

Rapport de Travaux Pratiques : Conception d'une API REST avec Node.js et Express.js

Objectif

L'objectif de ce TP est de développer une API REST pour la gestion des utilisateurs, des livres et des prêts, avec une authentification sécurisée basée sur JWT. Cette API permet de persister les données dans une base de données MySQL et inclut une conteneurisation avec Docker pour une meilleure portabilité et gestion des dépendances.

1. Configuration et Préparation

1.1 Environnement de Développement




- **Technologies utilisées:**
 - Node.js et Express.js pour le backend.
 - MySQL pour la base de données.
 - JWT pour l'authentification sécurisée.
 - Docker pour la conteneurisation.

1.2 Arborescence du Projet

- project/
 - src/
 - models/
 - controllers/
 - routes/
 - middleware/
 - config/
 - config.js
 - docker-compose.yml
 - Dockerfile
 - .env
 - package.json
 - server.js

2. Conception des Modèles

2.1 Modèle Utilisateur





```
models >  user.js >  User >  id
1  const { Sequelize, DataTypes } = require('sequelize');
2  const sequelize = require('../config/config');
3
4  const User = sequelize.define('User', {
5    id: {
6      type: DataTypes.INTEGER,
7      primaryKey: true,
8      autoIncrement: true
9    },
10   firstName: {
11     type: DataTypes.STRING,
12     allowNull: false
13   },
14   lastName: {
15     type: DataTypes.STRING,
16     allowNull: false
17   },
18   age: {
19     type: DataTypes.INTEGER,
20     allowNull: false
21   },
22   school: {
23     type: DataTypes.STRING,
24     allowNull: false
25   },
```

```
    email: {
      type: DataTypes.STRING,
      allowNull: false,
      unique: true,
      validate: {
        isEmail: true
      }
    },
    password: {
      type: DataTypes.STRING,
      allowNull: false
    }
  });

module.exports = User;
```

2.2 Modèle Livre



```

models >  book.js >  Book >  author >  allowNull
1  const { Sequelize, DataTypes } = require('sequelize');
2  const sequelize = require('../config/config');
3
4  const Book = sequelize.define('Book', {
5    id: {
6      type: DataTypes.INTEGER,
7      primaryKey: true,
8      autoIncrement: true
9    },
10   title: {
11     type: DataTypes.STRING,
12     allowNull: false
13   },
14   author: {
15     type: DataTypes.STRING,
16     allowNull: false
17   },
18   edition: {
19     type: DataTypes.STRING,
20     allowNull: false
21   }
22 });
23
24 module.exports = Book;

```

2.3 Modèle Prêt

```

models >  loan.js >  Loan
1  const { Sequelize, DataTypes } = require('sequelize');
2  const sequelize = require('../config/config');
3  const User = require('../user');
4  const Book = require('../book');
5
6  const Loan = sequelize.define('Loan', {
7    id: {
8      type: DataTypes.INTEGER,
9      primaryKey: true,
10     autoIncrement: true
11   },
12   startDate: {
13     type: DataTypes.DATE,
14     allowNull: false
15   },
16   endDate: {
17     type: DataTypes.DATE,
18     allowNull: false
19   }
20 });
21
22 // Relationship
23 Loan.belongsTo(User, { foreignKey: 'userId' });
24 Loan.belongsTo(Book, { foreignKey: 'bookId' });
25
26 module.exports = Loan;

```

3. Conteneurisation avec Docker

3.1 Fichier Dockerfile

```

Dockerfile X package.json doc
Dockerfile > COPY
1 FROM node:18
2
3 WORKDIR /usr/src/app
4
5 COPY package*.json ./
6
7 RUN npm install
8
9 COPY . .
10
11 EXPOSE 3000
12
13 CMD ["npm", "run", "start"]
14

```

3.2 Fichier docker-compose.yml

```

docker-compose.yml
1 version: '3.8'
2
3 services:
4   nodejs:
5     build:
6       context: .
7       dockerfile: Dockerfile
8     ports:
9       - "3000:3000"
10    volumes:
11      - ../usr/src/app
12      - /usr/src/app/node_modules
13    depends_on:
14      - db
15    environment:
16      - DB_HOST=db
17      - DB_USER=root
18      - DB_PASSWORD=root
19      - DB_NAME=library
20    networks:
21      - backend
22  db:
23    image: mysql:8.0
24    container_name: mysql_container
25    restart: always
26    environment:

```

```

docker-compose.yml
3 services:
22  db:
26    environment:
27      MYSQL_ROOT_PASSWORD: root
28      MYSQL_DATABASE: library
29    volumes:
30      - mysql_data:/var/lib/mysql
31    networks:
32      - backend
33
34  phpmyadmin:
35    image: phpmyadmin/phpmyadmin
36    container_name: phpmyadmin_container
37    restart: always
38    ports:
39      - "8080:80"
40    environment:
41      PMA_HOST: db
42      MYSQL_ROOT_PASSWORD: root
43    depends_on:
44      - db
45    networks:
46      - backend
47
48  vscode:
49    image: codercom/code-server:latest

```

```

docker-compose.yml
3  services:
48  vscode:
49    image: codercom/code-server:latest
50    container_name: vscode_container
51    restart: always
52    ports:
53      - "8443:8080"
54    volumes:
55      - ../home/coder/project
56    environment:
57      PASSWORD: "yourpassword"
58      SUDO_PASSWORD: "yourpassword"
59    networks:
60      - backend
61
62  networks:
63    backend:
64      driver: bridge
65
66  volumes:
67    mysql_data:

```

Visualisation des conteneurs après exécution du docker-compose :

C:\Users\dell	xps>docker ps						
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES	
60a2988a3215	api-rest-nodejs-nodejs	"docker-entrypoint.s..."	9 seconds ago	Up 4 seconds	0.0.0.0:3000->3000/tcp	api-rest-nodejs-no	
e63cb2d6d4ae	phpmyadmin/phpmyadmin	"/docker-entrypoint..."	9 seconds ago	Up 4 seconds	0.0.0.0:8080->80/tcp	phpmyadmin_contain	
05511b539b38	mysql:8.0	"docker-entrypoint.s..."	9 seconds ago	Up 6 seconds	3306/tcp, 33060/tcp	mysql_container	
704cd8ee2c0d	codercom/code-server:latest	"/usr/bin/entrypoint..."	9 seconds ago	Up 6 seconds	0.0.0.0:8443->8080/tcp	vscode_container	

4. Authentification JWT

4.1 Création du Middleware

```

middleware > JS authMiddleware.js > authMiddleware
1  const jwt = require('jsonwebtoken');
2  const JWT_SECRET = process.env.JWT_SECRET;
3
4  const authMiddleware = (req, res, next) => {
5    const token = req.header('Authorization');
6    if (!token) return res.status(401).json({ message: 'Access denied' });
7
8    try {
9      const decoded = jwt.verify(token, JWT_SECRET);
10     req.user = decoded;
11     next();
12   } catch (error) {
13     res.status(400).json({ message: 'Invalid token' });
14   }
15 };
16
17 module.exports = {authMiddleware};
18

```

5. Routes et Contrôleurs

Les routes sont tous protégées par le Middleware d'authentification qui valide le Token de chaque utilisateur avant de lui donnée la possibilité de Continuer son action.

5.1 Utilisateurs

```
routes > JS userRoutes.js > ...
1  const express = require('express');
2  const router = express.Router();
3  const { getAllUsers } = require('../controllers/userController');
4  const { authMiddleware } = require('../middleware/authMiddleware');
5
6
7  router.get('/users', authMiddleware, getAllUsers);
8
9  module.exports = router;
10
```

5.2 Livres

```
routes > JS bookRoutes.js > ...
1  const express = require('express');
2  const router = express.Router();
3  const { getAllBooks, getBookById, createBook, updateBook, deleteBook } =
4  require('../controllers/bookController');
5  const { authMiddleware } = require('../middleware/authMiddleware');
6
7  router.get('/books', authMiddleware, getAllBooks);
8
9  router.get('/books/:id', authMiddleware, getBookById);
10
11 router.post('/books', authMiddleware, createBook);
12
13 router.put('/books/:id', authMiddleware, updateBook);
14
15 router.delete('/books/:id', authMiddleware, deleteBook);
16
17 module.exports = router;
18
```

5.3 Prêts

```

routes > JS loanRoutes.js > ...
1  const express = require('express');
2  const router = express.Router();
3  const { createLoan, getAllLoans, getLoanById, updateLoan, deleteLoan } =
4  require('../controllers/loanController');
5  const { authMiddleware } = require('../middleware/authMiddleware');
6
7  router.post('/loans', authMiddleware, createLoan);
8
9  router.get('/loans', authMiddleware, getAllLoans);
10
11 router.get('/loans/:id', authMiddleware, getLoanById);
12
13 router.put('/loans/:id', authMiddleware, updateLoan);
14
15 router.delete('/loans/:id', authMiddleware, deleteLoan);
16
17 module.exports = router;
18

```

5.4 Authentication

```

routes > JS authRoutes.js > ...
1  const express = require('express');
2  const router = express.Router();
3  const { signup, login } = require('../controllers/authController');
4
5  router.post('/signup', signup);
6  router.post('/login', login);
7
8  module.exports = router;
9

```

6. Gestion des Erreurs

Les erreurs sont gérées avec des messages clairs et des codes HTTP appropriés :

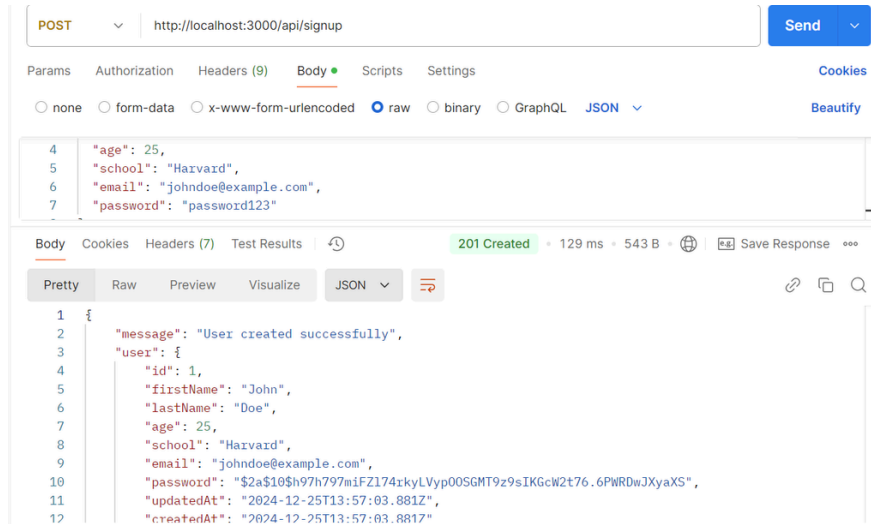
- **400** : Requête invalide.
- **401** : Non autorisé.
- **403** : Accès interdit.
- **404** : Ressource non trouvée.
- **500** : Erreur serveur.

7. Tests

Les tests ont été réalisés avec Postman :

- Vérification de toutes les routes CRUD.
- Validation des règles métiers (durée de 15 jours maximum, 2 livres par utilisateur).
- Tests de l'authentification et des tokens JWT.

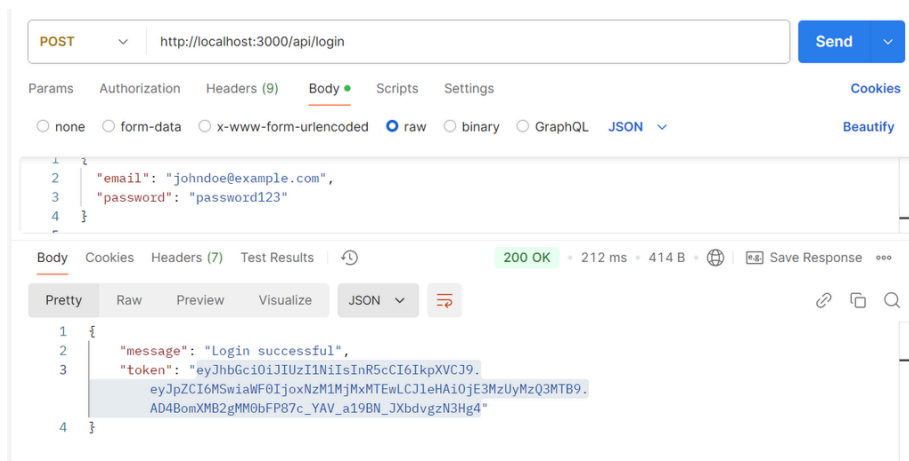
7.1 Inscription:



Résultat:

Utilisateur ajouté en Base de Données.

7.1 Login:



Résultat:

Message confirmant le Login avec succès.

Un Token unique sera livré a chaque Utilisateur. Ce token lui donne la possibilité de visiter les routes protégées

7.2 Exemple de CRUD :

Cas : Livres:

Get:

GET

http://localhost:3000/api/books/

Send

Params

Authorization

Headers (8)

Body

Scripts

Settings

Cookies

	Key	Value	Description	...	Bulk Edit	Presets
<input checked="" type="checkbox"/>	Authorization	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...				

Body

Cookies

Headers (7)

Test Results

200 OK

14 ms

554 B

Save Response

Pretty

Raw

Preview

Visualize

JSON

```
2 {
3   "id": 1,
4   "title": "New Book Title",
5   "author": "New Author",
6   "edition": "1st Edition",
7   "createdAt": "2024-12-25T14:00:17.000Z",
8   "updatedAt": "2024-12-25T14:00:17.000Z"
9 }
10 {
11   "id": 3,
12   "title": "New Book Title",
13   "author": "New Author",
14   "edition": "1st Edition",
15   "createdAt": "2024-12-25T14:01:21.000Z",
16   "updatedAt": "2024-12-25T14:01:21.000Z"
17 }
```

PUT

http://localhost:3000/api/books/2

Send

Params

Authorization

Headers (10)

Body

Scripts

Settings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

```
2 "title": "Updated Book",
3 "author": "Updated Author",
4 "edition": "4th Edition"
5 }
```

Body

Cookies

Headers (7)

Test Results

200 OK

92 ms

395 B

Save Response

Pretty

Raw

Preview

Visualize

JSON

```
1 {
2   "id": 2,
3   "title": "Updated Book",
4   "author": "Updated Author",
5   "edition": "4th Edition",
6   "createdAt": "2024-12-25T14:01:16.000Z",
7   "updatedAt": "2024-12-26T16:40:33.416Z"
8 }
```

Get/id:

GET

http://localhost:3000/api/books/3

Send

Params

Authorization

Headers (8)

Body

Scripts

Settings

7 hidden

	Key	Value	Description	...	Bu
<input checked="" type="checkbox"/>	Authorization	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...			
	Key	Value	Description		

Body

Cookies

Headers (7)

Test Results

200 OK

28 ms

393 B

Save Response

Pretty

Raw

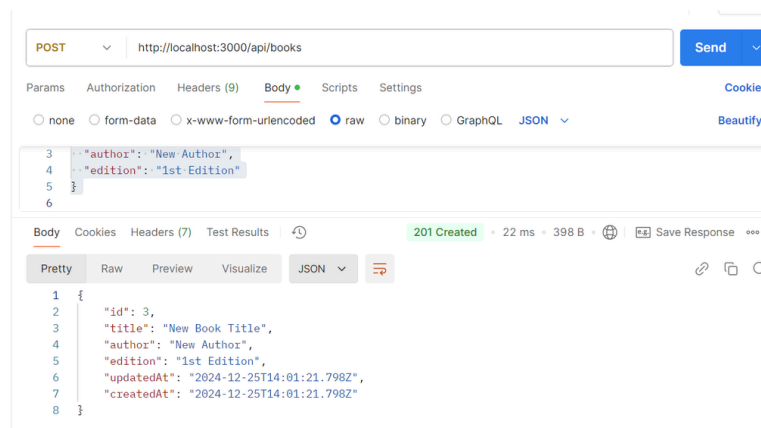
Preview

Visualize

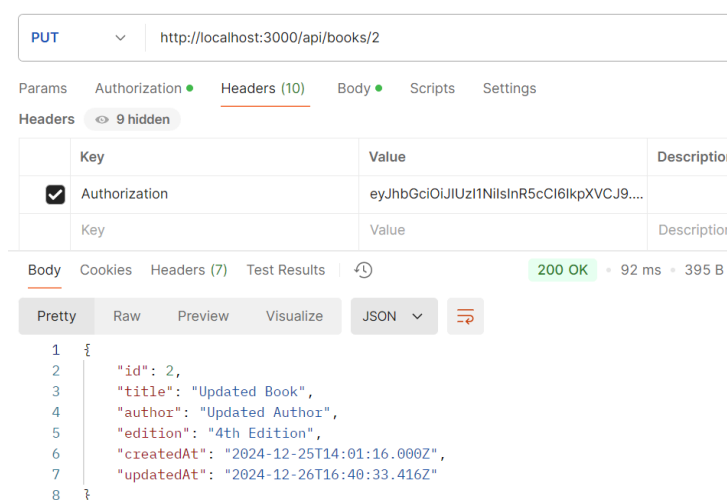
JSON

```
1 {
2   "id": 3,
3   "title": "New Book Title",
4   "author": "New Author",
5   "edition": "1st Edition",
6   "createdAt": "2024-12-25T14:01:21.000Z",
7   "updatedAt": "2024-12-25T14:01:21.000Z"
8 }
```

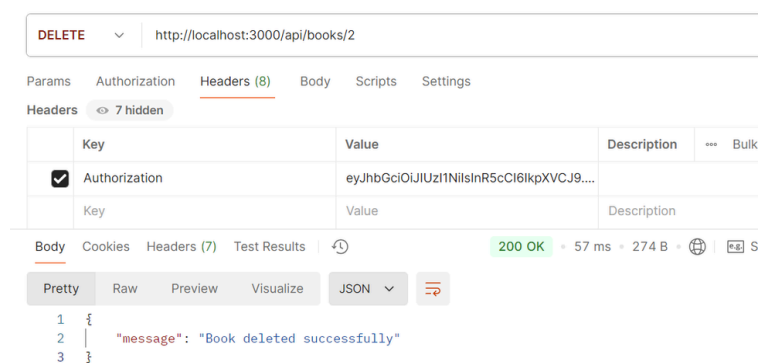
Add :



Update :



Delete :

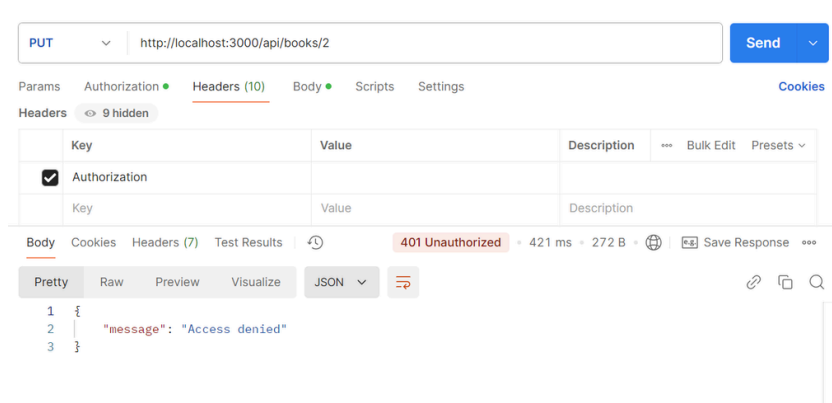


Nous ajoutons a chaque test le Token au Headers pour que l'utilisateur ait le droit d'accéder a ces routes

Résultat:

Liste des utilisateurs, un utilisateur ,ajout, modification, ou suppression avec succès.

Cas de token non valid:



Résultat:

Accès interdit.

8. Résultats

L'API répond aux besoins suivants :

- Gestion des utilisateurs, livres et prêts.
- Authentification sécurisée avec JWT.
- Respect des règles de prêt.
- Déploiement en conteneurs avec Docker.

9. Conclusion

Ce TP a permis de concevoir une API REST complète et sécurisée en respectant les bonnes pratiques de développement. La conteneurisation avec Docker assure une portabilité et une facilité de gestion des dépendances.