Node Js and Express Js with MySql

Node.js, Express, and MySQL are commonly used together to build robust, full-stack web applications. Here's a detailed overview of each component and how they work together.

1. Node.js

What is Node.js?

- Node.js is a JavaScript runtime environment built on Chrome's V8 JavaScript engine. It allows developers to use JavaScript for server-side scripting, enabling the

creation of scalable and high-performance web applications.

Key Features:

- **Asynchronous and Event-Driven**: Node.js handles multiple requests without waiting for previous ones to complete, making it highly efficient for I/O-heavy tasks.
- **Single-Threaded but Scalable**: Uses non-blocking I/O operations that allow a single thread to handle multiple concurrent connections.
- **NPM (Node Package Manager)**: Provides access to thousands of libraries and modules that simplify development.
- **Cross-Platform**: Runs on various platforms including Windows, macOS, and Linux.

Use Cases:

- Building RESTful APIs
- Real-time applications (e.g., chat apps, online games)
- Command-line tools
- Microservices

2. Express.js

What is Express.js?

- Express.js is a minimal and flexible Node.js web application framework that provides a robust set of features for building web and mobile applications. It simplifies the process of setting up routes, handling requests, and managing middleware.

Key Features:

- **Middleware**: Functions that execute during the lifecycle of a request to the server. They can modify the request and response objects, end requests, or call the next middleware function.
- **Routing**: Easily define URL routes to handle different HTTP methods (GET, POST, PUT, DELETE).
- **Template Engines**: Integrates with various template engines like EJS, Pug, or Handlebars to generate dynamic HTML.
- **Error Handling**: Provides robust mechanisms to manage errors and respond appropriately.
- **Built-in Features**: Simplifies working with cookies, sessions, form data, and file uploads.

Use Cases:

- Building APIs
- Single Page Applications (SPA)
- Server-side rendering
- Middleware-based web applications

3. MySQL

What is MySQL?

- MySQL is an open-source relational database management system (RDBMS) that uses Structured Query Language (SQL) for managing and manipulating data. It's widely used for its reliability, performance, and ease of use.

Key Features:

- **Relational Database**: Stores data in tables with defined relationships, making data retrieval efficient and organized.

- **Scalability**: Supports large databases and high-volume applications.
- **ACID Compliance**: Ensures data integrity through Atomicity, Consistency, Isolation, and Durability properties.
- **Secure**: Provides strong data protection with access control, data encryption, and user authentication.
- **Extensive Support**: Wide community support, extensive documentation, and integration with various programming languages.

Use Cases:

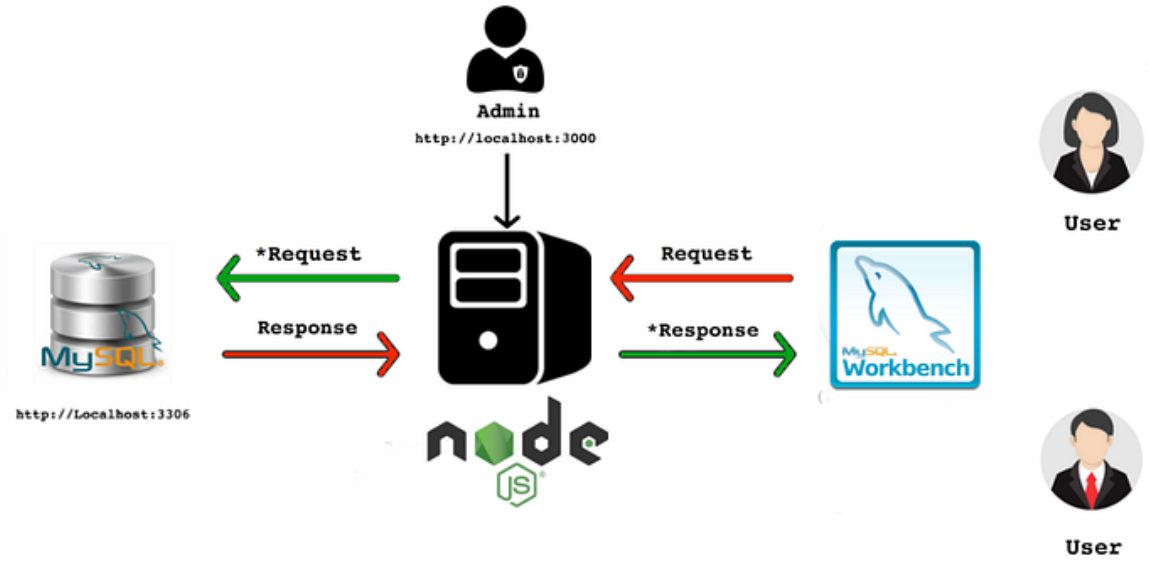
- Web applications (e.g., content management systems, ecommerce platforms)
- Data warehousing
- Embedded applications
- Online transaction processing (OLTP)

How They Work Together

- **Node.js** provides the runtime environment to execute JavaScript code on the server.

- **Express.js** acts as the web server framework, allowing easy setup of routes, middleware, and request handling.
- **MySQL** is used as the backend database to store, retrieve, and manage application data.

Together, Node.js, Express, and MySQL form a powerful stack for building full-fledged web applications, enabling developers to create server-side logic, manage data efficiently, and handle client requests seamlessly. This combination is particularly effective for building RESTful APIs, data-driven applications, and microservices.



Step 1: Install MySQL and MySQL Workbench

- Download MySQL from the official website: MySQL Downloads.

- After installation, add MySQL to your PATH in `.zshrc` (mac os)

```
export PATH=$PATH:/usr/local/mysql-9.0.1-macos14-x86_64/bin
```

- Restart the terminal or source the file:(mac os)

```
source ~/.zshrc
```

- Log into MySQL using:

```
mysql -u root -p
```

- Create a new database:

```
CREATE DATABASE mynode;
SHOW DATABASES;
```

Windows installation

- On windows download the installer .msi and select mysql and work bench with arrow from server and application
- Keep config. as default and use strong password and keep in note
- MySQL80 as service name rest keep it default
- Grant full permission

- Install MySQL Workbench to view and manage your database.

Download and Install MySQL Workbench (mac os)

1. **Download MySQL Workbench** from the official MySQL website: MySQL Workbench Downloads.
2. **Install MySQL Workbench** by opening the downloaded `.dmg` file and dragging the MySQL Workbench icon into your Applications folder.
3. **Launch MySQL Workbench**

- Open **MySQL Workbench** from your Applications folder.
- -Upon first launch, you might be prompted with a security warning; click “Open.”

Set Up a New Database Connection

Click on the **“+”** sign next to **“MySQL Connections”** to create a new connection.

Configure the Connection Settings:

- **Connection Name**: Give your connection a name (e.g., `Local MySQL`).
- **Connection Method**: Choose `Standard (TCP/IP)`.
- **Hostname**: `localhost` (if running MySQL locally).
- **Port**: `3306` (default port for MySQL).
- **Username**: `root` (or the user you've set up).
- **Password**: Click **"Store in Keychain"** and enter your MySQL password

Test Connection:

- Click **"Test Connection"** to verify that the connection settings are correct.
- If the connection is successful, you'll see a confirmation message. If not, double-check your hostname, port, username, and password.
- Click **"OK"** to save the connection.

Connect to the Database

- After saving the connection, you'll see it listed under **MySQL Connections** on the main screen.

- **Double-click the connection** you just created (`Local MySQL`) to connect to your MySQL server

Now, you can perform

- Creating a Database and Tables from workbench
- Importing and Exporting Data
- Managing Users and Permissions
- Backing Up Your Database

Step 2: Setup Node.js Project with Typescript

- Install Node.js and npm on your machine.
- Verify installations:

```
node -v
npm -v
```

- Initialize a new project and install Express:

```
npm init -y
npm install mysql2
npm install express typescript ts-node @types/node @types/express --save-dev
```

- Create and update tsconfig.json

```
npx tsc – init
```

```
{
  "compilerOptions": {
    "target": "es6",
    "module": "commonjs",
    "outDir": "./dist",
    "strict": true,
    "esModuleInterop": true,
    "skipLibCheck": true,
    "forceConsistentCasingInFileNames": true
  },
  "include": ["src/**/*.ts"],
  "exclude": ["node_modules"]
}
```

Project Structure

- /node-mysql-app

```
|
├── controllers
| └── employee.ts
|
├── routes
| └── employee.ts
|
```

```
├── database
│   └── db.ts
│   └── config.ts
│
├── helper
│   └── helper.ts
│
└── index.tsx
```

Step 3: Database Connection (src/`database/db.ts`)

```
import mysql from "mysql2/promise";
import config from "./config";
async function query(sql:string, params:any) {
  const connection = await mysql.createConnection(config.db);
  const [results] = await connection.query(sql, params);
  return results;
}

export default  {
  query,
};
```

Step 4: Define config (src/`database/config.ts`)

```
const config = {
    db: {
        host: "localhost",
```

```
        user: "root",
        password: "admin",
        database: "nodemysql",
    },
    port: 3000,
    listPerPage: 100
};
export default config;
```

Step 5: Define Routes (src/`routes/employee.js`)

with all CRUD operation

```
import express, { NextFunction, Request, response, Response } from
"express";
import employee from "../controller/employee";

const router = express.Router();
//create employee
router.post(
  "/create",
  async (req: Request, res: Response, next: NextFunction) => {
    try {
      res.status(200).json(await employee.create(req.body));
    } catch (err) {
      next(err);
    }
  }
```

```
);
// get employee by id

router.get("/all", async (req: Request, res: Response, next) => {
  try {
    const emp = await employee.getAllEmployees(req);
    res.json(emp);
  } catch (err: any) {
    console.error(`Error while getting Records `, err.message);
    next(err);
  }
});

//update employee by id

router.put("/:id", async (req: Request, res: Response, next:
NextFunction) => {
  try {
    const emp = await employee.update(req.params.id, req.body);
    res.json(emp);
  } catch (err: any) {
    console.error(`Error while getting Records `, err.message);
    next(err);
  }
});

// Delete employee by id
```

```
router.delete("/:id", async (req: Request, res:
Response,next:NextFunction) => {
  try {
    const emp = await employee.remove(req.params.id);
    res.json(emp);
  } catch (err: any) {
    console.error(`Error while getting Records `, err.message);
    next(err);
  }
});

export default router;
```

Step 6: Create Controller (src/`controllers/employee.ts`)

```
import db from "../database/db";
import { emptyOrRows } from "../helper/helper";
import { IEmployee } from "../models/IEmployee";

async function getAllEmployees(req: any): Promise<IEmployee[]> {
  const rows = await db.query(`SELECT * FROM employee`);
  const data: IEmployee[] = emptyOrRows(rows);
  return data;
}
async function getEmployeeByID(req: any): Promise<IEmployee[]> {
  const rows = await db.query(
    `SELECT * FROM employee WHERE id=${req.params.id}`
```

```
  );
  const data: IEmployee[] = emptyOrRows(rows);
  return data;
}
async function create(payload: any): Promise<Object> {
  const data = {
    name: payload.name,
    email: payload.email,
  };
  const result: any = await db.query(`INSERT INTO employee SET ?`,
data);
  let message = "Error in creating Record";
  if (result.affectedRows) {
    message = "Record created successfully";
  }
  return { message };
}
async function update(id: string, payload: any): Promise<Object> {
  const data = {
    name: payload.name,
    email: payload.email,
  };
  const result: any = await db.query(`UPDATE employee SET? WHERE id =
?`, [
    data,
    id,
  ]);
```

```
  let message = "Error in updating Record";
  if (result.affectedRows) {
    message = "Record updated successfully";
  }
  return { message };
}
async function remove(id: string): Promise<Object> {
  const result: any = await db.query(`DELETE FROM employee WHERE id=?`,
[id]);
  let message = "Error in deleting Record";
  if (result.affectedRows) {
    message = "Record deleted successfully";
  }
  return { message };
}
export default {
  getAllEmployees,
  create,
  update,
  remove,
  getEmployeeByID,
};
```

Step 7: Main Server File (src/`index.ts`) and package.json

```
import express, { NextFunction, Request, Response } from "express";
const app = express();
import db from "./database/config";
import employee from "./routes/employee";
const port = db.port;
app.use(express.json());
app.use(express.urlencoded({ extended: true }));

app.use("/employee", employee);
/* Error handler middleware */
app.use((err: any, req: Request, res: Response, next: NextFunction) => {
    const statusCode = err.statusCode || 500;
    res.status(statusCode).json({ message: err.message });
    return;
});
app.listen(port, () => {
    console.log(`Example app listening at http://localhost:${port}`);
});
```

Create Helper file at ***src/helper/helper.ts*** to check the empty rows form the database response.

```
export function emptyOrRows(rows: any) {
  if (!rows) {
    return [];
  }
```

```
  return rows;
}
```

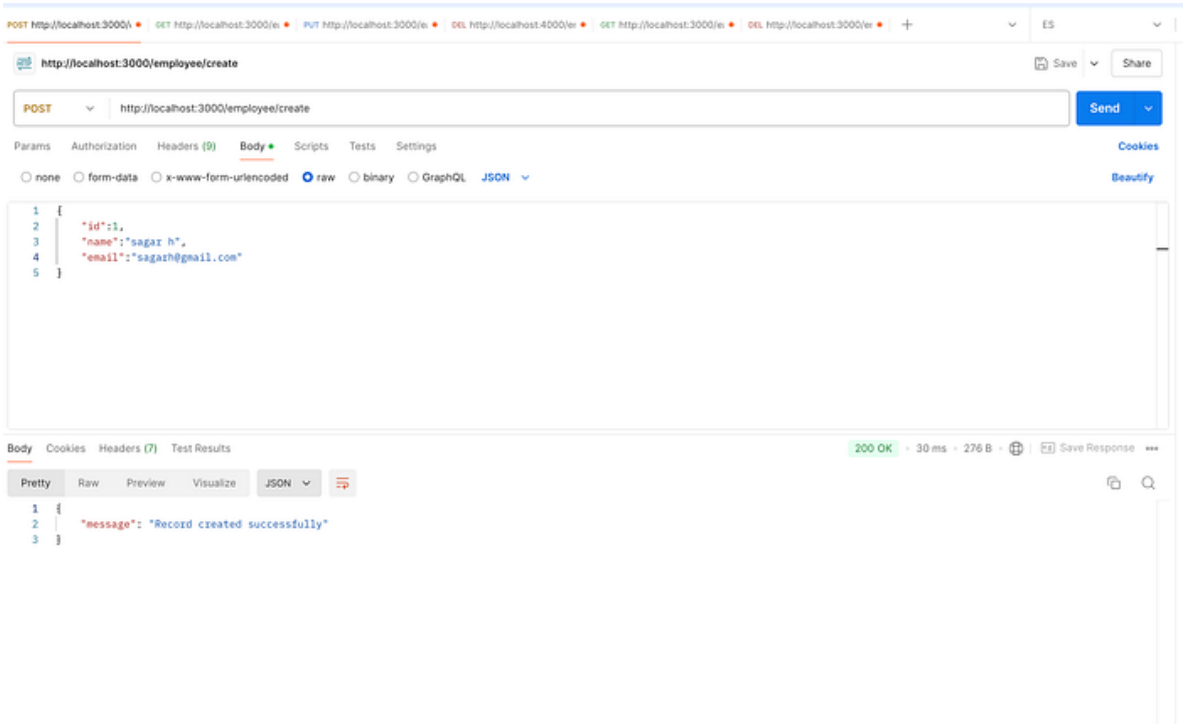
- **Update the *package.json***

```
"main": "index.ts",
...
"scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "ts-node src/index.ts",
    "build": "tsc",
    "serve": "node dist/index.js"
  },
```

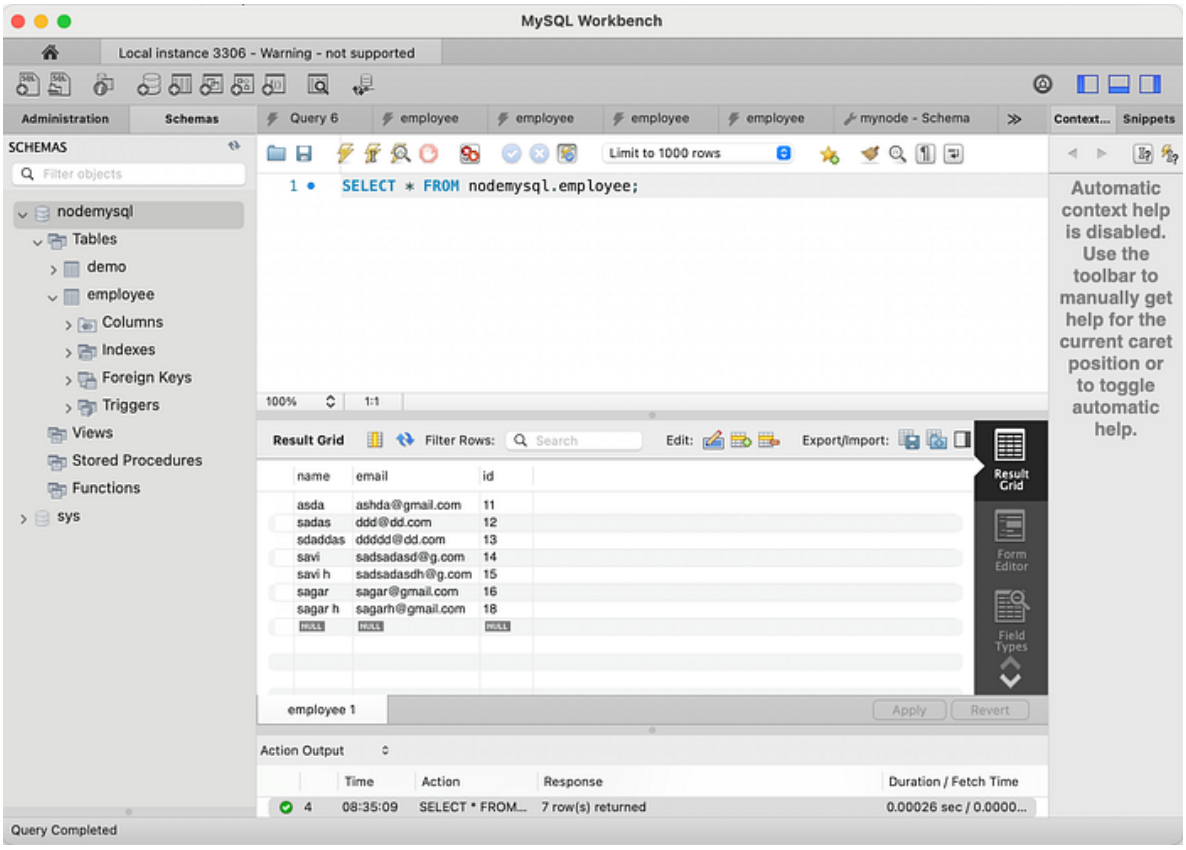
- **Start the server:**

```
node index.js
```

- Open your browser/postman and visit `http://localhost:3000/create` to create the employee .



Now, View database and table to see the records in mysqlworkbench



Conclusion

We now have a functioning API server that uses Node.js and MySQL with TYpoescript. This tutorial taught us how to set up MySQL and MySQL Workbench as a free service. We then created an Express.js server that can handle various HTTP methods (GET, POST,PUT,DELETE) concerning how it translates to SQL queries.

Happy Coding ….!

Github Project : https://github.com/sagarhudge/Node-Express-Mysql