



9530

**ST.MOTHER THERESA ENGINEERING COLLEGE
COMPUTER SCIENCE AND ENGINEERING**

**NM-ID:D2542380FBEB0F6
AA3A4B2106794562E**

REG NO:953023104135

DATE: 22-09-2025

Completed the project named as

Phase-3

Node.js Backend for Contact Form

**SUBMITTED BY
S. VIJI**

PH NO: 9843768709

Node.js Backend for Contact Form

1. Project Setup

Explanation:

Before starting development, the project environment must be set up. Node.js provides a runtime environment for JavaScript on the server side, and npm (Node Package Manager) helps in installing required packages. Express.js is a popular framework used for building REST APIs in Node.js because it simplifies request handling, routing, and middleware integration.

In this step, the folder structure is also organized:

- **server.js** → main entry file
- **routes/** → contains route definitions
- **controllers/** → business logic for APIs
- **models/** → database schemas

This modular design makes the backend scalable and easy to maintain.

Code Example:

```
const express = require("express"); const
bodyParser = require("body-parser"); const
app = express(); const PORT = 3000;
```

```
app.use(bodyParser.json());
```

```
// test route app.get("/", (req,
res) => {    res.send("Node.js
Backend Setup
Successful!");
});
```

```
app.listen(PORT, () => {
```

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

Output (CLI):

Server running at http://localhost:3000

Browser Output:

Node.js Backend Setup Successful!

2. Core Features Implementation

Explanation:

The main functionality of the backend is to handle incoming requests from the frontend contact form. A contact form usually collects

Name, Email, and Message. The backend must:

- **Receive Data** → Accept form submissions via POST request.
- **Validate Input** → Ensure fields are not empty and email format is valid.
- **Respond to Client** → Return success or error messages.
 - **Optional Action** → Send an email notification to admin using **Nodemailer** or store data in a database.

This ensures that every form submission is processed correctly and securely.

Code Example:

```
app.post("/contact", (req, res) => {  const  
  { name, email, message } = req.body;  
  
  if (!name || !email || !message) {  
    return res.status(400).json({ error: "All  
fields are required" });  
  }
```

```
res.status(200).json({
  success: true,    message: "Form
submitted successfully!",
  data: { name, email, message }
});
});
```

Sample Input (Postman – POST /contact):

```
{
  "name": "John Doe",
  "email": "john@example.com",
  "message": "Hello, I need some help!"
}
```

Output:

```
{
  "success": true,
  "message": "Form submitted successfully!",
  "data": {
    "name": "John Doe",
    "email": "john@example.com",
    "message": "Hello, I need some help!"
  }
}
```

3. Data Storage (Database Integration)

Explanation:

Storing data is important for future reference. Instead of only returning responses, we can store the form data in a database. MongoDB (NoSQL) is commonly used because it integrates smoothly with Node.js.

Steps:

1. Connect Node.js to MongoDB using Mongoose.
2. Create a schema for contact form data.
3. Save incoming submissions as documents in MongoDB.

4. Retrieve submissions later for analysis or admin viewing.

Code Example:

```
const mongoose = require("mongoose");

// Connect to MongoDB
mongoose.connect("mongodb://127.0.0.1:27017/contactFormDB", {  useUrlParser: true,
useUnifiedTopology: true
});

// Schema
const contactSchema = new mongoose.Schema({
name: String,  email: String,  message:
String
});

const Contact = mongoose.model("Contact",
contactSchema);

// Save data in DB app.post("/saveContact",
async (req, res) => {  const { name, email,
message } = req.body;  const newContact = new
Contact({ name, email, message });

  try {
    await newContact.save();
    res.json({ success: true, message: "Data
saved in DB!" });  } catch (err) {
    res.status(500).json({ error: "Failed to
save data" });
  }
});
```

Output (Saved in MongoDB):

```
{
  "_id": "65123456789abc",
```

```
"name": "Alice",
"email": "alice@example.com",
"message": "Hi, please contact me back!",
"__v": 0
}
```

4. Testing Core Features

Explanation:

Testing ensures the system works as expected before deployment. Without testing, the backend may fail under real-world use cases. There are **two main types of testing**:

- **Manual Testing** → Using Postman to send requests and check responses.
- **Automated Testing** → Writing test scripts with frameworks like Jest or Mocha to verify functionality continuously.

Automated tests save time and prevent human errors, especially in large projects.

Code Example (Jest + Supertest):

```
const request = require("supertest"); const
app = require("./server"); // assuming app
exported
```

```
test("POST /contact should return success",
  async () => {    const res = await
    request(app).post("/contact").send({
      name: "Sam",      email: "sam@example.com",
      message: "Testing API"
    });
    expect(res.statusCode).toBe(200);
    expect(res.body.success).toBe(true);
  });
```

Output (running `npm test`):

```
PASS  ./server.test.js
✓ POST /contact should return success (50ms)
```

5. Version Control (GitHub)

Explanation:

Version control tracks every change in the project, making collaboration and debugging easier. Git allows developers to commit updates, switch between branches, and roll back to older versions if needed. Hosting on GitHub ensures the code is safely stored in the cloud and accessible from anywhere. **Commands Example:**

```
git init
git add .
git commit -m "Initial commit - Contact form backend"
git branch -M main
git remote add origin https://github.com/username/contact-formbackend.git
git push -u origin main
```

Output:

```
Enumerating objects: 10, done.
Counting objects: 100% (10/10), done.
Compressing objects: 100% (8/8), done.
Writing objects: 100% (10/10), done. To
https://github.com/username/contact-formbackend.git
* [new branch] main -> main
```

GITHUB URL:

<https://github.com/vijiug23cs-bit/Smtecmibm-node-js-backend-for-contact-form.git>
