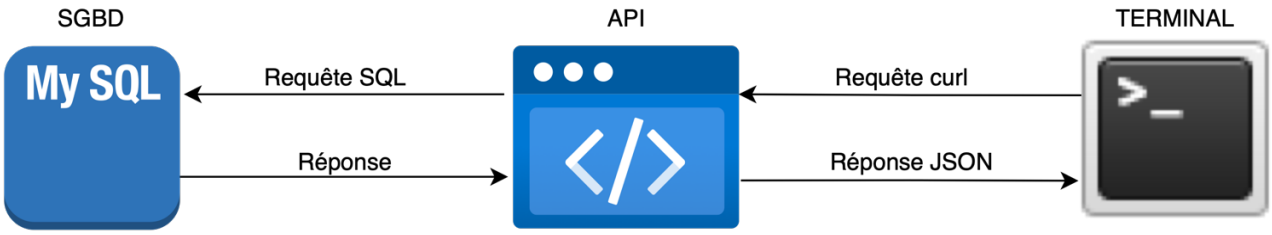
CODAGE D'UNE API REST



Création de la base de donnée Mysql

1. Créer une base de donnée que l'on nommera : students_db à partir de terminal.

2. Créer une table nommée students (id INT AUTO_INCREMENT PRIMARY KEY, name VARCHAR(255) NOT NULL, email VARCHAR(255) NOT NULL UNIQUE, age INT NOT NULL) à partir du terminal.

3. Insérer quelques données pour tester :
('Alice', 'alice@example.com', 20), ('Bob', 'bob@example.com', 22).

Création du projet :

1. Créer un répertoire api.
2. Créer dans ce répertoire une projet nodejs : `npm init -y`
3. Installer les modules express mysql2 dotenv.

Création de l'API REST :

Configuration de la connexion à MySQL

1. Créer un fichier .env pour stocker les informations de connexion :

```
DB_HOST=localhost
DB_USER=ciel
DB_PASSWORD=ciel
DB_NAME=students_db
```

2. Créer un fichier db.js pour gérer la connexion :

```
const mysql = require('mysql2');
const dotenv = require('dotenv');

dotenv.config();

const pool = mysql.createPool({
  host: process.env.DB_HOST,
  user: process.env.DB_USER,
```

```
  password: process.env.DB_PASSWORD,
  database: process.env.DB_NAME,
});

module.exports = pool.promise();
```

Explications du code :

Création d'un pool de connexions

dotenv.config(); // Charge les variables d'environnement avant de les utiliser

const pool = mysql.createPool({
* host: process.env.DB_HOST,*
* user: process.env.DB_USER,*
* password: process.env.DB_PASSWORD,*
* database: process.env.DB_NAME,*
});

Qu'est-ce qu'un pool de connexions ?

*Un **pool de connexions** est un ensemble de connexions pré-créées à la base de données.*

- *Avantage : Cela améliore les performances car il n'est pas nécessaire d'ouvrir et de fermer une connexion à chaque requête. Les connexions sont réutilisées.*

Les options utilisées :

- ***host** : L'adresse de votre serveur MySQL.*
- ***user** : Le nom d'utilisateur MySQL.*
- ***password** : Le mot de passe associé à l'utilisateur.*
- ***database** : Le nom de la base de données à utiliser.*

Pourquoi process.env ?

- ***process.env** est utilisé pour lire les variables d'environnement définies dans le fichier .env.*
    - *Avantage : Les informations sensibles ne sont pas directement dans le code source. Cela permet également de changer les configurations sans modifier le code.*

Utilisation de pool.promise()

*En appelant .promise(), on dit à mysql2 de fonctionner en mode **Promise**. Cela nous permettrai de faire par exemple :*

*const [rows] = await pool.query('SELECT * FROM table_name');*

Exportation du pool

module.exports = pool.promise();

module.exports : Permet de rendre le pool disponible dans d'autres fichiers.

Création de l'API :

1. Créer un fichier server.js pour gérer la connexion :

```
const db = require('./db');

async function getStudents() {
  const [rows] = await db.query('SELECT * FROM students');
  console.log(rows);
}

getStudents();
```

Explications du code :

- *async function*
  - *L'ajout de async transforme getStudents en une fonction asynchrone.*
  - *Cela permet d'utiliser await à l'intérieur de la fonction pour gérer des Promises de manière plus lisible.*

- *db.query : Appelle une méthode de requête MySQL depuis le pool de connexions.*
  - *db.query('SELECT * FROM students') envoie une requête SQL pour récupérer tous les enregistrements de la table students.*
  - *Cette méthode renvoie une Promise contenant deux éléments :*
    1. *rows : Les résultats de la requête sous forme d'un tableau d'objets.*
    2. *fields (non utilisé ici) : Les métadonnées des colonnes de la requête.*

- *await attend que la Promise retournée par db.query soit résolue.*
  - *Cela permet d'obtenir directement les résultats sous forme de tableau (rows), sans utiliser .then().*

- *Destructuration [rows]*
  - *La syntaxe [rows] extrait directement la première valeur renvoyée par la Promise (les lignes de la requête SQL).*

2. Modifier le fichier server.js :

```
const express = require('express');
const dotenv = require('dotenv');
const db = require('./db');

const app = express();
app.use(express.json());

const PORT = process.env.PORT || 3000;

// Routes
```

```
app.get('/', (req, res) => {
  res.send('API REST Node.js + MySQL');
});

// Démarrer le serveur
app.listen(PORT, () => {
  console.log(`Serveur démarré sur http://localhost:${PORT}`);
});
```

3. Tester maintenant l'api depuis un navigateur : http://localhost:3000

4. Ajouter le code suivant dans server.js :

```
app.get('/students', async (req, res) => {
try {
const [rows] = await db.query('SELECT * FROM students');
res.json(rows);
} catch (err) { res.status(500).json({ error: err.message }); }
});
```

5. Expliquer le code précédent.

6. Indiquer la signification du code 500.

7. Tester le code à l'aide d'une requête curl :

```
curl -X GET http://localhost:3000/students
```

8. Ajouter le code suivant au fichier server.js :

```
app.post('/students', async (req, res) => {
  const { name, email, age } = req.body;
  try {
    const sql = `INSERT INTO students (name, email, age) VALUES
('${name}', '${email}', ${age})`;
    const [result] = await db.query(sql);
    res.status(201).json({ id: result.insertId, name, email, age });
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});
```

9. Expliquer le code précédent.

10. Indiquer la signification du code 201.

11. Tester le code à l'aide d'une requête curl :

```
curl -X POST http://localhost:3000/students \
-H "Content-Type: application/json" \
-d '{"name": "Alice", "email": "alice@example.com", "age":
20}'
```

12. Ajouter le code suivant au fichier server.js :

```
app.get('/students/:id', async (req, res) => {
  try {
    // Construire la requête SQL directement
    const sql = `SELECT * FROM students WHERE id =${req.params.id}`;

    // Exécuter la requête
    const [rows] = await db.query(sql);

    // Vérifier si l'étudiant existe
    if (rows.length === 0) {
      return res.status(404).json({ error: 'Student not found' });
    }

    // Retourner les données de l'étudiant
    res.json(rows[0]);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});
```

13. Expliquer le code précédent.

14. Indiquer et tester la requête curl à saisir pour sélectionner l'étudiant 2.

15. Ajouter le code suivant au fichier server.js :

```
app.put('/students/:id', async (req, res) => {
  const { name, email, age } = req.body;
  const id = req.params.id;

  try {
    // Construire la requête SQL directement
    const sql = `UPDATE students SET name = '${name}', email =
'${email}', age = ${age} WHERE id = ${id}`;

    // Exécuter la requête
    const [result] = await db.query(sql);

    // Vérifier si un étudiant a été mis à jour
    if (result.affectedRows === 0) {
      return res.status(404).json({ error: 'Student not found' });
    }

    // Répondre avec succès
    res.json({ message: 'Student updated' });
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});
```

16. Expliquer le code précédent.

17. Indiquer et tester la requête curl à saisir pour modifier l'âge de l'étudiant1.

18. Proposer le code pour effacer un enregistrement.

19. Indiquer et tester la requête curl à saisir pour modifier l'âge de l'étudiant2.