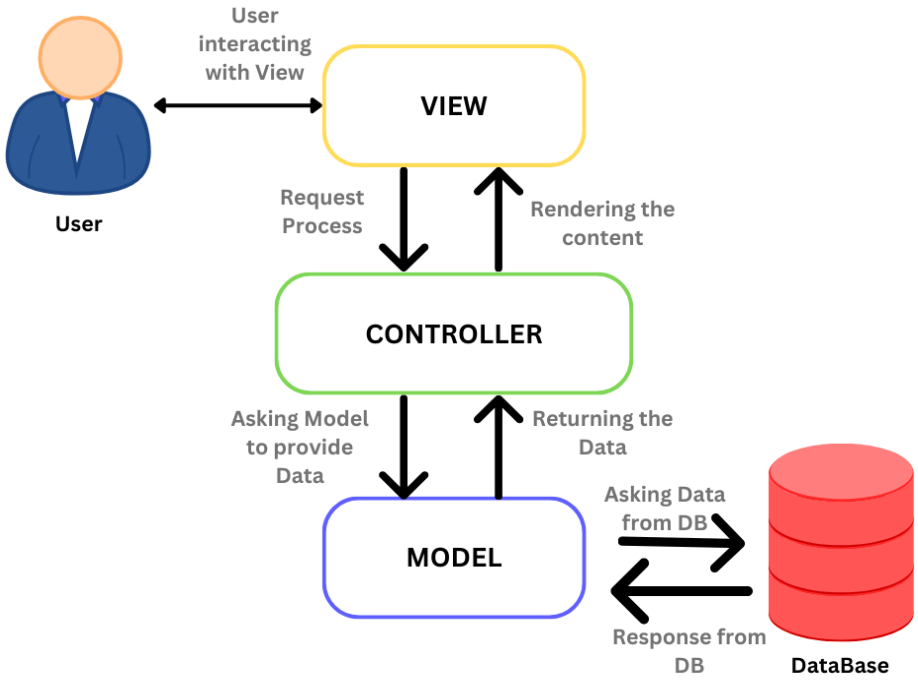
API Project

Node.js + MySQL Form

This project allows users to **submit their name and email**, stores it in **MySQL**, and shows how a **structured Node.js project (MVC)** works.

What We Will Build

- A **web form** (index.html) to collect Name and Email.
- **Backend API** in Node.js with **Express**.
- **Database storage** in MySQL.
- Project follows **MVC pattern**:
  - **Models** → Handle database queries.
  - **Controllers** → Business logic (decides what happens when someone submits form).
  - **Routes** → API endpoints (the URLs users or frontend talk to).
- Everything organized in **folders**.



Requirements

- Node.js installed on your computer.
- MySQL installed and running.
- Basic text editor (VS Code, Sublime, etc.).
- Browser to test the form.

Step 1: Folder Structure

Create a folder called `project`:

```
project/
├─ index.html                 # The web form
├─ server.js                  # Main server file
├─ package.json               # Node.js dependencies
├─ controllers/               # Business logic
│   └─ userController.js
├─ models/                    # Database logic
│   └─ userModel.js
├─ routes/                    # API routes
│   └─ userRoutes.js
├─ config/                    # Database config
│   └─ db.js
└─ public/                    # Optional CSS/JS
```

✅ Think of this like a school:

- `models` → library (database)
- `controllers` → teachers (logic/decisions)
- `routes` → hallways/doors (URLs)

Step 2: Install Node.js Packages

Open terminal in your project folder and type:

```
npm init -y
npm install express mysql2 body-parser cors
```

- express → Helps create a server quickly.
- mysql2 → Talks to MySQL database.
- body-parser → Reads form data sent by users.
- cors → Allows your browser to talk to the server without errors.

Step 3: Create MySQL Database

Open MySQL terminal:

```
mysql -u root -p
# enter password: user
```

Then type:

```
CREATE DATABASE formdb;

USE formdb;

CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

- AUTO_INCREMENT → automatically gives each user an ID.
- TIMESTAMP → saves the time user submitted the form.

Step 4: Database Connection

Create file: config/db.js

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

const pool = mysql.createPool({
    host: 'localhost',
    user: 'root',
```

```
    password: 'user',
    database: 'formdb'
});

module.exports = pool.promise(); // allows async/await
```

Think of this as a **telephone line** to the database.

Step 5: Model

Create file: models/userModel.js

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

const User = {
    create: async (name, email) => {
        const [result] = await db.execute(
            'INSERT INTO users (name, email) VALUES (?, ?)',
            [name, email]
        );
        return result;
    },
    getAll: async () => {
        const [rows] = await db.execute('SELECT * FROM users ORDER BY
id DESC');
        return rows;
    }
};

module.exports = User;
```

- Models are like **library books** → store & fetch data.
- ? prevents hackers from messing with your database (**SQL injection**).

Step 6: Controller

Create file: controllers/userController.js

```
const User = require('../models/userModel');

const createUser = async (req, res) => {
    try {
        const { name, email } = req.body;
        if (!name || !email) {
            return res.status(400).json({ status: 'error', message:
'All fields required' });
        }

        const result = await User.create(name, email);
        res.json({ status: 'success', message: 'User added', id:
result.insertId });
    } catch (err) {
        console.error(err);
        res.status(500).json({ status: 'error', message: 'Server
error' });
    }
};

const getUsers = async (req, res) => {
    try {
        const users = await User.getAll();
        res.json(users);
    } catch (err) {
        console.error(err);
        res.status(500).json({ status: 'error', message: 'Server
error' });
    }
};

module.exports = { createUser, getUsers };
```

- Controllers are like **teachers** → they take data from students (frontend) and decide what to do with it.

Step 7: Routes

Create file: routes/userRoutes.js

```
const express = require('express');
const router = express.Router();
```

```
const { createUser, getUsers } =
require('../controllers/userController');

router.post('/submit', createUser); // URL: /api/submit
router.get('/all', getUsers);       // URL: /api/all

module.exports = router;
```

- Routes are like **doors in a school** → `/submit` or `/all`.

Step 8: Server

Create `server.js`:

```
const express = require('express');
const bodyParser = require('body-parser');
const cors = require('cors');
const path = require('path');

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

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

// Middleware
app.use(cors());
app.use(bodyParser.json());
app.use(express.static(path.join(__dirname, 'public')));

// API routes
app.use('/api', userRoutes);

// Serve frontend
app.get('/', (req, res) => {
    res.sendFile(path.join(__dirname, 'index.html'));
});

app.listen(PORT, () => {
    console.log(`Server running at http://localhost:${PORT}`);
});
```

- Middleware = helpers that sit **between browser & server.**
- `express.static` → lets you serve CSS/JS files.
- `app.use('/api', ...)` → all API URLs start with `/api`.

Step 9: Frontend Form

Create `index.html`:

```
<!DOCTYPE html>
<html>
<head>
    <title>Submit Form</title>
</head>
<body>
    <h2>Submit Your Details</h2>
    <form id="userForm">
        <label>Name:</label><br>
        <input type="text" name="name" required><br><br>

        <label>Email:</label><br>
        <input type="email" name="email" required><br><br>

        <button type="submit">Submit</button>
    </form>

    <p id="response"></p>

    <script>
        const form = document.getElementById('userForm');
        form.addEventListener('submit', async (e) => {
            e.preventDefault();
            const formData = Object.fromEntries(new
FormData(form).entries());

            const res = await fetch('/api/submit', {
                method: 'POST',
                headers: { 'Content-Type': 'application/json' },
                body: JSON.stringify(formData)
            });

            const result = await res.json();
```

```
            document.getElementById('response').innerText =
result.message;
            form.reset();
        });
    </script>
</body>
</html>
```

- This is what the **student sees**.
- Uses **JavaScript fetch** to send data to the backend.

Step 10: Run the Project

```
node server.js
```

Open browser:

```
http://localhost:3000
```

- Fill Name and Email → click Submit → stored in MySQL.
- Open MySQL to see inserted data:

```
USE formdb;
SELECT * FROM users;
```

Step 11: Concepts Recap for Students

Concept	Explanation for 15-year-old
Node.js	JavaScript that runs on server, not browser.
Express	Makes building websites/APIs easy.
MySQL	Stores data (like a notebook).
MVC Pattern	Separates project into: Models (data), Controllers (logic), Routes (URLs).

Auto-increment	Gives each new user a unique ID automatically.
Timestamp	Remembers when user submitted.
Fetch API	Lets frontend talk to backend easily.