COLLEGE CODE : 9528

COLLEGE NAME : SCAD COLLEGE OF ENGINEERING AND TECHNOLOGY

DEPARTMENT : COMPUTER SCIENCE AND ENGINEERING

STUDENTNMID : BE96D422034491CBF0B482D5A1BD21C0

Roll no : 952823104145

DATE : 26.09.2025

Completed the project named as:

Phase 2

TECHNOLOGYPROJECTNAME :

NODEJS BACKEND FOR CONTACT FORM

SUBMITTED By,

NAME: C.SELVA NANTHINI

MOBILE NO: 7604999577

IBM-FE-NodeJS Backend for Contact Form

Phase 2: Solution Design & Architecture

Tech stack selection

For the backend, Node.js with Express.js will be used due to its event-driven, non-blocking architecture, which is highly suitable for handling multiple requests efficiently. Express provides lightweight yet powerful routing and middleware capabilities, making it ideal for building REST APIs for the contact form. MongoDB is chosen as the database since it allows for flexible schema design and scalability, enabling easy storage of form submissions without rigid relational constraints. Additionally, IBM Cloud services can be integrated to ensure secure data hosting, monitoring, and scalability with options like IBM Cloud Databases for MongoDB and IBM App ID for authentication

On the frontend, IBM Carbon Design System can be used to ensure consistent UI/UX aligned with IBM standards. React.js (or plain vanilla JS with Carbon components, depending on complexity) can be leveraged for building dynamic and responsive interfaces. For deployment, Docker containers orchestrated via Kubernetes on IBM Cloud will allow seamless CI/CD integration, ensuring smooth updates and rollback capabilities. This combination ensures a highly scalable, maintainable, and enterprise-grade architecture.

UI Structure / API Schema Design

The UI structure will consist of a simple yet structured form with fields such as Name, Email, Subject, and Message. The design will emphasize accessibility, responsiveness, and validation. Client-side validation (using HTML5 constraints or React form handling libraries) ensures that malformed inputs are caught early, while server-side validation ensures security. Error states, success messages, and loading indicators will be incorporated to guide user interactions and improve usability. The frontend will also make API calls to the backend for data submission and retrieval (if admin dashboards are planned).

The API schema will follow RESTful principles. The primary endpoint will be/api/contact for POST requests, where users can submit contact form details. An additional GET /api/contact may be included for admins to fetch stored messages. Request payloads will follow JSON structure, e.g.:

```
{
  "name": "John Doe",
  "email": "john@example.com",
  "subject": "Inquiry about services",
  "message": "Could you provide details about pricing?"
}
```

Responses will be standardized, including status codes, error messages, and data payloads to ensure consistency and ease of integration.

Data Handling Approach

On form submission, the backend will validate the incoming payload using middleware such as express-validator to ensure correctness of email formats, message length, and mandatory fields. After validation, the request will be sanitized to prevent injection attacks before storing in the database. MongoDB will store each form entry as a document with timestamp metadata for easy querying and retrieval. Data encryption can be applied at rest (via IBM Cloud-managed encryption) and in transit (via HTTPS/TLS).

For error handling, centralized error middleware will be implemented to catch and respond with structured error objects. Logs of each request and submission will be maintained using IBM Log Analysis or Winston for traceability. If required, IBM Event Streams (Kafka) can be integrated to handle large volumes of form submissions asynchronously, enabling further workflows such as triggering notifications or integrating with CRM tools. This ensures that the data handling process remains secure, scalable, and aligned with enterprise-grade requirements.

Component/Module Diagram

The system will be structured into well-defined modules. At the frontend, components will include a Form Component, Validation Component, and Notification Component (for success/error alerts). On the backend, the application will be divided into API Routes (handling requests), Controller Layer (business logic), and Model Layer (MongoDB interactions). Middleware components such as authentication (if applicable) and validation will serve as intermediaries between routes and controllers.

This modular approach improves maintainability by isolating responsibilities. For instance, the form submission logic can be modified independently from database operations. Additionally, abstraction layers for database connections allow switching databases (e.g., MongoDB to IBM Db2) with minimal code changes. IBM Cloud services, including monitoring and storage, will sit alongside these components in the overall architecture, ensuring integration into the enterprise ecosystem.

Frontend Components:
- **ContactForm**: Contains input fields and submit button.
- **Notification**: Displays success/error messages.
- **FormInput**: Reusable input component for name/email/message.

Backend Modules:
- **Routes (contact.js)**: Defines API endpoints.
- **Controllers (contactController.js)**: Handles request/response logic.
- **Models (contactModel.js)**: Defines MongoDB schema with Mongoose.
- **Middleware**: Includes input validation, authentication (future), and error handling.

System Layers:
- UI Layer → API Layer → Controller Layer → Database Layer.

Basic Flow Diagram

The basic flow begins with a user opening the contact form UI and entering their details. Once they hit “Submit,” the form data is validated on the client side and then sent via a POST request to the backend API. The backend first passes the data through validation middleware, then routes it to the controller, which processes and stores the data in the MongoDB database. Upon successful storage, the backend responds with a success message, which the frontend displays to the user.

In case of validation errors or failures (e.g., DB downtime), the backend returns an error response, which the frontend presents in a user-friendly format. Additional flows may include notifications sent to admins (via email or IBM Cloud Functions) when new submissions are recorded. The flow diagram highlights this cycle of Input → API → Processing → Database → Response → UI Feedback, ensuring that all key interactions between user, backend, and data storage are accounted for.