email,
      phone: req.body.phone,
      image: req.file.filename,
      CIN: req.body.CIN,
      bankAccounts: [newAccount]
    });

    await user.save();
    req.session.message = {
      type: "success",
      message: "L'utilisateur a été ajouté avec succès.",
    };
    res.redirect('/');
  } catch (error) {
    req.session.message = {

```

Lorsqu'un utilisateur est ajouté, le système vérifie d'abord s'il existe déjà un utilisateur avec le même numéro d'identification (CIN). Si tel est le cas, il affiche un message d'erreur indiquant qu'un utilisateur avec le même CIN existe déjà.

✕ Un utilisateur avec le même CIN existe déjà.

Rechercher les utilisateurs de ce compte :

Épargne

Rechercher

Chercher 10 entrées

Chercher

C'est ce n'est pas le cas, il crée un nouvel utilisateur avec les détails fournis, y compris le nom complet, l'email, le numéro de téléphone, l'image et le CIN. Il crée également un compte bancaire initial pour cet utilisateur avec les informations du compte fournies. Enfin, il enregistre ces informations dans la base de données et redirige vers la page d'accueil avec un message de réussite ou d'erreur en fonction du résultat de l'opération.

</> BankRS

Accueil Add Users About Us Contact

✕ L'utilisateur a été ajouté avec succès.

Rechercher les utilisateurs de ce compte :

Épargne

Rechercher

3 Modifier user

Cette page est structurée en utilisant Bootstrap et permet aux utilisateurs de modifier leurs informations personnelles ainsi que les détails de leurs comptes bancaires. Lorsqu'un utilisateur souhaite apporter des modifications, le formulaire de modification est présenté, affichant les détails actuels de l'utilisateur, y compris ses informations de compte bancaire.

Modifier l'utilisateur (ALAOUI Mohammed)

Nom Complet:

ALAOUI Mohammed

Email:

alaouimohammed@gmail.com


Phone:

0673456219

Image:

Choisir un fichier

Aucun fichier choisi



Informations du Compte Bancaire

Numéro de Compte:

223

Type de Compte:

Épargne

Solde du Compte:

90000

Modifier

Formule de modification

Le traitement de cette page est géré par deux routes dans votre application Express.

- La première route, lorsqu'elle reçoit une demande GET sur **'/edit/:id'**, recherche l'utilisateur en fonction de l'ID fourni. Si l'utilisateur est trouvé, les détails sont affichés dans le formulaire de modification.

Le code de la première route gère l'affichage des détails d'un utilisateur pour la modification en fonction de son ID.

```
router.get('/edit/:id', async (req, res) => {
  try {
    const id = req.params.id;
    const user = await User.findById(id);

    if (!user) {
      return res.redirect('/');
    }

    res.render('edit_users', {
      title: "Modifier utilisateur",
      user: user,
      bankAccounts: user.bankAccounts, // Envoi de
    });
  } catch (err) {
    res.redirect('/');
  }
});
```

- La deuxième route, en cas de demande POST sur **'/update/:id'**, gère la mise à jour des informations utilisateur. Elle vérifie les nouvelles données et met à jour les détails de l'utilisateur, y compris les informations du compte bancaire si elles ont été modifiées. Une fois les modifications enregistrées avec succès, l'utilisateur est redirigé vers la page d'accueil avec un message de succès.

</> BankRS

Accueil Add Users About Us Contact

✕ Modification enregistrée avec succès

Rechercher les utilisateurs de ce compte :

Épargne

Rechercher

Show 10 entries

Search:

Cette partie du code est responsable de la gestion de la modification des détails d'un utilisateur dans la base de données. Elle récupère l'ID de l'utilisateur et gère la mise à jour des informations, telles que le nom, l'e-mail et le numéro de téléphone. De plus, elle prend en charge la mise à jour de l'image de profil si un nouveau fichier est téléchargé, tout en prenant soin de supprimer l'ancienne image. En cas d'erreur lors de la mise à jour, elle renvoie un message indiquant le type d'erreur rencontré pour assurer une expérience utilisateur transparente.

```
//update router
router.post('/update/:id', upload, (req, res) => {
  let id = req.params.id;
  let new_image = '';
  if (req.file) {
    new_image = req.file.filename;
    try {
      fs.unlinkSync("./uploads/" + req.body.old_image);
    } catch (err) {
      console.log(err);
    }
  } else {
    new_image = req.body.old_image;
  }

  users.findById(id, (err, user) => {
    if (err) {
      res.json({ message: err.message, type: 'danger' });
    } else {
      // Mise à jour des champs utilisateur
      user.name = req.body.name;
      user.email = req.body.email;
      user.phone = req.body.phone;
      if (req.file) {
        user.image = new_image;
      }
    }
  })
})
```

Ce bloc de code gère la mise à jour des informations liées aux comptes bancaires d'un utilisateur spécifique. Il vérifie d'abord s'il existe des comptes bancaires associés à cet utilisateur. Ensuite, pour chaque compte bancaire, il met à jour les détails tels que le numéro de compte, le type de compte et le solde en fonction des nouvelles données fournies. Après avoir effectué ces mises à jour, il enregistre les modifications dans la base de données. En cas d'erreur lors de la sauvegarde, il renvoie un message d'erreur approprié, sinon, il confirme la réussite de la modification et redirige vers la page d'accueil.

```

// Mise à jour des champs du compte bancaire
if (user.bankAccounts && user.bankAccounts.length > 0) {
  user.bankAccounts.forEach((account, index) => {
    if (account && account.accountNumber && req.body.accountNumber && req.body.accountNumber[index]) {
      account.accountNumber = req.body.accountNumber[index];
      account.accountType = req.body.accountType[index];
      account.balance = req.body.balance[index];
    }
  });
}

// Sauvegarde des modifications
user.save((err, updatedUser) => {
  if (err) {
    res.json({ message: err.message, type: 'danger' });
  } else {
    req.session.message = {
      type: 'success',
      message: 'Modification enregistrée avec succès'
    };
    res.redirect('/');
  }
});
}

```

4. Ajouter compte

Cette section dédiée à l'ajout de comptes bancaires pour un utilisateur spécifique ,il permet de contribuer aux détails financiers des utilisateurs existants. On peut saisir des informations telles que le numéro de compte, le type de compte (comme Épargne, Chèques ou Jeunesse), et le solde du compte.

Ajouter des comptes (ALAOUI Mohammed)

Informations du Compte Bancaire

Numéro de Compte:

Type de Compte:

Choisissez le type de compte

▼

Solde du Compte:

Ajouter

Formule pour ajouter un compte

Cette section permet l'ajout de nouveaux comptes bancaires pour un utilisateur spécifique. Lorsqu'un utilisateur souhaite ajouter un compte, le système vérifie d'abord si cet utilisateur existe et s'il n'a pas déjà atteint le nombre maximum de comptes. De plus, il contrôle si le type de compte à ajouter est déjà associé à cet utilisateur pour éviter les doublons.

```

6 // Route pour ajouter un nouveau compte bancaire à l'utilisateur
7 router.post('/addAccount/:id', async (req, res) => {
8   try {
9     const id = req.params.id;
10    const user = await User.findById(id);
11    if (!user) {
12      return res.json({ message: "Utilisateur non trouvé", type: 'danger' });
13    }
14    // Vérifier le nombre de comptes actuels pour l'utilisateur
15    if (user.bankAccounts.length >= 3) {
16      return res.json({ message: "Nombre maximum de comptes atteint", type: 'danger' });
17    }
18    // Vérifier si le type de compte est déjà associé à l'utilisateur
19    const existingAccount = user.bankAccounts.find(account => account.accountType === req.body.accountType);
20    if (existingAccount) {
21      return res.json({ message: "Ce type de compte existe déjà pour cet utilisateur", type: 'danger' });
22    }
23    // Ajouter un nouveau compte bancaire au tableau 'bankAccounts'
24    if (req.body.accountNumber & req.body.accountType & req.body.balance) {
25
26      user.bankAccounts.push({
27        accountNumber: req.body.accountNumber,
28        accountType: req.body.accountType,
29        balance: req.body.balance
30      });
31    }
32    await user.save();
33    req.session.message = {
34      type: 'success',
35      message: 'Nouveau compte ajouté avec succès'
36    };
37    res.redirect('/');
38  } catch (err) {
39    // Gestion des erreurs
40  }
41 }

```

Si ces critères sont respectés, un nouveau compte est ajouté à la liste des comptes bancaires de l'utilisateur, enrichissant ainsi son profil.



5. Supprimer

Cette partie du code gère la suppression d'un utilisateur. Lorsqu'un utilisateur envoie une demande de suppression d'un compte, le système vérifie d'abord l'existence de cet utilisateur. Ensuite, s'il trouve l'utilisateur correspondant à l'identifiant spécifié, il demande une confirmation explicite de la part de l'utilisateur.

```

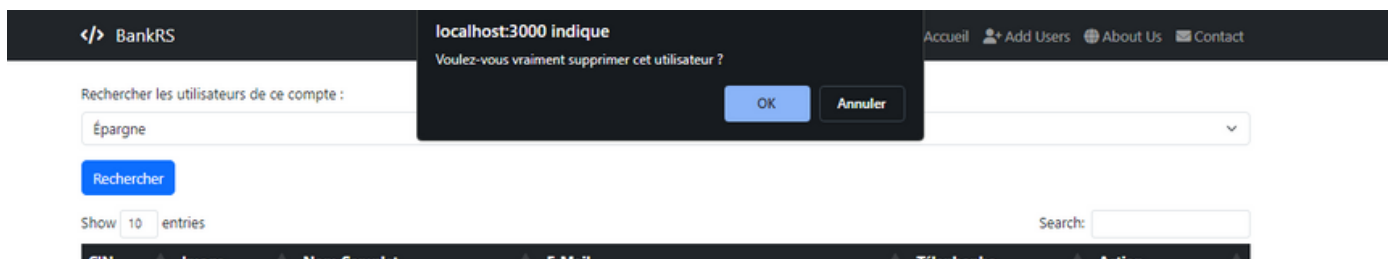
router.get('/delete/:id', async (req, res) => {
  try {
    let id = req.params.id;
    const user = await User.findByIdAndDelete(id);

    if (user && user.image !== '') {
      try {
        fs.unlinkSync('./uploads/' + user.image);
      } catch (err) {
        console.log(err);
      }
    }

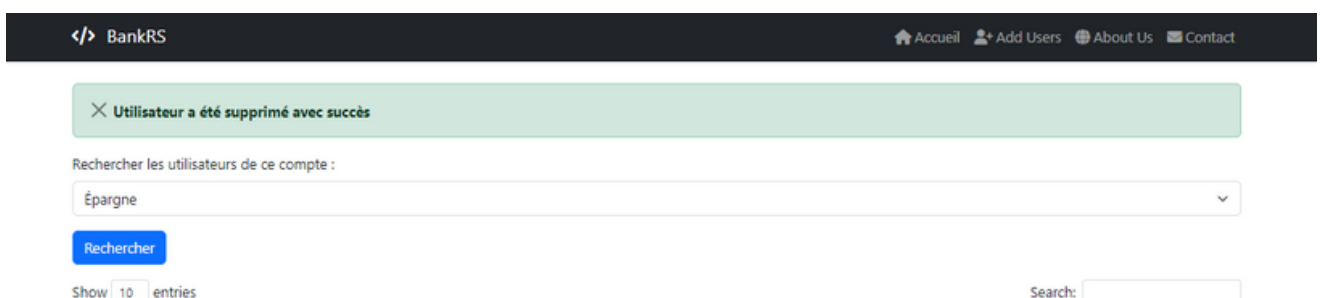
    req.session.message = {
      type: 'success',
      message: 'Utilisateur a été supprimé avec succès'
    };
    res.redirect('/');
  } catch (err) {
    res.json({ message: err.message });
  }
});

```

Lorsqu'une demande de suppression est initiée, le système vérifie l'existence de cet utilisateur dans la base de données. Si l'utilisateur est trouvé, une page de confirmation est affichée, demandant à l'utilisateur de confirmer son intention de supprimer le compte.



Une fois que l'utilisateur confirme cette action, le système procède à la suppression de l'utilisateur de la base de données. Si l'utilisateur avait une image de profil associée, sa suppression entraîne également la suppression de cette image depuis le répertoire des téléchargements. Enfin, une fois la suppression effectuée, le système redirige l'utilisateur vers la page d'accueil avec un message indiquant le succès de l'opération.



Conclusion

Nous avons réalisé un projet portant sur notre banque virtuelle, BankRS, en utilisant MongoDB MapReduce et Node.js. Notre système repose sur la base de données MongoDB en tant que SGBD, le back-end en Node.js, et le front-end en Express. Dans notre recherche utilisateur, nous avons explicitement mentionné l'utilisation de MapReduce pour la sélection des comptes, mais en raison de problèmes de version pour les opérations d'insertion, de suppression et de modification, nous avons opté pour une approche plus simple sans MapReduce dans la pratique. Néanmoins, nous avons approfondi notre compréhension de MapReduce au cours de la recherche, comprenant son fonctionnement et ses mécanismes, ce qui constitue l'objectif principal de notre démarche.