3 - Contrôleurs et architecture MVC

5.1 Qu'est-ce que l'architecture MVC ?

MVC signifie **Model-View-Controller**. C'est un pattern architectural qui sépare les responsabilités :

- **Model** : Gestion des données (notre base de données)
- **View** : Interface utilisateur (dans notre cas, les réponses JSON)
- **Controller** : Logique métier qui fait le lien entre Model et View

Pourquoi séparer en contrôleurs ?

1. **Organisation** : Code plus propre et structuré
2. **Réutilisabilité** : Même logique utilisable à plusieurs endroits
3. **Testabilité** : Plus facile de tester chaque partie
4. **Maintenance** : Plus facile de modifier et déboguer

5.2 Création du dossier controllers

Créez un dossier `controllers` à la racine du projet, puis créez le fichier `usersControllers.js` (oui, le nom vient du projet de référence même si on gère des voitures).

```
const db = require('../database'); // GET - Récupérer toutes les voiture
exports.getAllCars = (req, res) => { const query = 'SELECT * FROM cars
ORDER BY year DESC'; db.all(query, [], (err, rows) => { if (err) { retur
res.status(500).json({ error: 'Erreur lors de la récupération des
voitures', details: err.message }); } res.json({ success: true, message:
'Liste des voitures récupérée', count: rows.length, data: rows }); }); }
// GET - Récupérer une voiture par ID exports.getCarById = (req, res) =>
const id = req.params.id; const query = 'SELECT * FROM cars WHERE id = ?
db.get(query, [id], (err, row) => { if (err) { return
res.status(500).json({ error: 'Erreur serveur', details: err.message });
if (!row) { return res.status(404).json({ error: 'Voiture non trouvée',
message: `Aucune voiture avec l'ID ${id}` }); } res.json({ success: true
message: 'Voiture trouvée', data: row }); }); }; // POST - Créer une
nouvelle voiture exports.createCar = (req, res) => { const { brand, mode
year, color, price, mileage, description } = req.body; // Validation des
données if (!brand || !model || !year) { return res.status(400).json({
error: 'Données invalides', message: 'Les champs brand, model et year so
obligatoires' }); } const query = ` INSERT INTO cars (brand, model, year
color, price, mileage, description) VALUES (?, ?, ?, ?, ?, ?, ?) `;
db.run( query, [brand, model, year, color, price, mileage, description],
function(err) { if (err) { return res.status(500).json({ error: 'Erreur
lors de la création', details: err.message }); } res.status(201).json({
success: true, message: 'Voiture créée avec succès', data: { id:
this.lastID, brand, model, year, color, price, mileage, description } })
} ); }; // PUT - Modifier une voiture existante exports.updateCar = (req
res) => { const id = req.params.id; const { brand, model, year, color,
price, mileage, description } = req.body; // Vérifier si la voiture exis
db.get('SELECT * FROM cars WHERE id = ?', [id], (err, row) => { if (err)
return res.status(500).json({ error: 'Erreur serveur', details:
err.message }); } if (!row) { return res.status(404).json({ error:
'Voiture non trouvée' }); } const query = ` UPDATE cars SET brand = ?,
model = ?, year = ?, color = ?, price = ?, mileage = ?, description = ?
WHERE id = ? `; db.run( query, [brand, model, year, color, price, mileag
description, id], function(err) { if (err) { return res.status(500).json
error: 'Erreur lors de la mise à jour', details: err.message }); }
res.json({ success: true, message: 'Voiture mise à jour avec succès',
data: { id, brand, model, year, color, price, mileage, description } });
); }); }; // DELETE - Supprimer une voiture exports.deleteCar = (req, re
=> { const id = req.params.id; // Vérifier si la voiture existe
db.get('SELECT * FROM cars WHERE id = ?', [id], (err, row) => { if (err)
return res.status(500).json({ error: 'Erreur serveur', details:
err.message }); } if (!row) { return res.status(404).json({ error:
'Voiture non trouvée' }); } db.run('DELETE FROM cars WHERE id = ?', [id]
function(err) { if (err) { return res.status(500).json({ error: 'Erreur
lors de la suppression', details: err.message }); } res.json({ success:
```

```
true, message: 'Voiture supprimée avec succès', data: { id } }); }); });
};
```

5.3 Mise à jour du fichier index.js

Maintenant, simplifiez votre `index.js` en utilisant les contrôleurs :

```
const express = require('express'); const bodyParser = require('body-
parser'); const cors = require('cors'); const carsController =
require('./controllers/usersControllers'); const app = express(); const
PORT = process.env.PORT || 3000; // Middlewares app.use(cors());
app.use(bodyParser.json()); // Route de bienvenue app.get('/', (req, res)
=> { res.json({ message: 'Bienvenue sur l\'API de gestion de voitures
classiques', version: '1.0.0', endpoints: { getAllCars: 'GET /api/cars',
getCarById: 'GET /api/cars/:id', createCar: 'POST /api/cars', updateCar:
'PUT /api/cars/:id', deleteCar: 'DELETE /api/cars/:id' } }); }); // Routes
CRUD app.get('/api/cars', carsController.getAllCars);
app.get('/api/cars/:id', carsController.getCarById); app.post('/api/cars',
carsController.createCar); app.put('/api/cars/:id',
carsController.updateCar); app.delete('/api/cars/:id',
carsController.deleteCar); // Gestion des routes non trouvées
app.use((req, res) => { res.status(404).json({ error: 'Route non trouvée',
message: `La route ${req.method} ${req.url} n'existe pas` }); }); //
Démarrage du serveur app.listen(PORT, () => { console.log(`🚀 Serveur
démarré sur le port ${PORT}`); console.log(`📍 http://localhost:${PORT}`);
});
```

✏️ **Exercice 8** :

- Créez le dossier `controllers` et le fichier `usersControllers.js`
- Mettez à jour votre `index.js`
- Testez toutes les opérations CRUD avec Postman :
  - GET toutes les voitures
  - GET une voiture par ID
  - POST créer une nouvelle voiture