Barangay Management System

Overview

A comprehensive web-based management system for Barangay administration. This system allows managing residents, documents, events, officials, complaints, and users through an intuitive web interface.

Status: Fully operational; deployed using Render.com (web service) with PostgreSQL hosted on Render

Last Updated: October 25, 2025

Features

• **Resident Management:** Add, edit, view, and delete resident records
• **Document Management:** Track and manage barangay documents (clearances, certificates, permits)
• **Event Management:** Schedule and manage community events
• **Officials Management:** Maintain records of barangay officials
• **Complaints Management:** Track and handle resident complaints
• **User Management:** Admin can manage user accounts and roles
• **Data Import/Export:** Backup and restore system data
• **Role-Based Access:** Separate user and admin interfaces

Web-Based Barangay Management System — Manuscript

This document is the application development manuscript for the Web-Based Barangay Management System (BMS) implemented for Barangay Cataggaman Nuevo. It follows academic manuscript structure and documents the system goals, technical choices, architecture, methodology, and significance.

ABSTRACT

This manuscript documents the design and development of a Web-Based Barangay Management System (BMS) for Barangay Cataggaman Nuevo. The system replaces traditional paper-based workflows with a centralized, secure web application for resident profiling, document issuance, incident/blotter tracking, event management, and administrative reporting. Implemented using Node.js/Express and PostgreSQL, the application supports user registration/login, CRUD operations for residents/documents/events/officials/complaints, data import/export, and monitoring via Render.com and UptimeRobot. The project aims to reduce processing time, minimize human error, and improve record security and accessibility. SMS notification support is planned for a future release to enhance communication between barangay officials and residents.

1. INTRODUCTION

Barangays play a vital role in the Philippine government as the first line of service to the community at the grassroots level. Barangay Cataggaman Nuevo currently uses a paper-based management system that is inefficient and error-prone. This project aims to design and implement a web-based system to improve record keeping, document issuance, incident reporting, and event coordination while preserving data security and accessibility.

1.1 PROJECT CONTEXT

The BMS provides a centralized repository for resident records and administrative tasks. It is intended for barangay administrators, health workers (BHWs), and residents for common services like certificate issuance and complaints filing. The system integrates a backend API, a PostgreSQL database, and static front-end pages.

1.2 PURPOSE DESCRIPTION

The purpose of this project is to design and develop a web-based management system to address inefficiencies in manual record-keeping. Specifically, the system is intended to:

- Provide a centralized database for resident information.
- Automate issuance of certificates and related documentation.
- Record and monitor community incidents efficiently.
- Allow faster communications (web-based notifications; SMS planned for future release).

By implementing this system the barangay can reduce processing time, minimize human error, and improve record security and retrieval.

1.3 OBJECTIVES OF THE STUDY

Main objective: Design, implement, and evaluate the Web-Based Barangay Management System to improve operational efficiency and transaction services.

Specific objectives:

- Describe the profile of barangay residents (age, gender, employment status, length of residency).
- Assess the effectiveness of the web-based system in automating document requests compared with the manual system.
- Enhance profile management, data verification, and attendance/leave records.
- Use Likert-scale surveys before and after implementation to measure employee satisfaction.
- Determine the impact on data security and loss-risk reduction versus paper records.

1.3.1 GENERAL OBJECTIVES

Create, build, and test a web-based barangay management system that improves daily operations for Barangay Cataggaman Nuevo.

1.3.2 SPECIFIC OBJECTIVES

- Create a secure database to store resident information.
- Develop modules for certificate requests, incident reports, and barangay clearances.
- Implement notification channels (web-based now, SMS planned).
- Evaluate usability, reliability, and effectiveness.
- Compare the system’s efficiency against the manual process.

1.4 RESEARCH PARADIGM

This study uses the Technology Acceptance Model (TAM). Core constructs used are Perceived Usefulness (PU) and Perceived Ease of Use (PEOU). The independent variable is the web-based BMS; dependent variables include operational efficiency, user satisfaction, and service quality.

1.5 SCOPE AND DELIMITATION

Scope:

- Design and development of a web-based BMS for Barangay Cataggaman Nuevo.
- Data collection and evaluation: September–December 2025.
- Focus: document tracking, blotter management, certificate issuance, resident profiling.

Delimitations:

- The results are specific to Barangay Cataggaman Nuevo and may require adaptation for other barangays.

1.6 SIGNIFICANCE OF THE STUDY

- Barangay officials: Streamline operations and increase transparency.
- Barangay staff/BHWs: Improved workflows and record accuracy.
- Residents: Faster access to services.
- Policymakers: Local evidence supporting e-governance adoption.
- Researchers: Reference for future local e-governance studies.

1.7 DEFINITION OF TERMS

- Barangay Management System (BMS): A software application to manage barangay functions and records.
- Data Accuracy: Degree to which stored data reflect real-world facts.
- Database: A structured collection of stored information (PostgreSQL in this project).
- Digital Transformation: Process of adopting digital technologies to improve business processes.
- E-Governance: Use of information technologies to provide public services and governance.
- Efficiency: Speed and resource usage in completing tasks.
- Information System: Organized set of components that collect, process, store, and distribute information.
- Paper-Based System: Traditional manual system relying on physical documents.
- Service Transaction Time: Time required to complete a service request.
- System Usability: Ease with which users can achieve goals using the system.
- Transparency: Clarity and openness of operations and record availability.

2. TECHNICAL BACKGROUND

This chapter documents the technical components, system setup, tools, and requirements. The application uses a modern web stack: front-end pages built with HTML5/CSS3/JavaScript, and a back-end API using Node.js and Express that connects to PostgreSQL via a centralized `db.js` helper. Environment configuration uses a `.env` file (loaded with `dotenv`). Static files and client pages are served from the project root using Express static middleware.

Hosting and deployment:

- Usability — Simple, responsive UI for administrative staff and residents.
- Reliability — Consistent operation, data integrity, DB backups (managed by Render), and graceful handling of transient errors.
- Security — Password hashing (bcrypt recommended), environment secrets in `.env` / Render environment variables, and access control for admin actions.
- Maintainability — Modular code structure: API routers, DB helper, static assets separated for easier maintenance.
- Scalability — Use of connection pooling (`pg.Pool`) in `db.js` and ability to scale web and database tiers independently on Render.

Notes on monitoring/cold starts: The `/health` endpoint performs a lightweight DB check and returns metrics that monitoring tools can use.

2.2 SYSTEM REQUIREMENTS

2.2.1 HARDWARE

Cloud hosting limits local hardware needs. Recommended cloud specs for small deployments: 512MB–1GB RAM and small vCPU allocations. Client devices: modern smartphones and laptops are adequate.

2.2.2 SOFTWARE

- Frontend: HTML5, CSS3, JavaScript (vanilla)
- Backend: Node.js, Express
- Database: PostgreSQL (Render managed)
- Monitoring: UptimeRobot (optional)
- Environment: `dotenv` for local testing; Render environment variables for production.

2.2.3 PEOPLEWARE

Roles: Project manager, system analyst, developers, DB administrator, UI/UX designer, QA/testers, barangay officials (end users), maintenance personnel. Responsibilities are documented in the project notes.

3. METHODOLOGY

Development followed an iterative process: design !’ implement !’ test !’ deploy !’ monitor. Key activities included requirements gathering, API design, database schema creation, integration tests, and deployment to Render.

3.1 REQUIREMENTS ANALYSIS

Functional requirements:

- User registration and authentication
- Admin panel for residents, documents, officials, events, complaints
- CRUD API endpoints under `/api/*`
- Import/export and DB initialization endpoints
- Health check endpoint `/health`

Nonfunctional requirements were covered in section 2.1.

3.2 USER REQUIREMENT

User stories:

- Admin: log in, manage residents/documents/events/complaints/officials.
- Resident: register, log in, request documents.
- System operator: deploy to Render and monitor with UptimeRobot.

Key API endpoints:

- `GET /api/:collection`, `GET /api/:collection/:id`, `POST /api/:collection`, `PUT /

api/:collection/:id`, `DELETE /api/:collection/:id`
- `POST /api/login`, `POST /api/register`, `GET /api/export`, `POST /api/import`, `POST /api/init-database`, `GET /health`

3.3 CONTEXT LEVEL DIAGRAM (textual)

Actors:

- Admin (web browser) !” Web Application (Express)
- Resident (web browser) !” Web Application (Express)
- Web Application !” PostgreSQL (Render-hosted)
- Web Application !” UptimeRobot (health pings)
- GitHub !” Render (continuous deploys)

High-level flow: Browser !’ Express routes !’ business logic !’ DB (`db.js`) !’ response !’ client UI.

3.4 DATA FLOW DIAGRAM (textual)

Data sources: user form inputs, admin CRUD forms.

Flow: Client !’ Express API !’ SQL queries !’ PostgreSQL !’ Express returns JSON !’ Client renders UI.

Import/export: JSON payloads via `/api/import` and `/api/export`.

Health monitoring: UptimeRobot !’ GET `/health` !’ server returns status.

3.5 ENTITY RELATIONSHIP DIAGRAM (textual)

Primary entities:

- `residents(id, name, age, address, contact, ...)`
- `users(id, username, password, role, resident_id)`
- `documents(id, type, resident_id, date, status, approved_by)`
- `events(id, title, date, time, location)`
- `officials(id, name, position, contact)`
- `complaints(id, resident_id, title, details, date)`

Relationships:

- `users.resident_id -> residents.id` (optional)
- `documents.resident_id -> residents.id`
- `complaints.resident_id -> residents.id`

Note: Ensure PostgreSQL-compatible SQL in `create_tables.sql` (use double quotes or unquoted identifiers, not single quotes around identifiers).

3.6 HIERARCHICAL DIAGRAMS (textual)

Module hierarchy:

- `server.js` — entry point, middleware, API routes, static serving
- `db.js` — centralized database helper (pg.Pool)
- Frontend pages — HTML files in repo root
- Client JS — files in `js/` and `Javascript_User_&_admin/`

REFERENCES

- Express — https://expressjs.com/

- node-postgres (pg) — https://node-postgres.com/
- Render — https://render.com/
- UptimeRobot — https://uptimerobot.com/
- dotenv — https://www.npmjs.com/package/dotenv
- bcrypt / bcryptjs — https://www.npmjs.com/package/bcrypt
- PostgreSQL — https://www.postgresql.org/

Notes: the repository contains `render.yaml` and a `.env` pointing to a Render-managed PostgreSQL instance. Some front-end assets use the Cloudflare CDN (cdnjs.cloudflare.com) for icons; those are CDN references only.