WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

SMART WASTE MANAGEMENT SYSTEM COMMUNITY WASTE BIN MONITORING AND COLLECTION

KEERTHANA K[1], MITHRALAXMI SARAVANAN[2], PANDISELVI P[3], I.A. JANNATHUL FIRTHOUS[4]

[4] Project Guide:
Department of Information Technology Sri Shakthi Institute of Engineering and Technology Coimbatore, Tamil Nadu, India
Email ID: samyuktha86101@gmail.com

ABSTRACT :

Waste bins in residential areas, markets, parks and apartment blocks can overflow when the people responsible for collection do not know which bins need attention. This project presents the Smart Community Waste Management System, a web-based dashboard for monitoring waste bins and requesting their collection. The application uses a Node.js and Express backend, a built-in SQLite database accessed through the node:sqlite module, and a frontend written in HTML, CSS, and JavaScript. The database stores bins with a name, location, fill level from 0 to 100 percent, and collection status, together with a table of collection requests. The backend exposes REST endpoints for listing bins, reading dashboard statistics, listing alerts for bins that are at least 80 percent full, updating a fill level, requesting a collection, marking a bin as collected, and adding a bin. The dashboard shows five summary cards, bin fill-level indicators, bin alerts, a collection status table, and recycling guidance. Ten functional checks run against a local copy of the project returned the expected responses. The supplied version does not include sensor hardware, user authentication, or a fully dynamic interface, and these are identified as future work. The project demonstrates a small full-stack application for community waste management.

Keywords: Smart Waste Management, Waste Bin Monitoring, Node.js, Express, SQLite, REST API, Dashboard, Waste Collection.

1. INTRODUCTION

Communities generate waste every day, and the bins that receive it are spread across homes, markets, roads, parks, and apartment buildings. When a bin is not emptied in time, waste can spill over, attract pests, and affect public hygiene. Collection staff often work from fixed schedules, so a bin that fills quickly may wait for the next round while another bin is visited although it is nearly empty. These challenges motivate simple digital tools that show the fill state of each bin and let a person request collection when needed.

Web technologies make such a tool easy to deliver, because a dashboard can be opened in any browser without installing software. In this project, the Smart Community Waste Management System combines a database of bins, a REST backend, and a browser dashboard in one application. The dashboard presents summary figures, bin fill levels, alerts, collection status, and recycling information.

1.1 Existing challenges

Manual inspection of bins is time-consuming and gives no continuous view of which locations are filling fastest. Without a shared record, collection requests may be passed on verbally and be forgotten, and there is no history of when a bin was requested or emptied. A useful system should therefore keep the fill level of each bin, highlight bins that are close to full, record each collection request with a time, and show the current position on a single screen. It should also validate the data it accepts so that an incorrect value does not corrupt the stored records.

1.2 Role of web technologies

Node.js allows the backend to be written in JavaScript, the same language used in the browser. Express provides routing and JSON handling for the REST endpoints. SQLite stores the data in a single local file, which suits a small application and removes the need for a separate database server. In this project the database is accessed through the node:sqlite module that is built into recent Node.js versions. The frontend uses HTML for structure, CSS for layout, and JavaScript with the Fetch API to read statistics and send collection requests to the backend.

1.3 Problem statement and objectives

The project addresses the need for a convenient dashboard that shows the state of community waste bins, highlights bins that are full or nearly full, and

allows a collection to be requested and recorded. The supplied version is a working prototype that uses sample bins and does not read from physical sensors.

The objectives of the Smart Community Waste Management System are:

- To store waste bins, their locations, fill levels, and collection status in a database.
- To provide REST API endpoints for listing bins, dashboard statistics, and alerts.
- To classify each bin as low, medium, near-full, or full from its fill level.
- To raise alerts for bins that have reached 80 percent or more of their capacity.
- To let a user request a collection and to record the request with a timestamp.
- To allow a bin to be marked as collected, emptying it and completing the request record.
- To validate fill levels and new-bin data and to return clear error messages.
- To present the information in a browser dashboard with recycling guidance for the community.

2. MATERIALS AND METHODS

2.1 Development approach

The project was developed through requirement identification, database design, backend implementation, frontend development, integration, and testing. The backend was written in Node.js with Express and the node:sqlite module. The frontend consists of one HTML page, one stylesheet, and one JavaScript file. Express serves the frontend folder as static files, so the dashboard and the API are delivered by the same server process and the browser can call the API using relative addresses.

2.2 System architecture

The system follows a client-server architecture. The browser loads the dashboard from the Express server. When the page opens, the JavaScript requests the dashboard statistics from the API and writes the values into the summary cards. When the user requests a collection, the script finds the bin through the API and sends a collection request. The server validates each request, runs the corresponding SQL statement on the SQLite database, and returns the result as JSON.

Layer / component	**Responsibility**
Frontend – HTML, CSS, JavaScript	Shows the dashboard cards, bin indicators, alerts, collection table, and recycling information; sends API requests.
Fetch API calls (script.js)	Loads dashboard statistics and sends collection requests to the backend and shows a confirmation message.
Express server (server.js)	Receives requests, validates inputs, runs database statements, and returns JSON responses; serves the frontend files.
node:sqlite database access	Opens the database file, creates the tables, seeds sample data, and executes prepared statements.
SQLite database (waste.db)	Stores the bins table and the collections table in a single file.
Helper function level()	Converts a fill level into the category full, near, medium, or low.

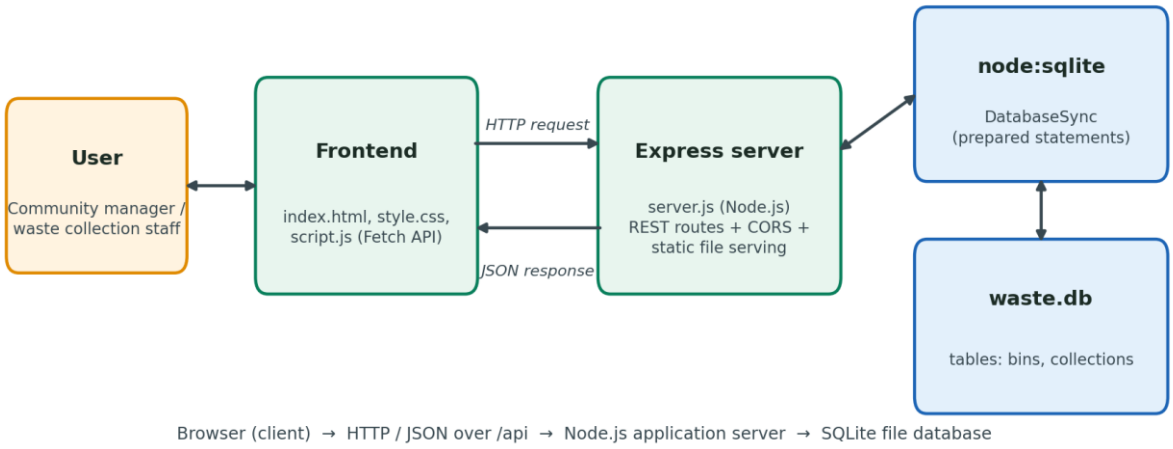


Figure 1. System architecture of the Smart Community Waste Management System.

2.3 Technology stack and development environment

Technology	Use in the project
Node.js	Runtime for the backend server.
Express (^4.19.2)	Web framework for routing, JSON parsing, and static file serving.
node:sqlite (DatabaseSync)	Built-in module used to open the database and run SQL.
SQLite	File-based relational database (waste.db).
cors (^2.8.5)	Allows cross-origin requests to the API.
HTML / CSS	Page structure, layout, colours, and responsive cards.
JavaScript (Fetch API)	Loads statistics and sends collection requests from the browser.
npm	Installs dependencies and starts the server with npm start.

The server listens on the port given by the PORT environment variable and uses port 3000 when none is set. The package file defines the start script as node server.js. The node:sqlite module is marked experimental in Node.js, and the server prints an experimental-feature warning when it starts. The project also contains a Visual Studio Code launch configuration that opens Chrome at http://localhost:8080, so the server must be run on that port, or the configuration changed, when that shortcut is used.

2.4 Database design and sample data

The server creates two tables when it starts, if they do not already exist. The bins table stores one row for each bin. The collections table stores one row for each collection request and refers to a bin through bin_id. A check constraint keeps the fill level between 0 and 100. When the bins table is empty, the server inserts five sample bins: Bin #01 in the Residential Area (30 percent), Bin #02 in the Market Area (55 percent), Bin #03 in the Apartment Block (100 percent), Bin #04 in the Public Park (100 percent), and Bin #07 on the Main Road (80 percent).

Table.field	Type	Purpose
bins.id	INTEGER, primary key	Unique bin number, assigned automatically.
bins.name	TEXT, required	Bin label, for example Bin #01.
bins.location	TEXT, required	Place where the bin is installed.
bins.fill_level	INTEGER, 0–100, default 0	Current fill level in percent.
bins.collection_status	TEXT, default none	One of none, requested, or collected.
collections.id	INTEGER, primary key	Unique request number.
collections.bin_id	INTEGER, references bins	Bin for which collection was requested.
collections.requested_at	TEXT, default current time	Time the request was created.
collections.completed_at	TEXT, may be empty	Time the collection was completed.

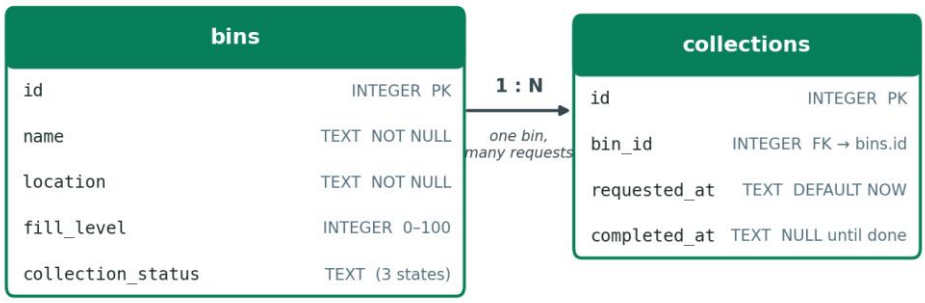


Figure 2. Database schema of the project: the bins table and the collections table (one bin, many collection requests).

2.5 Fill-level categories and dashboard statistics

The server converts a fill level into a category using the helper function level(). A bin at 100 percent is full, a bin from 80 to 99 percent is near full, a bin from 50 to 79 percent is medium, and a bin below 50 percent is low. The list of bins returned by the API contains this category in a field named level. The statistics endpoint returns five values for the dashboard cards. Total is the number of bins. Full is the number of bins with a fill level of 100 percent or more. Empty is calculated as the total number of bins minus the full bins. Pending is the number of bins whose collection status is not collected and whose fill level is at least 50 percent. The recycled percentage is a fixed value of 65 that is returned by the server and is not calculated from stored data.

2.6 Frontend design

The dashboard page contains a header, a navigation bar with five links (Dashboard, Waste Bins, Alerts, Collection, and Recycling), and five sections. The dashboard section shows five summary cards. The waste bin section shows a card for each bin with an icon, name, location, a coloured progress bar, and a status label. The alerts section lists bins that need attention. The collection section contains a table with the bin, location, fill level, status, and an action button. The recycling section describes paper, plastic, organic, and electronic waste. The stylesheet uses a green colour scheme, a sticky navigation bar, and card layouts.

In the supplied version, the JavaScript file performs two tasks. On page load it requests /api/stats and replaces the values in the five summary cards. When a Collect button is pressed, it requests /api/bins, finds the bin whose name matches the button, sends a collection request, shows a short confirmation message for about two and a half seconds, and reloads the statistics. The bin cards, the alert list, and the rows of the collection table are written directly in the HTML.

2.7 Collection workflow

1. The user opens the dashboard, which Express serves from the Frontend folder.
2. The script requests GET /api/stats and writes the returned values into the five summary cards.
3. The user reviews the bin indicators, the alerts, and the collection table.
4. The user presses Collect for a bin, which calls the collectBin function with the bin name.
5. The script requests GET /api/bins and finds the identifier of the bin with that name.
6. The script sends POST /api/bins/:id/collect to the server.
7. The server looks up the bin and returns a not-found error if it does not exist.
8. For an existing bin, the server inserts a row into the collections table and sets the collection status of the bin to requested.
9. The server returns a confirmation message, the dashboard shows it, and the statistics are reloaded.
10. After the waste is removed, POST /api/bins/:id/collected sets the fill level to 0, the status to collected, and records the completion time on the open request.

2.8 Monitoring and alert workflow

The fill level of a bin can be changed through PATCH /api/bins/:id/fill, which the source code describes as a sensor or manual update. The endpoint accepts a whole number from 0 to 100 and returns an error for any other value or for an unknown bin. The alerts endpoint returns every bin with a fill level of 80 percent or more, ordered from the highest level to the lowest. The supplied project does not contain sensor hardware or code that reads a sensor, so fill levels are entered through the API or taken from the sample data.

2.9 Backend API design

Endpoint	Method	Purpose
/api/bins	GET	Returns all bins with their fill-level category.
/api/stats	GET	Returns total, empty, full, pending, and recycled percentage.
/api/alerts	GET	Returns bins at 80 percent or more, highest first.
/api/bins/:id/fill	PATCH	Updates the fill level of a bin (0–100).
/api/bins/:id/collect	POST	Creates a collection request and sets status to requested.
/api/bins/:id/collected	POST	Empties the bin and completes the open request.
/api/bins	POST	Adds a new bin from name and location.
/	GET	Serves the dashboard and its static files.

2.10 Input validation and safety

The fill-level endpoint rejects values that are not whole numbers between 0 and 100, and the database enforces the same range with a check constraint. The add-bin endpoint requires a name and a location and returns status 400 when either is missing. Requests for a bin that does not exist return status 404. All SQL statements are written as prepared statements with parameters instead of joining user text into the query, which reduces the risk of SQL injection. The browser code writes messages and statistics as plain text. These controls do not make the application production-secure: the supplied version has no login, no user roles, no rate limiting, and it allows requests from any origin.

2.11 Testing and evaluation method

The project does not include an automated test suite. For this journal, the server was started on a local copy of the project with the supplied database, and each API route was called with an HTTP client for valid and invalid input. The dashboard page was loaded in a headless browser to confirm that it renders and that the summary cards receive values from the API. The observed responses are recorded in Table 3. No performance, load, or usability measurements were taken, so none are reported.

3. RESULTS AND DISCUSSION

3.1 Implemented application features

The supplied project contains an Express backend with seven API routes and static file serving, a SQLite database with two tables, and a dashboard page with five sections. The backend can list bins, calculate dashboard statistics, list alerts, update a fill level, create and complete collection requests, and add a bin. The dashboard loads its summary cards from the API and can send a collection request. These are implementation features verified from the project source and the checks in Section 3.5; the usefulness of the system for real collection staff would need separate evaluation.

3.2 Implementation summary

Item	Value
API routes (excluding static files)	7
Database tables	2
Dashboard sections	5
Dashboard summary cards	5
Sample bins inserted on first run	5
Fill-level categories	4 (low, medium, near, full)
Alert threshold	80% fill level
Functional checks run	10
Checks that returned the expected response	10

Table 1. Summary of the implementation and of the functional checks run for this journal.

3.3 Dashboard data

The supplied database contains the five sample bins and one collection request, created for Bin #03 on 12 October 2026 and not yet completed. With this data, the statistics endpoint returns a total of 5 bins, 3 bins counted as empty, 2 full bins, 4 pending collections, and a recycled percentage of 65. Bin #03 and Bin #04 are full, and Bin #07 is at 80 percent, so the alerts endpoint returns these three bins in that order of fill level.

Bin	Location	Fill level (%)	Category	Collection status
Bin #01	Residential Area	30	low	none
Bin #02	Market Area	55	medium	none
Bin #03	Apartment Block	100	full	requested
Bin #04	Public Park	100	full	none
Bin #07	Main Road	80	near	none

Table 2. Bins stored in the supplied waste.db database and their categories.

The pending value of 4 counts every bin that is at least 50 percent full and not yet marked collected, which includes Bin #02, Bin #03, Bin #04, and Bin #07. The empty value of 3 is the total of 5 minus the 2 full bins, so it includes bins that are partly filled.

3.4 Dashboard screenshot



Figure 3. Dashboard of the Smart Community Waste Management System showing the five summary cards and the bin monitoring section, captured from the running application.

3.5 Functional test scenarios

Test ID	Module	Test scenario	Observed result
F1	Static hosting	Open / on the server	Dashboard page returned with status 200.
F2	Bins API	GET /api/bins	Five bins returned, each with a level field.
F3	Stats API	GET /api/stats	Returned total 5, empty 3, full 2, pending 4, recycled 65.
F4	Alerts API	GET /api/alerts	Returned Bin #03, Bin #04 (100%) and Bin #07 (80%).
F5	Validation	PATCH fill_level 150	Status 400 with a message that the level must be 0–100.
F6	Fill update	PATCH fill_level 85 for bin 1	Status 200 and ok returned.
F7	Not found	PATCH for bin 99	Status 404, Bin not found.
F8	Collection request	POST /api/bins/2/collect	Status 200 with the message “Bin #02 collection request sent!”.
F9	Mark collected	POST /api/bins/2/collected	Status 200; completion time recorded in the collections table.
F10	Add bin	POST with name and location; POST with name only	Status 201 with the new id; status 400 for the missing location.

Table 3. Functional checks run against a local copy of the project. They are not a formal test campaign.

3.6 Discussion and limitations

The system shows that a small full-stack application can store bin data, classify fill levels, raise alerts, and record collection requests with a Node.js backend and a single-file database. The database and the API are separate from the page, so the interface can later be replaced or extended without changing the stored data. The API already includes functions that the page does not yet use, such as the alerts endpoint, the fill-level update, the collected action, and adding a bin.

Several limitations of the supplied version should be stated. First, only the five summary cards and the Collect action use the API; the bin cards, the alert list, and the collection table are fixed HTML, so they do not change when the database changes, and the table lists four of the five stored bins. Second,

the Empty Bins card shows total minus full bins, and the recycled percentage is a constant, so neither is a measured value. Third, once a bin is marked collected, its status stays collected, so a bin that fills again is not counted as pending, and repeated collection requests for one bin are not prevented. Fourth, the project has no sensors, login, user roles, notifications, map view, or route planning. Fifth, the node:sqlite module is experimental, and the debug configuration uses a different port from the server default. These points are suitable targets for future work.

4. CONCLUSION

The Smart Community Waste Management System is a web-based application for monitoring community waste bins and requesting their collection. It uses Node.js, Express, SQLite, and a browser dashboard written in HTML, CSS, and JavaScript. The backend stores bins and collection requests, classifies fill levels, calculates dashboard statistics, lists bins that need attention, and validates the data it receives. The dashboard shows summary cards, bin indicators, alerts, collection status, and recycling guidance.

Ten functional checks on a local copy of the project returned the expected responses. The supplied version is a prototype: parts of the page are static, some dashboard values are fixed or approximate, and the project has no sensor input or authentication. Future work includes generating the bin cards, alerts, and collection table from the API, connecting fill-level sensors to the update endpoint, calculating the recycling figure from recorded data, preventing duplicate requests, adding login and roles, sending notifications, and adding automated tests. With these improvements the system could support real community waste collection.

5. ACKNOWLEDGMENTS

The authors express their sincere gratitude to Sri Shakthi Institute of Engineering and Technology for providing the academic environment and facilities required for this project. The authors thank the project guide, I.A. JANATHUL FERTHOUS, for guidance, encouragement, and support throughout the development of the Smart Community Waste Management System. The authors also acknowledge the faculty members and contributors who supported the learning and implementation process.

6. REFERENCES

[1] Kaza, S., Yao, L. C., Bhada-Tata, P., & Van Woerden, F.; What a Waste 2.0: A Global Snapshot of Solid Waste Management to 2050, 2018, World Bank, Washington, DC.

[2] World Health Organization; Waste and Human Health: Evidence and Needs, 2015, World Health Organization, Geneva, Switzerland.

[3] Folianto, F., Low, Y. S., & Yeoh, W.; Smartbin: Smart Waste Management System, 2015, IEEE Tenth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP).

[4] United Nations Environment Programme; Global Waste Management Outlook 2024: Beyond an Age of Waste – Turning Rubbish into a Resource, 2024, UNEP.

[5] Arduino; Arduino Documentation, Arduino Official Documentation.

[6] Espressif Systems; ESP32 Technical Reference Manual, Espressif Systems.

[7] Flask Documentation; Flask Web Development Framework Documentation, Pallets Projects.

[8] Oracle Corporation; MySQL Reference Manual, Oracle Corporation.

[9] PyMySQL Contributors; PyMySQL Documentation, Python MySQL Client Library.

[10] Chart.js Contributors; Chart.js Documentation, JavaScript Data Visualization Library.

[11] OpenStreetMap Contributors; OpenStreetMap Documentation, OpenStreetMap Foundation.

[12] Leaflet Contributors; Leaflet Documentation, JavaScript Interactive Mapping Library.

[13] Python Software Foundation; Python Documentation, Python Official Documentation.

[14] World Bank; Solid Waste Management and Sustainable Urban Development Resources, World Bank Publications.

[15] Node.js, “Node.js Documentation.” Available: https://nodejs.org/docs/latest/api/

[16]Node.js, “SQLite (node:sqlite),” Node.js API Documentation. Available: https://nodejs.org/api/sqlite.html

[17] Express, “Express – Node.js web application framework.” Available: https://expressjs.com/

[18] SQLite, “SQLite Documentation.” Available: https://www.sqlite.org/docs.html

[19]MDN Web Docs, “Fetch API.” Available: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

[20] npm, “cors,” npm package. Available: https://www.npmjs.com/package/cors

[21] R. T. Fielding, “Architectural Styles and the Design of Network-based Software Architectures,” Ph.D. dissertation, University of California, Irvine, 2000.

[22] S. Kaza, L. Yao, P. Bhada-Tata, and F. Van Woerden, “What a Waste 2.0: A Global Snapshot of Solid Waste Management to 2050,” World Bank, Washington, DC, 2018. doi: 10.1596/978-1-4648-1329-0.

[23] Project source code, “Smart Community Waste Management System,” supplied project archive, waste.zip. Includes server.js, package.json, Frontend/index.html, Frontend/style.css, Frontend/script.js, and waste.db.