9530

ST.MOTHER THERESA ENGINEERING COLLEGE
COMPUTER SCIENCE AND ENGINEERING

NM-ID:21138DB95511CA3A3
4684679AB9A32

REG NO:953023104109
DATE:14-09-2025

Completed the project named as Phase-1

Node.js Backend for Contact Form

SUBMITTED BY
S.SELVA KAVIYA

PH NO: 7094816164

IBM-FE-Node.js Backend for Contact Form

Project Title:

IBM-FE-Node.js Backend for Contact Form

Technologies Used:

Frontend: HTML, CSS, JavaScript

Backend: Node.js, Express.js

Database: MongoDB

Validation: Regex and Express Validator

Optional Tools: Bootstrap, Postman

Problem Statement:

In many websites, contact forms are used to collect messages and inquiries from users. However,in most cases: there’s no real-time validation of user input, backend servers are exposed to spamor invalid data, users don’t get confirmation whether their message is sent or not, and many formslack security and proper storage of form data.

Objective:

To design and develop a full-stack contact form system with: real-time frontend validation,backend API using Node.js and Express.js, data storage using MongoDB, and input sanitization and validation on both client and server sides.

AIM 1:

To create a responsive and interactive contact form UI using HTML, CSS, and JavaScript thatvalidates: Name, Email, and Message.

AIM 2:

To build a secure and efficient Node.js backend that: Receives validated input from the frontend,Saves form data to MongoDB, Sends success or error responses, Optionally sends email confirmations.

Frontend Code (HTML + JavaScript):

```
<form id="contactForm">
<input type="text" id="name" placeholder="Your Name" required />
<span class="error" id="nameError">Name is required.</span>
<input type="email" id="email" placeholder="Your Email" required />
<span class="error" id="emailError">Valid email required.</span>
<textarea id="message" placeholder="Your Message" required></textarea>
<span class="error" id="messageError">Message cannot be empty.</span>
<button type="submit">Send</button>
</form>
<script>
document.getElementById('contactForm').addEventListener('submit', function(e) {
 let valid = true;
 const name = document.getElementById('name').value.trim();
 const email = document.getElementById('email').value;
 const message = document.getElementById('message').value.trim();
```

```
 if (!name) {
 document.getElementById('nameError').style.display = 'block';
 valid = false;
 }
 const emailRegex = /^[^ ]+@[^ ]+\.[a-z]{2,3}$/;
 if (!emailRegex.test(email)) {
 document.getElementById('emailError').style.display = 'block';
 valid = false;
 }
 if (!message) {
 document.getElementById('messageError').style.display = 'block';
 valid = false;
 }
 if (!valid) e.preventDefault();
 });
</script>
```

Backend Code (Node.js + Express):

```
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');
const app = express();
app.use(cors());
app.use(express.json());
mongoose.connect('mongodb://localhost:27017/contactFormDB');
```

```
const ContactSchema = new mongoose.Schema({
 name: String,
 email: String,
 message: String
});
const Contact = mongoose.model('Contact', ContactSchema);
app.post('/api/contact', async (req, res) => {
 const { name, email, message } = req.body;
 if (!name || !email || !message) {
 return res.status(400).json({ error: 'All fields are required' });
 }
 try {
 const newContact = new Contact({ name, email, message });
 await newContact.save();
 res.status(200).json({ success: true, message: 'Message received' });
 } catch (err) {
 res.status(500).json({ error: 'Server error' });
 }
});
app.listen(3000, () => console.log('Server running on port 3000'));
```

Output Page:

Sample Input (Form Submission):

- Name: John Doe

- Email: john@example.com

- Message: Hello, this is a test message.

Frontend Output:

- If valid → "Form submitted successfully!"

- If invalid → "Name is required" / "Valid email required" / "Message cannot be empty"

Backend Output (API Response):

{

"success": true,

"message": "Message received"

}

Database Entry (MongoDB):

{

"_id": "650cbd23f8a7d8123456abcd",

"name": "John Doe",

"email": "john@example.com",

"message": "Hello, this is a test message."

}

Project Hurdles:

1. Validation Conflicts
2. CORS Issues
3. MongoDB Setup
4. Security
5. UI Feedback

Architecture & Tech Stack:

Frontend: HTML, CSS, JS

Backend: Node.js, Express

Database: MongoDB

Validation: Regex, Express Validator

Optional Tool: Bootstrap, Nodemailer

Data Handling Approach:

1. Frontend checks input with JS and regex
2. Valid data sent to backend via /api/contact
3. Server stores the data into MongoDB
4. Response sent back to user
5. (Optional) Send confirmation email

Users & Stakeholders:

Users: Website visitors submitting contact forms

Stakeholders: Website owners, administrators, and developers who need to store, view, and respond to messages

User Stories:

As a visitor, I want to submit my name, email, and message through a contact form.

As a website owner, I want to receive and securely store messages submitted through the form.

As a developer, I want a simple REST API to handle form submissions and allow future integrations (like email notifications).

MVP Features:

API endpoint to receive contact form submissions (POST /api/contact)Validation of input fields (name, email, message)

Save submissions into a database (e.g., MongoDB / PostgreSQL)

Optional: Send email notification to admin on new submission

Wireframes / API Endpoint List

Frontend Wireframe (contact page):

Input fields: Name, Email, Message

Submit button

API Endpoints:

POST /api/contact → Store new submission

GET /api/contact (admin only) → View all submissions

Acceptance Criteria

API must accept valid contact form data and reject invalid inputs.

Messages must be stored securely in the database.

Admin should be able to retrieve messages with proper authentication.

Optional: Email notifications are sent for each new submission.

Component / Module Diagram:

Frontend: UI, form validation, AJAX request

Validation Module: Input sanitization and pattern checks

Backend: Express server with REST API

Database: MongoDB collection for storing messages

Conclusion:

The IBM-FE-Node.js backend project for the Contact Form successfully demonstrates the use of full-stack technologies. It ensures secure and validated data collection from users, improves form usability with real-time feedback, and stores submissions safely in a database.