REST API - TP

TP - Développement Web 2024-2025

Création d'une API REST pour la gestion de voitures classiques

Licence Informatique

Hiver 2025

---

1. Introduction

Bienvenue dans ce travail pratique sur le développement d'API REST. Voici quelques règles essentielles pour le bon déroulement de ces séances :

- À ce niveau universitaire, vous êtes responsables de la qualité de votre code. Les erreurs de syntaxe basiques doivent être résolues de manière autonome.
- Le travail se fait sur vos propres ordinateurs. **Merci de venir en séance avec tous les logiciels installés et testés au préalable**.
- La sauvegarde régulière de votre travail est **VOTRE responsabilité**.
- Nous recommandons **fortement** l'usage d'un logiciel de gestion de version comme Git.
- En cas de blocage, consultez d'abord la documentation officielle, puis demandez de l'aide.

1.1 Prérequis techniques

Avant la première séance, vous devez avoir installé et testé les éléments suivants :

- **Node.js** (version 16 ou supérieure) : https://nodejs.org
- **Visual Studio Code** : https://code.visualstudio.com
- **Postman** ou **Insomnia** : pour tester vos routes API
- **Git** : pour la gestion de version

- Un compte **GitHub** : pour héberger votre code

Pour vérifier que Node.js est correctement installé, ouvrez un terminal et exécutez :

```
node --version npm --version
```

Vous devriez voir s'afficher les numéros de version.

---

2. Présentation générale du projet

2.1 Contexte

Dans le monde du développement web moderne, les applications sont généralement divisées en deux parties distinctes : le **frontend** (interface utilisateur) et le **backend** (logique métier et données). Ces deux parties communiquent via des **API REST** (Application Programming Interface - Representational State Transfer).

Une API REST permet à différents systèmes de communiquer entre eux de manière standardisée en utilisant le protocole HTTP. C'est le même protocole que celui utilisé par votre navigateur web, mais au lieu de renvoyer des pages HTML, une API REST renvoie des données au format JSON.

2.2 Objectif du projet

Au cours de ces séances, vous allez construire une **API REST complète** pour gérer un système de voitures classiques. Cette API permettra de :

- **Créer** de nouvelles voitures dans la base de données (CREATE)
- **Lire** les informations sur les voitures existantes (READ)
- **Modifier** les informations d'une voiture (UPDATE)
- **Supprimer** une voiture de la base de données (DELETE)

Ces quatre opérations constituent ce qu'on appelle les opérations **CRUD** (Create, Read, Update, Delete), qui sont la base de toute application de gestion de données.

2.3 Technologies utilisées

Node.js

Node.js est un environnement d'exécution JavaScript côté serveur. Contrairement au JavaScript qui s'exécute dans le navigateur, Node.js permet d'exécuter du JavaScript sur un serveur pour créer des applications backend.

Pourquoi Node.js ?

- Utilise JavaScript, un langage que vous connaissez déjà
- Très performant pour les applications web
- Énorme écosystème de bibliothèques (npm)
- Très demandé sur le marché du travail

Express

Express est un framework web minimaliste pour Node.js. Il simplifie grandement la création de serveurs web et d'API REST.

Ce qu'Express nous apporte :

- Gestion simplifiée des routes HTTP (GET, POST, PUT, DELETE)
- Middleware pour traiter les requêtes
- Gestion des erreurs
- Parsing automatique du JSON

SQLite

SQLite est un système de gestion de base de données relationnelle léger qui stocke les données dans un simple fichier.

Avantages pour l'apprentissage :

- Aucune installation de serveur de base de données nécessaire
- Parfait pour le développement et les prototypes
- Syntaxe SQL standard
- Facilement remplaçable par PostgreSQL ou MySQL en production

2.4 Le domaine : Gestion de voitures classiques

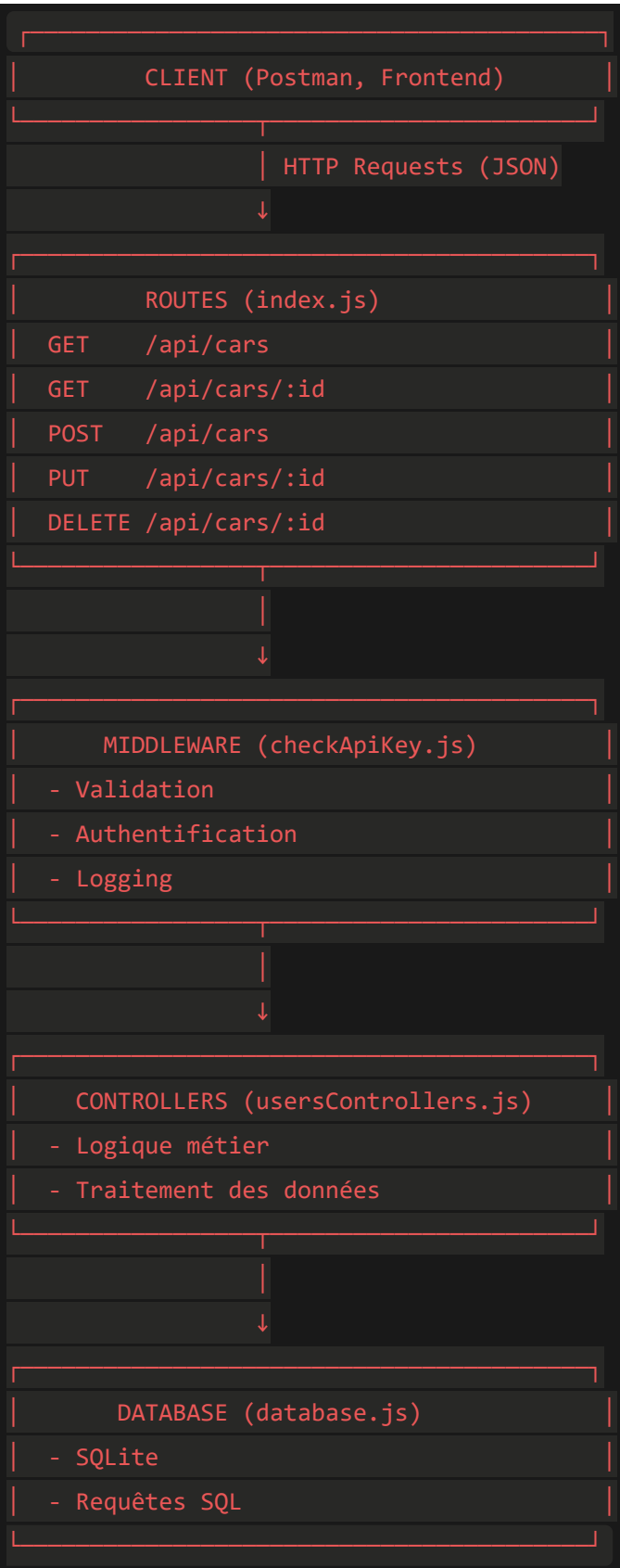
Nous avons choisi le domaine des voitures classiques car il est concret et facile à comprendre. Chaque voiture dans notre système possède les attributs suivants :

- **id** : Identifiant unique (généré automatiquement)
- **brand** : Marque (ex: Ferrari, Porsche, Jaguar)
- **model** : Modèle (ex: 250 GTO, 911, E-Type)

- **year** : Année de fabrication
- **color** : Couleur
- **price** : Prix en euros
- **mileage** : Kilométrage
- **description** : Description détaillée (optionnelle)

2.5 Architecture de l'application

Votre API REST suivra une architecture en couches, une pratique standard dans le développement professionnel :



2.6 Structure du projet

Voici la structure de fichiers que vous allez créer :

```
node-js-api/ │ ├── controllers/ │ └── usersControllers.js # Logique métier et contrôleurs │ ├── middleware/ │ └── checkApiKey.js # Vérification de la clé API │ ├── database.js # Configuration et connexion DB ├── index.js # Point d'entrée, routes principales ├── seed.js # Données de test initiales ├── package.json # Dépendances du projet ├── .gitignore # Fichiers à ignorer par Git └── README.md # Documentation du projet
```

2.7 Compétences visées

À la fin de ce TP, vous serez capable de :

1. **Comprendre** le fonctionnement d'une API REST et le protocole HTTP
2. **Créer** un serveur web avec Node.js et Express
3. **Concevoir** des routes RESTful suivant les bonnes pratiques
4. **Interagir** avec une base de données SQLite
5. **Implémenter** les opérations CRUD complètes
6. **Sécuriser** une API avec une authentification par clé
7. **Tester** une API avec des outils professionnels (Postman)
8. **Gérer** un projet avec Git et GitHub
9. **Documenter** votre code et votre API

2.8 Projet de référence

Un projet exemple complet est disponible sur GitHub pour vous guider :

https://github.com/web-rest-api/node-js-api/tree/sql

Vous pouvez consulter ce projet à tout moment pour :

- Comprendre la structure globale
- Voir des exemples de code fonctionnel
- Vérifier votre implémentation
- Vous inspirer pour résoudre un problème

⚠️ **Important** : N'hésitez pas à consulter ce projet, mais essayez d'abord de comprendre et d'implémenter par vous-même avant de copier du code.

Etapes:

8. Travail à la maison : Améliorations et extensions

8.1 Fonctionnalités supplémentaires à implémenter

1. **Recherche et filtrage :**
   - Ajouter une route GET `/api/cars/search?brand=Ferrari&year=1962`
   - Permettre de filtrer par marque, année, prix min/max
2. **Pagination :**
   - Ajouter des paramètres `page` et `limit`
   - Exemple : `/api/cars?page=1&limit=10`
3. **Tri :**
   - Permettre de trier par différents champs
   - Exemple : `/api/cars?sortBy=price&order=desc`
4. **Validation avancée :**
   - Installer le package `express-validator`
   - Valider les types de données, formats, etc.
5. **Variables d'environnement :**
   - Installer `dotenv`
   - Créer un fichier `.env` pour les configurations

8.2 Exemple d'implémentation de la recherche

```
// Dans usersControllers.js exports.searchCars = (req, res) => { const {
brand, model, minYear, maxYear, minPrice, maxPrice } = req.query; let
query = 'SELECT * FROM cars WHERE 1=1'; const params = []; if (brand) {
query += ' AND brand LIKE ?'; params.push(`%${brand}%`); } if (model) {
query += ' AND model LIKE ?'; params.push(`%${model}%`); } if (minYear) {
query += ' AND year >= ?'; params.push(minYear); } if (maxYear) { query +=
' AND year <= ?'; params.push(maxYear); } if (minPrice) { query += ' AND
price >= ?'; params.push(minPrice); } if (maxPrice) { query += ' AND price
<= ?'; params.push(maxPrice); } query += ' ORDER BY year DESC';
db.all(query, params, (err, rows) => { if (err) { return
res.status(500).json({ error: 'Erreur lors de la recherche', details:
err.message }); } res.json({ success: true, message: 'Recherche
effectuée', count: rows.length, data: rows }); }); };
```

8.3 Défis supplémentaires

🎯 **Niveau facile :**

- Ajouter un champ `favorite` (booléen) aux voitures
- Créer une route pour récupérer uniquement les favorites

🎯 **Niveau moyen :**

- Implémenter un système de catégories (sportive, berline, cabriolet)
- Ajouter des statistiques (voiture la plus chère, moyenne des prix, etc.)

🎯 **Niveau difficile :**

- Créer un système d'upload d'images pour les voitures
- Implémenter l'authentification avec JWT (JSON Web Tokens)
- Ajouter des tests unitaires avec Jest

9. Conclusion et ressources

9.1 Ce que vous avez appris

Félicitations ! Au cours de ce TP, vous avez :

✅ Compris le fonctionnement d'une API REST

✅ Créé un serveur web avec Node.js et Express

✅ Manipulé une base de données SQLite

✅ Implémenté les opérations CRUD complètes

✅ Sécurisé une API avec un middleware d'authentification

✅ Structuré un projet selon les bonnes pratiques

✅ Documenté votre code et votre API

✅ Utilisé Git pour versionner votre projet

9.2 Pour aller plus loin

Documentation officielle :

- Node.js : https://nodejs.org/docs
- Express : https://expressjs.com
- SQLite : https://www.sqlite.org/docs.html

Tutoriels et ressources :

- MDN Web Docs (HTTP, API) : https://developer.mozilla.org
- FreeCodeCamp : https://www.freecodecamp.org
- The Odin Project : https://www.theodinproject.com

Prochaines étapes :

1. Apprendre un ORM comme Sequelize ou Prisma
2. Découvrir les bases de données PostgreSQL ou MongoDB
3. Créer un frontend (React, Vue) qui consomme votre API
4. Déployer votre API sur Heroku, Railway ou Vercel
5. Apprendre Docker pour containeriser votre application

9.3 Critères d'évaluation

Votre projet sera évalué sur :

- **Fonctionnalité (35%)** : Toutes les routes CRUD fonctionnent correctement
- **Code (25%)** : Propreté, organisation, respect des conventions
- **Sécurité (15%)** : Middleware d'authentification, validation des données
- **Documentation (15%)** : README complet, commentaires pertinents
- **Déploiement (10%)** : Application fonctionnelle en ligne, accessible via une URL

9.4 Livrables

À rendre avant le 20 décembre 2025 :

1. Repository GitHub avec le code complet
2. README.md détaillé
3. Fichier `TESTS.md` avec des captures d'écran Postman montrant que toutes les routes fonctionnent
4. (Optionnel) Vidéo de démonstration (3-5 minutes)

10. Aide et support

10.1 Erreurs courantes

Erreur : "Cannot find module 'express'"

- Solution : Vérifiez que vous avez exécuté `npm install`

Erreur : "Port 3000 already in use"

- Solution : Un autre processus utilise le port 3000. Tuez-le ou changez de port