Creazione di un'Applicazione Backend con JavaScript e SQLite

Creazione di un'Applicazione Backend con JavaScript e SQLite

Obiettivo della Lezione

Questa lezione ha lo scopo di insegnarti come:

1. Creare un'applicazione backend utilizzando Node.js.
2. Gestire un database SQLite per archiviare e recuperare dati.
3. Utilizzare un'architettura RESTful per gestire le richieste.

---

1. Configurazione dell'Ambiente di Lavoro

Prerequisiti

- **Node.js** (installato sulla tua macchina)
- **SQLite** (incluso nella libreria Node.js tramite il pacchetto `sqlite3`)
- **Postman** (per testare le API)

Creazione del Progetto

1. Crea una nuova directory per il progetto:

```
mkdir my-backend-app
cd my-backend-app
```

2. Inizializza un progetto Node.js:

```
npm init -y
```

3. Installa il pacchetto `sqlite3`:

```
npm install sqlite3
```

4. Installa anche il pacchetto `express` per creare il server:

```
npm install express
```

---

2. Creazione del Server con Express

Crea un file chiamato `server.js` nella directory principale del progetto.

Contenuto di `server.js`:

```
const express = require('express');
const sqlite3 = require('sqlite3').verbose();

const app = express();
const PORT = 3000;
```

```
// Connessione al database SQLite
const db = new sqlite3.Database('./database.db', (err) => {
  if (err) {
    console.error('Errore durante la connessione al database:', err);
  } else {
    console.log('Connesso al database SQLite');
  }
});

// Middleware per gestire il parsing dei JSON
app.use(express.json());

// Avvio del server
app.listen(PORT, () => {
  console.log(`Server in esecuzione su http://localhost:${PORT}`);
});
```

Esegui il server:

```
node server.js
```

Dovresti vedere il messaggio **"Server in esecuzione su http://localhost:3000"**.

---

3. Creazione del Database e delle Tabelle

Modifica il file `server.js` per creare una tabella utente nel database:

Aggiorna `server.js`:

```
// Creazione della tabella utenti
const createTableQuery = `
  CREATE TABLE IF NOT EXISTS utenti (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    nome TEXT NOT NULL,
    email TEXT NOT NULL,
    password TEXT NOT NULL
  );
`;

db.run(createTableQuery, (err) => {
  if (err) {
    console.error('Errore durante la creazione della tabella:', err);
  } else {
    console.log('Tabella utenti creata con successo');
  }
});
```

Riavvia il server per eseguire la query di creazione della tabella.

---

4. Creazione delle API RESTful

Endpoint per Creare un Utente (POST)

Aggiungi il seguente codice in `server.js`:

```
// Endpoint per creare un nuovo utente
app.post('/utenti', (req, res) => {
  const { nome, email, password } = req.body;
```

```
  if (!nome || !email || !password) {
    return res.status(400).json({ error: 'Tutti i campi sono obbligatori' });
  }

  const insertQuery = `INSERT INTO utenti (nome, email, password) VALUES (?, ?, ?)`;

  db.run(insertQuery, [nome, email, password], function(err) {
    if (err) {
      return res.status(500).json({ error: 'Errore durante l'inserimento dell'utente' });
    }

    res.status(201).json({ id: this.lastID, nome, email });
  });
});
```

Testa questo endpoint con Postman inviando una richiesta POST a `http://localhost:3000/utenti` con un body JSON simile a:

```
{
  "nome": "Mario Rossi",
  "email": "mario.rossi@example.com",
  "password": "123456"
}
```

Endpoint per Recuperare Tutti gli Utenti (GET)

Aggiungi questo codice:

```
// Endpoint per recuperare tutti gli utenti
app.get('/utenti', (req, res) => {
  const selectQuery = `SELECT * FROM utenti`;

  db.all(selectQuery, [], (err, rows) => {
    if (err) {
      return res.status(500).json({ error: 'Errore durante il recupero degli utenti' });
    }

    res.json(rows);
  });
});
```

Testa l'endpoint inviando una richiesta GET a `http://localhost:3000/utenti`.

---

5. Aggiornamento e Cancellazione degli Utenti

Endpoint per Aggiornare un Utente (PUT)

Aggiungi il codice:

```
// Endpoint per aggiornare un utente
app.put('/utenti/:id', (req, res) => {
  const { id } = req.params;
  const { nome, email, password } = req.body;

  if (!nome || !email || !password) {
    return res.status(400).json({ error: 'Tutti i campi sono obbligatori' });
  }

  const updateQuery = `UPDATE utenti SET nome = ?, email = ?, password = ? WHERE id = ?`;
```

```
  db.run(updateQuery, [nome, email, password, id], function(err) {
    if (err) {
      return res.status(500).json({ error: 'Errore durante l'aggiornamento dell'utente' });
    }

    res.json({ message: 'Utente aggiornato con successo' });
  });
});
```

Endpoint per Eliminare un Utente (DELETE)

Aggiungi il codice:

```
// Endpoint per eliminare un utente
app.delete('/utenti/:id', (req, res) => {
  const { id } = req.params;

  const deleteQuery = `DELETE FROM utenti WHERE id = ?`;

  db.run(deleteQuery, id, function(err) {
    if (err) {
      return res.status(500).json({ error: 'Errore durante l'eliminazione dell'utente' });
    }

    res.json({ message: 'Utente eliminato con successo' });
  });
});
```

---

6. Conclusione

Hai ora creato un'applicazione backend completa utilizzando Node.js e SQLite. Hai imparato a:

- Configurare un server Express.
- Creare e gestire un database SQLite.
- Creare API RESTful per creare, leggere, aggiornare ed eliminare utenti.

Continua a esplorare e migliorare l'applicazione aggiungendo funzionalità come autenticazione e gestione degli errori.