NODE JS:

**Node.js is a tool that lets you run JavaScript code on the backend (server-side), not just in the browser.**

**📦 Why is Node.js popular for backend?**

✅ Uses JavaScript (easy if you already know it from frontend)  
✅ Fast and efficient (built on Google’s V8 engine)  
✅ Can handle many users at once (non-blocking system)  
✅ Huge community and free libraries (called npm packages)

EXAMPLE:

**DB.JS**

const my sql = require('mysql2'); // Import the mysql2 module to use MySQL database in Node.js

const db = mysql.createConnection({ // Create a connection to the database with given settings

host: 'localhost', // The database server is on the same machine (local)

user: 'root', // Username for the MySQL database (default is 'root')

password: 'root', // Password to connect to the database

database: 'nagaraju', // Name of the specific database to connect to

});

db.connect((err) => { // Try to connect to the database

if (err) { // If there's an error during connection

console.error('❌ Database connection failed:', err); // Print the error message

return; // Stop the function if connection fails

}

console.log('✅ Connected to MySQL database'); // Print success message if connection works

});

module.exports = db; // Export the db connection so it can be used in other files

SERVER.JS

const express = require('express'); // Import Express framework to build the server

const cors = require('cors'); // Import CORS to allow cross-origin requests (frontend ↔ backend)

const bodyParser = require('body-parser'); // Import body-parser to handle form and JSON data from requests

const registerRouter = require('./routes/register'); // Import the router file that handles /register routes

const app = express(); // Create an Express application

const port = 5000; // Define the port number where server will run

// Middleware

app.use(cors()); // Enable CORS for all routes

app.use(bodyParser.json()); // Parse incoming JSON request bodies

app.use(bodyParser.urlencoded({ extended: true })); // Parse form data (URL-encoded), allows nested objects

// Routes

app.use('/api', registerRouter); // All routes in registerRouter will be used under /api path

// Start server

app.listen(port, () => { // Start the server and listen on the given port

console.log(`✅ Server is running on http://localhost:${5000}`); // Show a success message when server starts

});

REGISTER.JS

const express = require("express"); // Load Express framework

const router = express.Router(); // Create a router object to define routes

const db = require("../db"); // Import the database connection (MySQL)

const connection = require('../db'); // Also importing db again (redundant but doesn't break)

const { v4: uuidv4 } = require('uuid'); // Import UUID generator (v4) for unique IDs (not used directly here)

const nodemailer = require('nodemailer'); // Import nodemailer to send emails

// SQL command to create 'users' table if it doesn't already exist

const createTable = `

CREATE TABLE IF NOT EXISTS users (

id INT AUTO\_INCREMENT PRIMARY KEY, // Auto-incrementing numeric ID

uuid VARCHAR(6) NOT NULL, // Custom 6-character unique ID

first\_name VARCHAR(255) NOT NULL, // User's first name

last\_name VARCHAR(255) NOT NULL, // User's last name

user\_name VARCHAR(255) UNIQUE NOT NULL, // Username, must be unique

email VARCHAR(255) UNIQUE NOT NULL, // Email address, must be unique

password VARCHAR(255) NOT NULL // Password (should be hashed ideally)

)

`;

// Run the query to create the table

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

if (err) {

console.error("Error creating table:", err); // If error occurs, log it

} else {

console.log("✅ Users table ready"); // Success message if table is ready

}

});

// ---------- REGISTER ROUTE ----------

router.post("/register", (req, res) => { // POST route to handle user registration

const { first\_name, last\_name, user\_name, email, password } = req.body; // Extract data from request body

// Check if any field is missing

if (!first\_name || !last\_name || !user\_name || !email || !password) {

return res.status(400).json({ message: "All fields are required" }); // Send error if fields are missing

}

// Generate a random 6-character alphanumeric UUID

const generateCustomUUID = () => {

const chars = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789'; // Possible characters

let uuid = '';

for (let i = 0; i < 6; i++) {

uuid += chars.charAt(Math.floor(Math.random() \* chars.length)); // Pick random char 6 times

}

return uuid;

};

const uuid = generateCustomUUID(); // Call the function to create UUID

// Create email transporter using Gmail and App Password

const transporter = nodemailer.createTransport({

service: 'gmail',

auth: {

user: ' nagarajupindi333@gmail.com', // Your Gmail address

pass: 'yaim hkmz rgav mlxg' // Your Gmail App Password (secure key)

}

});

// Email content settings

const mailOptions = {

from: 'nagarajupindi333@gmail.com', // Sender email

to: email, // Recipient email (from registration form)

subject: 'Registration Successful', // Email subject

html: `<h3>Welcome, ${first\_name}!</h3><p>Your registration was successful. Your user ID is: <strong>${uuid}</strong></p>` // Email body

};

// Send the confirmation email

transporter.sendMail(mailOptions, (error, info) => {

if (error) {

console.error('Error sending email:', error); // If email fails, log error

return res.status(500).json({ error: 'Submission saved, but email failed' }); // Inform frontend

} else {

console.log('Email sent:', info.response); // Log success

res.status(201).json({ message: 'Submission successful, confirmation email sent' }); // Respond to frontend

}

});

// Insert user into MySQL database

const insertUser = "INSERT INTO users (uuid, first\_name, last\_name, user\_name, email, password) VALUES (?, ?, ?, ?, ?, ?)";

db.query(insertUser, [uuid, first\_name, last\_name, user\_name, email, password], (err) => {

if (err) {

console.error("Insert error:", err); // If insert fails, log error

return res.status(500).json({ message: "Registration failed" }); // Respond with failure

}

res.status(201).json({ message: "Registration successful" }); // Respond with success

});

});

// ---------- LOGIN ROUTE ----------

router.post('/login', (req, res) => { // POST route for login

const { email, password } = req.body; // Extract email and password from body

if (!email || !password) {

return res.status(400).json({ message: 'Email and password are required.' }); // Validation

}

// Query the user by email

db.query('SELECT \* FROM users WHERE email = ?', [email], (err, results) => {

if (err) {

return res.status(500).json({ message: 'Database error.' }); // If query fails

}

if (results.length === 0) {

return res.status(401).json({ message: 'Invalid email address' }); // If no user found

}

const user = results[0]; // Get the user record

if (user.password !== password) { // Compare passwords (plaintext for now)

return res.status(401).json({ message: 'Invalid password.' }); // If password is wrong

}

res.status(200).json({ message: 'Login successful.', id: user.id }); // Successful login

});

});

// ---------- GET USER BY ID ----------

router.get('/user/:id', (req, res) => { // Route to fetch user profile by ID

const userId = req.params.id; // Get ID from URL

db.query('SELECT id, user\_name AS username, email FROM users WHERE id = ?', [userId], (err, results) => {

if (err) {

console.error('❌ Error fetching user:', err); // Log any DB error

return res.status(500).json({ message: 'Database error.' }); // Respond with DB error

}

if (results.length === 0) {

return res.status(404).json({ message: 'User not found.' }); // If user doesn't exist

}

res.status(200).json(results[0]); // Return the user data

});

});

// ---------- UPDATE USER ----------

router.put('/user/:id', (req, res) => { // PUT route to update user

const userId = req.params.id; // Get user ID

const { username, email, password } = req.body; // Get new data from body

if (!username || !email || !password) {

return res.status(400).json({ message: 'All fields are required.' }); // Validate

}

// Update user in DB

db.query('UPDATE users SET user\_name = ?, email = ?, password = ? WHERE id = ?',

[username, email, password, userId], (err) => {

if (err) {

return res.status(500).json({ message: 'Database error.' }); // Handle error

}

res.status(200).json({ message: 'User updated successfully.' }); // Success

}

);

});

// ---------- DELETE USER ----------

router.delete('/user/:id', (req, res) => { // DELETE route to remove user

const userId = req.params.id; // Get user ID

db.query('DELETE FROM users WHERE id = ?', [userId], (err) => {

if (err) {

return res.status(500).json({ message: 'Database error.' }); // Handle error

}

res.status(200).json({ message: 'User deleted successfully.' }); // Success message

});

});

// Export the router to be used in main app.js

module.exports = router;

REGISTER.HTML

<!DOCTYPE html>

<html>

<head>

<title>Signup Form</title>

</head>

<body style="font-family: Arial, sans-serif; background-color: #f2f2f2; display: flex; justify-content: center; align-items: center; height: 100vh; margin: 0;">

<form id="registerForm" style="background-color: white; padding: 30px; border-radius: 10px; box-shadow: 0 0 10px rgba(0,0,0,0.1); width: 300px;">

<h2 style="text-align: center; margin-top: 0;">Signup</h2>

<label for="first\_name" style="display: block; margin-top: 15px; font-weight: bold;">First Name</label>

<input type="text" id="first\_name" name="first\_name" required style="width: 100%; padding: 8px; margin-top: 5px; border: 1px solid #ccc; border-radius: 5px; box-sizing: border-box;">

<label for="last\_name" style="display: block; margin-top: 15px; font-weight: bold;">Last Name</label>

<input type="text" id="last\_name" name="last\_name" required style="width: 100%; padding: 8px; margin-top: 5px; border: 1px solid #ccc; border-radius: 5px; box-sizing: border-box;">

<label for="user\_name" style="display: block; margin-top: 15px; font-weight: bold;">Username</label>

<input type="text" id="user\_name" name="user\_name" required style="width: 100%; padding: 8px; margin-top: 5px; border: 1px solid #ccc; border-radius: 5px; box-sizing: border-box;">

<label for="email" style="display: block; margin-top: 15px; font-weight: bold;">Email</label>

<input type="email" id="email" name="email" required style="width: 100%; padding: 8px; margin-top: 5px; border: 1px solid #ccc; border-radius: 5px; box-sizing: border-box;">

<label for="password" style="display: block; margin-top: 15px; font-weight: bold;">Password</label>

<input type="password" id="password" name="password" required minlength="6" style="width: 100%; padding: 8px; margin-top: 5px; border: 1px solid #ccc; border-radius: 5px; box-sizing: border-box;">

<label for="confirm\_password" style="display: block; margin-top: 15px; font-weight: bold;">Confirm Password</label>

<input type="password" id="confirm\_password" name="confirm\_password" required style="width: 100%; padding: 8px; margin-top: 5px; border: 1px solid #ccc; border-radius: 5px; box-sizing: border-box;">

<button type="submit" style="margin-top: 20px; width: 100%; padding: 10px; background-color: #007bff; color: white; border: none; border-radius: 5px; cursor: pointer;">Sign Up</button>

<div id="message" style="margin-top: 15px; text-align: center; color: red;"></div>

</form>

<script>

// Attach a listener to the form's submit event

document.getElementById('registerForm').addEventListener('submit', async function(e) {

e.preventDefault(); // Prevents the default form submission behavior (which reloads the page)

// Get the values entered by the user in each input field

const first\_name = document.getElementById('first\_name').value;

const last\_name = document.getElementById('last\_name').value;

const user\_name = document.getElementById('user\_name').value;

const email = document.getElementById('email').value;

const password = document.getElementById('password').value;

const confirm\_password = document.getElementById('confirm\_password').value;

// Get reference to the div that shows error or success messages

const messageDiv = document.getElementById('message');

// Check if any field is empty

if (!first\_name || !last\_name || !user\_name || !email || !password || !confirm\_password) {

messageDiv.textContent = 'All fields are required';

return; // Stop the function here

}

// Check if password and confirm password match

if (password !== confirm\_password) {

messageDiv.textContent = 'Passwords do not match';

return;

}

// Ensure password is long enough

if (password.length < 6) {

messageDiv.textContent = 'Password must be at least 6 characters';

return;

}

try {

// Send the form data to the backend using fetch()

const response = await fetch(`http://localhost:5000/api/register`, {

method: 'POST', // HTTP method for sending data

headers: {

'Content-Type': 'application/json' // Let the backend know we're sending JSON

},

body: JSON.stringify({

// Send only the required fields to the backend

first\_name,

last\_name,

user\_name,

email,

password

// Don't send confirm\_password — it's only used on frontend

})

});

// Wait for the response and parse it as JSON

const data = await response.json();

// If the response was successful (status 200-299)

if (response.ok) {

messageDiv.style.color = 'green'; // Set message color to green

messageDiv.textContent = 'Registration successful!'; // Show success message

document.getElementById('registerForm').reset(); // Clear form fields

// Redirect to login page after 2 seconds

setTimeout(() => {

window.location.href = 'login.html'; // Replace with your actual login page path

}, 2000);

} else {

// If the backend returned an error (e.g., user already exists)

messageDiv.textContent = data.message || 'Registration failed';

}

} catch (error) {

// If fetch itself failed (e.g., server is down)

console.error('Error:', error);

messageDiv.textContent = 'An error occurred. Please try again.';

}

});

</script>

</body>

</html>

LOGIN.HTML

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Login Page</title>

</head>

<body style="margin: 0; font-family: Arial, sans-serif; background-color: #e8f0fe; display: flex; justify-content: center; align-items: center; height: 100vh;">

<!-- Login container -->

<div style="background-color: #ffffff; padding: 40px; border-radius: 10px; box-shadow: 0 0 10px rgba(0, 0, 0, 0.1); width: 100%; max-width: 400px;">

<h2 style="text-align: center; color: #333; margin-bottom: 30px;">Login to Your Account</h2>

<!-- Login form -->

<form id="loginForm">

<label for="email" style="display: block; margin-bottom: 8px; color: #333;">Email:</label>

<input type="email" id="email" name="email" required

style="width: 100%; padding: 10px; margin-bottom: 20px; border: 1px solid #ccc; border-radius: 5px;">

<label for="password" style="display: block; margin-bottom: 8px; color: #333;">Password:</label>

<input type="password" id="password" name="password" required

style="width: 100%; padding: 10px; margin-bottom: 30px; border: 1px solid #ccc; border-radius: 5px;">

<button type="submit"

style="width: 100%; background-color: #4CAF50; color: white; padding: 12px; border: none; border-radius: 5px; font-size: 16px; cursor: pointer;">

Login

</button>

</form>

<!-- Message display area -->

<div id="message" style="margin-top: 15px; text-align: center; color: red;"></div>

<!-- Redirect to register -->

<p style="text-align: center; margin-top: 20px;">

Don't have an account?

<a href="/register" style="color: #4CAF50; text-decoration: none;">Register here</a>

</p>

</div>

<!-- Backend interaction and validation logic -->

<script>

// Add event listener to the login form submission

document.getElementById('loginForm').addEventListener('submit', async function(e) {

e.preventDefault(); // Prevent default form behavior (page refresh)

// Get email and password values from input fields

const email = document.getElementById('email').value;

const password = document.getElementById('password').value;

// Reference to the message display div

const messageDiv = document.getElementById('message');

// Validate empty fields

if (!email || !password) {

messageDiv.textContent = 'Both email and password are required';

return;

}

// Disable the login button and show a loading message

const submitBtn = e.target.querySelector('button[type="submit"]');

submitBtn.disabled = true;

submitBtn.textContent = 'Logging in...';

try {

// Make POST request to backend login API

const response = await fetch('http://localhost:5000/api/login', {

method: 'POST', // HTTP method

headers: {

'Content-Type': 'application/json' // Indicate JSON format

},

body: JSON.stringify({ email, password }) // Send email and password to backend

});

// Parse the JSON response from backend

const data = await response.json();

if (response.ok) {

// ✅ If login is successful

messageDiv.style.color = 'green';

messageDiv.textContent = 'Login successful!';

// ✅ Save user ID to localStorage

if (data.id) {

localStorage.setItem('userId', data.id);

}

// ✅ Optionally save token if available

if (data.token) {

localStorage.setItem('token', data.token);

}

// ✅ Optionally save entire user object

if (data.user) {

localStorage.setItem('user', JSON.stringify(data.user));

}

// Redirect to profile page after 1 second

setTimeout(() => {

window.location.href = 'profile.html'; // Change to your desired page

}, 1000);

} else {

// ❌ If credentials are incorrect or other error

messageDiv.style.color = 'red';

messageDiv.textContent = data.message || 'Login failed. Please check your credentials.';

}

} catch (error) {

// ❌ If network/server error occurs

console.error('Error:', error);

messageDiv.style.color = 'red';

messageDiv.textContent = 'An error occurred. Please try again.';

} finally {

// Re-enable the login button and reset text

submitBtn.disabled = false;

submitBtn.textContent = 'Login';

}

});

</script>

</body>

</html>

PROFILE.HTML

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>User Profile</title>

</head>

<body style="background: #f4f4f4; font-family: Arial, sans-serif;">

<div style="width: 400px; margin: 60px auto; background: #fff; padding: 30px 25px; border-radius: 8px; box-shadow: 0 2px 8px rgba(0,0,0,0.1);">

<h2 style="text-align: center; margin-bottom: 25px; color: #333;">User Profile</h2>

<form id="profileForm">

<div style="margin-bottom: 18px;">

<label for="username" style="display: block; margin-bottom: 6px; color: #555;">Username</label>

<input type="text" id="username" name="username" required style="width: 100%; padding: 10px 8px; border: 1px solid #ccc; border-radius: 4px; font-size: 15px;">

</div>

<div style="margin-bottom: 18px;">

<label for="email" style="display: block; margin-bottom: 6px; color: #555;">Email</label>

<input type="email" id="email" name="email" required style="width: 100%; padding: 10px 8px; border: 1px solid #ccc; border-radius: 4px; font-size: 15px;">

</div>

<div style="margin-bottom: 18px;">

<label for="password" style="display: block; margin-bottom: 6px; color: #555;">Password</label>

<input type="password" id="password" name="password" required style="width: 100%; padding: 10px 8px; border: 1px solid #ccc; border-radius: 4px; font-size: 15px;">

</div>

<button type="submit" style="width: 100%; padding: 10px; background: #0078d7; color: #fff; border: none; border-radius: 4px; font-size: 16px; cursor: pointer; margin-top: 10px;">Update Profile</button>

<button type="button" id="deleteBtn" style="width: 100%; padding: 10px; background: #d8000c; color: #fff; border: none; border-radius: 4px; font-size: 16px; cursor: pointer; margin-top: 8px;">Delete Account</button>

<div id="message" style="margin-top: 15px; text-align: center; color: #d8000c;"></div>

</form></div>

<script>

// Step 1: Get the user ID from localStorage

const userId = localStorage.getItem('userId') || '';

// Step 2: Fetch the user's profile using their ID

async function fetchProfile() {

if (!userId) {

document.getElementById('message').textContent = 'No user ID found.';

return;

}

try {

const res = await fetch(`http://localhost:5000/api/user/${userId}`);

const data = await res.json();

if (res.ok) {

// Fill form fields with user data

document.getElementById('username').value = data.username;

document.getElementById('email').value = data.email;

} else {

document.getElementById('message').textContent = data.message || 'Failed to load profile.';

}

} catch {

document.getElementById('message').textContent = 'Error connecting to server.';

}

}

// Call the fetch function when the page loads

fetchProfile();

// Step 3: Handle profile update on form submission

document.getElementById('profileForm').addEventListener('submit', async function(e) {

e.preventDefault(); // Prevent page reload

const username = document.getElementById('username').value;

const email = document.getElementById('email').value;

const password = document.getElementById('password').value;

const messageDiv = document.getElementById('message');

messageDiv.textContent = ''; // Clear message

try {

const res = await fetch(`http://localhost:5000/api/user/${userId}`, {

method: 'PUT', // Update operation

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({ username, email, password }) // Send updated data

});

const data = await res.json();

if (res.ok) {

messageDiv.style.color = 'green';

messageDiv.textContent = 'Profile updated successfully!';

} else {

messageDiv.style.color = '#d8000c';

messageDiv.textContent = data.message || 'Update failed.';

}

} catch {

messageDiv.style.color = '#d8000c';

messageDiv.textContent = 'Error connecting to server.';

}

});

// Step 4: Handle account deletion

document.getElementById('deleteBtn').addEventListener('click', async function() {

if (!confirm('Are you sure you want to delete your account?')) return;

const messageDiv = document.getElementById('message');

messageDiv.textContent = ''; // Clear message

try {

const res = await fetch(`http://localhost:5000/api/user/${userId}`, {

method: 'DELETE' // Delete operation

});

const data = await res.json();

if (res.ok) {

messageDiv.style.color = 'green';

messageDiv.textContent = 'Account deleted successfully!';

// Optional: Redirect or clear localStorage

} else {

messageDiv.style.color = '#d8000c';

messageDiv.textContent = data.message || 'Delete failed.';

}

} catch {

messageDiv.style.color = '#d8000c';

messageDiv.textContent = 'Error connecting to server.';

}

});

</script>

</body>

</html>

ERROR CODES EXPLAINATION

| **Code** | **Meaning** | **When to Use** |
| --- | --- | --- |
| **200 OK** | Success | Data fetched or action completed successfully. |
| **201 Created** | Resource created | User registered successfully, new data saved. |
| **204 No Content** | Success, no response body | For successful deletion when no data is returned. |
| **400 Bad Request** | Client error | Missing fields, invalid input. |
| **401 Unauthorized** | Authentication failed | Invalid token, not logged in. |
| **403 Forbidden** | Access denied | Logged in but not allowed to perform action. |
| **404 Not Found** | Resource missing | User ID or data not found. |
| **409 Conflict** | Duplicate conflict | Email already exists, username taken. |
| **422 Unprocessable Entity** | Validation failed | Data format correct, but content invalid. |
| **500 Internal Server Error** | Server crash | Database errors, coding bugs, etc. |
| **Keyword** | Meaning / Use | Example |
| **require()** | Imports external modules (Node.js built-in or custom). | const express = require('express'); |
| **module.exports** | Exports code from one file to use in another. | module.exports = router; |
| **express()** | Initializes the Express framework. | const app = express(); |
| **app.use()** | Applies middleware (e.g., JSON parser, routes). | app.use(express.json()); |
| **router** | Handles grouped routes (GET, POST, etc). | const router = express.Router(); |
| **req** | Request object: holds data sent by client. | req.body.email |
| **res** | Response object: used to send data back to client. | res.send('Hello'); |
| **res.status()** | Sets HTTP status code for response. | res.status(200).json(...) |
| **res.json()** | Sends JSON-formatted response to client. | res.json({ message: 'OK' }); |
| **async / await** | Handles asynchronous operations (like DB queries) more cleanly than callbacks. | await db.query(...) |
| **try...catch** | Handles errors in async code safely. | try { ... } catch (err) { ... } |
| **POST** | HTTP method for creating new data. | app.post('/register', ...) |
| **GET** | HTTP method for reading/fetching data. | app.get('/user/:id', ...) |
| **PUT** | HTTP method for updating existing data. | app.put('/user/:id', ...) |
| **DELETE** | HTTP method for deleting data. | app.delete('/user/:id', ...) |
| **params** | Route parameters (e.g., ID in /user/:id). | req.params.id |
| **body** | Holds data sent from client in POST/PUT request. | req.body.username |
| **status codes** | HTTP response codes indicating success/failure. | res.status(404) |
| **fetch()** | Used in frontend to call backend API. | fetch('/api/login') |
| **JSON.stringify()** | Converts JS object to JSON string. | JSON.stringify({email, password}) |
| **JSON.parse()** | Converts JSON string to JS object. | JSON.parse(data) |
| **localhost** | Refers to the current machine (your local server). | http://localhost:5000 |
| **CORS** | Allows/disallows frontend apps from accessing the backend. | app.use(cors()); |
| **middleware** | Functions that run before the final route handler (e.g., auth, logger). | app.use(express.json()) |
| **database (DB)** | External storage like MySQL, MongoDB, PostgreSQL, etc. | mysql.createConnection(...) |
| **JWT (token)** | Used for user authentication (JSON Web Token). | jwt.sign({id}, secret) |
| **bcrypt** | Library for hashing passwords securely. | bcrypt.hash(password) |