**Step-by-Step Guide**

**1. Setting Up the Project**

**What are we doing?**

We are creating a folder for our project and setting up the tools needed to start building.

**Steps:**

1. Create a new folder:

mkdir student-manager

cd student-manager

1. Initialize the project:

npm init -y

* + **Why:** This sets up a file to manage the tools we will install.

1. Install the required tools:

npm install express mysql body-parser cors nodemon

* + **What these tools do:**
    - **Express:** Helps us create the app.
    - **MySQL:** Helps us connect to the database.
    - **Body-Parser:** Helps us handle incoming data (like form submissions).
    - **CORS:** Allows our app to interact with other systems safely.
    - **Nodemon:** Restarts our server automatically when we make changes.

**1. app.use(cors());**

**What It Does:**

* **CORS** stands for **Cross-Origin Resource Sharing**.
* By default, web browsers block requests made from one origin (domain) to another for security reasons. For example:
  + If your frontend (e.g., http://localhost:3000) tries to access your backend (e.g., http://localhost:5000), the browser might block this as a "cross-origin" request.
* **cors()** is middleware that allows your server to handle these cross-origin requests by including the proper headers.

**Why It's Important:**

* It allows your frontend (e.g., React, Angular) to communicate with your backend (Express) even if they are hosted on different domains or ports.
* Without it, the browser will block requests, and you'll see an error like:
  + **"Access to XMLHttpRequest at 'http://localhost:5000/students' from origin 'http://localhost:3000' has been blocked by CORS policy."**

**What Happens Without It:**

* Your frontend cannot fetch data or interact with the backend.
* This is especially critical during development when frontend and backend are often running on different ports.

**2. app.use(bodyParser.json());**

**What It Does:**

* **Body-Parser** is middleware that helps your app understand and work with incoming JSON data.
* In a typical web app, data is sent from the frontend to the backend as part of the request body (e.g., when submitting a form or sending data using fetch or axios).
* **bodyParser.json()** parses this data and converts it into a format (JavaScript object) that your app can easily use.

**Why It's Important:**

* By default, Express doesn’t know how to handle JSON in the request body. Without parsing it, your app won’t understand the data.
* For example, if your frontend sends this data:

json

Copy code

{

"name": "Ismail",

"email": "ismail@gmail.com",

"grade": "A"

}

You can access it in your backend as req.body.name, req.body.email, etc., thanks to bodyParser.json().

**What Happens Without It:**

* Without this middleware, req.body will be undefined.
* Any attempt to access data from the request body will fail, and your app won’t be able to process user input or send it to the database.
* You’ll likely see errors or unexpected behavior.

**2. Setting Up the Database**

**What are we doing?**

We are creating a database and a table to store information about students.

**Steps:**

1. Open MySQL (or a tool like phpMyAdmin).
2. Create a database:

CREATE DATABASE student\_db;

* + **Why:** This is the space where all our app’s data will be stored.

1. Use the database:

USE student\_db;

CREATE TABLE students (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100),

email VARCHAR(100),

grade VARCHAR(10)

);

* + **What this does:** Creates a table with:
    - **id:** A unique number for each student.
    - **name:** The student’s name.
    - **email:** The student’s email address.
    - **grade:** The student’s grade (e.g., A, B, C).

1. Insert a student (example):

INSERT INTO students (name, email, grade) VALUES ('Ismail', 'ismail@gmail.com', 'A');

* + **What this does:** Adds a new student record with the name "Ismail", email "ismail@gmail.com", and grade "A".

**3. Writing the Server Code**

**What are we doing?**

We are creating a server that:

* Connects to the database.
* Sends and receives data.

**Steps:**

1. Create a file called index.js:

touch index.js

1. Write the basic server code:

const express = require('express');

const mysql = require('mysql');

const bodyParser = require('body-parser');

const cors = require('cors');

const app = express();

const PORT = 5000;

app.use(cors());

app.use(bodyParser.json());

const db = mysql.createConnection({

host: 'localhost',

user: 'root',

password: '',

database: 'student\_db',

});

db.connect(err => {

if (err) {

console.log('Error connecting to database:', err);

return;

}

console.log('Connected to database');

});

app.listen(PORT, () => {

console.log(`Server running at http://localhost:${PORT}`);

});

* + **What this does:**
    - Starts the server so we can handle requests.
    - Connects to the student\_db database.

**4. CRUD Operations**

**1. Create (Add a Student)**

**What are we doing?** We are teaching the app how to add a new student to the database.

**Code:**

app.post('/students', (req, res) => {

const { name, email, grade } = req.body;

const sql = `INSERT INTO students (name, email, grade) VALUES ('${name}', '${email}', '${grade}')`;

db.query(sql, (err) => {

if (err) {

res.status(500).send('Error adding student');

return;

}

res.send('Student added!');

});

});

* **What this does:**
  + Takes the name, email, and grade from the user.
  + Adds the information to the database.

**2. Read (View Students)**

**What are we doing?** We are teaching the app how to display all students in the database.

**Code:**

app.get('/students', (req, res) => {

const sql = 'SELECT \* FROM students';

db.query(sql, (err, results) => {

if (err) {

res.status(500).send('Error fetching students');

return;

}

res.json(results);

});

});

* **What this does:**
  + Retrieves all students from the database.
  + Sends the data back to the user.

**3. Update (Edit a Student)**

**What are we doing?** We are teaching the app how to update a student’s details.

**Code:**

app.put('/students/:id', (req, res) => {

const { id } = req.params;

const { name, email, grade } = req.body;

const sql = `UPDATE students SET name = '${name}', email = '${email}', grade = '${grade}' WHERE id = ${id}`;

db.query(sql, (err) => {

if (err) {

res.status(500).send('Error updating student');

return;

}

res.send('Student updated!');

});

});

* **What this does:**
  + Updates the name, email, or grade of the student with the given ID.

**4. Delete (Remove a Student)**

**What are we doing?** We are teaching the app how to delete a student from the database.

**Code:**

app.delete('/students/:id', (req, res) => {

const { id } = req.params;

const sql = `DELETE FROM students WHERE id = ${id}`;

db.query(sql, (err) => {

if (err) {

res.status(500).send('Error deleting student');

return;

}

res.send('Student deleted!');

});

});

* **What this does:**
  + Deletes the student with the given ID from the database.