Rapport

Étape 1

Ayant déjà mongoDB sur mon ordinateur, je n'ai pas eu besoin d'installer ce dernier. Me dirigeant alors dans le terminal et écrivant la commande mongosh puis show dbs je retrouve bien les 3 bases de données initiales (admin, config et local) mais aussi les autres bases que j'ai pu créer.

```
PS C:\Users\LINos> mongosh
Current Mongosh Log ID: 65def27c80d3a74ad57a352a
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2
.0.2
Using MongoDB:          7.0.2
Using Mongosh:          2.0.2
mongosh 2.1.5 is available for download: https://www.mongodb.com/try/download/shell

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

------
   The server generated these startup warnings when booting
   2024-02-27T12:37:36.559+01:00: Access control is not enabled for the database. Read and write access to data and conf
iguration is unrestricted
------

test> show dbs
DatabaseIsidor  256.00 KiB
admin            40.00 KiB
config           72.00 KiB
local            92.00 KiB
test             25.98 MiB
test>
```

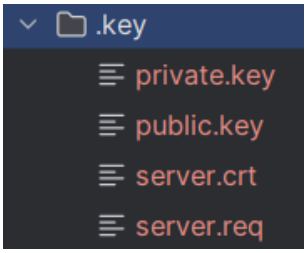
Étape 2

J'utilise les commandes openssl correspondant pour créer une clé privée et extraire la clé publique de cette dernière.

```
PS C:\Users\LINos\OneDrive\Bureau\Travaux\3A\Dev avancé\tp5\.key> openssl genrsa -out private.key
PS C:\Users\LINos\OneDrive\Bureau\Travaux\3A\Dev avancé\tp5\.key> openssl rsa -pubout -in private.key -out public.key
writing RSA key
```

Il serait bien dans un réel projet de rajouter la clé privée dans le .gitignore pour éviter de la partager dans mon dépôt.

Toujours en me basant sur les diapos du cours, je génère alors un certificat à partir de la clé privée pour fastify.



J'ajoute à la la configuration classique de fastify, l'ajout du certificat afin d'ajouter le https au serveur.

```
const fastify = Fastify({
    logger: true,
    http2: true,
    https: {
        key: fs.readFileSync(path.join(__dirname, '..', '.key', 'private.key')),
        cert: fs.readFileSync(path.join(__dirname, '..', '.key', 'server.crt'))
    }
})
```

Dans un fichier config.js dans le répertoire databases, je crée alors la connexion à la base de donnée que j'ajoute à l'initialisation du serveur pour réaliser des tests.

```
import mongoose from "mongoose"

const dbURI = "mongodb://127.0.0.1:27017/"
const dbName = "bibliotheque"

1+ usages  Oscar Lin
export function connectDB() {
    mongoose.connect(`${dbURI}${dbName}`).then(r => console.log(r))
}
```

Je me suis ensuite aidé de la documentation mongoose afin de créer le schéma suivant por correspondre aux attentes.

```
import mongoose from 'mongoose';
const { Schema } = mongoose;

const bookSchema = new Schema( definition: {
    title: {
        type: String,
        required: true
    },
    author: {
        type: String,
        required: true
    },
    description: String,
    format: {
        type: String,
        enum: ['poche', 'manga', 'audio'],
        default: 'poche'
    }
});
```

Étape 3

Afin de rajouter les diverses fonctionnalités de l'api j'ai créer les liens des requêtes et affecter à chacun un handler qui s'occupera des opérations

```
export default async (app,opts) => {
    app.route( opts: {
        method : "PUT",
        url:"/livre",
        handler : addOuvrage,
        schema : bookSchema
    });
    app.route( opts: {
        method : "GET",
        url:"/livre",
        handler : getOuvrage,
        schema : getBookSchema
    });
    app.route( opts: {
        method:"POST",
        url:"/livre",
        handler: updateOuvrage,
        schema : updateBookSchema
    })
    app.route( opts: {
        method:"DELETE",
        url:"/livre",
        handler: deleteOuvrage,
        scheme : deleteBookSchema
    })
}
```

Ces handles se situent dans controlleurs/bibliotheque.js et appelle simplement des fonctions de api.js pour faire ce qui nous intéresse avec l'ODM mongoose.

```
export const addOuvrage = async(request, reply) =>{
    try{
        const data = request.body
        await addOuvrageBD(data.title,data.author,data.description,data.format)
        reply.send("ajout réussi")
    }
    catch(e){
        reply.send(e)
    }
}
```

La fonction d'ajout se résume simplement à la création d'un nouvel objet selon le schéma du livre qu'on avait déterminé à l'étape 2

```
export const addOuvrageBD = async(title, author, description, format) => {
    try{
        await connectDB()
        const book = new Book( doc: {title,author,description,format})
        await book.save()
        await disconnectDB()
    }
    catch(e){
        throw e;
    }
}
```

La récupération de livre récupère simplement les 20 premiers livres de la BD avec la commande find et le limit qui limite le résultat à 20 ouvrages.

```
export const getOuvrageBD = async() => {
    try{
        await connectDB()
        const res = await Book.find( filter: {}).limit( val: 20);
        await disconnectDB()
        return res;
    }
    catch(e){
        throw e;
    }
}
```

Pour le update, je vérifie que le champs qu'on souhaite modifier n'est pas vide avant de réaliser la modification avec un updateOne.

```
export const updateOuvrageBD = async(title, author, description, format)=>{
    try{
        await connectDB()
        if(author){
            await Book.updateOne( filter: {title:title}, update: {author:author}).exec()
        }
        if(description){
            await Book.updateOne( filter: {title:title}, update: {description:description}).exec()
        }
        if(format){
            await Book.updateOne( filter: {title:title}, update: {format:format}).exec()
        }
        await disconnectDB()
    }
    catch(e){
        throw e;
    }
}
```

Enfin pour le delete, j'utilise simplement deleteOne pour supprimer le livre demandé.

```
export const deleteOuvrageBD = async (title) => {
    try{
        await connectDB()
        await Book.deleteOne( filter: {title:title}).exec();
        await disconnectDB()
    }
    catch(e){
        throw e;
    }
}
```

Pour toutes ses requêtes j'ai implémenté un try catch pour capturer toute erreur possible et le renvoyer à l'envoyer à l'utilisateur.

On peut aussi voir que dans le fichier routes.js des schémas sont implémentés. Je me suis aidé de la documentation fastify pour réaliser ses schémas json car il y avait une petite subtilité qui était d'intégrer le schéma json qu'on a vu en cours dans un body.

Pour la création d'un livre je demande obligatoirement le titre ainsi que l'auteur et je force pour le format des certaines valeurs.

```
export const bookSchema = {body : {
        type: "object",
        properties: {
            title: {type: "string"},
            author: {type: "string"},
            description: {type: "string"},
            format: {
                type: "string",
                enum: ["poche", "manga", "audio"]
            }
        },
        required: ["title", "author"],
    }
}
```

Dans le cas de la consultation, la vérification se fait dans les données que j'envoie pour éviter d'envoyer des données sensibles ou inutiles de la bd

```
export const getBookSchema = {
    response: {
        200: {
            type : "array" ,
            items : bookSchema
        }
    }
}
```

Pour la mise à jour c'est similaire au schéma de la création mais ici le seul champ obligatoire est le titre afin de trouver le document à modifier

```
export const updateBookSchema = {body :{
        type: "object",
        properties: {
            title: {type: "string"},
            author: {type: "string"},
            description: {type: "string"},
            format: {
                type: "string",
                enum: ["poche", "manga", "audio"]
            },
        },
        required: ["title"],
    }}
```

Enfin pour la suppression, je récupère seulement le titre de la requête pour supprimer le document.

```
export const deleteBookSchema ={body : {
        type: "object",
        properties: {
            title: {type: "string"}
        },
        required: ["title"],
    }}
```

Conclusion

Ce TP a permis d'approfondir ma compréhension des json schema et plus particulièrement comment en instaurer. J'ai fait attention de respecter le format d'API REST que j'ai déjà pu réaliser dans un projet. J'avais déjà utilisé mongoose, cela a permis de revoir mes connaissances sur cet ODM.