TP - Documentation d'API REST avec Swagger/OpenAPI 3.0

Express, SQLite et Documentation Interactive

IUT Nord Franche-Comté Objectif: Créer une API REST documentée avec Swagger/OpenAPI 3.0 et générer une interface interactive

---

Contexte

Vous êtes développeur backend dans une startup et devez créer une API REST pour gérer une bibliothèque de livres. L'API doit être **parfaitement documentée** pour faciliter l'intégration par l'équipe frontend et les partenaires externes. Vous utiliserez **Swagger/OpenAPI 3.0** pour générer une documentation interactive et standardisée.

Objectifs d'Apprentissage

À la fin de ce TP, vous saurez:

- ✅ Comprendre la spécification OpenAPI 3.0
- ✅ Écrire un fichier `swagger.yaml` complet
- ✅ Documenter les endpoints avec méthodes HTTP
- ✅ Définir des schémas de données réutilisables
- ✅ Utiliser `$ref` pour éviter la duplication
- ✅ Intégrer Swagger UI dans Express
- ✅ Tester l'API directement depuis la documentation

Prérequis

Avant de commencer, assurez-vous d'avoir: - Node.js installé (v16 ou supérieur) - Un éditeur de code (VS Code recommandé) - Des connaissances de base en Express et REST

Partie 1 : Mise en Place du Projet

Exercice 1.1 : Initialiser le Projet

📝 **Task 1:** Créez la structure du projet

```bash
# Créer le dossier du projet
mkdir bibliotheque-api
cd bibliotheque-api

# Initialiser npm
npm init -y

# Installer les dépendances
npm install express sqlite3 swagger-ui-express yamljs

# Installer nodemon en dev
npm install --save-dev nodemon
```

💡 **Packages installés:** - `express` - Framework web - `sqlite3` - Base de données - `swagger-ui-express` - Interface Swagger UI - `yamljs` - Parser YAML pour charger la documentation - `nodemon` - Rechargement automatique en développement

---

📝 **Task 2:** Créez la structure de dossiers

```bash
mkdir -p app/config app/models app/routes app/controllers app/services docs

# Votre structure devrait ressembler à:
# bibliotheque-api/
# ├── app/
# │   ├── config/
# │   ├── models/
# │   ├── routes/
# │   ├── controllers/
# │   ├── services/
# │   └── app.js
# ├── docs/
# │   └── swagger.yaml
# ├── server.js
# └── package.json
```

📝 **Task 3:** Modifiez `package.json` pour ajouter les scripts

```json
{
  "name": "bibliotheque-api",
  "version": "1.0.0",
  "main": "server.js",
  "scripts": {
    "start": "node server.js",
    "dev": "nodemon server.js"
  }
}
```

Exercice 1.2 : Créer la Base de Données SQLite

📝 **Task 4:** Créez `app/config/database.js`

```
const sqlite3 = require('sqlite3').verbose();
const path = require('path');

// Chemin vers le fichier de base de données
const dbPath = path.resolve(__dirname, '../../database.sqlite');

// Créer la connexion
const db = new sqlite3.Database(dbPath, (err) => {
  if (err) {
    console.error('Erreur lors de la connexion à la base de données:',
err.message);
  } else {
    console.log('Connecté à la base de données SQLite');
  }
});

// Activer les clés étrangères
db.run('PRAGMA foreign_keys = ON');

module.exports = db;
```

📝 **Task 5:** Créez `app/models/Book.js`

```
const db = require('../config/database');

class Book {
  /**
   * Initialiser la table books
   */
  static initTable() {
    return new Promise((resolve, reject) => {
      const sql = `
        CREATE TABLE IF NOT EXISTS books (
          id INTEGER PRIMARY KEY AUTOINCREMENT,
          title TEXT NOT NULL,
          author TEXT NOT NULL,
          isbn TEXT UNIQUE NOT NULL,
          published_year INTEGER,
          genre TEXT,
          available BOOLEAN DEFAULT 1,
          created_at DATETIME DEFAULT CURRENT_TIMESTAMP
        )
```

```
      `;

      db.run(sql, (err) => {
        if (err) reject(err);
        else resolve();
      });
    });
  }

  /**
   * Créer un nouveau livre
   */
  static create(bookData) {
    return new Promise((resolve, reject) => {
      const { title, author, isbn, published_year, genre } = bookData;
      const sql = `
        INSERT INTO books (title, author, isbn, published_year, genre)
        VALUES (?, ?, ?, ?, ?)
      `;

      db.run(sql, [title, author, isbn, published_year, genre], function(err)
{
        if (err) reject(err);
        else {
          Book.getById(this.lastID).then(resolve).catch(reject);
        }
      });
    });
  }

  /**
   * Récupérer tous les livres
   */
  static getAll() {
    return new Promise((resolve, reject) => {
      const sql = 'SELECT * FROM books ORDER BY created_at DESC';
      db.all(sql, [], (err, rows) => {
        if (err) reject(err);
        else resolve(rows);
      });
    });
  }

  /**
   * Récupérer un livre par ID
   */
  static getById(id) {
    return new Promise((resolve, reject) => {
      const sql = 'SELECT * FROM books WHERE id = ?';
```

```
      db.get(sql, [id], (err, row) => {
        if (err) reject(err);
        else resolve(row);
      });
    });
  }

  /**
   * Mettre à jour un livre
   */
  static update(id, bookData) {
    return new Promise((resolve, reject) => {
      const { title, author, isbn, published_year, genre, available } =
bookData;
      const sql = `
        UPDATE books
        SET title = ?, author = ?, isbn = ?, published_year = ?, genre = ?,
available = ?
        WHERE id = ?
      `;

      db.run(sql, [title, author, isbn, published_year, genre, available,
id], function(err) {
        if (err) reject(err);
        else if (this.changes === 0) reject(new Error('Book not found'));
        else {
          Book.getById(id).then(resolve).catch(reject);
        }
      });
    });
  }

  /**
   * Supprimer un livre
   */
  static delete(id) {
    return new Promise((resolve, reject) => {
      const sql = 'DELETE FROM books WHERE id = ?';
      db.run(sql, [id], function(err) {
        if (err) reject(err);
        else if (this.changes === 0) reject(new Error('Book not found'));
        else resolve(true);
      });
    });
  }
}

module.exports = Book;
```

Partie 2 : Créer l'API Express (sans documentation)

Exercice 2.1 : Service Layer

📝 **Task 6:** Créez `app/services/bookService.js`

```
const Book = require('../models/Book');

class BookService {
  static async createBook(bookData) {
    // Validation
    if (!bookData.title || !bookData.author || !bookData.isbn) {
      throw new Error('Title, author, and ISBN are required');
    }

    // Validation ISBN (simplifié)
    if (bookData.isbn.length < 10) {
      throw new Error('Invalid ISBN format');
    }

    // Validation année de publication
    if (bookData.published_year && (bookData.published_year < 1000 ||
bookData.published_year > new Date().getFullYear())) {
      throw new Error('Invalid published year');
    }

    return await Book.create(bookData);
  }

  static async getAllBooks() {
    return await Book.getAll();
  }

  static async getBookById(id) {
    const book = await Book.getById(id);
    if (!book) {
      throw new Error('Book not found');
    }
    return book;
  }

  static async updateBook(id, bookData) {
    // Validation
    if (!bookData.title || !bookData.author || !bookData.isbn) {
      throw new Error('Title, author, and ISBN are required');
    }

    return await Book.update(id, bookData);
  }
```

```
  static async deleteBook(id) {
    return await Book.delete(id);
  }
}

module.exports = BookService;
```

Exercice 2.2 : Controller Layer

📝 **Task 7:** Créez app/controllers/bookController.js

```
const BookService = require('../services/bookService');

class BookController {
  static async createBook(req, res) {
    try {
      const book = await BookService.createBook(req.body);
      res.status(201).json({
        success: true,
        message: 'Book created successfully',
        data: book
      });
    } catch (error) {
      res.status(400).json({
        success: false,
        message: error.message
      });
    }
  }

  static async getAllBooks(req, res) {
    try {
      const books = await BookService.getAllBooks();
      res.status(200).json({
        success: true,
        message: 'Books retrieved successfully',
        data: books,
        count: books.length
      });
    } catch (error) {
      res.status(500).json({
        success: false,
        message: error.message
      });
    }
  }
```

```
  static async getBookById(req, res) {
    try {
      const book = await BookService.getBookById(parseInt(req.params.id));
      res.status(200).json({
        success: true,
        message: 'Book retrieved successfully',
        data: book
      });
    } catch (error) {
      const statusCode = error.message === 'Book not found' ? 404 : 500;
      res.status(statusCode).json({
        success: false,
        message: error.message
      });
    }
  }

  static async updateBook(req, res) {
    try {
      const book = await BookService.updateBook(parseInt(req.params.id),
req.body);
      res.status(200).json({
        success: true,
        message: 'Book updated successfully',
        data: book
      });
    } catch (error) {
      const statusCode = error.message === 'Book not found' ? 404 : 400;
      res.status(statusCode).json({
        success: false,
        message: error.message
      });
    }
  }

  static async deleteBook(req, res) {
    try {
      await BookService.deleteBook(parseInt(req.params.id));
      res.status(200).json({
        success: true,
        message: 'Book deleted successfully'
      });
    } catch (error) {
      const statusCode = error.message === 'Book not found' ? 404 : 500;
      res.status(statusCode).json({
        success: false,
        message: error.message
      });
    }
  }
```

```
}

module.exports = BookController;
```

Exercice 2.3 : Routes

📝 **Task 8:** Créez app/routes/bookRoutes.js

```
const express = require('express');
const router = express.Router();
const BookController = require('../controllers/bookController');

// Create a new book
router.post('/', BookController.createBook);

// Get all books
router.get('/', BookController.getAllBooks);

// Get book by ID
router.get('/:id', BookController.getBookById);

// Update book
router.put('/:id', BookController.updateBook);

// Delete book
router.delete('/:id', BookController.deleteBook);

module.exports = router;
```

Exercice 2.4 : Application Express

📝 **Task 9:** Créez app/app.js

```
const express = require('express');
const app = express();

// Import routes
const bookRoutes = require('./routes/bookRoutes');

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

// API Routes
app.use('/api/books', bookRoutes);
```

```
// Root route
app.get('/', (req, res) => {
  res.json({
    success: true,
    message: 'Bienvenue sur l\'API Bibliothèque',
    endpoints: {
      books: '/api/books',
      documentation: '/api-docs'
    }
  });
});

// 404 handler
app.use((req, res) => {
  res.status(404).json({
    success: false,
    message: 'Route not found'
  });
});

module.exports = app;
```

📝 **Task 10:** Créez `server.js`

```
const app = require('./app/app');
const Book = require('./app/models/Book');

const PORT = process.env.PORT || 3000;

async function startServer() {
  try {
    // Initialiser la base de données
    await Book.initTable();
    console.log('Database initialized');

    // Démarrer le serveur
    app.listen(PORT, () => {
      console.log(`Server running on port ${PORT}`);
      console.log(`API: http://localhost:${PORT}/api`);
    });
  } catch (error) {
    console.error('Error starting server:', error);
    process.exit(1);
  }
}

startServer();
```

📝 **Task 11:** Testez votre API

```
# Démarrer le serveur
npm start

# Dans un autre terminal, testez avec curl:
# Créer un livre
curl -X POST http://localhost:3000/api/books \
  -H "Content-Type: application/json" \
  -d '{
    "title": "Le Petit Prince",
    "author": "Antoine de Saint-Exupéry",
    "isbn": "978-2-07-061275-8",
    "published_year": 1943,
    "genre": "Fiction"
  }'

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

✅ **Checkpoint:** Votre API fonctionne, mais elle n'est **pas documentée** ! C'est ce qu'on va corriger maintenant avec Swagger.

Partie 3 : Documenter l'API avec Swagger/OpenAPI 3.0

Exercice 3.1 : Structure de Base du Fichier Swagger

📝 **Task 12:** Créez `docs/swagger.yaml` avec la structure de base

```yaml
openapi: 3.0.3
info:
  title: API Bibliothèque
  description: |
    Une API REST pour gérer une bibliothèque de livres.

    Fonctionnalités :
    - Gestion complète des livres (CRUD)
    - Recherche et filtrage
    - Disponibilité des livres

  version: 1.0.0
  contact:
    name: Support API
    email: support@bibliotheque.fr
  license:
    name: MIT
    url: https://opensource.org/licenses/MIT
```

```yaml
servers:
  - url: http://localhost:3000/api
    description: Serveur de développement

tags:
  - name: Books
    description: Gestion des livres de la bibliothèque

paths:
  # À compléter dans les exercices suivants

components:
  schemas:
    # À compléter dans les exercices suivants
  parameters:
    # À compléter dans les exercices suivants
  responses:
    # À compléter dans les exercices suivants
```

💡 **Explications:** - `openapi: 3.0.3` - Version de la spécification - `info` - Métadonnées de l'API - `servers` - Liste des URLs de serveurs - `tags` - Groupes pour organiser les endpoints - `paths` - Les endpoints (à venir) - `components` - Éléments réutilisables

---

Exercice 3.2 : Définir les Schémas de Données

📝 **Task 13:** Ajoutez les schémas dans la section `components/schemas`

```yaml
components:
  schemas:
    Book:
      type: object
      properties:
        id:
          type: integer
          description: Identifiant unique du livre
          example: 1
        title:
          type: string
          description: Titre du livre
          example: Le Petit Prince
        author:
          type: string
          description: Auteur du livre
          example: Antoine de Saint-Exupéry
        isbn:
          type: string
          description: Numéro ISBN du livre
```

```yaml
        example: "978-2-07-061275-8"
      published_year:
        type: integer
        description: Année de publication
        minimum: 1000
        maximum: 2100
        example: 1943
      genre:
        type: string
        description: Genre littéraire
        example: Fiction
      available:
        type: boolean
        description: Disponibilité du livre
        example: true
      created_at:
        type: string
        format: date-time
        description: Date de création dans la base
        example: "2025-01-15T10:30:00Z"

  BookInput:
    type: object
    required:
      - title
      - author
      - isbn
    properties:
      title:
        type: string
        minLength: 1
        maxLength: 200
        example: Le Petit Prince
      author:
        type: string
        minLength: 1
        maxLength: 100
        example: Antoine de Saint-Exupéry
      isbn:
        type: string
        minLength: 10
        maxLength: 17
        example: "978-2-07-061275-8"
      published_year:
        type: integer
        minimum: 1000
        maximum: 2100
        example: 1943
      genre:
        type: string
```

```yaml
          example: Fiction
        available:
          type: boolean
          example: true

    Error:
      type: object
      properties:
        success:
          type: boolean
          example: false
        message:
          type: string
          example: An error occurred
```

💡 **Explications:** - `Book` - Schéma complet d'un livre (réponse) - `BookInput` - Schéma pour créer/modifier un livre (requête) - `Error` - Schéma pour les erreurs - `required` - Liste des champs obligatoires - `minLength`, `maxLength`, `minimum`, `maximum` - Contraintes de validation

---

Exercice 3.3 : Définir les Paramètres Réutilisables

📝 **Task 14:** Ajoutez les paramètres dans `components/parameters`

```yaml
components:
  parameters:
    BookId:
      name: id
      in: path
      required: true
      description: Identifiant unique du livre
      schema:
        type: integer
        minimum: 1
      example: 1
```

Exercice 3.4 : Définir les Réponses Réutilisables

📝 **Task 15:** Ajoutez les réponses communes dans `components/responses`

```yaml
components:
  responses:
    NotFound:
      description: Ressource non trouvée
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/Error'
```

```yaml
          example:
            success: false
            message: Book not found

    BadRequest:
      description: Requête invalide
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/Error'
          example:
            success: false
            message: Title, author, and ISBN are required

    InternalServerError:
      description: Erreur serveur interne
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/Error'
          example:
            success: false
            message: Internal Server Error
```

Exercice 3.5 : Documenter les Endpoints

📝 **Task 16:** Documentez `GET /api/books` (récupérer tous les livres)

```yaml
paths:
  /books:
    get:
      tags:
        - Books
      summary: Récupérer tous les livres
      description: Renvoie la liste complète de tous les livres de la
bibliothèque
      operationId: getAllBooks
      responses:
        '200':
          description: Livres récupérés avec succès
          content:
            application/json:
              schema:
                type: object
                properties:
                  success:
                    type: boolean
                    example: true
```

```yaml
                message:
                  type: string
                  example: Books retrieved successfully
                data:
                  type: array
                  items:
                    $ref: '#/components/schemas/Book'
                count:
                  type: integer
                  example: 10
        '500':
          $ref: '#/components/responses/InternalServerError'
```

📝 **Task 17:** Documentez `POST /api/books` (créer un livre)

```yaml
paths:
  /books:
    post:
      tags:
        - Books
      summary: Créer un nouveau livre
      description: Ajoute un nouveau livre à la bibliothèque
      operationId: createBook
      requestBody:
        required: true
        description: Informations du livre à créer
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/BookInput'
            examples:
              complete:
                summary: Livre complet
                value:
                  title: Le Petit Prince
                  author: Antoine de Saint-Exupéry
                  isbn: "978-2-07-061275-8"
                  published_year: 1943
                  genre: Fiction
                  available: true
              minimal:
                summary: Champs obligatoires uniquement
                value:
                  title: 1984
                  author: George Orwell
                  isbn: "978-0-452-28423-4"
      responses:
        '201':
```

```
          description: Livre créé avec succès
          content:
            application/json:
              schema:
                type: object
                properties:
                  success:
                    type: boolean
                    example: true
                  message:
                    type: string
                    example: Book created successfully
                  data:
                    $ref: '#/components/schemas/Book'
        '400':
          $ref: '#/components/responses/BadRequest'
```

💡 **Points importants:** - `requestBody` - Décrit le corps de la requête - `required: true` - Le corps est obligatoire - `examples` - Fournir plusieurs exemples de requêtes

---

📝 **Task 18:** Documentez `GET /api/books/{id}` (récupérer un livre par ID)

```yaml
paths:
  /books/{id}:
    get:
      tags:
        - Books
      summary: Récupérer un livre par ID
      description: Récupère les détails d'un livre spécifique
      operationId: getBookById
      parameters:
        - $ref: '#/components/parameters/BookId'
      responses:
        '200':
          description: Livre récupéré avec succès
          content:
            application/json:
              schema:
                type: object
                properties:
                  success:
                    type: boolean
                    example: true
                  message:
                    type: string
                    example: Book retrieved successfully
                  data:
                    $ref: '#/components/schemas/Book'
```

```yaml
        '404':
          $ref: '#/components/responses/NotFound'
        '500':
          $ref: '#/components/responses/InternalServerError'
```

📝 **Task 19:** Documentez `PUT /api/books/{id}` (mettre à jour un livre)

```yaml
paths:
  /books/{id}:
    put:
      tags:
        - Books
      summary: Mettre à jour un livre
      description: Modifie les informations d'un livre existant
      operationId: updateBook
      parameters:
        - $ref: '#/components/parameters/BookId'
      requestBody:
        required: true
        description: Nouvelles informations du livre
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/BookInput'
            examples:
              update_availability:
                summary: Changer la disponibilité
                value:
                  title: Le Petit Prince
                  author: Antoine de Saint-Exupéry
                  isbn: "978-2-07-061275-8"
                  available: false
      responses:
        '200':
          description: Livre mis à jour avec succès
          content:
            application/json:
              schema:
                type: object
                properties:
                  success:
                    type: boolean
                    example: true
                  message:
                    type: string
                    example: Book updated successfully
                  data:
                    $ref: '#/components/schemas/Book'
```

```yaml
        '400':
          $ref: '#/components/responses/BadRequest'
        '404':
          $ref: '#/components/responses/NotFound'
```

📝 **Task 20:** Documentez `DELETE /api/books/{id}` (supprimer un livre)

```yaml
paths:
  /books/{id}:
    delete:
      tags:
        - Books
      summary: Supprimer un livre
      description: Supprime définitivement un livre de la bibliothèque
      operationId: deleteBook
      parameters:
        - $ref: '#/components/parameters/BookId'
      responses:
        '200':
          description: Livre supprimé avec succès
          content:
            application/json:
              schema:
                type: object
                properties:
                  success:
                    type: boolean
                    example: true
                  message:
                    type: string
                    example: Book deleted successfully
        '404':
          $ref: '#/components/responses/NotFound'
        '500':
          $ref: '#/components/responses/InternalServerError'
```

Partie 4 : Intégrer Swagger UI dans Express

Exercice 4.1 : Charger la Documentation

📝 **Task 21:** Modifiez `app/app.js` pour intégrer Swagger UI

```javascript
const express = require('express');
const YAML = require('yamljs');
const swaggerUi = require('swagger-ui-express');
const path = require('path');
```

```
const app = express();

// Import routes
const bookRoutes = require('./routes/bookRoutes');

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

// Charger la documentation Swagger
const swaggerDocument = YAML.load(path.join(__dirname,
'../docs/swagger.yaml'));

// Servir la documentation Swagger UI
app.use('/api-docs', swaggerUi.serve, swaggerUi.setup(swaggerDocument, {
  customCss: '.swagger-ui .topbar { display: none }',
  customSiteTitle: 'Documentation API Bibliothèque'
}));

// API Routes
app.use('/api/books', bookRoutes);

// Root route
app.get('/', (req, res) => {
  res.json({
    success: true,
    message: 'Bienvenue sur l\'API Bibliothèque',
    documentation: '/api-docs',
    endpoints: {
      books: '/api/books'
    }
  });
});

// 404 handler
app.use((req, res) => {
  res.status(404).json({
    success: false,
    message: 'Route not found'
  });
});

module.exports = app;
```

💡 **Options de personnalisation:** - `customCss` - CSS personnalisé (ici, on cache la barre supérieure) - `customSiteTitle` - Titre de l'onglet du navigateur

---

Exercice 4.2 : Tester la Documentation

📝 **Task 22:** Testez votre documentation

```
# Démarrer le serveur
npm run dev

# Ouvrir dans le navigateur:
# http://localhost:3000/api-docs
```

✅ **Ce que vous devriez voir:** - Une interface interactive Swagger UI - Tous vos endpoints listés et groupés par tag - La possibilité de tester chaque endpoint directement depuis l'interface - Les schémas, paramètres et réponses documentés

---

📝 **Task 23:** Testez les endpoints depuis Swagger UI

1. Cliquez sur `POST /api/books`
2. Cliquez sur "Try it out"
3. Modifiez le JSON d'exemple
4. Cliquez sur "Execute"
5. Vérifiez la réponse

Répétez pour tous les endpoints (GET, PUT, DELETE).

Partie 5 : Aller Plus Loin (Bonus)

Exercice 5.1 : Ajouter des Query Parameters

📝 **Task 24:** Ajoutez un paramètre de recherche pour filtrer par genre

Modifiez `docs/swagger.yaml` :

```yaml
paths:
  /books:
    get:
      tags:
        - Books
      summary: Récupérer tous les livres
      description: Renvoie la liste de tous les livres, avec filtrage
optionnel par genre
      operationId: getAllBooks
      parameters:
        - name: genre
          in: query
          description: Filtrer les livres par genre
          required: false
          schema:
```

```
            type: string
            example: Fiction
        - name: available
          in: query
          description: Filtrer par disponibilité
          required: false
          schema:
            type: boolean
            example: true
      responses:
        # ... (identique)
```

Puis implémentez le filtrage dans `app/services/bookService.js` :

```
static async getAllBooks(filters = {}) {
  let sql = 'SELECT * FROM books WHERE 1=1';
  const params = [];

  if (filters.genre) {
    sql += ' AND genre = ?';
    params.push(filters.genre);
  }

  if (filters.available !== undefined) {
    sql += ' AND available = ?';
    params.push(filters.available ? 1 : 0);
  }

  sql += ' ORDER BY created_at DESC';

  return new Promise((resolve, reject) => {
    db.all(sql, params, (err, rows) => {
      if (err) reject(err);
      else resolve(rows);
    });
  });
}
```

Et adaptez le controller :

```
static async getAllBooks(req, res) {
  try {
    const filters = {
      genre: req.query.genre,
      available: req.query.available === 'true' ? true : req.query.available
=== 'false' ? false : undefined
    };
    const books = await BookService.getAllBooks(filters);
    res.status(200).json({
      success: true,
```

```
      message: 'Books retrieved successfully',
      data: books,
      count: books.length
    });
  } catch (error) {
    res.status(500).json({
      success: false,
      message: error.message
    });
  }
}
```

Exercice 5.2 : Valider avec Swagger Editor

📝 **Task 25:** Validez votre fichier YAML

1. Allez sur https://editor.swagger.io/
2. Copiez-collez votre fichier `swagger.yaml`
3. Vérifiez qu'il n'y a pas d'erreurs de syntaxe
4. Corrigez les éventuelles erreurs

🎯 Critères de Réussite

Votre TP est réussi si :

✅ L'API fonctionne et répond correctement à toutes les requêtes ✅ La documentation Swagger est accessible à `/api-docs` ✅ Tous les endpoints sont documentés ✅ Les schémas sont correctement définis avec des exemples ✅ Les réponses d'erreur sont documentées (400, 404, 500) ✅ Vous pouvez tester l'API directement depuis Swagger UI ✅ Le fichier `swagger.yaml` est valide (pas d'erreurs dans Swagger Editor)

📚 Points Clés à Retenir

OpenAPI 3.0

- **Spécification standard** pour documenter les APIs REST
- Format **YAML** ou JSON
- Sections principales : `info`, `servers`, `tags`, `paths`, `components`

Swagger UI

- **Interface interactive** générée automatiquement
- Permet de **tester les endpoints** directement
- Intégration facile avec Express via `swagger-ui-express`

Bonnes Pratiques

- ✅ Garder la documentation dans un fichier séparé (`swagger.yaml`)
- ✅ Utiliser `$ref` pour éviter la duplication
- ✅ Fournir des **exemples** pour tous les schémas et paramètres
- ✅ Documenter **toutes les réponses** possibles (succès ET erreurs)
- ✅ Maintenir la documentation **à jour** avec le code

Structure Recommandée

```
components:
  schemas:       # Modèles de données
  parameters:    # Paramètres réutilisables
  responses:     # Réponses réutilisables
```

🔗 Ressources Utiles

- Spécification OpenAPI 3.0
- Swagger Editor
- Documentation swagger-ui-express
- Documentation yamljs

📝 Questions de Réflexion

1. **Pourquoi est-il important de documenter une API ?**
   - Facilite l'intégration par d'autres développeurs
   - Réduit les erreurs de communication
   - Sert de contrat entre frontend et backend
   - Permet de tester l'API sans code client
2. **Quelle est la différence entre OpenAPI et Swagger ?**
   - OpenAPI = spécification (le format)
   - Swagger = outils (Swagger UI, Editor, Codegen)
3. **Pourquoi utiliser `$ref` ?**
   - Évite la duplication de code
   - Facilite la maintenance
   - Rend le document plus lisible et organisé
4. **Quand utiliser `components/schemas` vs définition inline ?**
   - `components/schemas` pour les schémas réutilisés plusieurs fois
   - Inline pour les schémas très spécifiques à un seul endpoint

---