TP - Gestion de Bibliothèque avec SQLite et Documentation Swagger

API REST complète avec relations et documentation interactive

Date: 17 Novembre 2025

🎯 Contexte

Vous êtes développeur dans une start-up qui crée une application de gestion de bibliothèque. Votre mission est de développer une API REST permettant de gérer des **livres** et des **auteurs**, avec leurs relations.

Le système doit permettre :

- 📚 Gérer les livres (titre, année de publication, ISBN)
- ✍️ Gérer les auteurs (nom, prénom, nationalité)
- 🔗 Associer des livres à leurs auteurs (un livre peut avoir plusieurs auteurs)
- 📖 Documenter l'API avec Swagger pour faciliter son utilisation

---

🎓 Objectifs d'Apprentissage

À la fin de ce TP, vous saurez :

- ✅ Créer une base de données SQLite avec relations (tables avec clés étrangères)
- ✅ Implémenter une architecture MVC complète (Routes, Controllers, Services)
- ✅ Gérer des relations Many-to-Many entre entités
- ✅ Créer des routes CRUD pour deux entités différentes
- ✅ Documenter une API REST avec Swagger/OpenAPI
- ✅ Tester vos endpoints via l'interface Swagger UI

🚀 Configuration Initiale

Étape 1 : Créer le projet

```
# Créer le dossier du projet
mkdir tp-bibliotheque
cd tp-bibliotheque

# Initialiser npm
npm init -y

# Installer les dépendances
npm install express better-sqlite3 validator
npm install swagger-jsdoc swagger-ui-express
npm install --save-dev nodemon
```

Étape 2 : Configurer package.json

Ajoutez le script de développement dans `package.json` :

```
{
  "scripts": {
    "dev": "nodemon index.js"
  }
}
```

Étape 3 : Créer la structure du projet

```
# Créer les dossiers
mkdir db controllers routes services middlewares

# Créer les fichiers principaux
touch index.js
touch swagger.js
touch db/init.js
touch controllers/auteurs.controller.js
touch controllers/livres.controller.js
touch routes/auteurs.router.js
touch routes/livres.router.js
touch services/auteurs.service.js
touch services/livres.service.js
touch middlewares/validation.middleware.js
```

Structure finale attendue :

```
tp-bibliotheque/
├── index.js
├── swagger.js
├── package.json
├── db/
│   ├── init.js
│   └── bibliotheque.db (sera créé automatiquement)
├── controllers/
│   ├── auteurs.controller.js
│   └── livres.controller.js
├── routes/
│   ├── auteurs.router.js
│   └── livres.router.js
├── services/
│   ├── auteurs.service.js
│   └── livres.service.js
└── middlewares/
    └── validation.middleware.js
```

□ PARTIE 1 : Base de Données SQLite

Étape 1.1 : Initialisation de la base de données

Fichier `db/init.js` :

```
const Database = require('better-sqlite3');
const path = require('path');

// Créer ou ouvrir la base de données
const dbPath = path.join(__dirname, 'bibliotheque.db');
const db = new Database(dbPath);

console.log('□ Connexion à la base de données bibliothèque...');

// Activer les clés étrangères (important pour SQLite)
db.pragma('foreign_keys = ON');

// Créer la table AUTEURS
db.exec(`
    CREATE TABLE IF NOT EXISTS auteurs (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        nom TEXT NOT NULL,
        prenom TEXT NOT NULL,
        nationalite TEXT,
        created_at DATETIME DEFAULT CURRENT_TIMESTAMP
```

```
    )
`);

// Créer la table LIVRES
db.exec(`
    CREATE TABLE IF NOT EXISTS livres (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        titre TEXT NOT NULL,
        isbn TEXT UNIQUE NOT NULL,
        annee_publication INTEGER,
        created_at DATETIME DEFAULT CURRENT_TIMESTAMP
    )
`);

// Créer la table de relation Many-to-Many
db.exec(`
    CREATE TABLE IF NOT EXISTS livres_auteurs (
        livre_id INTEGER NOT NULL,
        auteur_id INTEGER NOT NULL,
        PRIMARY KEY (livre_id, auteur_id),
        FOREIGN KEY (livre_id) REFERENCES livres(id) ON DELETE CASCADE,
        FOREIGN KEY (auteur_id) REFERENCES auteurs(id) ON DELETE CASCADE
    )
`);

// Vérifier si les tables sont vides et insérer des données de test
const countAuteurs = db.prepare('SELECT COUNT(*) as count FROM auteurs').get();
const countLivres = db.prepare('SELECT COUNT(*) as count FROM livres').get();

if (countAuteurs.count === 0) {
    console.log('📝 Insertion des auteurs de test...');

    const insertAuteur = db.prepare(`
        INSERT INTO auteurs (nom, prenom, nationalite)
        VALUES (?, ?, ?)
    `);

    insertAuteur.run('Hugo', 'Victor', 'Française');
    insertAuteur.run('Dumas', 'Alexandre', 'Française');
    insertAuteur.run('Tolkien', 'J.R.R.', 'Britannique');
    insertAuteur.run('Rowling', 'J.K.', 'Britannique');

    console.log('✅ Auteurs insérés avec succès');
}

if (countLivres.count === 0) {
    console.log('📝 Insertion des livres de test...');

    const insertLivre = db.prepare(`
        INSERT INTO livres (titre, isbn, annee_publication)
        VALUES (?, ?, ?)
    `);

    const livre1 = insertLivre.run('Les Misérables', '978-2-07-036633-4', 1862);
    const livre2 = insertLivre.run('Le Comte de Monte-Cristo', '978-2-07-045620-6', 1844);
    const livre3 = insertLivre.run('Le Seigneur des Anneaux', '978-2-266-15410-9', 1954);
    const livre4 = insertLivre.run('Harry Potter à l\'école des sorciers', '978-2-07-054127-8', 1997);

    // Associer les livres aux auteurs
    const insertRelation = db.prepare(`
        INSERT INTO livres_auteurs (livre_id, auteur_id)
        VALUES (?, ?)
    `);

    insertRelation.run(livre1.lastInsertRowid, 1); // Les Misérables -> Victor Hugo
    insertRelation.run(livre2.lastInsertRowid, 2); // Monte-Cristo -> Alexandre Dumas
```

```
    insertRelation.run(livre2.lastInsertRowid, 2); // Monte-Cristo -> Alexandre Dumas
    insertRelation.run(livre3.lastInsertRowid, 3); // LOTR -> Tolkien
    insertRelation.run(livre4.lastInsertRowid, 4); // Harry Potter -> Rowling

    console.log('✅ Livres et relations insérés avec succès');
}

console.log('✅ Base de données initialisée avec succès');

module.exports = db;
```

❓ Questions de compréhension :

1. Pourquoi active-t-on `foreign_keys = ON` ?
2. Qu'est-ce que `ON DELETE CASCADE` signifie ?
3. Pourquoi utilise-t-on une table `livres_auteurs` séparée ?

🔧 PARTIE 2 : Services (Couche d'accès aux données)

Étape 2.1 : Service Auteurs

Fichier `services/auteurs.service.js` :

```
const db = require('../db/init');

// Créer un auteur
const createAuteur = (nom, prenom, nationalite, callback) => {
    try {
        const stmt = db.prepare(`
            INSERT INTO auteurs (nom, prenom, nationalite)
            VALUES (?, ?, ?)
        `);
        const result = stmt.run(nom, prenom, nationalite);

        const newAuteur = {
            id: result.lastInsertRowid,
            nom,
            prenom,
            nationalite
        };

        callback(null, newAuteur);
    } catch (error) {
        callback(error, null);
    }
};

// Récupérer tous les auteurs
const getAllAuteurs = (callback) => {
    try {
        const stmt = db.prepare('SELECT * FROM auteurs ORDER BY nom, prenom');
        const auteurs = stmt.all();
        callback(null, auteurs);
    } catch (error) {
        callback(error, null);
    }
};

// Récupérer un auteur par ID avec ses livres
const getAuteurById = (id, callback) => {
    try {
        const stmtAuteur = db.prepare('SELECT * FROM auteurs WHERE id = ?');
        const auteur = stmtAuteur.get(id);

        if (!auteur) {
```

```
        if (!auteur) {
            return callback(new Error('Auteur non trouvé'), null);
        }

        // Récupérer les livres de cet auteur
        const stmtLivres = db.prepare(`
            SELECT l.* FROM livres l
            JOIN livres_auteurs la ON l.id = la.livre_id
            WHERE la.auteur_id = ?
        `);
        const livres = stmtLivres.all(id);

        auteur.livres = livres;
        callback(null, auteur);
    } catch (error) {
        callback(error, null);
    }
};

// Supprimer un auteur
const deleteAuteur = (id, callback) => {
    try {
        const checkStmt = db.prepare('SELECT * FROM auteurs WHERE id = ?');
        const auteur = checkStmt.get(id);

        if (!auteur) {
            return callback(new Error('Auteur non trouvé'), null);
        }

        const deleteStmt = db.prepare('DELETE FROM auteurs WHERE id = ?');
        deleteStmt.run(id);

        callback(null, { message: 'Auteur supprimé avec succès' });
    } catch (error) {
        callback(error, null);
    }
};

module.exports = {
    createAuteur,
    getAllAuteurs,
    getAuteurById,
    deleteAuteur
};
```

Étape 2.2 : Service Livres (À COMPLÉTER)

Fichier `services/livres.service.js` :

```
const db = require('../db/init');

// TODO: Créer un livre
const createLivre = (titre, isbn, annee_publication, auteur_ids, callback) => {
    try {
        // 1. Insérer le livre
        const stmtLivre = db.prepare(`
            INSERT INTO livres (titre, isbn, annee_publication)
            VALUES (?, ?, ?)
        `);
        const result = stmtLivre.run(titre, isbn, annee_publication);
        const livreId = result.lastInsertRowid;

        // 2. Associer les auteurs (auteur_ids est un tableau)
        if (auteur_ids && auteur_ids.length > 0) {
            const stmtRelation = db.prepare(`
                INSERT INTO livres_auteurs (livre_id, auteur_id)
```

```
                INSERT INTO livres_auteurs (livre_id, auteur_id)
                VALUES (?, ?)
            `);

            for (const auteurId of auteur_ids) {
                stmtRelation.run(livreId, auteurId);
            }
        }

        // 3. Récupérer le livre créé avec ses auteurs
        // TODO: Compléter cette partie

    } catch (error) {
        callback(error, null);
    }
};

// TODO: Récupérer tous les livres avec leurs auteurs
const getAllLivres = (callback) => {
    try {
        // Astuce: Utilisez une requête avec GROUP_CONCAT pour joindre les auteurs
        const stmt = db.prepare(`
            SELECT
                l.*,
                GROUP_CONCAT(a.nom || ' ' || a.prenom, ', ') as auteurs
            FROM livres l
            LEFT JOIN livres_auteurs la ON l.id = la.livre_id
            LEFT JOIN auteurs a ON la.auteur_id = a.id
            GROUP BY l.id
            ORDER BY l.titre
        `);
        const livres = stmt.all();
        callback(null, livres);
    } catch (error) {
        callback(error, null);
    }
};

// TODO: Récupérer un livre par ID
const getLivreById = (id, callback) => {
    // À implémenter
    // Indice: Similaire à getAuteurById mais inversé
};

// TODO: Supprimer un livre
const deleteLivre = (id, callback) => {
    // À implémenter
};

module.exports = {
    createLivre,
    getAllLivres,
    getLivreById,
    deleteLivre
};
```

✏ **EXERCICE 1** : Complétez les fonctions `getLivreById` et `deleteLivre`.

🎮 PARTIE 3 : Controllers

Étape 3.1 : Controller Auteurs

Fichier `controllers/auteurs.controller.js` :

```
const auteursService = require('../services/auteurs.service');

exports.createAuteur = async (req, res, next) => {
    try {
        const { nom, prenom, nationalite } = req.body;

        auteursService.createAuteur(nom, prenom, nationalite, (err, data) => {
            if (err) {
                return res.status(500).json({ error: 'Erreur lors de la création de l\'auteur' });
            }
            res.status(201).json({
                message: 'Auteur créé avec succès',
                auteur: data
            });
        });
    } catch (error) {
        next(error);
    }
};

exports.getAllAuteurs = async (req, res, next) => {
    try {
        auteursService.getAllAuteurs((err, data) => {
            if (err) {
                return res.status(500).json({ error: 'Erreur lors de la récupération des auteurs' });
            }
            res.status(200).json(data);
        });
    } catch (error) {
        next(error);
    }
};

exports.getAuteurById = async (req, res, next) => {
    try {
        const { id } = req.params;

        auteursService.getAuteurById(id, (err, data) => {
            if (err) {
                return res.status(404).json({ error: err.message });
            }
            res.status(200).json(data);
        });
    } catch (error) {
        next(error);
    }
};

exports.deleteAuteur = async (req, res, next) => {
    try {
        const { id } = req.params;

        auteursService.deleteAuteur(id, (err, data) => {
            if (err) {
                return res.status(404).json({ error: err.message });
            }
            res.status(200).json(data);
        });
    } catch (error) {
        next(error);
    }
};
```

Étape 3.2 : Controller Livres (À COMPLÉTER)

Fichier `controllers/livres.controller.js` :

```
const livresService = require('../services/livres.service');

// TODO: Implémenter les 4 fonctions suivantes
exports.createLivre = async (req, res, next) => {
    // À compléter
};

exports.getAllLivres = async (req, res, next) => {
    // À compléter
};

exports.getLivreById = async (req, res, next) => {
    // À compléter
};

exports.deleteLivre = async (req, res, next) => {
    // À compléter
};
```

✏ **EXERCICE 2 :** Complétez les 4 fonctions du controller Livres en suivant le modèle du controller Auteurs.

🛣 PARTIE 4 : Routes

Étape 4.1 : Routes Auteurs

Fichier `routes/auteurs.router.js` :

```
const express = require('express');
const router = express.Router();
const auteursController = require('../controllers/auteurs.controller');

/**
 * @swagger
 * /api/auteurs:
 *   post:
 *     summary: Créer un nouvel auteur
 *     tags: [Auteurs]
 *     requestBody:
 *       required: true
 *       content:
 *         application/json:
 *           schema:
 *             type: object
 *             required:
 *               - nom
 *               - prenom
 *             properties:
 *               nom:
 *                 type: string
 *                 example: "Zola"
 *               prenom:
 *                 type: string
 *                 example: "Émile"
 *               nationalite:
 *                 type: string
 *                 example: "Française"
 *     responses:
 *       201:
 *         description: Auteur créé avec succès
 *       500:
 *         description: Erreur serveur
```

```
 */
router.post('/', auteursController.createAuteur);

/**
 * @swagger
 * /api/auteurs:
 *   get:
 *     summary: Récupérer tous les auteurs
 *     tags: [Auteurs]
 *     responses:
 *       200:
 *         description: Liste des auteurs
 *         content:
 *           application/json:
 *             schema:
 *               type: array
 *               items:
 *                 type: object
 *                 properties:
 *                   id:
 *                     type: integer
 *                   nom:
 *                     type: string
 *                   prenom:
 *                     type: string
 *                   nationalite:
 *                     type: string
 */
router.get('/', auteursController.getAllAuteurs);

/**
 * @swagger
 * /api/auteurs/{id}:
 *   get:
 *     summary: Récupérer un auteur par ID avec ses livres
 *     tags: [Auteurs]
 *     parameters:
 *       - in: path
 *         name: id
 *         required: true
 *         schema:
 *           type: integer
 *     responses:
 *       200:
 *         description: Détails de l'auteur
 *       404:
 *         description: Auteur non trouvé
 */
router.get('/:id', auteursController.getAuteurById);

/**
 * @swagger
 * /api/auteurs/{id}:
 *   delete:
 *     summary: Supprimer un auteur
 *     tags: [Auteurs]
 *     parameters:
 *       - in: path
 *         name: id
 *         required: true
 *         schema:
 *           type: integer
 *     responses:
 *       200:
 *         description: Auteur supprimé
 *       404:
```

```
 *         description: Auteur non trouvé
 */
router.delete('/:id', auteursController.deleteAuteur);

module.exports = router;
```

Étape 4.2 : Routes Livres (À COMPLÉTER)

Fichier `routes/livres.router.js` :

```
const express = require('express');
const router = express.Router();
const livresController = require('../controllers/livres.controller');

/**
 * @swagger
 * /api/livres:
 *   post:
 *     summary: Créer un nouveau livre
 *     tags: [Livres]
 *     requestBody:
 *       required: true
 *       content:
 *         application/json:
 *           schema:
 *             type: object
 *             required:
 *               - titre
 *               - isbn
 *             properties:
 *               titre:
 *                 type: string
 *                 example: "1984"
 *               isbn:
 *                 type: string
 *                 example: "978-2-07-036822-2"
 *               annee_publication:
 *                 type: integer
 *                 example: 1949
 *               auteur_ids:
 *                 type: array
 *                 items:
 *                   type: integer
 *                 example: [1, 2]
 *     responses:
 *       201:
 *         description: Livre créé avec succès
 */
router.post('/', livresController.createLivre);

// TODO: Ajouter les routes suivantes avec leur documentation Swagger:
// - GET /api/livres (récupérer tous les livres)
// - GET /api/livres/:id (récupérer un livre par ID)
// - DELETE /api/livres/:id (supprimer un livre)

module.exports = router;
```

✏ **EXERCICE 3 :** Complétez les 3 routes manquantes avec leur documentation Swagger.

⚙ PARTIE 5 : Configuration Swagger

Fichier `swagger.js` :

```
 const swaggerJsdoc = require('swagger-jsdoc');
const path = require('path');

const options = {
  definition: {
    openapi: '3.0.0',
    info: {
      title: 'API Gestion Bibliothèque',
      version: '1.0.0',
      description: 'API REST pour gérer une bibliothèque (livres et auteurs)',
      contact: {
        name: 'Votre Nom',
        email: 'votre.email@example.com'
      }
    },
    servers: [
      {
        url: 'http://localhost:3000',
        description: 'Serveur de développement',
      },
    ],
    tags: [
      {
        name: 'Auteurs',
        description: 'Gestion des auteurs'
      },
      {
        name: 'Livres',
        description: 'Gestion des livres'
      }
    ]
  },
  apis: [path.join(__dirname, './routes/*.js')],
};

const swaggerSpec = swaggerJsdoc(options);

module.exports = swaggerSpec;
```

PARTIE 6 : Serveur Principal

Fichier `index.js` :

```
const express = require('express');
const swaggerUi = require('swagger-ui-express');
const swaggerSpec = require('./swagger');

// Initialiser la base de données
require('./db/init');

const app = express();
const port = 3000;

// Middlewares
app.use(express.json());
app.use(express.urlencoded({ extended: true }));

// Documentation Swagger
app.get('/api-docs.json', (req, res) => {
  res.setHeader('Content-Type', 'application/json');
  res.send(swaggerSpec);
});
app.use('/api-docs', swaggerUi.serve, swaggerUi.setup(swaggerSpec));

// Routes
const auteursRouter = require('./routes/auteurs.router');
const livresRouter = require('./routes/livres.router');

app.use('/api/auteurs', auteursRouter);
app.use('/api/livres', livresRouter);

// Route par défaut
app.get('/', (req, res) => {
  res.json({
    message: '🚀 Bienvenue sur l\'API Bibliothèque',
    documentation: 'http://localhost:3000/api-docs',
    endpoints: {
      auteurs: 'http://localhost:3000/api/auteurs',
      livres: 'http://localhost:3000/api/livres'
    }
  });
});

// Gestion 404
app.use('*', (req, res, next) => {
  const error = new Error('Route non trouvée');
  error.status = 404;
  next(error);
});

// Gestion des erreurs
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(err.status || 500).json({
    error: err.message || 'Erreur interne du serveur'
  });
});

app.listen(port, () => {
  console.log(`🚀 Serveur démarré sur http://localhost:${port}`);
  console.log(`📚 Documentation Swagger: http://localhost:${port}/api-docs`);
});
```

🧪 PARTIE 7 : Tests et Validation

Étape 7.1 : Démarrer le serveur

```
npm run dev
```

Étape 7.2 : Tester avec Swagger UI

1. Ouvrez votre navigateur à `http://localhost:3000/api-docs`
2. Testez chaque endpoint :

Tests à effectuer :

Auteurs

- ☐ GET `/api/auteurs` - Lister tous les auteurs
- ☐ POST `/api/auteurs` - Créer un nouvel auteur
- ☐ GET `/api/auteurs/1` - Récupérer l'auteur avec ID 1
- ☐ DELETE `/api/auteurs/1` - Supprimer un auteur

Livres

- ☐ GET `/api/livres` - Lister tous les livres
- ☐ POST `/api/livres` - Créer un nouveau livre avec auteurs
- ☐ GET `/api/livres/1` - Récupérer le livre avec ID 1
- ☐ DELETE `/api/livres/1` - Supprimer un livre

Étape 7.3 : Tests manuels avec cURL (optionnel)

```
# Créer un auteur
curl -X POST http://localhost:3000/api/auteurs \
  -H "Content-Type: application/json" \
  -d '{"nom":"Orwell","prenom":"George","nationalite":"Britannique"}'

# Créer un livre
curl -X POST http://localhost:3000/api/livres \
  -H "Content-Type: application/json" \
  -d '{"titre":"1984","isbn":"978-2-07-036822-2","annee_publication":1949,"auteur_ids":[5]}'

# Récupérer tous les livres
curl http://localhost:3000/api/livres
```

Exercices Bonus (Optionnels)

Bonus 1 : Validation avec Middleware

Créez un middleware de validation dans `middlewares/validation.middleware.js` :

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

exports.validateAuteur = (req, res, next) => {
    const { nom, prenom } = req.body;

    if (!nom || !prenom) {
        return res.status(400).json({
            error: 'Les champs nom et prenom sont requis'
        });
    }

    if (!validator.isLength(nom, { min: 2, max: 50 })) {
        return res.status(400).json({
            error: 'Le nom doit contenir entre 2 et 50 caractères'
        });
    }

    next();
};

exports.validateLivre = (req, res, next) => {
    // TODO: Valider titre (requis, 2-200 caractères)
    // TODO: Valider isbn (requis, format ISBN)
    // TODO: Valider annee_publication (optionnel, entre 1000 et année actuelle)
};
```

Intégrez ces middlewares dans vos routes.

Bonus 2 : Route de Recherche

Ajoutez une route de recherche de livres par titre :

```
// Dans livres.router.js
/**
 * @swagger
 * /api/livres/search:
 *   get:
 *     summary: Rechercher des livres par titre
 *     tags: [Livres]
 *     parameters:
 *       - in: query
 *         name: titre
 *         schema:
 *           type: string
 *         required: true
 *     responses:
 *       200:
 *         description: Résultats de recherche
 */
router.get('/search', livresController.searchLivres);
```

Implémentez la fonction correspondante avec une requête SQL `LIKE`.

Bon courage ! 💪