7. Documentation et bonnes pratiques

7.1 Création du README.md

Un bon README est essentiel pour tout projet. Créez un fichier `README.md` :

```
# 🚗 API REST - Gestion de Voitures Classiques API REST pour gérer une collection de voitures classiques. Développée avec Node.js, Express et SQLite. ## 📋 Prérequis - Node.js (v16 ou supérieure) - npm ou yarn - Postman (pour tester l'API) ## 🚀 Installation 1. Cloner le projet ```bash git clone [URL_DE_VOTRE_REPO] cd car-api ``` 2. Installer les dépendances ```bash npm install ``` 3. Initialiser la base de données ```bash npm run seed ``` 4. Lancer le serveur ```bash npm run dev ``` Le serveur démarre sur `http://localhost:3000` ## 🔑 Authentification Toutes les routes API nécessitent une clé API dans les headers :
```

x-api-key: ma-super-cle-api-2024

```
## 📚 Endpoints ### GET /api/cars Récupère toutes les voitures **Réponse (200 OK) :** ```json { "success": true, "message": "Liste des voitures récupérée", "count": 5, "data": [...] } ``` ### GET /api/cars/:id Récupère une voiture par son ID **Paramètres :** - `id` (number) : ID de la voiture **Réponse (200 OK) :** ```json { "success": true, "message": "Voiture trouvée", "data": {...} } ``` ### POST /api/cars Crée une nouvelle voiture **Body (JSON) :** ```json { "brand": "Ferrari", "model": "250 GTO", "year": 1962, "color": "Rouge", "price": 45000000, "mileage": 12000, "description": "Voiture exceptionnelle" } ``` **Réponse (201 Created) :** ```json { "success": true, "message": "Voiture créée avec succès", "data": {...} } ``` ### PUT /api/cars/:id Modifie une voiture existante **Paramètres :** - `id` (number) : ID de la voiture **Body (JSON) :** Même structure que POST **Réponse (200 OK) :** ```json { "success": true, "message": "Voiture mise à jour avec succès", "data": {...} } ``` ### DELETE /api/cars/:id Supprime une voiture **Paramètres :** - `id` (number) : ID de la voiture **Réponse (200 OK) :** ```json { "success": true, "message": "Voiture supprimée avec succès", "data": { "id": 1 } } ``` ## 🏗️ Structure du projet
```

CAR API

```
car-api/ ├── controllers/ │ └── usersControllers.js # Logique métier ├── middleware/ │ └── checkApiKey.js # Authentification ├── database.js # Configuration DB ├── index.js # Point d'entrée ├── seed.js # Données de test ├── package.json └── README.md
```

```
## 🛠️ Technologies utilisées - **Node.js** : Environnement d'exécution JavaScript - **Express** : Framework web minimaliste - **SQLite3** : Base de données légère - **body-parser** : Parsing des requêtes JSON - **cors** : Gestion des requêtes cross-origin ## 👨‍💻 Développeur [Votre Nom] - Licence Informatique 2024-2025 ## 📝 Licence Ce projet est un exercice pédagogique.
```

```
7.2 Initialisation Git Initialisez Git dans votre projet : git init git add . git commit -m "Initial commit: API REST voitures classiques"
```

Créez un repository sur GitHub et poussez votre code :

```
git remote add origin [URL_DE_VOTRE_REPO] git branch -M main git push -u origin main
```

7.3 Bonnes pratiques à retenir

1. **Nommage :**
   - Routes : kebab-case (`/api/classic-cars`)
   - Variables : camelCase (`firstName`)
   - Constantes : UPPER_SNAKE_CASE (`API_KEY`)
2. **Codes HTTP :**
   - Utilisez toujours le bon code de statut
   - 2xx pour succès, 4xx pour erreurs client, 5xx pour erreurs serveur
3. **Validation :**
   - Validez toujours les données entrantes
   - Renvoyez des messages d'erreur clairs
4. **Sécurité :**
   - Ne jamais exposer de données sensibles
   - Utiliser des variables d'environnement (fichier `.env`)
   - Valider et assainir les entrées utilisateur
5. **Documentation :**