TP Projet – API REST avec Express, Sequelize & Swagger

Formateur : Bastien Flanquart
Barème : /20 (+2 points bonus front possible)
Travail : individuel ou en groupe (2 à 3 étudiants)

🎯 Objectif du TP

Concevoir, développer et documenter une **API REST** complète en Node.js avec **Express**, connectée à une base de données SQL via **Sequelize**.

Le thème fonctionnel est libre (au choix du groupe), à condition de respecter toutes les contraintes techniques.

Exemples de thèmes possibles :

- 📚 Gestion de bibliothèque (livres, auteurs, emprunts)
- 🍳 Application de recettes (recettes, ingrédients, catégories)
- 📋 Gestion de projets (projets, tâches, utilisateurs, commentaires)
- 🏪 Plateforme de petites annonces (annonces, utilisateurs, catégories)
- 🎪 Gestion d'événements (événements, lieux, participants)

⚙️ Contraintes techniques OBLIGATOIRES

Stack technique

```
Node.js + Express
├── Sequelize (ORM SQL)
├── MySQL/PostgreSQL/SQLite
├── Swagger (swagger-jsdoc + swagger-ui-express)
├── JWT (jsonwebtoken)
├── bcrypt (hash mots de passe)
├── CORS (middleware)
└── Validation (express-validator ou Joi)
```

Optionnel recommandé : Nodemailer (email)

📋 Spécifications fonctionnelles MINIMALES

1. Ressource User obligatoire

```
-> Inscription (signup) → bcrypt
-> Connexion (login) → JWT
-> Profil utilisateur → route protégée JWT
```

2. DEUX ressources métier minimum avec relations

Exemples : Livre↔Auteur (1-N), Projet↔Tâche (1-N), User↔Annonce (1-N)

3. CRUD complet pour chaque ressource

```
- GET /resources (liste)
- GET /resources/:id (détail)
- POST /resources (créer)
- PUT/PATCH /resources/:id (modifier)
- DELETE /resources/:id (supprimer)
```

⚠ **Certaines routes protégées JWT obligatoire**

🔧 Spécifications techniques détaillées

1. Sequelize OBLIGATOIRE

À implémenter :

```
- Modèles avec types de données Sequelize
- Migrations pour TOUTES les modifications
- Au moins 1 association (1-N ou N-N)
- Seeders avec données de test
```

2. Express & Routes

```
- 2 types de paramètres : req.params, req.body
- Codes HTTP corrects (200, 201, 400, 401, 404, 500)
- Séparation routes/controllers/services
```

3. Sécurité OBLIGATOIRE

```
- bcrypt → hashage mots de passe
- JWT → authentification
- CORS → configuration
- Validation données → express-validator/Joi
```

4. Swagger OBLIGATOIRE

```
npm i swagger-jsdoc swagger-ui-express

Route : http://localhost:3000/api-docs

- TOUTES les routes principales documentées
- Schémas request/response
- Codes d'erreur documentés
```

Bonus optionnels

Fonctionnalité	Points
Nodemailer (email fonctionnel)	+1 pt
Front-end consommant l'API	**+2 pts**

Front minimum attendu : formulaire login/signup + affichage 1 ressource

---

Livrables attendus

1. Code source complet (Git recommandé)

```
- README.md avec :

Le ou les membres du projet

Sujet fonctionnel choisi

Instructions installation

npm install → npx sequelize db:migrate → npm start

URL Swagger : /api-docs
```

2. Base de données fonctionnelle

```
- Migrations exécutables
- Seeders avec données démo
```

3. Swagger fonctionnel

```
- http://localhost:3000/api-docs
- Testable directement
```

Barème détaillé /20

Critère	Points
Fonctionnalités (User + 2 ressources + relations)	4 pts
Sequelize (modèles/migrations/seeders/associations)	4 pts
Express (structure/CRUD/codes HTTP)	4 pts
Sécurité (bcrypt/JWT/CORS/validation)	3 pts
Swagger (couverture/qualité doc)	3 pts
Qualité code (organisation/README)	2 pts

Bonus : Nodemailer (+1) | Front-end (+2)

Exemple concret – Gestion de tâches

```
Ressources :
├── User (id, email, password, role)
├── Project (id, name, description)
└── Task (id, title, status, projectId, userId)

Relations :
├── User → hasMany Tasks
└── Project → hasMany Tasks

Routes protégées :
├── POST /projects
├── DELETE /tasks/:id
└── PUT /tasks/:id
```

Bonne chance à tous !

Questions ? -> Demandez au formateur