///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

express boiler plate for mysql database connetion and controller logic for every request and routing

Setting up an Express.js boilerplate with MySQL database connection and a structured approach to controller logic and routing involves several steps. Here’s a comprehensive guide to get you started:

**Step 1: Set Up Your Project**

1. **Initialize a new Node.js project:**

bash

mkdir express-mysql-api

cd express-mysql-api

npm init -y

1. **Install required dependencies:**

bash

Copy code

npm install express mysql2

npm install --save-dev nodemon

**Step 2: Project Structure**

Organize your project into a clear structure:

Copy code

express-mysql-api/

│

├── config/

│ └── db.js

│

├── controllers/

│ └── itemController.js

│

├── routes/

│ └── itemRoutes.js

│

├── index.js

└── package.json

**Step 3: Database Configuration**

1. **Create config/db.js:**

javascript

Copy code

const mysql = require('mysql2');

const pool = mysql.createPool({

host: 'localhost',

user: 'your-username',

password: 'your-password',

database: 'your-database',

waitForConnections: true,

connectionLimit: 10,

queueLimit: 0

});

module.exports = pool.promise();

**Step 4: Controller Logic**

1. **Create controllers/itemController.js:**

javascript

Copy code

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

// Get all items

exports.getItems = async (req, res) => {

try {

const [rows] = await db.query('SELECT \* FROM items');

res.json(rows);

} catch (err) {

res.status(500).json({ error: err.message });

}

};

// Create a new item

exports.createItem = async (req, res) => {

const { name, description } = req.body;

try {

const [result] = await db.query('INSERT INTO items (name, description) VALUES (?, ?)', [name, description]);

res.status(201).json({ id: result.insertId, name, description });

} catch (err) {

res.status(500).json({ error: err.message });

}

};

**Step 5: Define Routes**

1. **Create routes/itemRoutes.js:**

javascript

Copy code

const express = require('express');

const router = express.Router();

const itemController = require('../controllers/itemController');

// Routes for items

router.get('/items', itemController.getItems);

router.post('/items', itemController.createItem);

module.exports = router;

**Step 6: Main Application File**

1. **Create index.js:**

javascript

Copy code

const express = require('express');

const app = express();

const port = 3000;

// Middleware to parse JSON bodies

app.use(express.json());

// Import routes

const itemRoutes = require('./routes/itemRoutes');

// Use routes

app.use('/api', itemRoutes);

// Start the server

app.listen(port, () => {

console.log(`Server is running on http://localhost:${port}`);

});

**Step 7: Update package.json**

1. **Update package.json to use nodemon for development:**

json

Copy code

"scripts": {

"start": "node index.js",

"dev": "nodemon index.js"

}

**Step 8: Run the Server**

1. **Start the server in development mode:**

bash

Copy code

npm run dev

**Additional Notes:**

* Ensure your MySQL database is running and accessible.
* Create a table named items in your MySQL database:

sql

Copy code

CREATE TABLE items (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(255) NOT NULL,

description TEXT

);

This setup provides a boilerplate for an Express.js API connected to a MySQL database, with a structured approach to controller logic and routing. You can extend this further by adding more controllers, routes, middleware, and other functionality as needed.

insdustry level setup

The setup provided is a good starting point for small to medium-sized projects or prototypes, but for industry-level applications, you need to incorporate more features, best practices, and tooling to ensure scalability, maintainability, and robustness. Here’s a more comprehensive guide to setting up an industry-level Express.js application with MySQL:

**Step 1: Project Structure**

Organize your project into a clear structure:

express-mysql-api/

│

├── config/

│ └── db.js

│

├── controllers/

│ └── itemController.js

│

├── middleware/

│ └── errorHandler.js

│

├── models/

│ └── itemModel.js

│

├── routes/

│ └── itemRoutes.js

│

├── utils/

│ └── logger.js

│

├── .env

├── .gitignore

├── index.js

└── package.json

**Step 2: Environment Configuration**

1. **Install dotenv:**

bash

Copy code

npm install dotenv

1. **Create .env file:**

env

Copy code

DB\_HOST=localhost

DB\_USER=your-username

DB\_PASSWORD=your-password

DB\_NAME=your-database

PORT=3000

1. **Update config/db.js to use environment variables:**

javascript

Copy code

const mysql = require('mysql2');

require('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,

waitForConnections: true,

connectionLimit: 10,

queueLimit: 0

});

module.exports = pool.promise();

**Step 3: Logger**

1. **Install winston:**

bash

Copy code

npm install winston

1. **Create utils/logger.js:**

javascript

Copy code

const { createLogger, format, transports } = require('winston');

const logger = createLogger({

level: 'info',

format: format.combine(

format.timestamp(),

format.printf(({ timestamp, level, message }) => {

return `${timestamp} ${level}: ${message}`;

})

),

transports: [

new transports.Console(),

new transports.File({ filename: 'error.log', level: 'error' }),

new transports.File({ filename: 'combined.log' })

],

});

module.exports = logger;

**Step 4: Middleware**

1. **Create middleware/errorHandler.js:**

javascript

Copy code

const logger = require('../utils/logger');

const errorHandler = (err, req, res, next) => {

logger.error(err.message);

res.status(500).json({ error: 'Internal Server Error' });

};

module.exports = errorHandler;

**Step 5: Models**

1. **Create models/itemModel.js:**

javascript

Copy code

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

const getAllItems = async () => {

const [rows] = await db.query('SELECT \* FROM items');

return rows;

};

const createItem = async (name, description) => {

const [result] = await db.query('INSERT INTO items (name, description) VALUES (?, ?)', [name, description]);

return { id: result.insertId, name, description };

};

module.exports = {

getAllItems,

createItem,

};

**Step 6: Controllers**

1. **Update controllers/itemController.js:**

javascript

Copy code

const itemModel = require('../models/itemModel');

exports.getItems = async (req, res, next) => {

try {

const items = await itemModel.getAllItems();

res.json(items);

} catch (err) {

next(err);

}

};

exports.createItem = async (req, res, next) => {

const { name, description } = req.body;

try {

const newItem = await itemModel.createItem(name, description);

res.status(201).json(newItem);

} catch (err) {

next(err);

}

};

**Step 7: Routes**

1. **Update routes/itemRoutes.js:**

javascript

Copy code

const express = require('express');

const router = express.Router();

const itemController = require('../controllers/itemController');

router.get('/items', itemController.getItems);

router.post('/items', itemController.createItem);

module.exports = router;

**Step 8: Main Application File**

1. **Update index.js:**

javascript

Copy code

const express = require('express');

const app = express();

const errorHandler = require('./middleware/errorHandler');

const itemRoutes = require('./routes/itemRoutes');

const logger = require('./utils/logger');

require('dotenv').config();

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

app.use(express.json());

app.use('/api', itemRoutes);

app.use(errorHandler);

app.listen(port, () => {

logger.info(`Server is running on http://localhost:${port}`);

});

**Step 9: Additional Enhancements**

1. **Add validation using a library like joi:**

bash

Copy code

npm install joi

1. **Add request logging using morgan:**

bash

Copy code

npm install morgan

1. **Add unit and integration tests using jest or mocha and chai:**

bash

Copy code

npm install jest --save-dev

1. **Ensure proper security with libraries like helmet and cors:**

bash

Copy code

npm install helmet cors

By incorporating these additional elements, your Express.js application will be more robust and ready for production environments, following industry-level best practices.