IUT de Paris - Rives de Seine
Université Paris Cité

BUT 3 INFORMATIQUE

Parcours A FI

Réalisations d'applications

RAPPORT TP5

Albums et Mongoose

Prénom et Nom de l'étudiant : Akash Selvaratnam

Groupe : 303

Promotion : 2023-2024

Sommaire

Table des matières

Étape 1 – C’est la base

Pour la première étape, j’avais déjà installé MongoDB Community sur mon ordinateur, je me suis donc tout simplement connecté a ma base de données MongoDB avec la commande **mongosh** puis j’ai lancé la commande **show dbs** afin de voir l’ensemble des bases de données existantes.

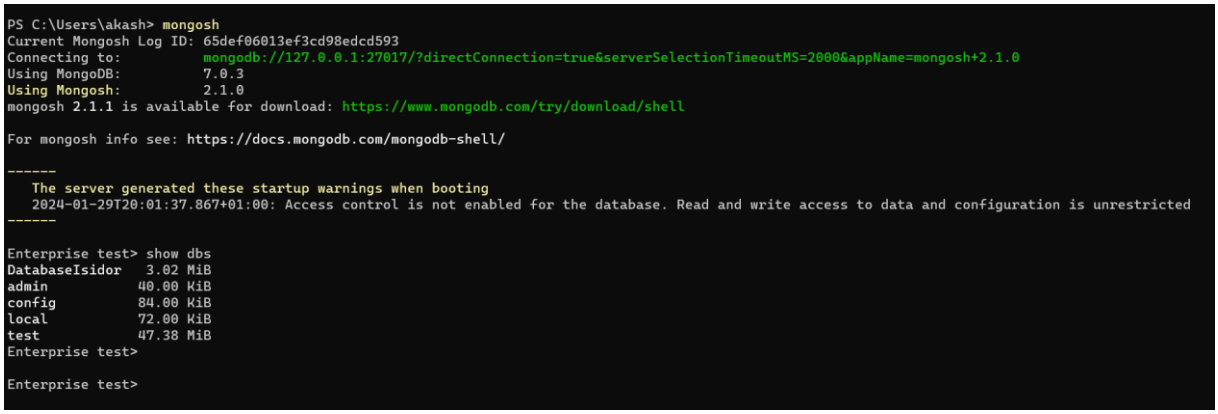

Figure 1 : Terminale mongosh avec l'ensemble des bases de données existante

Étape 2 – C’est la base

Pour la deuxième étape, j’ai installé les modules **Fastify** et **mongoose** comme on peut le voir sur le fichier **package.json.**

Figure 2 : Package.json (dépendances)

Ensuite, j’ai créé un nouveau fichier que j’ai nommé **databases.js** ou je me connecte à ma base de données **MongoDB** dans une fonction asynchrone que j’exporte.

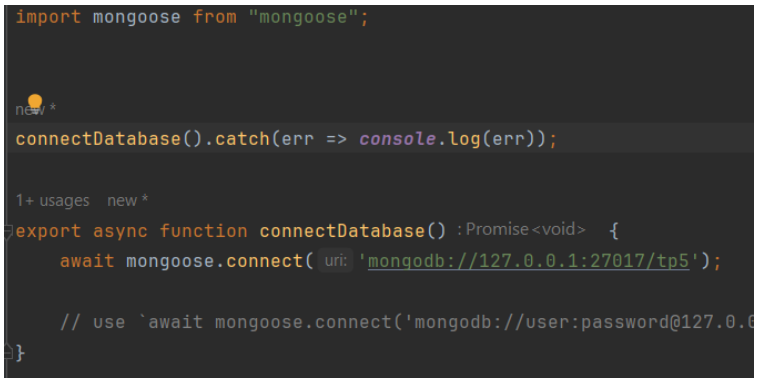

Figure 3 : databases.js

J'ai créé un nouveau fichier nommé **model.js** pour pouvoir créer un nouveau document que j'appellerai **Livre**. Pour cela, j'ai créé un nouveau schéma avec la méthode **Shema** proposé par **Mongoose**. Ce schéma contient 4 différents paramètres (**le titre et l'auteur** qui est de type **String** et qui est requis, **la description** qui est également de type **String** et le **format** qui est de type **String** et avec **comme valeur de défaut poche**, les trois valeurs acceptées pour le format sont **poche, manga et audio**).

Figure 4 : Schéma du Livre

Ensuite, j'ai créé le modèle associé au **schéma du Livre**, ce modèle, je l'ai **nommé Livre** puis **j'exporte mon Model**.

Étape 3 – Les routes

Pour l'étape 3, j'ai défini l'ensemble des routes de l'application pour les 4 opérations suivantes, l'ajout d'un livre (**POST**), la suppression d'un livre (**DELETE**), la mise à jour d'un livre (**PUT**) et l'obtention des livres dans la base de données (**GET**).

```
const livres :{…}  = {
    type : "object",
    properties: {
        title: {type: 'string'},
        author: {type: 'string'},
        description: {type: 'string'},
        format: {type: "string"}
    },
    required: ["title", "author", "format"]
}
app.route( opts: {
    method: "GET",
    url: "/Livre",
    handler : RecupereLivre,
    schema: {
        response: {
            200: {
                type: "array",
                items: livres
            }
        }
    },
});
```

Figure 5 : Route permettant de récupérer l'ensemble des livres

Ensuite, j'ai créé 4 différentes méthodes, **une première méthode permettant d'ajouter un livre**, pour cela, je me connecte à ma base de données puis je réalise les opérations nécessaires pour ajouter un nouveau livre dans mon document puis je me déconnecte de ma base de données et **je retourne le titre, le nom de l'auteur, la description et le format du livre que j'ai ajouté avec un code status 200.**

Pour cette première méthode, j'ai réalisé un **schéma JSON d'entrée** avec les **propriétés title (requis), author (requis), description et format.**

J'ai également réalisé un **schéma JSON de réponse/sortie** avec les propriété **title (requis), author (requis), description et format (requis).**

```
export async function ajoutLivre(req, rep) :Promise<…>  {
    const data = req.body;
    try {
        await connectDatabase();
        let livre :HydratedDocument<InferSchemaType = new LivreModel( doc: {
            title : data.title,
            author : data.author,
            description : data.description,
            format : data.format
        });
        await livre.save();
        await DisconnectDatabase();
        const resultatJSON :{…}  = {
            title : livre.title,
            author : livre.author,
            description : livre.description,
            format : livre.format
        }
        return rep.status(200).send(resultatJSON)
    }
    catch(e){
        console.error(e);
    }
}
```

Figure 6 : Méthode permettant d'ajouter un livre

```
app.route( opts: {
    method: "POST",
    url: "/Livre",
    handler : ajoutLivre,
    schema: {
        body: {
            type: "object",
            properties: {
                title: {type: "string"},
                author: {type: "string"},
                description : {type : "string"},
                format : {type : "string"}
            },
            required: ["title", "author"],
        },
        response: {
            200: {
                type: "object",
                properties: {
                    title: {type: 'string'},
                    author: {type: 'string'},
                    description: {type: 'string'},
                    format: {type: "string"}
                },
                required: ["title", "author", "format"]
            }
        }
    },
});
```

Figure 7 : Route de la méthode permettant d'ajouter un livre

Pour la méthode permettant de **récupérer l'ensemble des livres**, je me suis tout d'abord connecté à ma base de données puis j'ai réalisé **la méthode find()** proposé par le **module mongoose** sur tous les éléments de ma base de données, je me déconnecte ensuite de ma base de données et je **retourne un tableau JSON** qui contient **le titre, le nom de l'auteur, la description et le format du livre que j'ai récupéré avec un code status 200.**

J'ai également réalisé un **schéma JSON de réponse/sortie** avec les propriété **title (requis), author (requis), description et format (requis).**

```
export async function RecupereLivre(req, rep) : Promise<…>  {
    try {
        await connectDatabase();
        const res : Query<…> & ObtainSchemaGeneric<module:mon…  = await LivreModel.find( filter: {});
        await DisconnectDatabase();
        const resultatJSON = res.map(livres => ({
            title : livres.title,
            author : livres.author,
            description : livres.description,
            format : livres.format
        }))
        return rep.status(200).send(resultatJSON);
    }
    catch(e){
        console.error(e);
    }
}
```

Figure 8 : Méthode permettant de récupère l'ensemble des livre de la base de données

```
const livres :{…}  = {
    type : "object",
    properties: {
        title: {type: 'string'},
        author: {type: 'string'},
        description: {type: 'string'},
        format: {type: "string"}
    },
    required: ["title", "author", "format"]
}
app.route( opts: {
    method: "GET",
    url: "/Livre",
    handler : RecupereLivre,
    schema: {
        response: {
            200: {
                type: "array",
                items: livres
            }
        }
    },
});
```

Figure 9 : Route permettant de récupérer l'ensemble des livres

J'ai réalisé ensuite une méthode permettant de **mettre à jour les informations d'un livre**, je me connecte à ma base de données puis j'exécute la méthode **findOneAndUpdate()** avec **l'id du livre** qu'on souhaite mettre à jour et **les données qu'on souhaite modifier**, il s'agit d'une méthode proposée par le **module mongoose** et je me déconnecte ensuite de ma base de données. Pour finir, je retourne un objet JSON qui contient **le titre, le nom de l'auteur, la description et le format du livre avec un code status 200.**

Pour cette méthode, j'ai réalisé un **schéma JSON d'entrée** avec la propriété **id (requis).**

J'ai également réalisé un **schéma JSON de réponse/sortie** avec les propriété **title (requis), author (requis), description et format (requis).**

```
export async function MiseAJourLivre(req, rep) : Promise<…>  {
    try {
        await connectDatabase();
        const data = req.body;
        const res : ModifyResult<InferSchemaType  = await LivreModel.findOneAndUpdate( filter: { id: data.id}, update: {title : data.title , a
        await DisconnectDatabase();
        const resultatJSON : {…}  ={
            title : res.title,
            author : res.author,
            description : res.description,
            format : res.format
        };
        return rep.status(200).send(resultatJSON);
    }
    catch(e){
        console.error(e);
    }
}
```

Figure 10 : Méthode permettant de mettre à jour un livre

```
app.route( opts: {
    method: "PUT",
    url: "/Livre",
    handler : MiseAJourLivre,
    schema: {
        body: {
            type: "object",
            properties: {
                id : {type : "string"},
                title: {type: "string"},
                author: {type: "string"},
                description : {type : "string"},
                format : {type : "string"}
            },
            required: ["id"],
        },
        response: {
            200: {
                type: "object",
                properties: {
                    title: {type: 'string'},
                    author: {type: 'string'},
                    description: {type: 'string'},
                    format: {type: "string"}
                },
                required: ["title", "author", "format"]
            }
        }
    }
});
```

Figure 11 : Route permettant de mettre à jour un livre

Pour finir, j'ai réalisé une méthode permettant de **supprimer un livre**, tout d'abord, je me connecte à ma base de données puis j'exécute la **fonction findOneAndDelete()** avec comme **paramètre l'id du livre** que je souhaite supprimer puis je me déconnecte de la base de données et **je retourne un objet JSON** qui contient **le titre, le nom de l'auteur, la description et le format du livre avec un code status 200.**

Pour cette méthode, j'ai réalisé un **schéma JSON d'entrée** avec la propriété **id (requis).**

J'ai également réalisé un **schéma JSON de réponse/sortie** avec les propriété **title (requis), author (requis), description et format (requis).**

```
export async function SupprimerLivre(req, rep) :Promise<…>  {
    try {
        await connectDatabase();
        const data = req.body;
        const res :FlattenMaps<InferSchemaType<mo… & {…}  = await LivreModel.findOneAndDelete( filter: { id : data.id}).exec();
        await DisconnectDatabase();
        const resultatJSON :  = {
            title : res.title,
            author : res.author,
            description : res.description,
            format : res.format
        }
        return rep.status(200).send(resultatJSON);
    }
    catch(e){
        console.error(e);
    }
}
```

Figure 12 : Méthode permettant de supprimer un livre

```
app.route( opts: {
    method: "DELETE",
    url: "/Livre",
    handler : SupprimerLivre,
    schema: {
        body: {
            type: "object",
            properties: {
                id : {type : "string"},
            },
            required: ["id"],
        },
        response: {
            200: {
                type: "object",
                properties: {
                    title: {type: 'string'},
                    author: {type: 'string'},
                    description: {type: 'string'},
                    format: {type: "string"}
                },
                required: ["title", "author", "format"]
            }
        }
    }
});
```

Figure 13 : Route permettant de supprimer un livre