9530

ST.MOTHER THERESA ENGINEERING COLLEGE
COMPUTER SCIENCE AND ENGINEERING

NM-ID:CE36C6739E08215AA
2C45AB3E397FCB7

REG NO:953023104126

DATE: 29-09-2025

Completed the project named as Phase-4

Node.js Backend for Contact Form

SUBMITTED BY
S. SUJITHA

PH NO: 9042427346

NODE.JS BACKEND FOR CONTACT FORM

1. Additional Features

Content:
Adding new functionalities to improve the backend of the contact form.

Explanation:
Beyond basic form submissions, additional features may include:

- Email notifications to admin when a new message is received.
- Storing messages in a database (MongoDB/MySQL).
- Adding spam protection with reCAPTCHA.

Example:
When a user submits a message through the contact form, an email notification is sent automatically to the admin.

Program (Node.js + Nodemailer):

```
const express = require("express");
const nodemailer = require("nodemailer");
const bodyParser = require("body-parser");

const app = express();
app.use(bodyParser.json());

app.post("/contact", async (req, res) => {
    const { name, email, message } = req.body;

    // Configure mail transporter
    let transporter = nodemailer.createTransport({
        service: "gmail",
        auth: {
            user: "yourgmail@gmail.com",
            pass: "yourpassword"
        }
    });

    // Email options
    let mailOptions = {
        from: email,
        to: "admin@gmail.com",
```

```
            subject: `New Message from ${name}`,
            text: message
        };

        // Send email
        try {
            await transporter.sendMail(mailOptions);
          res.status(200).send("Message sent successfully!");
      } catch (error) {
          res.status(500).send("Error sending message");
      }
});

app.listen(5000, () => console.log("Server running on port 5000"));
```

Output:

- If the user submits the form → **"Message sent successfully!"**
- If an error occurs → **"Error sending message"**

---

2. UI/UX Improvements

Content:
Improving user interface and experience for the contact form.

Explanation:
UI/UX focuses on making the form simple, responsive, and visually appealing. This includes:

- Adding validation (name, email, and message must be filled).
- Clear success/error messages.
- Mobile-friendly layout.

Example:
If a user tries to submit without entering an email, the form shows → "Email is required."

Program (Frontend HTML + JavaScript):

```
<form id="contactForm">
  <input type="text" id="name" placeholder="Your
Name" required><br>
  <input type="email" id="email" placeholder="Your
Email" required><br>
  <textarea id="message" placeholder="Your Message"
required></textarea><br>
  <button type="submit">Send</button>
```

```
</form>

<script>
document.getElementById("contactForm").addEventListen
er("submit", function(e){
    e.preventDefault();
    alert("Form submitted successfully!");
});
</script>
```

Output:

- If the form is correctly filled → **Alert: "Form submitted successfully!"**
- If fields are empty → HTML5 validation error.

---

3. API Enhancements

Content:

Improving backend API for efficiency and security.

Explanation:

Enhancements include:

- Using validation libraries (Joi/Validator.js).
- Sanitizing inputs to prevent SQL Injection/XSS.
- Adding response formats (JSON).

Example:

If a user sends empty data, the API should return → `{ "error": "All fields are required" }`.

Program (Node.js + Express + Joi):

```
const Joi = require("joi");

app.post("/contact", (req, res) => {
    const schema = Joi.object({
        name: Joi.string().min(3).required(),
        email: Joi.string().email().required(),
        message: Joi.string().min(5).required()
    });

   const { error } = schema.validate(req.body);
    if (error) return res.status(400).json({ error:
error.details[0].message });
```

```
    res.status(200).json({ success: "Message
received" });
});
```

Output:

- Valid input → `{ "success": "Message received" }`
- Invalid input → `{ "error": "\"email\" is required" }`

---

4. Performance & Security Checks

Content:
Ensuring the backend is secure and optimized.

Explanation:

- Use Helmet.js to secure HTTP headers.
- Enable rate limiting to prevent spam.
- Optimize server response time with caching (Redis).

Example:
If one IP sends 100 requests in 1 minute → Block further requests.

Program (Node.js + Helmet + Rate Limit):

```
const helmet = require("helmet");
const rateLimit = require("express-rate-limit");

app.use(helmet());

const limiter = rateLimit({
    windowMs: 1 * 60 * 1000, // 1 minute
    max: 10 // limit each IP to 10 requests per
minute
});

app.use(limiter);
```

Output:

- If user sends normal requests → Works fine.
- If user exceeds limit → **"Too many requests, please try again later."**

---

5. Testing of Enhancements

Content:
Testing all new features before deployment.

Explanation:
- Unit testing with Jest/Mocha.
- API testing with Postman.
- UI testing on multiple devices.

Example:
Run API test with Postman → Check if `/contact` API returns correct JSON response.

Program (Jest test example):
```
test("Contact API should return success message",
async () => {
    const response = await
request(app).post("/contact").send({
        name: "John",
        email: "john@gmail.com",
        message: "Hello"
    });
    expect(response.status).toBe(200);
});
```

Output:
✔ Test passed – API returns success message.

---

6. Deployment (Netlify, Vercel, or Cloud Platform)

Content:
Hosting the project online.

Explanation:
- Frontend → Deploy on **Netlify** or **Vercel**.
- Backend → Deploy on **Render, Railway, or AWS**.

Example:
- URL after deployment: `https://mycontactform.vercel.app`

Steps (Vercel Deployment):
1. Push project to GitHub.

2. Import repo into Vercel.
3. Configure environment variables.
4. Deploy.

Output:
Live project available at → `https://yourproject.vercel.app`
