Guideline — Connect MySQL with JavaScript (Node.js)

A step-by-step guide to build a simple user registration app using Node.js, Express, MySQL and EJS. Readers will learn how to connect to MySQL, run queries from JavaScript, and render a simple UI.

Prerequisites

- Node.js (v16+ recommended)
- MySQL server running (this guide uses port `3307` as an example)
- Basic terminal and editor knowledge

Project Overview & Files

We'll create a small Express app with these responsibilities:

- Serve registration form and users list (EJS views)
- Connect to MySQL using `mysql2` promise API
- Hash passwords with `bcryptjs`
- Serve a small stylesheet from `public/`

Important files readers should review:

- `index.js` — Express server and routes
- `db.js` — MySQL pool and connection logic
- `views/index.ejs` & `views/users.ejs` — templates
- `public/styles.css` — UI styles
- `.env` — environment variables (DB credentials)

Step 1 — Create project and install packages

From PowerShell or terminal:

```
 mkdir MySqlJsApp
cd MySqlJsApp
npm init -y
npm install express ejs mysql2 bcryptjs dotenv
```

Step 2 — Create database and table

Use the MySQL client (this example uses port `3307`):

```
mysql -u root -P 3307 -p -e "CREATE DATABASE IF NOT EXISTS myappdb;
USE myappdb;
CREATE TABLE IF NOT EXISTS users (id INT AUTO_INCREMENT PRIMARY KEY,
name VARCHAR(100) NOT NULL, email VARCHAR(255) NOT NULL UNIQUE,
password VARCHAR(255) NOT NULL,
created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP);"
```

Step 3 — (Recommended) Create an app DB user

```
mysql -u root -P 3307 -p -e "CREATE USER 'appuser'@'localhost'
      IDENTIFIED BY 'StrongPass123'; GRANT ALL PRIVILEGES ON myappdb.*
      TO 'appuser'@'localhost'; FLUSH PRIVILEGES;"
```

> Use a stronger password in production and restrict privileges appropriately.

Step 4 — Environment variables

Create a `.env` file in project root (do not commit this file):

```
DB_HOST=localhost
DB_PORT=3307
DB_USER=appuser
DB_PASSWORD=StrongPass123
DB_NAME=myappdb
PORT=3000
```

The app uses `dotenv` to load these values.

Step 5 — DB connector (`db.js`)

Use `mysql2/promise` and a connection pool. Example code:

```
const mysql = require('mysql2/promise');
require('dotenv').config();

const pool = mysql.createPool({
  host: process.env.DB_HOST,
  port: Number(process.env.DB_PORT),
  user: process.env.DB_USER,
  password: process.env.DB_PASSWORD,
  database: process.env.DB_NAME,
  connectionLimit: 10
});

module.exports = pool;
```

Step 6 — Express server and routes (`index.js`)

Main points:

- Load `dotenv`, create Express app, set EJS view engine.
- Serve static files from `public/` for CSS.
- `POST /register` hashes the password and inserts into DB using parameterized query.
- `GET /users` queries and renders the users list.

```
app.post('/register', async (req, res) => {
  const { name, email, password } = req.body;
  const hashed = await bcrypt.hash(password, 10);
  await pool.execute('INSERT INTO users (name, email, password)
  VALUES (?, ?, ?)', [name, email, hashed]);
  res.redirect('/users');
});
```

Step 7 — Views & UI

Templates are EJS files in `views/`. EJS basics:

- `<% code %>` — run JS without output
- `<%= value %>` — escaped output

We added a simple `public/styles.css` for a modern look.

Step 8 — Start the app

```
npm install
npm start
# open http://localhost:3000
```

Troubleshooting

Access denied for user 'root'@'localhost' (using password: NO)

This means Node tried to connect without a password. Fix by creating a proper `.env` with `DB_USER` and `DB_PASSWORD`, then restart the server.

ER_DUP_ENTRY

Caused by inserting a duplicate value into a UNIQUE column (e.g., email). Either change the email or handle duplicates in code.

Security & Best Practices (brief)

- Never commit `.env` into version control.
- Hash passwords (we use `bcryptjs`).
- Use parameterized queries to avoid SQL injection.
- Create a dedicated DB user with least privileges for the app.

Future Scope

- Add login and sessions with `express-session`.
- Add client-side validation and nice success messages.
- Add pagination to the users list.
- Write unit tests for DB modules (use a separate test DB).

Cloning Instruction - (If you do not want to create this project manually from scratch)

After cloning this GitHub repository, you can run the following commands to get the project running locally. Commands assume Windows PowerShell (replace the git URL with your repo URL). If using macOS/Linux, use `cp` instead of `copy` for the `.env` step.

```
 git clone https://github.com/your-username/your-repo.git
cd your-repo
npm install

# Create the database and table (MySQL on port 3307)
mysql -u root -P 3307 -p -e "CREATE DATABASE IF NOT EXISTS myappdb;
USE myappdb;
CREATE TABLE IF NOT EXISTS users (id INT AUTO_INCREMENT
PRIMARY KEY, name VARCHAR(100) NOT NULL, email VARCHAR(255) NOT NULL UNIQUE,
password VARCHAR(255) NOT NULL,
created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP);"

# Create an app DB user and grant privileges
```

```
mysql -u root -P 3307 -p -e "CREATE USER 'appuser'@'localhost'
IDENTIFIED BY 'YourStrongPassword'; GRANT ALL PRIVILEGES ON myappdb.*
TO 'appuser'@'localhost'; FLUSH PRIVILEGES;"

# Copy example environment file and edit credentials
copy .env.example .env
# (Windows) then open .env in an editor and set DB_PASSWORD
# (macOS/Linux) use: cp .env.example .env

# Start the app
npm start
# Open http://localhost:3000 in the browser
```

> Remember to update `.env` with the correct `DB_USER` and `DB_PASSWORD` before starting the app.

Generated documentation — follow the steps in order. For automatic PDF creation run `npm run make-pdf` after installing dependencies.