BUT3 VCOD R5.VCOD.07
Prog. Web Visualisation

TP 8 – Application FullStack avec NodeJS

Cas d'étude. Inscription et Connexion avec Node.js, MySQL (XAMPP) et Docker

Dans ce travail pratique, vous allez apprendre à créer une petite application web permettant à un utilisateur de s'inscrire et de se connecter à l'aide de Node.js et d'une base de données MySQL. Le tout sera exécuté dans un conteneur Docker, ce qui évite d'installer Node.js localement. MySQL sera géré via XAMPP.

Étape 1: Préparer l'environnement

Avant de commencer, assurez-vous que:

1. **Docker** est installé et fonctionne correctement sur votre machine.
    a. Ouvrez **Docker Desktop** (ou le lanceur Docker de votre machine).
    b. Attendez que l'icône Docker devienne **verte ou bleue** selon votre version. Cela indique que le **daemon Docker** est bien lancé.
2. **XAMPP** est démarré, avec le module **MySQL** activé.
    a. Ouvrez **XAMPP Control Panel**.
    b. Cliquez sur **Start** pour les modules :
        i. **Apache** (serveur web local)
        ii. **MySQL** (base de données)

    Une fois les deux modules démarrés, les voyants deviennent **verts**.
3. Vous pouvez maintenant accéder à **phpMyAdmin** via : http://localhost/phpmyadmin

Étape 2 : Créer la base de données MySQL

1. Ouvrez **phpMyAdmin**.
2. Cliquez sur **Nouvelle base de données** et nommez-la **compte_utilisateur**.
3. Une fois la base créée, allez dans l'onglet **SQL** et exécutez les commandes suivantes 1 par 1 :

```sql
CREATE DATABASE IF NOT EXISTS compte_utilisateur;
USE compte_utilisateur;
CREATE TABLE IF NOT EXISTS users (
  id INT AUTO_INCREMENT PRIMARY KEY,
  username VARCHAR(50) NOT NULL UNIQUE,
  password VARCHAR(255) NOT NULL
);
```

Cette table permettra de stocker les identifiants des utilisateurs.

Étape 3 : Créer la structure du projet

Dans votre dossier de travail (par exemple C:\Users\VotreNom\Downloads\), créez un dossier appelé **tp-login**. À l'intérieur, créez les fichiers et dossiers suivants :

```
compte_utilisateur/
├── server.js
├── package.json
├── Dockerfile
└── public/
    ├── index.html
    ├── login.html
    └── register.html
```

Étape 4 : Configurer le projet Node.js

1. **Fichier** *package.json*

```
{
  "name": "compte_utilisateur",
  "version": "1.0.0",
  "main": "server.js",
  "dependencies": {
    "express": "^4.18.2",
    "body-parser": "^1.20.2",
    "mysql2": "^3.6.0",
    "bcryptjs": "^2.4.3"
  }
}
```

Ce fichier définit les dépendances nécessaires pour exécuter votre application.

2. **Fichier** server.js

Voici le code principal de votre serveur, à ajouter au fur à mésure.

   a. **Importation des modules et création de l'application**

```
// Importation des modules nécessaires
const express = require('express');
const mysql = require('mysql2');
const bcrypt = require('bcryptjs');
const bodyParser = require('body-parser');
const path = require('path');

// Création de l'application Express
const app = express();
const port = 3000;
```

b. Middleware et dossier des fichiers statiques

```
// Middleware pour lire les données des formulaires
app.use(bodyParser.urlencoded({ extended: true }));

// Définir le dossier des fichiers statiques (HTML)
app.use(express.static(path.join(__dirname, 'public')));
```

c. Connexion à la base de données MySQL

```
// Connexion à la base MySQL (hébergée par XAMPP)
const db = mysql.createConnection({
  host: 'host.docker.internal', // permet d'accéder à MySQL local depuis Docker
  user: 'root',
  password: '', // mot de passe MySQL (vide par défaut)
  database: 'compte_utilisateur'
});

// Vérification de la connexion MySQL
db.connect(err => {
  if (err) throw err;
  console.log('✅ Connecté à MySQL');
});
```

d. Route principale

```
// Route principale
app.get('/', (req, res) => res.sendFile(path.join(__dirname, 'public', 'index.html')));
```

e. Route d'inscription

```
// Route d'inscription
app.post('/register', async (req, res) => {
  const { username, password } = req.body;
  const hashed = await bcrypt.hash(password, 10);

  db.query(
    'INSERT INTO users (username, password) VALUES (?, ?)',
    [username, hashed],
    (err) => {
      if (err) return res.send('⚠️ Erreur : ' + err.message);
      res.send(`🎉 Bravo ${username} ! Inscription réussie ! <a href='/login.html'>Connexion</a>`
    }
  );
});
```

f. Route de connexion

```
// Route de connexion
app.post('/login', (req, res) => {
  const { username, password } = req.body;

  db.query('SELECT * FROM users WHERE username = ?', [username], async (err, results) => {
    if (err) return res.send('⚠️ Erreur : ' + err.message);
    if (results.length === 0) return res.send('🙈 Utilisateur non trouvé');

    const match = await bcrypt.compare(password, results[0].password);
    if (!match) return res.send('❌ Mot de passe incorrect');

    res.send(`👋 Bienvenue ${results[0].username} ! Heureux de te revoir !`);
  });
});
```

g. Lancement du serveur

```
// Lancement du serveur
app.listen(port, () => console.log(`🚀 Serveur démarré sur http://localhost:${port}`));
```

Étape 5 : Créer les fichiers HTML

1. index.html

```
<h1>Accueil</h1>
<a href="/register.html">Inscription</a> | <a href="/login.html">Connexion</a>
```

2. register.html

```
<h2>Inscription</h2>
<form action="/register" method="POST">
  <input name="username" placeholder="Nom d'utilisateur" required><br>
  <input type="password" name="password" placeholder="Mot de passe" required><br>
  <button type="submit">S'inscrire</button>
</form>
<a href="/login.html">Connexion</a>
```

3. Login.html

```
<h2>Connexion</h2>
<form action="/login" method="POST">
  <input name="username" placeholder="Nom d'utilisateur" required><br>
  <input type="password" name="password" placeholder="Mot de passe" required><br>
  <button type="submit">Se connecter</button>
</form>
<a href="/register.html">Inscription</a>
```

Étape 6 : Créer le fichier Dockerfile

```
# Image de base légère avec Node.js 20
FROM node:20-alpine

# Définir le répertoire de travail dans le conteneur
WORKDIR /app

# Copier les fichiers de dépendances
COPY package*.json ./

# Installer les dépendances Node.js
RUN npm install

# Copier le reste du code du projet
COPY . .

# Exposer le port 3000 pour accéder à l'application
EXPOSE 3000

# Commande de démarrage du serveur
CMD ["node", "server.js"]
```

Ce fichier indique à Docker comment construire l'image du serveur Node.js.

Étape 7 : Construire et exécuter le projet dans Docker

1. Ouvrez un terminal dans le dossier **compte_utilisateur**.
2. Construisez l'image avec la commande :

```
docker build -t compte_utilisateur .
```

3. Exécutez le conteneur :

```
docker run -p 3000:3000 compte_utilisateur
```

Celle-ci démarre un conteneur à partir de l'image compte_utilisateur et mappe le port **3000** du conteneur sur le port **3000** de votre machine.

4. Ouvrez votre navigateur à l'adresse : http://localhost:3000
5. Vérification et test :
    a. Inscrivez un utilisateur via la page **Inscription**.

b. Connectez-vous avec les mêmes identifiants.
c. Vérifiez dans **phpMyAdmin** que l'utilisateur est bien ajouté à la table users

Étape 8 : Vérifier dans MySQL

Retournez dans phpMyAdmin et ouvrez la base tp_login. Vous verrez la table users contenant les utilisateurs inscrits. Les mots de passe sont hachés pour plus de sécurité.

Conclusion

Vous venez de créer un petit système d'authentification avec Node.js, Express, MySQL et Docker. Ce TP vous a permis de :

- Comprendre comment connecter un serveur Node.js à une base MySQL locale via Docker.
- Créer des formulaires HTML simples.
- Gérer l'inscription et la connexion avec hachage de mot de passe.
- Rendre le TP plus ludique avec des messages amusants.

Deuxième partie du TP : Passage à Docker Compose avec MySQL intégré

Dans cette deuxième partie, vous allez transformer le projet précédent (basé sur XAMPP) pour qu'il fonctionne entièrement dans Docker.
Le serveur Node.js et la base MySQL seront exécutés dans deux conteneurs distincts, orchestrés via **Docker Compose**.

Étape 1 — Préparation du projet existent

Vous partez du projet compte_utilisateur déjà fonctionnel avec XAMPP.
Voici l'arborescence que vous devez avoir à ce stade :

```
compte_utilisateur/
├── server.js
├── package.json
├── Dockerfile
└── public/
    ├── index.html
    ├── login.html
    └── register.html
```

Nous allons simplement ajouter un fichier docker-compose.yml et adapter le code pour que la connexion à MySQL se fasse dans un conteneur.

Étape 2 — Création du fichier docker-compose.yml

```
services:
  # Node.js application
  app:
    build: .
    container_name: compte_app
    ports:
      - "3000:3000"
    depends_on:
      db:
        condition: service_healthy
    environment:
      - DB_HOST=db
      - DB_USER=appuser
      - DB_PASSWORD=apppassword
      - DB_NAME=tp_login
    volumes:
      - .:/app
      - /app/node_modules  # Prevent local directory from overwriting
node_modules
    command: sh -c "npm install && node server.js"

  # MySQL service
  db:
    image: mysql:8.0
    container_name: compte_db
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: rootpassword
      MYSQL_DATABASE: tp_login
      MYSQL_USER: appuser
      MYSQL_PASSWORD: apppassword
    ports:
      - "3306:3306"
    volumes:
      - db_data:/var/lib/mysql
    healthcheck:
      test: ["CMD", "mysqladmin", "ping", "-h", "localhost"]
      timeout: 20s
      retries: 10

volumes:
  db_data:
```

Remarque : Vérifiez soigneusement l'indentation après avoir copié et collé le contenu du fichier YAML, car le YAML est sensible à l'indentation.

- **app** correspond auserveur Node.js.
- **db** est le service MySQL.
- **depends_on** garantit que MySQL démarre avant ton application.
- **Les variables d'environnement** permettent à server.js de savoir comment se connecter à la base.

Étape 3 — Adapter le code du serveur

- Avant (avec XAMPP)

```
const db = mysql.createConnection({
  host: 'host.docker.internal',
  user: 'root',
  password: '',
  database: 'compte_utilisateur'
});
```

- Après avec docker-compose

```
const db = mysql.createConnection({
  host: process.env.DB_HOST || 'localhost',
  user: process.env.DB_USER || 'root',
  password: process.env.DB_PASSWORD || '',
  database: process.env.DB_NAME || 'compte_utilisateur'
});
```

Maintenant que nous n'avons pas accès à phpmysql pour créer la table user à la main au préalable on va le créer en utilizer le code du serveur. C'est comme ça que ça se pense en réalité.

- Avant docker-compose

```
// Connexion
db.connect(err => {
  if (err) {
    console.error('❌ Erreur de connexion à MySQL :', err.message);
    return;
  }
});
```

- Après docker compose

```
// Connect to MySQL and create table if it doesn't exist
db.connect(err => {
  if (err) throw err;
  console.log('✅ Connecté à MySQL (via Docker)');

  // Create users table if it doesn't exist
  const createTableQuery = `
    CREATE TABLE IF NOT EXISTS users (
      id INT AUTO_INCREMENT PRIMARY KEY,
      username VARCHAR(50) NOT NULL UNIQUE,
      password VARCHAR(255) NOT NULL
    )
  `;

  db.query(createTableQuery, (err) => {
    if (err) {
      console.error('❌ Erreur lors de la création de la table:', err.message);
    } else {
      console.log('✅ Table "users" prête');
    }
  });
});
```

Étape 4 — Construire et lancer les conteneurs

Dans un terminal

```
cd compte_utilisateur
docker compose up --build
```

La première fois, Docker va :

- Télécharger les images nécessaires (**node:20-alpine, mysql:8.0**)
- Construire l'image du projet **Node.js**
- Démarrer les deux services (**app** et **db**)

Étape 6 — Tester l'application

- Ouvre ton navigateur à l'adresse : http://localhost:3000
- Ouvre ton navigateur à l'adresse : http://localhost:3306 (base de donées,; pas très visible sans outil, vous le verez en binaire)

Exemple d'utilisation de la base de donnée sans outil comme phpMyAdmin

```
docker exec -it compte_db mysql -uroot -prootpassword -e "SELECT * FROM compte_utilisateur.users;"
```

```
docker exec -it compte_db mysql -uroot -prootpassword -e "CREATE TABLE IF NOT EXISTS compte_utilisateur.users (id INT AUTO_INCREMENT PRIMARY KEY, username VARCHAR(50) UNIQUE, password VARCHAR(255));
```

Bilan pédagogique

Grâce à cette deuxième partie du TP, vous avez :

- Supprimé la dépendance à XAMPP
- Créé une infrastructure multi-conteneur Node.js + MySQL
- Utilisé Docker Compose pour orchestrer vos services
- Géré la persistance des données grâce à un volume Docker (db_data)
- Compris comment les variables d'environnement facilitent la configuration