Nao Libre

IUT Nantes
Pôle Sciences et technologie
Nantes Université

BUT2 Initiaux
SAÉ4.Real.01 : Qualité et au-delà du relationnel
Partie : R403 | QADR

LEPORT Clovis
MARTEL Floran
GROMARD Alexys
JOUAULT Lancelot
CHEVREUX Arthur

Année Scolaire 2023 - 2024
Réf : eq_01_01

Introduction

Dans le cadre de notre deuxième année de BUT, nous avons dû développer un site web répondant à un problème actuel. Nous avons décidé de simplifier la vie des utilisateurs de transports en commun au sein de Nantes. Pour ce faire, nous avons voulu créer un site web permettant de donner l'encombrement de chaque transport pour que les personnes qui sont repoussées par l'utilisation des bus de par sa surpopulation, puissent constater que ce n'est pas toujours le cas.

Notre site web permet aux utilisateurs de pouvoir consulter le temps d'attente des bus et tram de leurs arrêts favoris et ils peuvent aussi observer les moyennes d'encombrement par heure sur chaque ligne. Les utilisateurs sont récompensés par un système de points à chaque fois qu'ils évaluent une ligne. De cette façon, ils se sentent plus impliqués et prennent le temps de nous aider à chaque fois.

Afin de développer ce site, nous avons utilisé des technologies variées, à savoir : React pour le développement de la partie client, Express et Node.js pour la partie serveur, ainsi que MongoDB et Mongoose pour l'utilisation de la base de données.

Équipe de projet

Ce rapport a été préparé par :
- Alexys GROMARD
- Floran MARTEL
- Lancelot JOUAULT
- Clovis LEPORT
- ARTHUR CHEVREUX

Equipe : eq_01_01

Informations complémentaires

📍 IUT de Nantes (Département Informatique)
📆 Quand : 2024 (4ème semestre)

Table des matières

Introduction SGBD

Dans le cadre de notre projet, nous avons utilisé *MongoDB* pour stocker la donnée de notre application. Nous avons choisi d’utiliser une base de données afin de conserver les informations relatives aux utilisateurs et aux avis d’encombrement sur les lignes. Concernant l'API de la Tan, nous n'avons pas stocker ses données dans notre propre base, mais nous avons plutôt mis en place un filtrage de l’API Naolibe pour afficher les informations dans un format plus adapté à notre utilisation.

Installation Base de donné

Prérequis :

- Mongodb
- Node.js v20.12.2

Pour mettre en place la base donnée il faudra cloner notre projet puis aller sur le terminal dans le répertoire ./**database/dist**.
Lancer la commande **npm i** qui permet d’installer les modules nécessaires pour lancer les fichiers.
Vérifier que le service mongodb est bien lancé, sinon faite **sudo service mongodb start**.
Puis lancer la commande **node ./mongodb.js** qui est le fichier qui créer la base de donné et les collections.
Si on souhaite avoir des données par défaut, lancer **node ./implementation/Insert_data.js**

Technologie

Dans ce projet nous avons utilisé la technologie *NoSQL* qui nous a offert la possibilité de stocker des données (sous forme de collections) de manière plus flexible et plus rapide que les bases de données relationnelles. Cette approche nous a permis d'établir des connexions plus cohérentes entre les données et de les manipuler plus aisément à l'aide de requêtes en JavaScript (langage utilisé par MongoDB et notre API) .

NaoLibre

Dans notre base de données `NaoLibre`, nous avons créé deux collections : `Users` et `Avis`. La collection `Users` contient les informations des utilisateurs de notre application telles que leur nom, leur adresse e-mail, leur mot de passe, leurs favoris et leurs points. Quant à la collection "Avis", elle recueille les évaluations des utilisateurs concernant les lignes de bus et de tramway de la Tan.

La collection User permet de stocker les information personnelle dû user, comme sont nom ou son email mais aussi c'est point est ces arrêts en favorie.

Collection :	USER	
Nom	Type de donnée	Description
_id	ObjectId	id de l'utilisateur avec les id de mongodb
name	String	name d'un utilisateur
email	String	email d'un utilisateur
password	String	password crypter avec bcrypt
point	Int32	point correspond au nombre d'avis ajouter
favori	Array[String]	Arret ajouter en favori

Exemple de Json d'un User

```
{
        "_id": new ObjectId("661562334edca916f0028b2f"),
        "name": "Suzanne Camus",
        "email": "Camus.Suzanne",
        "password": "azerty",
        "point": 0,
        "favori": []
    }
```

Le champ 'nomLigne' correspond au nom d'une ligne de bus, vérifié grâce à l'API Tan. De plus, l'objet contenant l'avis possède également un identifiant.
Enfin comme MongoDB est obligé d'avoir un _id dans les documents nous en avons une dans notre base de données mais elle n'est pas utilisée dans notre système.

Collection :	AVIS	
Nom	Type de donnée	Description
nomLigne	ObjectId	id de l'utilisateur avec les id de mongodb
avis	Array[Object]	Liste de l'ensemble des avis pour une ligne
Object qui contient l'avis		
iduser	ObjectId	id de l'utilisateur
note	Int32	note donnée à une ligne
date	String	date du moment ou a été mis l'avis
dayweek	String	jour de la semaine

Exemple de Json d'AVIS

```
{
  "nomLigne": "C2",
  "avis": [
    {
      "iduser": new ObjectId("661562334edca916f0028b2f"),
      "note": 1,
      "date": "2024-04-09T21:49:18.625Z",
      "dayweek": "Tuesday"
    }
  ]
}
```

Qualité des sources de données

Comme notre base de données est en NoSQL il n'y a pas de contraste comme SQL pour avoir une clé primaire unique ou n'autoriser qu'une plage de valeurs à un attribut. Or, nous voulons que certaines valeurs soient uniques comme l'adresse mail de l'utilisateur et nous voulons aussi que les notes d'encombrement ne puissent qu'être entre 1 et 3.

Alors pour pouvoir avoir s'est contrainte malgré l'utilisation de NoSQL nous avons ajouté de la logique à notre DAO qui effectue en amont des vérifications de certaines contraintes pour prévenir de condition non respectée. Pour donner d'autres exemples, pour la collection "Avis", nous avons mis en place une vérification pour s'assurer qu'un utilisateur ne puisse pas publier deux avis sur la même ligne de bus ou de tramway dans un intervalle de temps donné (par exemple, une heure).

Et pour terminer, les données doivent respecter le validateur ci-dessous (schéma mongodb), qui permet de vérifier le typage et la mise en forme de la base de données.

Choix de modélisation

Nous avons opté pour une modélisation de données qui les rend facilement manipulables et connectables tout en maintenant leur cohérence.

La base de données `Naolibre` est une base de données NoSQL qui a été mise en place avec les collections `User` et `Avis`. Chaque collection contient des documents représentant les données de notre application, intégrées avec un schéma de validation pour assurer la solidité des données.

Voici le schéma créé dans mongoDB pour la collection `User` de la base de données `Naolibre`:

```
{
  $jsonSchema: {
    required: [
      '_id',
      'name',
      'email',
      'password',
      'point',
      'favori'
    ],
    type: 'object',
    properties: {
      _id: {
        bsonType: 'objectId',
        description: 'must be a string and is required'
      },
      name: {
        bsonType: 'string',
        description: 'must be a string and is required'
```

```
      },
      email: {
        bsonType: 'string',
        description: 'the email is required and must be a string'
      },
      password: {
        bsonType: 'string',
        description: 'the password is required and must be a string'
      },
      point: {
        bsonType: 'int',
        description: 'this attribute represente the point partisipation
of users'
      },
      favori: {
        bsonType: 'array',
        description: 'this attribute represente the favori ligne of
users',
        items: {
          type: 'string'
        }
      }
    }
  }
}
```

Maintenant, voici le schéma de la collection `Avis` dans la base de données `Naolibre`:

```
{
  $jsonSchema: {
    required: [
      'nomLigne',
      'avis'
    ],
    type: 'object',
    properties: {
      nomLigne: {
        bsonType: 'string',
        description: 'must be a string and is required'
      },
      avis: {
        type: 'array',
        items: {
```

```
          type: 'object',
          properties: {
            iduser: {
              bsonType: 'objectId',
              description: 'must be a string and is required'
            },
            note: {
              bsonType: 'int',
              description: 'must be a string and is required'
            },
            date: {
              bsonType: 'date',
              description: 'must be a date and is required'
            },
            dayweek: {
              bsonType: 'string',
              description: 'must be a string and is required'
            }
          },
          required: [
            'iduser',
            'note',
            'date',
            'dayweek'
          ]
        }
      }
    }
  }
}
```

Exemple Json des Données insérer

“User”

```
[
    {
        "_id": new ObjectId("661562334edca916f0028b26"),
        "name": "Jacqueline Hoareau",
        "email": "Hoareau@laposte.net",
        "password": "123456",
```

```
        "point": 0,
        "favori": []
    },
    {
        "_id": new ObjectId("661562334edca916f0028b27"),
        "name": "Suzanne Camus",
        "email": "Camus.Suzanne",
        "password": "azerty",
        "point": 0,
        "favori": []
    },
    {
        "_id": new ObjectId("661562334edca916f0028b28"),
        "name": "Michèle Fleury",
        "email": "Michè@exemple.com",
        "password": "motdepasse",
        "point": 0,
        "favori": []
    }
]
```

“Avis”

```
[
    {
        nomLigne: "C6",
        avis: [
            {
                iduser: new ObjectId("661562334edca916f0028b28"),
                note: 3,
                date: (new Date(2024, 3, 26, 17, 21, 0)).toISOString(),
                dayweek: "Tuesday"
            }
        ]
    }
]
```

Retour sur l'utilisation de MongoDB

Le choix d'utiliser la base de données NoSQL MongoDB pour stocker nos données a eu un impact très positif sur l'utilisation de celles-ci dans nos applications et/ou services. La flexibilité de la base de données "Naolibre" nous permet de stocker les données de manière rapide et efficace.

Cette décision a également facilité l'accès aux informations via l'API, notamment pour consulter les avis sur une ligne de bus ou de tramway spécifique. L'utilisation de MongoDB a simplifié le code de notre application en réduisant le nombre de requêtes et en minimisant les erreurs potentielles dans notre js.

Cependant, nous avons rencontré des défis, notamment en ce qui concerne les contraintes sur les collections. Il a été difficile de mettre en place des règles pour empêcher un utilisateur de soumettre deux fois le même avis sur une même ligne de bus ou de tramway.

Enfin, la principale difficulté que nous avons rencontrée, est l'utilisation de MongoDB puisque nous n'avions pas beaucoup d'expérience avec celui-ci. Mais grâce au problème que nous avons rencontré durant le projet, nous sommes maintenant bien plus familier avec l'utilisation de mongoDB et du NoSQL.

Connexion de l'API via des routes

Nous avons utilisé un MVC simplifié avec une architecture simple au niveau des dossiers qui est :

```
C:./api
├────model
├────repositories
├────routes
└────tools
```

* model : Ce dossier contient les classes ou les fichiers qui définissent la structure des données de votre application, également appelées "modèles". Les modèles représentent généralement les entités principales de notre système, telles que les utilisateurs, les avis, etc.

* repositories : Ce dossier contient les DAO qui permettent de faire les requêtes sur mongodb pour les ajouts, delete, update, find. Chaque DAO est connecté à une collection et database. On utilise la librairie mongoose pour simplifier l'extraction des documents via des schémas qui permette de sortir les documents sous la forme souhaitée.

* routes : Ce dossier contient les fichiers de routage de notre application. Ces fichiers définissent les points de terminaison de l'API ou les routes que l'application expose, ainsi que les fonctions associées à chaque route. C'est là que nous définissons les actions à effectuer en réponse à différentes requêtes HTTP.
Nous avons utilisé la librairie *express* pour créer les routes et pour gérer les requêtes HTTP.

* tools : Ce dossier contient des fichiers qui regroupent des fonctions ou des utilitaires réutilisables qui ne sont pas directement liés à un modèle, à un dépôt ou à une route spécifique. Ces fichiers peuvent contenir des fonctions utilitaires, des fonctions de validation, des fonctions de formatage, etc: cryptage, qui permet de crypter les mots de passe des utilisateurs.

l'utilisation du MVC dans notre API permet de mettre un intermédiaire entre la base de données et les requêtes des utilisateurs, puisque leurs requêtes sont obligée de passer par l'API qui se charge de faire des vérifications et de modifier les données, par exemple si une personne demande tous les utilisateurs via notre API les password des utilisateur seront remplacer par la valeur "null" pour ne pas les dévoilés.

Conclusion

Avec MongoDB, nous avons pu manipuler les données de façon simple grâce à des requêtes en JavaScript. Cela a grandement facilité le développement et la maintenance de notre application. De plus, l'organisation des données en collections et l'utilisation de schémas nous ont aidés à garder une structure claire et adaptable aux évolutions de notre projet.

Bien sûr, il y a eu des défis, mais ce sont grâce à ce défi, que notre expérience avec MongoDB et NoSQL à augmenter notre expérience.

En résumé, l'utilisation de MongoDB a largement contribué au succès de notre projet, en fournissant une base solide et flexible pour gérer nos données de manière efficace.