# COLLEGE CODE : 5105

**COLLEGE NAME** : Bharathidasan Engineering College

**DEPARTMENT** : Computer science and engineering

**STUDENT\_NM-ID : dfa9a3454aa20fcba974a5b887f79ef7**

# REGISTER NO : 510523104082

# DATE :27- 10 - 2025

# Completed the project named as Phase 4

**Technology Project Name** : IBM-NJ- RESTFULL Contact Management APL

.

# SUBMITTED BY,

**NAME** :S.SURIYA

**MOBILE NO** :9361947680

# Phase 4: Enhancements, Testing, and Deployment

## 1. Additional Features:

**Feature Identification & Planning**

* Gather requirements for new features or improvements.
* Prioritize features based on project goals or stakeholder feedback.
* Examples:
  1. Bulk import/export of contacts (CSV, JSON).
  2. Tagging or grouping contacts.
  3. Advanced search/filter functionality. iv. Activity logs or audit trails.

v. API versioning support.

**Implementation**

* Design API endpoints and data models to support new features.
* Write backend logic and update database schema if needed.
* Ensure backward compatibility with existing API consumers.

**Integration**

* Connect new features to the UI, if applicable.
* Update frontend components to support added functionalities.

## 2. UI/UX Improvements

* User Interface Enhancements
  + Refine the layout and design for better user experience.
  + Ensure responsiveness across devices (mobile, tablet, desktop).
  + Improve forms for creating and editing contacts with real-time validation and helpful error messages.
  + Implement loading spinners and notifications to improve perceived performance.
  + Enhance accessibility (ARIA labels, keyboard navigation, color contrast).
* **User Experience Enhancements**
  + Simplify navigation flows and reduce the number of clicks to complete tasks.
  + Add helpful tooltips or onboarding tips.
  + Implement undo/redo functionality for critical actions.
  + Provide better feedback for user actions (e.g., success/failure messages).

## 3. API Enhancements

* **Functional Improvements** o Add **pagination** to handle large lists of contacts efficiently. o Support **sorting** by name, date created, or other relevant fields. o Implement **filtering** based on parameters like contact group, email domain, etc.
* **Versioning** o Introduce API versioning (e.g., /api/v1/contacts). o Document changes to support smooth transition for API consumers.
* **Rate Limiting & Throttling** o Protect API from abuse by limiting the number of requests per client/IP. o Implement using tools or middleware (e.g., Spring RateLimiter, API Gateway features).
* **Improved Error Handling** o Use consistent and descriptive error responses with standardized codes. o Include troubleshooting guidance in error messages.

**4. Performance & Security Checks**

* **Performance Optimization** o Optimize database queries (indexes, query restructuring). o Add caching layers (in-memory cache like Redis or HTTP cache headers).
  + Analyze response times and identify bottlenecks.
* **Security Hardening** o Secure endpoints using authentication mechanisms (JWT, OAuth2, API keys). o Implement role-based access control if needed. o Sanitize all inputs to prevent injection attacks.
  + Enable HTTPS everywhere.
  + Configure CORS policies correctly.
  + Conduct vulnerability scans and penetration testing.
* **Logging & Monitoring** o Add detailed logging for important events and errors. o Integrate monitoring tools for uptime and performance (e.g., Prometheus, Grafana).
  + Set up alerting on anomalies or failures.

## 5. Testing of Enhancements

* **Expand Test Suite** o Add unit tests for new features. o Update existing tests to cover changes and refactoring.
  + Write integration tests for full API workflows.
* **Manual & Exploratory Testing** o Perform exploratory testing focusing on new features and edge cases.
  + Test across different devices and browsers if UI is involved.
* **Regression Testing** o Ensure new changes have not broken existing functionality.
* **Performance Testing** o Run load tests to simulate multiple concurrent users.
* **Security Testing** o Validate all security improvements through penetration tests and code reviews.  **Documentation of Tests**

## 6. Deployment

* **Prepare Deployment Packages** o Build and package backend API (JAR/WAR or Docker image).
  + Bundle frontend code (if applicable).
* **Environment Configuration** o Manage environment-specific variables securely.
  + Set up database connections and secrets.
* **Deployment Targets**
  + **Frontend (UI)**: Deploy on **Netlify**, **Vercel**, or similar.
    - Use serverless functions if API endpoints need proxying.
  + **Backend API**: Deploy on cloud platforms like **IBM Cloud**, **AWS Elastic Beanstalk**, **Heroku**, or **Azure App Service**.
    - Alternatively, deploy via Docker containers on cloud VMs or Kubernetes clusters.
* **Continuous Integration/Continuous Deployment (CI/CD)** o Automate builds, tests, and deployments using tools like GitHub Actions, Jenkins, or Travis CI.
* **Post-Deployment Checks** o Smoke test deployed app for basic functionality. o Monitor logs and metrics for any issues. o Confirm environment variables and external integrations are working.

## 7. Project Deliverables & Documentation

* Final source code in version control.
* Updated and detailed API documentation (Swagger/OpenAPI).
* User manuals or README files covering new features.
* Postman collection with all API endpoints.
* Deployment guides for backend and frontend.
* Test reports and logs.
* Demo video or live walkthrough of the complete application.

# Source code:

**index.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>Registration Form</title>

<style>

body {

margin: 0;

font-family: 'Segoe UI', sans-serif;

background-color: #f0f4f8;

display: flex;

justify-content: center;

align-items: center;

min-height: 100vh;

}

.form-wrapper {

width: 100%;

padding: 20px;

}

.container {

background-color: #fff;

padding: 30px 40px;

border-radius: 12px;

box-shadow: 0 8px 20px rgba(0,0,0,0.1);

max-width: 500px;

margin: 0 auto;

}

h2 {

text-align: center;

margin-bottom: 25px;

color: #333;

font-size: 24px;

}

.form-group {

margin-bottom: 18px;

}

label {

display: block;

margin-bottom: 6px;

font-weight: 500;

color: #333;

}

input, textarea {

width: 100%;

padding: 10px 12px;

border-radius: 6px;

border: 1px solid #ccc;

font-size: 14px;

transition: all 0.3s;

}

input:focus, textarea:focus {

border-color: #2575fc;

box-shadow: 0 0 6px rgba(37,117,252,0.3);

outline: none;

}

textarea {

min-height: 60px;

resize: none;

}

.gender-group {

margin-bottom: 18px;

}

.gender-options {

display: flex;

gap: 20px;

margin-top: 6px;

}

.gender-options label {

display: flex;

align-items: center;

gap: 5px;

cursor: pointer;

}

.gender-options input[type="radio"] {

accent-color: #2575fc;

}

.terms-group {

display: flex;

align-items: center;

gap: 10px;

}

.terms-group input[type="checkbox"] {

accent-color: #2575fc;

width: 18px;

height: 18px;

}

button {

width: 100%;

padding: 12px;

background-color: #2575fc;

border: none;

color: #fff;

border-radius: 8px;

font-weight: bold;

font-size: 16px;

cursor: pointer;

transition: all 0.3s;

}

button:hover {

background-color: #145dcc;

transform: scale(1.02);

}

.error-msg {

color: red;

font-size: 13px;

margin-top: 3px;

display: block;

}

input.error, select.error, textarea.error {

border: 2px solid #ff4d4d;

background-color: rgba(255, 0, 0, 0.05);

}

.error-border {

border: 2px solid #ff4d4d !important;

background-color: rgba(255, 0, 0, 0.05);

}

#toast {

position: fixed;

top: 20px;

right: 20px;

min-width: 250px;

padding: 15px 20px;

border-radius: 8px;

font-weight: bold;

font-size: 14px;

display: none;

z-index: 1000;

box-shadow: 0 4px 15px rgba(0,0,0,0.2);

opacity: 0;

transition: opacity 0.5s, transform 0.5s;

}

#toast.show {

display: block;

opacity: 1;

transform: translateY(0);

}

#toast.success {

background-color: #d4edda;

color: green;

}

#toast.warning {

background-color: #fff3cd;

color: #856404;

}

</style>

</head>

<body>

<div class="form-wrapper">

<div class="container">

<h2>Registration Form</h2>

<form id="registerForm">

<div class="form-group">

<label for="name">Full Name</label>

<input type="text" id="name" placeholder="Enter your full name">

<span class="error-msg"></span>

</div>

<div class="form-group">

<label for="email">Email</label>

<input type="email" id="email" placeholder="Enter your email">

<span class="error-msg"></span>

</div>

<div class="form-group">

<label for="password">Password</label>

<input type="password" id="password" placeholder="Enter password">

<span class="error-msg"></span>

</div>

<div class="form-group">

<label for="confirmPassword">Confirm Password</label>

<input type="password" id="confirmPassword" placeholder="Re-enter password">

<span class="error-msg"></span>

</div>

<div class="form-group">

<label for="phone">Phone</label>

<input type="text" id="phone" placeholder="10-digit phone number">

<span class="error-msg"></span>

</div>

<div class="form-group gender-group">

<label>Gender</label>

<div class="gender-options">

<label><input type="radio" name="gender" value="Male"> Male</label>

<label><input type="radio" name="gender" value="Female"> Female</label>

<label><input type="radio" name="gender" value="Other"> Other</label>

</div>

<span class="error-msg"></span>

</div>

<div class="form-group">

<label for="dob">Date of Birth</label>

<input type="date" id="dob">

<span class="error-msg"></span>

</div>

<div class="form-group">

<label for="address">Address</label>

<textarea id="address" placeholder="Enter your address"></textarea>

<span class="error-msg"></span>

</div>

<div class="form-group">

<label for="city">City</label>

<input type="text" id="city" placeholder="Enter your city">

<span class="error-msg"></span>

</div>

<div class="form-group">

<label for="state">State</label>

<input type="text" id="state" placeholder="Enter your state">

<span class="error-msg"></span>

</div>

<div class="form-group">

<label for="zip">ZIP Code</label>

<input type="text" id="zip" placeholder="Enter ZIP code">

<span class="error-msg"></span>

</div>

<div class="form-group terms-group">

<input type="checkbox" id="terms">

<label for="terms">I agree to the terms & conditions</label>

<span class="error-msg"></span>

</div>

<button type="submit">Register</button>

</form>

</div>

</div>

<div id="toast"></div>

<script>

const registerForm = document.getElementById('registerForm');

const toast = document.getElementById('toast');

registerForm.addEventListener('submit', (e) => {

e.preventDefault();

document.querySelectorAll('.error-msg').forEach(el => el.textContent = '');

document.querySelectorAll('input, textarea').forEach(el => el.classList.remove('error-border'));

const name = value('name');

const email = value('email');

const password = value('password');

const confirmPassword = value('confirmPassword');

const phone = value('phone');

const gender = document.querySelector('input[name="gender"]:checked')?.value;

const dob = value('dob');

const address = value('address');

const city = value('city');

const state = value('state');

const zip = value('zip');

const terms = document.getElementById('terms').checked;

let isValid = true;

if (!name) showError('name', 'Full Name is required');

if (!email) showError('email', 'Email is required');

else if (!/\S+@\S+\.\S+/.test(email)) showError('email', 'Invalid email format');

if (!password) showError('password', 'Password is required');

else if (password.length < 6) showError('password', 'Password must be at least 6 characters');

if (!confirmPassword) showError('confirmPassword', 'Confirm your password');

else if (password !== confirmPassword) showError('confirmPassword', 'Passwords do not match');

if (!phone) showError('phone', 'Phone number is required');

else if (!/^\d{10}$/.test(phone)) showError('phone', 'Phone must be 10 digits');

if (!gender) {

document.querySelector('.gender-group .error-msg').textContent = 'Select gender';

isValid = false;

}

if (!dob) showError('dob', 'Date of Birth is required');

else if (new Date(dob) > new Date()) showError('dob', 'Date of Birth cannot be in the future');

if (!address) showError('address', 'Address is required');

if (!city) showError('city', 'City is required');

if (!state) showError('state', 'State is required');

if (!zip) showError('zip', 'ZIP code is required');

else if (!/^\d{5,6}$/.test(zip)) showError('zip', 'ZIP must be 5 or 6 digits');

if (!terms) {

document.querySelector('.terms-group .error-msg').textContent = 'You must agree to the terms';

isValid = false;

}

if (!isValid) {

showToast('⚠️ Please correct highlighted fields.', 'warning');

return;

}

// If everything is valid

showToast('✅ Registration Successful!', 'success');

registerForm.reset();

function value(id) {

return document.getElementById(id)?.value.trim();

}

function showError(id, message) {

const input = document.getElementById(id);

const error = input?.nextElementSibling;

if (error) error.textContent = message;

input?.classList.add('error-border');

isValid = false;

}

function showToast(message, type = 'success') {

toast.textContent = message;

toast.className = `show ${type}`;

setTimeout(() => { toast.className = toast.className.replace('show', ''); }, 3000);

}

});

</script>

</body>

</html>

**Server.js**

const express = require('express');

const connectDB = require('./config/db');

const userRoutes = require('./routes/userRoutes');

const cors = require('cors');

const path = require('path');

const app = express();

connectDB();

app.use(cors());

app.use(express.json());

app.use(express.static(path.join(\_\_dirname, '../frontend')));

app.get('/', (req, res) => {

res.sendFile(path.join(\_\_dirname, '../frontend/index.html'));

});

app.use('/api/users', userRoutes);

const PORT = 3000;

app.listen(PORT, () => console.log(`✅

README.md

IBM Node.js Client-Side form Validation Project

: S. Aswan : B.E.C College of Engineering

Project Overview

This project demonstrates client-side form validation using HTML, CSS, and JavaScript, with a Node.js server to handle submissions. It ensures user inputs are validated before sending data to the server, enhancing data integrity and user experience.

Aim

To implement robust client-side validation for web forms using JavaScript, with server-side handling via Node.js.

Features

Real-time input validation (e.g., email, phone, password strength)

Error messages displayed dynamically

Clean, responsive UI design

Node.js backend for handling form submissions

Easy to extend for other forms and validations

Technology Stack

Frontend: HTML, CSS, JavaScript

Backend: Node.js

Database: Mongodb

Validation: JavaScript (client-side)

Project Structure NM-Project/ │ ├─ server.js # Node.js server ├─ index.html # Main HTML form ├─ style.css # CSS for styling ├─ script.js # JS for client-side validation └─ README.md # Project documentation

How to Run Locally

Navigate to the project folder:

C:\xampp\htdocs\NM-Project\backend

Install dependencies:

npm install

Start the server:

node server.js

Open index.html in your browser and test the form.

Demo

Challenges & Solutions

Challenge: Ensuring proper validation for all input types. Solution: Used JavaScript regex and conditional checks for real-time validation.

Challenge: Handling server submission errors. Solution: Implemented Node.js backend with proper error handling.

References

Node.js Documentation

MDN Web Docs - JavaScript

Screenshots

all screenshots included in phase 5

License

This project is open-source and free to use.