COLLEGE CODE: 8207

COLLEGE NAME: As-salam College of engineering and technology

DEPARTMENT :B.Tech IT

STUDENTNM-ID:F7FA7C97AA7457485413436F718C21785,
2576EB9D76161302A2014430C9236E15C,
4FBBA860013F6618DB44043139DC781D0,
74D8652D9B84C55BBA947274E7FE413F,
368585D86E7E89DB9044C9007D831360

ROLL NO: 820723205012, 820723205016, 820723205018

DATE:03/10/2025

Completed the project named as phase node JS

Phase__ TECHNOLOGY PROJECT NAME :IBM-NJ-REST full contact management API

SUBMITTED BY,

U.Malathi:9080867437

K.Monisha:6379759824

M.Keerthika:8531064969

R.Ragavi:9585692593

S.Jaiseela:8248598737

IBM-NJ-REST — Contact Management API

Enhancement & Deployment Plan (Deadline: Week 9)
Generated: 2025-10-04 14:30:09

Overview

This document summarizes requested enhancements, UI/UX improvements, API changes, performance & security checks, testing plan, and recommended deployment options for the IBM-NJ-REST Contact Management API. Also included is a sample Node.js (Express) implementation of a Contacts REST API and sample curl commands demonstrating outputs.

1. Additional features

- Bulk contact import (CSV / vCard) with validation & de-duplication. - Contact groups / tags and group-level permissions. - Notes & activity timeline per contact (create/update/delete logs). - Photo/avatar upload (S3 / IBM COS) and image resizing. - Search: full-text search across name/email/notes with fuzzy matching. - Export contact(s) to CSV / vCard / JSON. - Webhooks for contact events (create/update/delete). - Role-based access control (Admin, Editor, Viewer).

2. UI/UX Improvements

- Responsive layout: desktop and mobile-first design. - Clear primary actions (Add Contact) and contextual actions for lists. - Inline editing for quick updates. - Bulk actions UI (select multiple -> export / delete). - Avatar upload with client-side crop/preview. - Accessibility: keyboard navigation & ARIA labels. - Performance: client-side pagination + server-side filtering.

3. API Enhancements

- Pagination (cursor-based preferred) for list endpoints. - Filter and sort query parameters (filter by tag, search q, sort by name/updatedAt). - PATCH endpoints for partial updates (RFC 5789). - Rate limiting headers and standardized error responses (RFC 7807 - Problem Details). - OpenAPI (Swagger) specification and example requests/responses. - API versioning (e.g., /api/v1/).

4. Performance & Security checks

Performance: - Add caching for hot endpoints (Redis). - Database indexing for search/filter fields. - Connection pooling and query optimization. - Stress testing (k6 or Artillery) with target SLAs. Security: - OAuth2 / OIDC with JWT for authentication. - Enforce TLS (HTTPS) and HSTS headers. - Input validation & output encoding; protect against injection. - Rate limiting and IP throttling. - Vulnerability scanning and dependency audits (Snyk, npm audit).

5. Testing of Enhancements

- Unit tests for business logic. - Integration tests for API endpoints (SuperTest / Jest or Mocha + Chai). - Contract tests against OpenAPI using Dredd or Pact. - End-to-end tests for UI (Playwright / Cypress). - Security testing (OWASP ZAP). - CI pipeline with automated tests and linting.

6. Deployment Options

Recommended options: - Netlify / Vercel: for static frontends (React/Vite) and serverless functions. - Cloud Providers (AWS/GCP/Azure) with containerized backend (Docker + Kubernetes/ECS). - Use managed DB (RDS / Cloud SQL / IBM DB services) and managed object storage (S3 / IBM COS). - CI/CD: GitHub Actions for build, test, and deploy pipelines.

Sample: Node.js (Express) Contacts API

Below is a minimal, self-contained example of an Express-based Contacts REST API (for illustration).

```
// app.js - minimal Contacts API (Node.js + Express)
// Run: node app.js
const express = require('express');
const bodyParser = require('body-parser');
const app = express();
app.use(bodyParser.json());

let contacts = [
  { id: '1', name: 'Alice Example', email: 'alice@example.com', phone: '555-0101', tags: ['client']
  { id: '2', name: 'Bob Sample', email: 'bob@example.com', phone: '555-0202', tags: ['lead'] }
];

// GET /api/v1/contacts
app.get('/api/v1/contacts', (req, res) => {
  const q = (req.query.q || '').toLowerCase();
  const result = contacts.filter(c => (!q) || c.name.toLowerCase().includes(q) || c.email.toLowerCa
  res.json({ data: result, total: result.length });
});

// GET /api/v1/contacts/:id
app.get('/api/v1/contacts/:id', (req, res) => {
  const c = contacts.find(x => x.id === req.params.id);
  if (!c) return res.status(404).json({ error: 'Not found' });
  res.json(c);
});

// POST /api/v1/contacts
app.post('/api/v1/contacts', (req, res) => {
  const id = String(Date.now());
  const newC = { id, ...req.body, createdAt: new Date().toISOString() };
  contacts.push(newC);
  res.status(201).json(newC);
});

// PATCH /api/v1/contacts/:id
app.patch('/api/v1/contacts/:id', (req, res) => {
  const idx = contacts.findIndex(x => x.id === req.params.id);
  if (idx === -1) return res.status(404).json({ error: 'Not found' });
  contacts[idx] = { ...contacts[idx], ...req.body, updatedAt: new Date().toISOString() };
  res.json(contacts[idx]);
});

// DELETE /api/v1/contacts/:id
app.delete('/api/v1/contacts/:id', (req, res) => {
  contacts = contacts.filter(x => x.id !== req.params.id);
  res.status(204).send();
});

app.listen(3000, () => console.log('Contacts API running on http://localhost:3000'));
```

Sample curl requests & expected outputs

```
# List contacts
curl -s http://localhost:3000/api/v1/contacts | jq
# Response:
# {
#   "data": [
#     { "id":"1", "name":"Alice Example", "email":"alice@example.com", "phone":"555-0101", "tags":[
#     { "id":"2", "name":"Bob Sample", "email":"bob@example.com", "phone":"555-0202", "tags":["lead
#   ],
#   "total": 2
# }
```

```
# Create a contact
curl -s -X POST http://localhost:3000/api/v1/contacts -H "Content-Type: application/json" -d '{"nam
```

Closing Notes

Notes: - Replace in-memory storage with PostgreSQL and use an ORM (Prisma / TypeORM / Sequelize) for production. - Add centralized logging (ELK, Loki) and monitoring (Prometheus + Grafana). - Create an OpenAPI spec and generate client SDKs if needed.

GitHub link:

K .monisha : https://github.com/monishakumar966/IBM-NJ-REST-ful-contact-management-API.git

R.ragavi : https://github.com/prakashragavi2006-lgtm/IBM_NJ_REST-ful-contact-management-API.git

S. Jaiseela: https://github.com/jaishu7526/Enhancement-Deployment-week-9-.git