TP 1 : Création du Backend CRUD pour les Produits

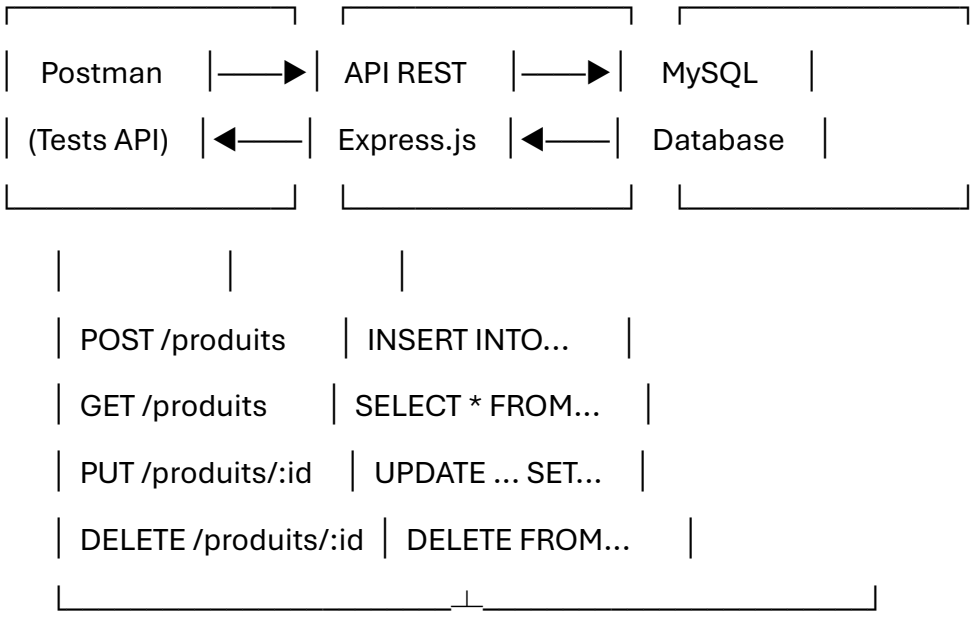
Objectif

Créer une API REST complète avec les 4 opérations CRUD (Create, Read, Update, Delete) pour gérer des produits avec MySQL.

Ce que nous allons construire

- Base de données MySQL avec table produits
- API REST avec Express.js
- Routes pour Create, Read, Update, Delete
- Tests avec Postman

Architecture finale



Structure du projet

backend-crud-produits/

├— config/

│ └— database.js

├— routes/

│ └— produits.js

├— .env

├— server.js

└── package.json

Étape 1 : Préparation de l'environnement

1.1 Créer le projet

mkdir backend-crud-produits

cd backend-crud-produits

npm init -y

1.2 Installer les dépendances

npm install express mysql2 dotenv cors body-parser

npm install --save-dev nodemon

Étape 2 : Base de données MySQL

2.1 Créer la base de données

Ouvrez MySQL (phpMyAdmin, MySQL Workbench ou terminal) :

```
-- Créer la base de données
CREATE DATABASE IF NOT EXISTS gestion_produits;

-- Utiliser la base de données
USE gestion_produits;

-- Créer la table produits
CREATE TABLE produits (
  id INT AUTO_INCREMENT PRIMARY KEY,
  nom VARCHAR(100) NOT NULL,
  description TEXT,
  prix DECIMAL(10, 2) NOT NULL,
  stock INT DEFAULT 0,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

-- Insérer quelques données de test

INSERT INTO produits (nom, description, prix, stock) VALUES

('iPhone 13', 'Smartphone Apple', 899.99, 15),

('Samsung Galaxy S21', 'Smartphone Samsung', 799.99, 20),

('iPad Pro', 'Tablette Apple', 1099.99, 10);

2.2 Structure de la table

Table: produits

├— id (INT, PRIMARY KEY, AUTO_INCREMENT)

├— nom (VARCHAR 100, NOT NULL)

├— description (TEXT)

├— prix (DECIMAL 10,2, NOT NULL)

├— stock (INT, DEFAULT 0)

└— created_at (TIMESTAMP)

Étape 3 : Configuration du serveur

3.1 Variables d'environnement (.env)

Base de données

DB_HOST=localhost

DB_USER=root

DB_PASSWORD=

DB_NAME=gestion_produits

DB_PORT=3306

Serveur

PORT=3000

3.2 Configuration de la base de données (config/database.js)

const mysql = require('mysql2');

require('dotenv').config();

```
// Créer la connexion
const connection = mysql.createConnection({
  host: process.env.DB_HOST,
  user: process.env.DB_USER,
  password: process.env.DB_PASSWORD,
  database: process.env.DB_NAME,
  port: process.env.DB_PORT
});

// Tester la connexion
connection.connect((err) => {
  if (err) {
    console.error(' Erreur de connexion MySQL:', err);
    return;
  }
  console.log(' Connecté à MySQL!');
});

// Utiliser les promesses
const db = connection.promise();

module.exports = db;
```

📝 **Étape 4 : Routes CRUD (routes/produits.js)**

```
const express = require('express');
const router = express.Router();
const db = require('../config/database');

//  CREATE - Créer un nouveau produit
```

```
router.post('/produits', async (req, res) => {
  try {
    const { nom, description, prix, stock } = req.body;

    // Validation simple
    if (!nom || !prix) {
      return res.status(400).json({
        success: false,
        message: 'Le nom et le prix sont obligatoires'
      });
    }

    // Requête SQL
    const query = 'INSERT INTO produits (nom, description, prix, stock) VALUES (?, ?, ?, ?)';
    const [result] = await db.execute(query, [nom, description, prix, stock || 0]);

    res.status(201).json({
      success: true,
      message: 'Produit créé avec succès',
      data: {
        id: result.insertId,
        nom,
        description,
        prix,
        stock: stock || 0
      }
    });
  } catch (error) {
```

```
    res.status(500).json({
      success: false,
      message: 'Erreur lors de la création',
      error: error.message
    });
  }
});

//  READ - Obtenir tous les produits
router.get('/produits', async (req, res) => {
  try {
    const query = 'SELECT * FROM produits ORDER BY created_at DESC';
    const [produits] = await db.execute(query);

    res.json({
      success: true,
      count: produits.length,
      data: produits
    });
  } catch (error) {
    res.status(500).json({
      success: false,
      message: 'Erreur lors de la récupération',
      error: error.message
    });
  }
});
```

```
// 🔍 READ - Obtenir un produit par ID
router.get('/produits/:id', async (req, res) => {
  try {
    const { id } = req.params;
    const query = 'SELECT * FROM produits WHERE id = ?';
    const [produits] = await db.execute(query, [id]);

    if (produits.length === 0) {
      return res.status(404).json({
        success: false,
        message: 'Produit non trouvé'
      });
    }

    res.json({
      success: true,
      data: produits[0]
    });
  } catch (error) {
    res.status(500).json({
      success: false,
      message: 'Erreur lors de la récupération',
      error: error.message
    });
  }
});

// 4️⃣ UPDATE - Modifier un produit
```

```
router.put('/produits/:id', async (req, res) => {
  try {
    const { id } = req.params;
    const { nom, description, prix, stock } = req.body;

    // Vérifier si le produit existe
    const [existing] = await db.execute('SELECT id FROM produits WHERE id = ?', [id]);
    if (existing.length === 0) {
      return res.status(404).json({
        success: false,
        message: 'Produit non trouvé'
      });
    }

    // Mise à jour
    const query = 'UPDATE produits SET nom = ?, description = ?, prix = ?, stock = ?
WHERE id = ?';
    await db.execute(query, [nom, description, prix, stock, id]);

    // Récupérer le produit mis à jour
    const [updated] = await db.execute('SELECT * FROM produits WHERE id = ?', [id]);

    res.json({
      success: true,
      message: 'Produit mis à jour avec succès',
      data: updated[0]
    });
  } catch (error) {
```

```
    res.status(500).json({
      success: false,
      message: 'Erreur lors de la mise à jour',
      error: error.message
    });
  }
});

//  DELETE - Supprimer un produit
router.delete('/produits/:id', async (req, res) => {
  try {
    const { id } = req.params;

    // Vérifier si le produit existe
    const [existing] = await db.execute('SELECT id FROM produits WHERE id = ?', [id]);
    if (existing.length === 0) {
      return res.status(404).json({
        success: false,
        message: 'Produit non trouvé'
      });
    }

    // Supprimer
    const query = 'DELETE FROM produits WHERE id = ?';
    await db.execute(query, [id]);

    res.json({
      success: true,
```

```
      message: 'Produit supprimé avec succès',
      deletedId: parseInt(id)
    });
  } catch (error) {
    res.status(500).json({
      success: false,
      message: 'Erreur lors de la suppression',
      error: error.message
    });
  }
});

module.exports = router;
```

Étape 5 : Serveur principal (server.js)

```
const express = require('express');
const cors = require('cors');
const bodyParser = require('body-parser');
require('dotenv').config();

// Import des routes
const produitsRoutes = require('./routes/produits');

// Créer l'application
const app = express();
const PORT = process.env.PORT || 3000;

// Middlewares
app.use(cors());
```

```
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));

// Route de test
app.get('/', (req, res) => {
  res.json({
    message: 'API CRUD Produits',
    endpoints: {
      'GET /api/produits': 'Liste tous les produits',
      'GET /api/produits/:id': 'Obtenir un produit',
      'POST /api/produits': 'Créer un produit',
      'PUT /api/produits/:id': 'Modifier un produit',
      'DELETE /api/produits/:id': 'Supprimer un produit'
    }
  });
});

// Routes API
app.use('/api', produitsRoutes);

// Gestion des erreurs 404
app.use((req, res) => {
  res.status(404).json({
    success: false,
    message: 'Route non trouvée'
  });
});
```

```
// Démarrer le serveur
app.listen(PORT, () => {
 console.log(`
╔═══════════════════════════════════════════════════╗
║   Serveur CRUD démarré!           ║
║   URL: http://localhost:${PORT}         ║
║   API: http://localhost:${PORT}/api       ║
╚═══════════════════════════════════════════════════╝
 `);
});
```

Étape 6 : Configuration package.json

```
{
 "name": "backend-crud-produits",
 "version": "1.0.0",
 "description": "API CRUD pour gestion de produits",
 "main": "server.js",
 "scripts": {
  "start": "node server.js",
  "dev": "nodemon server.js"
 },
 "dependencies": {
  "express": "^4.18.2",
  "mysql2": "^3.6.0",
  "dotenv": "^16.3.1",
  "cors": "^2.8.5",
  "body-parser": "^1.20.2"
 },
 "devDependencies": {
```

```
    "nodemon": "^3.0.1"
  }
}
```

🧪 Étape 7 : Tests avec Postman

7.1 CREATE - Créer un produit

POST http://localhost:3000/api/produits

Content-Type: application/json

```
{
  "nom": "MacBook Pro M1",
  "description": "Ordinateur portable Apple",
  "prix": 1999.99,
  "stock": 5
}
```

Réponse attendue :

```
{
  "success": true,
  "message": "Produit créé avec succès",
  "data": {
    "id": 4,
    "nom": "MacBook Pro M1",
    "description": "Ordinateur portable Apple",
    "prix": 1999.99,
    "stock": 5
  }
}
```

7.2 READ - Lister tous les produits

GET http://localhost:3000/api/produits

Réponse attendue :


```
{
  "success": true,
  "count": 4,
  "data": [
    {
      "id": 4,
      "nom": "MacBook Pro M1",
      "description": "Ordinateur portable Apple",
      "prix": "1999.99",
      "stock": 5,
      "created_at": "2024-01-15T10:30:00.000Z"
    },
    ...
  ]
}
```


7.3 READ - Obtenir un produit

GET http://localhost:3000/api/produits/1

7.4 UPDATE - Modifier un produit

PUT http://localhost:3000/api/produits/1

Content-Type: application/json


```
{
  "nom": "iPhone 13 Pro",
  "description": "Smartphone Apple Pro",
  "prix": 1099.99,
  "stock": 12
}
```

7.5 DELETE - Supprimer un produit

DELETE http://localhost:3000/api/produits/1

Lancement du serveur

Mode développement (avec rechargement automatique)

npm run dev

Mode production

npm start