COLLEGECODE : 9528
COLLEGENAME : 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 1

TECHNOLOGYPROJECTNAME :

NODEJS BACKEND FOR CONTACT FORM

SUBMITTED By,

NAME: C.SELVA NANTHINI

MOBILE NO: 7604999577

IBM-FE-NodeJS Backend for Contact Form

Project Overview

Thisdocument defines a concise, developer-friendly backend design for a Contact Form API implemented in Node.js. The goal is to deliver a simple, secure, and reusable RESTful service that stores contact records (name, email, phone, message, metadata) and exposes CRUD endpoints. The backend will be easy to integrate with front-end forms (IBM FE or other frameworks), mobile apps, and automation tools.

Problem Statement

Individuals andorganizations need a reliable, central place to collect and manage contact messages submitted through web forms. Typical problems with ad-hoc solutions include data loss when devices fail, inaccurate or duplicate records, lack of validation and security controls for submitted data, and difficulty integrating collected messages with existing workflows (email, ticketing, CRMs). This backend aims to solve these issues by providing a validated, persistent API with predictable behaviour, clear integration points, and extendable features like authentication, rate-limiting, and analytics.

Goals

• Provide a secure RESTful API for storing and retrieving contact messages.

• Validate incoming data to avoid duplicates and malformed records.

• Make the API trivial to integrate with front-end forms and no-code tools.

• Ensure persistence, resilience, and basic performance for expected load.

Users & Stakeholders

PrimaryUsers:

- End Users - people filling the contact form (submitters).
- Administrators - staff who review, reply, or export messages.
- Developers/Integrators - frontend or third party developers who connect to this API.

Secondary Stakeholders

- Organizations that want a lightweight inbox for web leads and enquiries.
- Product teams that need to automate routing of contact messages into ticketing systems.
- Students and developers who will reuse this backend as a learning or prototype system.

Contact Data Model (Suggested)

Acompact schema suitablefor relational or document databases:

Contact { id (uuid), name (string), email (string), phone (string), message (string), source (string - e.g. 'website'), createdAt (ISO8601), updatedAt (ISO8601), tags (array) }.

Validation rules: name required (1-120 chars), email optional but if present must be RFC5322-like, phone optional but must follow E.164 or normalized local format, message required (max 2000 chars).

Security & Privacy Considerations

- UseHTTPS only (TLS). Do not accept plaintextHTTP submissions in production.
- Sanitize all inputs to avoid injection attacks.
- Rate-limit endpoints to prevent spam and abuse.
- Store sensitive fields encrypted at rest if required by policy.
- Add optional authentication for management endpoints (e.g., JWT or API keys).

User Stories

•As a visitor: I want to submit my name, email, phone and message so that the website owner cancontact me.

Acceptance: Valid input stored; duplicates prevented; correct HTTP status codes.

• As an admin: I want to list and filter all submissions so I can respond to requests.

Acceptance: Valid input stored; duplicates prevented; correct HTTP status codes.

• As a developer: I want a stable API with predictable JSON responses so I can connect the front-end reliably.

Acceptance: Valid input stored; duplicates prevented; correct HTTP status codes.

• As an integrator: I want to export or forward submissions (webhook/email) so messages enter our support workflow.

Acceptance: Valid input stored; duplicates prevented; correct HTTP status codes.

• As a security officer: I want rate-limiting and spam protection so the service remains available.

Acceptance: Valid input stored; duplicates prevented; correct HTTP status codes.

MVP Features (Priority)

• **Create Contact (POST /contacts):** Accept validated payload and persist contact.

• **Read Contacts (GET /contacts):** Return paginated list with basic filters (name, email).

• **Read Contact (GET /contacts/{id}):** Return single contact by ID.

• **Update Contact (PUT /contacts/{id}):** Allow editing of non immutable fields with validation.

• **Delete Contact (DELETE /contacts/{id}):** Soft delete or hard delete depending on retention policy.

Nice-to-have

• Webhooks / integrations to forward new messages.

• Admin UI to view, search, tag, and export messages.

• Spam protection (CAPTCHA integration), and analytics dashboards.

Wireframes / API Endpoint List

TheAPIisintentionallysmalland follows REST conventions. Below is a compact endpoint list, example requests/responses, and simple wireframe notes for front-end needs.

Endpoint	Method	
/contacts	POST	Description Create a new contact (returns
/contacts	GET	201 + resource) List contacts (supports ?q=,
/contacts/{id}	GET	?page=, ?limit=) Get contact by ID Update a
/contacts/{id}	PUT	contact by ID Delete a contact by ID Health
/contacts/{id}	DELETE	check endpoint (200)
/health	GET	

Example Request (Create)

```
POST /contacts {"name":"Arun
Kumar","email":"arun.kumar@example.com","phone":"+919876543210","message":"I
need info about your services","source":"website"}
```

Example Response (201 Created)

```
{"id":"c0a8011a-...","name":"Arun
Kumar","email":"arun.kumar@example.com","phone":"+919876543210","message":"I
need info about your services","createdAt":"2025-09-14T12:34:56Z"}
```

Acceptance Criteria & Test Cases

Functional Acceptance:

- The API must support full CRUD with correct HTTP codes (201, 200, 204, 400, 404).
- Input validation: required fields, email and phone format checks.
- Duplicate detection: if the same email+message is received within a short window, treat as duplicate.

Non Functional Acceptance:

- Response times: p95 < 2s under typical load; baseline should handle small production loads.
- Persistence across restarts (use a database or persistent file store).
- Meaningful error messages and proper logging for audit/debug. Sample Test Cases:

TC001: Create valid contact	Expect 201 and returned object (id present).
TC002: Create invalid contact (missing message)	Expect 400 Bad Request
TC003: Fetch non-existing contact	Expect 404 Not Found.
TC004: Delete existing contact	Expect 204 No Content and subsequent GET returns 404.

Operational Notes

- Deployment: containerize with Docker, expose only necessary ports, use environment variables for secrets.
- Monitoring: add a health endpoint, logs, and basic metrics (requests/sec, error rates).
- Backups & retention: implement data export and retention policy (e.g., keep messages for 1 year unless configured).