Développement Backend – Node JS

PARTIE 4 : UTILISER UNE BDD

INTERVENANT : THOMAS JAMME

Utiliser une BDD

- MongoDB
- Créer un schéma de données
- Enregistrer des données
- Récupérer des données
- Modifier des données
- Supprimer des données

MongoDB

Mongo DB - Généralités

- Dans le cadre de ce cours, nous allons utiliser une base de donnée de type **MongoDB**.
- Ce SGBD utilise une **BDD non relationnelle** (**NoSQL**).
- Ce concept s'oppose aux bases de données relationnelle tel que MySQL ou PostgreSQL.
- Aucune **contraintes de clés** n'existe.
- Les tables que l'on connait en SQL peuvent êtres assimilées à des « **collections** ».
- MongoDB permet de manipuler des objets structurés au format **JSON**, sans schéma prédéterminé.
- Ainsi, des entrées peuvent être ajoutées à tout moment et des structures d'objets peuvent être différentes dans une même collection.

```
{
"_id": ObjectId("4efa8d2b7d284dad101e4bc7"),
"Nom": "DUMOND",
"Prénom": "Jean",
"Âge": 43
},

{
"_id": ObjectId("4efa8d2b7d284dad101e4bc8"),
"Nom": "PELLERIN",
"Prénom": "Franck",
"Adresse": "1 chemin des Loges",
"Ville": "VERSAILLES"
},

{
"_id": ObjectId("4efa8d2b7d284dad101e4bc9"),
"Nom": "KING",
"Âge": "51",
"Adresse": "36 quai des Orfèvres",
"Ville": "PARIS"
}
```

Mongo DB - Installation

- Pour que notre server Node.js puisse communiquer avec la base de donnée, un connecteur doit être installé.

- La base de donnée sera partagée entre tous.

- Pour cela, installez la librairie mongoose :

- **`npm install mongoose`**

- Importez ensuite mongoose dans le fichier **app.js**

```
const mongoose = require('mongoose');
```

Mongo DB - Installation

- Ajouter à présent l'instruction de connexion en haut de **app.js (avant les middlewares)**:

```
mongoose.connect('mongodb+srv://esn:castres81@cluster0.7vtsq.mongodb.net/myFirstDatabase?retryWrites=true&w=majority', {
    useNewUrlParser: true,
    useUnifiedTopology: true
  })
  .then(() => console.log('Connexion à MongoDB réussie !'))
  .catch(() => console.log('Connexion à MongoDB échouée !'));
```

- Testez la connexion en lançant le serveur.

Créer un schéma de données

TS user.model.ts ×

```ts
import mongoose, { Schema, Document } from 'mongoose';

export interface IUser extends Document {
  email: string;
  firstName: string;
  lastName: string;
}

const UserSchema: Schema = new Schema({
  email: { type: String, required: true, unique: true },
  firstName: { type: String, required: true },
  lastName: { type: String, required: true }
});

// Export the model and return your IUser interface
export default mongoose.model<IUser>('User', UserSchema);
```

Créer un schéma de données – Schéma Catégorie

- Nous allons créer des modèles d'objet qui nous permettront de manipuler la BDD.
- Un schéma décrivant l'objet doit ainsi être créé.
- Créez un dossier nommé « **models** » à la racine du projet.
- Créez un fichier nommé « **categorie.js** ».
- Dans ce fichier, on créé un schéma. On lui défini des propriétés ainsi que leur type.
- Enfin on exporte le modèle.

```
const mongoose = require('mongoose')

const categorieSchema = mongoose.Schema({
  name: { type: String, required: true },
  order: { type: Number, required: true }
})

module.exports =
mongoose.model('Categorie', categorieSchema)
```

TP - Créer un schéma de données

Créez un schéma pour l'objet Produit

Propriété	Type	Required
name	Texte (String)	Oui
categorieId	Lien (Schema.Types.ObjectId)	Oui
categorieName	Texte	Oui
price	Entier	Oui
photo	Texte	Oui

Enregistrer des données

Enregistrer des données

- Pour pouvoir enregistrer des données, il faut importer le modèle.

```
const categorie = require('./models/categorie')
```

- A présent, il faut créer une instance de l'objet dans le code la route d'API qui se **charge de créer une catégorie.**

```
app.post('/api/categorie', (req, res, next) =>
{
  const categorie = new Categorie({
    ...req.body
  });
  categorie.save().then(() =>
      res.status(201).json({
        message: 'Categorie créée !'
      }))
    .catch(error => res.status(400).json({
      error
    }))
});
```

Enregistrer des données

- La syntaxe « **…req.body** » est un raccourcie permettant de faire une copie clé par clé .
- On appelle ensuite la méthode « **save** » de l'objet mongoose « **categorie** ».
- Cette méthode renvoie une **promesse** qui doit être traitée.
- On renvoie une réponse avec le statut **200** si la sauvegarde s'est bien réalisée.
- Sinon on renvoie le statut **400** dans le catch.

```
app.post('/api/categorie', (req, res, next) =>
{
  const categorie = new Categorie({
    ...req.body
  });
  categorie.save().then(() =>
      res.status(201).json({
        message: 'Categorie créée !'
      }))
    .catch(error => res.status(400).json({
      error
    }))
});
```

Récupérer des données

Récupérer des données

- Pour récupérer des données, la syntaxe sera sensiblement la même que pour créer.
- La méthode find() devra être utilisée pour récupérer les données.
- Cette méthode peut prendre en paramètre une requête Mongo.

```
Categorie.find()
    .then(categories =>
       res.status(200).json(categories))
    .catch(error => res.status(400).json({
      error
    }));
```

Récupérer des données

- Voici un cas où l'on voudrait récupérer une catégorie en particulier.
- On spécifie dans la requête l'id de la catégorie que l'on souhaite récupérer.
- La méthode « **findOne** » peut être aussi utilisée

```
Categorie.find({_id: req.params.catId })
    .then(categories =>
       res.status(200).json(categories))
    .catch(error => res.status(400).json({
      error
    }));
```

Mettre à jour des données

Mettre à jour des données

- Pour mettre à jour des données, la méthode **updateOne** doit être utilisée.
- **UpdateOne** prend en premier paramètre l'identifiant de l'objet à modifier.
- **UpdateOne** prend en second paramètre les données à modifier.

```
app.put('/api/categories/:id', (req, res, next) => {
  Categorie.updateOne({
      _id: req.params.id
    }, {
      ...req.body
    })
    .then(() => res.status(200).json({
      message: 'Categorie modifiée !'
    }))
    .catch(error => res.status(400).json({
      error
    }));
});
```

Supprimer des données

Supprimer des données

- Pour supprimer une donnée, la méthode **deleteOne** doit être utilsée.

- Elle prend en paramètre l'identifiant de l'objet à supprimer.

```
app.delete('/api/categories/:id', (req, res, next) =>
{
  Categorie.deleteOne({
      _id: req.params.id
    })
    .then(() => res.status(200).json({
      message: 'Categorie supprimée !'
    }))
    .catch(error => res.status(400).json({
      error
    }));
});
```

TP – CRUD des données

Utilisez la base de données pour toutes les entrées d'API créées

Des questions ?