1 - Initialisation du projet et première route

1 - Initialisation du projet et première route

3.1 Création du projet Node.js

Ouvrez un terminal et créez un nouveau dossier pour votre projet :

```
mkdir car-api cd car-api
```

Initialisez un nouveau projet Node.js :

bash

```
npm init -y
```

Cette commande crée un fichier `package.json` qui contiendra toutes les informations sur votre projet et ses dépendances.

3.2 Installation des dépendances

Installez les packages nécessaires :

bash

```
npm install express sqlite3 cors
```

Explication des packages :

- **express** : Framework web pour créer le serveur et les routes
- **sqlite3** : Driver pour interagir avec SQLite
- **cors** : Middleware pour autoriser les requêtes cross-origin

Installez également nodemon en dépendance de développement :

```
npm install --save-dev nodemon
```

nodemon redémarre automatiquement le serveur à chaque modification de fichier, ce qui facilite grandement le développement.

3.3 Configuration du package.json

Ouvrez le fichier `package.json` et modifiez la section `scripts` :

```
"scripts": { "start": "node index.js", "dev": "nodemon index.js" }
```

Maintenant vous pourrez lancer votre serveur en mode développement avec `npm run dev`.

3.4 Création du fichier .gitignore

Créez un fichier `.gitignore` à la racine du projet pour éviter de versionner les fichiers inutiles :

```
node_modules/ *.db .env .DS_Store
```

✏️ **Exercice 1** : Créez le fichier `.gitignore` avec le contenu ci-dessus.

3.5 Création du serveur Express basique

Créez un fichier `index.js` à la racine du projet. Ce sera le point d'entrée de votre application.

Concepts clés avant de coder :

1. **require()** : Permet d'importer des modules Node.js
2. **express()** : Crée une application Express
3. **app.use()** : Ajoute un middleware à l'application
4. **app.listen()** : Démarre le serveur sur un port spécifique

Voici la structure de base d'un serveur Express :

```
// Importation des modules nécessaires const express = require('express');
const cors = require('cors'); // Création de l'application Express const
app = express(); // Configuration du port const PORT = process.env.PORT ||
3000; // Middlewares globaux app.use(cors()); // Autorise les requêtes
cross-origin app.use(express.json()); // Parse le JSON des requêtes//
Route de test app.get('/', (req, res) => { res.json({ message: 'Bienvenue
sur l\'API de gestion de voitures classiques', version: '1.0.0' }); }); //
Démarrage du serveur app.listen(PORT, () => { console.log(`🚀 Serveur
démarré sur le port ${PORT}`); console.log(`📍 http://localhost:${PORT}`);
});
```

✏️ **Exercice 2 :**

- Créez le fichier `index.js` avec le code ci-dessus
- Lancez le serveur avec `npm run dev`
- Ouvrez votre navigateur et allez sur `http://localhost:3000`
- Vous devriez voir le message JSON de bienvenue

3.6 Comprendre les routes HTTP

Une route définit comment l'application répond à une requête client sur un endpoint spécifique. Chaque route a :

1. **Une méthode HTTP** : GET, POST, PUT, DELETE
2. **Un chemin (path)** : l'URL de l'endpoint
3. **Une fonction de callback** : qui traite la requête et envoie la réponse

Les méthodes HTTP principales :

- **GET** : Récupérer des données (lecture)
- **POST** : Créer une nouvelle ressource
- **PUT** : Modifier une ressource existante
- **DELETE** : Supprimer une ressource

Exemple de route :

```
app.get('/api/cars', (req, res) => { // req = request (requête entrante)// res = response (réponse à envoyer) res.json({ message: 'Liste des voitures' }); });
```

✏️ **Exercice 3** : Ajoutez les routes suivantes dans votre `index.js` (avant `app.listen`) :

```
// GET - Récupérer toutes les voitures app.get('/api/cars', (req, res) => { res.json({ message: 'Liste de toutes les voitures', data: [] }); }); // GET - Récupérer une voiture par son ID app.get('/api/cars/:id', (req, res) => { const id = req.params.id; res.json({ message: `Voiture avec l'ID ${id}`, data: null }); }); // POST - Créer une nouvelle voiture app.post('/api/cars', (req, res) => { const carData = req.body; res.status(201).json({ message: 'Voiture créée avec succès', data: carData }); }); // PUT - Modifier une voiture existante app.put('/api/cars/:id', (req, res) => { const id = req.params.id; const carData = req.body; res.json({ message: `Voiture ${id} modifiée`, data: carData }); }); // DELETE - Supprimer une voiture app.delete('/api/cars/:id', (req, res) => { const id = req.params.id; res.json({ message: `Voiture ${id} supprimée` }); });
```

3.7 Tester les routes avec Postman

Postman est un outil professionnel pour tester les API REST. Voici comment tester vos routes :

1. **Ouvrez Postman**
2. **Créez une nouvelle requête** (New Request)
3. **Testez chaque route** :

Test 1 - GET toutes les voitures :

- Méthode : GET
- URL : `http://localhost:3000/api/cars`
- Cliquez sur "Send"

Test 2 - GET une voiture spécifique :

- Méthode : GET
- URL : `http://localhost:3000/api/cars/1`
- Cliquez sur "Send"

Test 3 - POST créer une voiture :

- Méthode : POST
- URL : `http://localhost:3000/api/cars`
- Dans l'onglet "Body", sélectionnez "raw" et "JSON"
- Entrez ce JSON :

```
{ "brand": "Ferrari", "model": "250 GTO", "year": 1962, "color": "Rouge",
"price": 45000000, "mileage": 12000 }
```

- Cliquez sur "Send"

✏️ **Exercice 4** : Testez toutes les routes avec Postman et vérifiez que vous recevez