# [DEBRIS MANAGEMENT SYSTEM]

ON

Submitted in partial fulfillment of the requirements
of the degree of

Bachelor of Engineering
(Information Technology)

By

JAPLEEN KAUR ARORA - Roll No (03)

Under the guidance of

MRS . RUPALI LOHAR

Department of Information Technology

VIVEKANAND EDUCATION SOCIETY'S INSTITUTE OF TECHNOLOGY, Chembur, Mumbai 400074

(An Autonomous Institute, Affiliated to University of Mumbai)

# Abstract

Keeping our environment clean, especially beaches, is important. Sometimes trash (debris) washes up or gets left behind. This project creates a computer program (a web application) to help manage this cleanup process. Instead of drones cleaning the debris themselves, which is complicated, our drones simply fly around, spot the debris, and report its location and estimated amount. Our web application then takes these reports and helps decide which cleanup truck depot is closest and has trucks available to send for cleanup. The system also makes sure only authorized people (Admins) can dispatch trucks, while others (Users) can only watch the progress. We used common web technologies like Node.js, MySQL, HTML, and JavaScript to build this tool.



**Keywords:** Debris Management, Dispatch System, Node.js, MySQL, Web Application, Nearest Neighbor Algorithm, Authentication, JWT.

# Contents

# 1. Introduction

Trash on beaches and in public spaces is a growing problem. Cleaning it up efficiently requires coordination. Imagine using drones to quickly find trash hotspots. This project builds a web-based tool to manage the information collected by these drones.

Our system, the "Debris Detection & Dispatch System," acts as a central dashboard. Drones (simulated in our project) report where they found debris. Operators using our website can see these reports in real-time. The most important part is that the system helps figure out the best (closest) truck depot to send a cleanup crew from.

We built this using standard tools for web development:

- Node.js with Express.js for the backend (the "engine" running on a server).
- MySQL as the database (where all the information is stored).
- HTML, CSS (Tailwind), and JavaScript for the frontend (the website pages you interact with).

## 2. Problem Statement

Consider the scenario of utilizing drones for environmental surveillance, specifically for identifying debris accumulations in areas such as beaches. While drones excel at detection and reporting location coordinates, the subsequent step of initiating physical cleanup requires manual intervention and coordination. Without a centralized system, managing these incoming drone reports becomes inefficient. This lack of organization can lead to operational confusion regarding cleanup status, potential duplication of efforts (dispatching multiple teams to the same location), and missed debris sites, ultimately resulting in wasted time and resources, including fuel.

The main challenge is choosing the best truck to dispatch.This requires knowing where the debris is, where the depots are, and how many trucks each depot has ready. We need a smart, automated way to make this decision based on location.

Not everyone involved in the cleanup operation should have the authority to dispatch vehicles. We need a secure system that understands different roles. Managers or supervisors (Admins) should be able to dispatch trucks, while other staff (Users) should only be able to view the reports and monitor the situation. This prevents mistakes and ensures resources are used correctly.

Challenges Addressed:

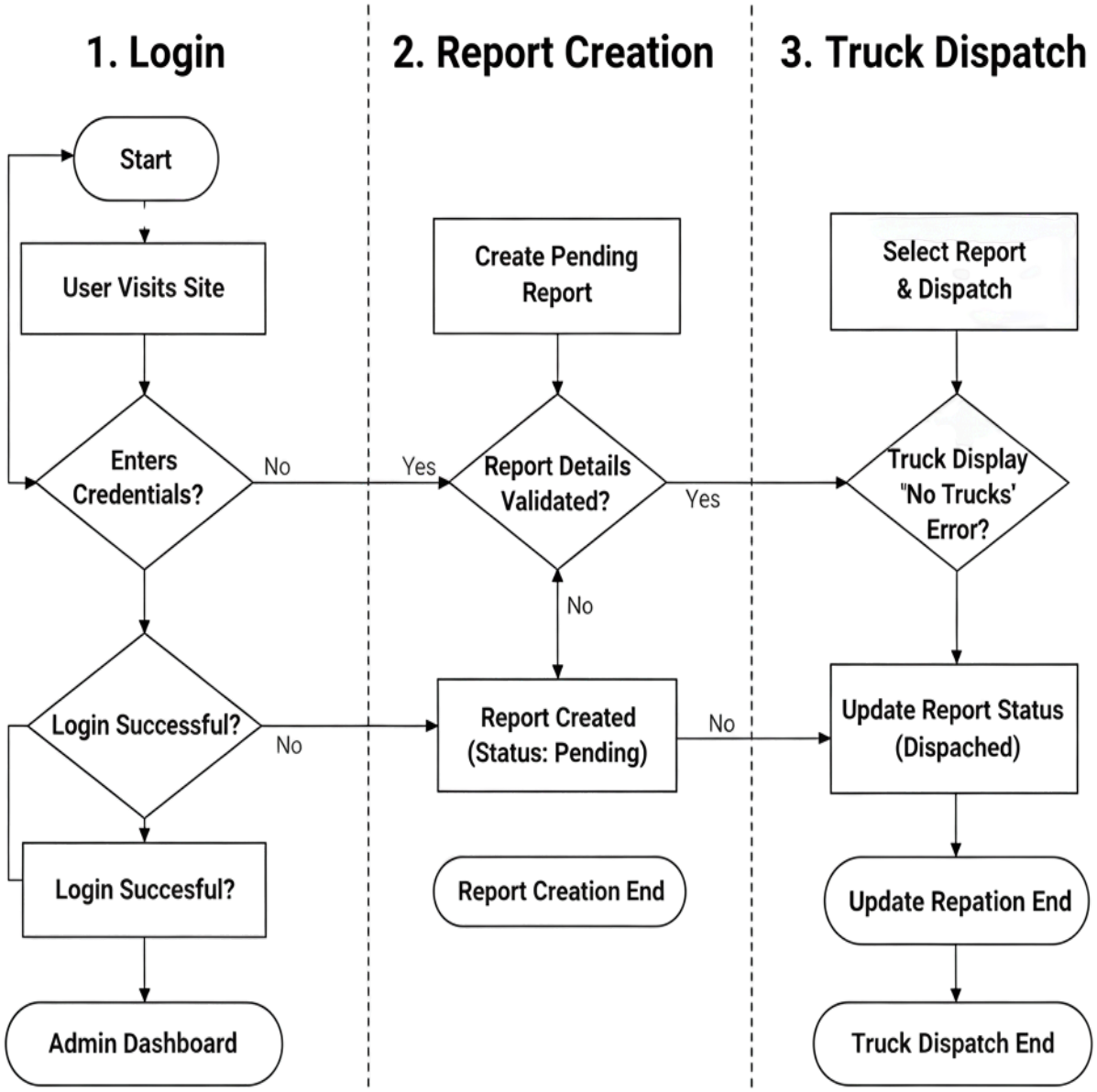
- We need a central place to see all reported debris locations.
- We need a smart way to decide which truck depot should handle each report based on location and truck availability.
- We need different access levels: some people should only be able to view the situation (Users), while others need the authority to send out trucks (Admins).

## 3.Objectives

The main goals of this project were to:

- **Build a Web Dashboard:** Create a user-friendly website to display debris reports and drone status.
- **Implement Secure Login:** Create a login system that differentiates between 'Admin' and 'User' roles using secure methods (JWT and password hashing).
- **Display Real-time Reports:** Show a list of debris reports fetched from the database, including status (Pending, Dispatched, Completed).
- **Develop a Dispatch Algorithm:** Implement a simple but effective algorithm (Nearest Neighbor) to find the closest available truck depot for a given debris report.
- **Enable Admin Actions:** Allow logged-in Admins to trigger the dispatch algorithm and simulate new drone reports via a web form.
- **Ensure Data Integrity:** Validate user inputs (like Site ID, Drone ID) on the backend before creating new reports to prevent errors.

# 4. Proposed Design (Flow chart)

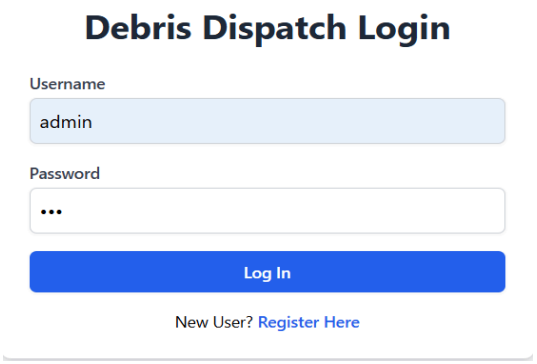


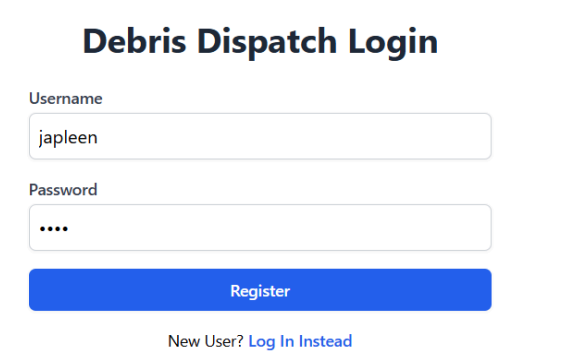
# 5. Implementation (Logic)

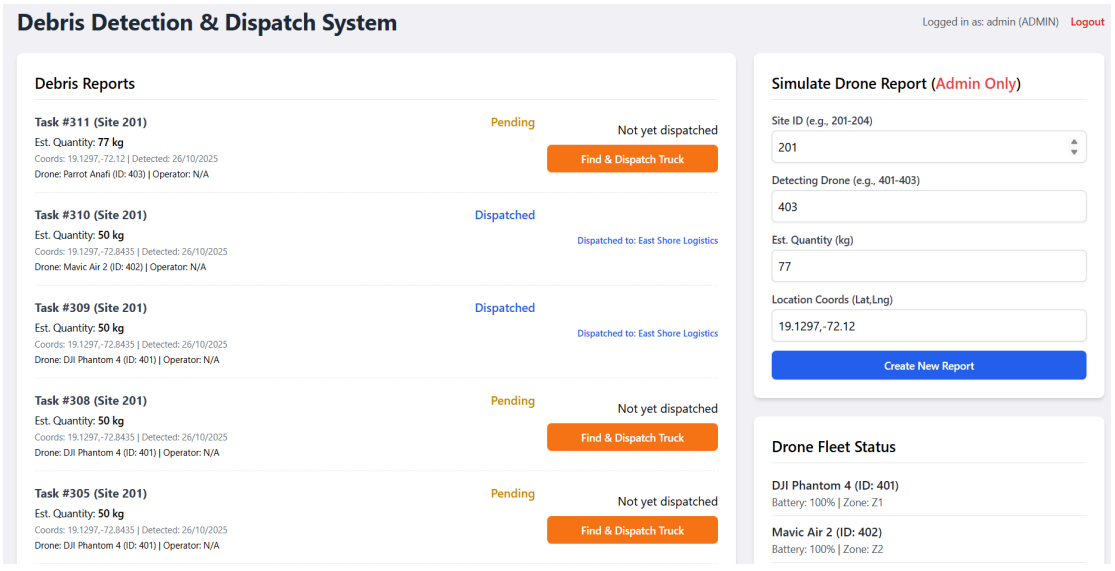
System was built using three main parts:

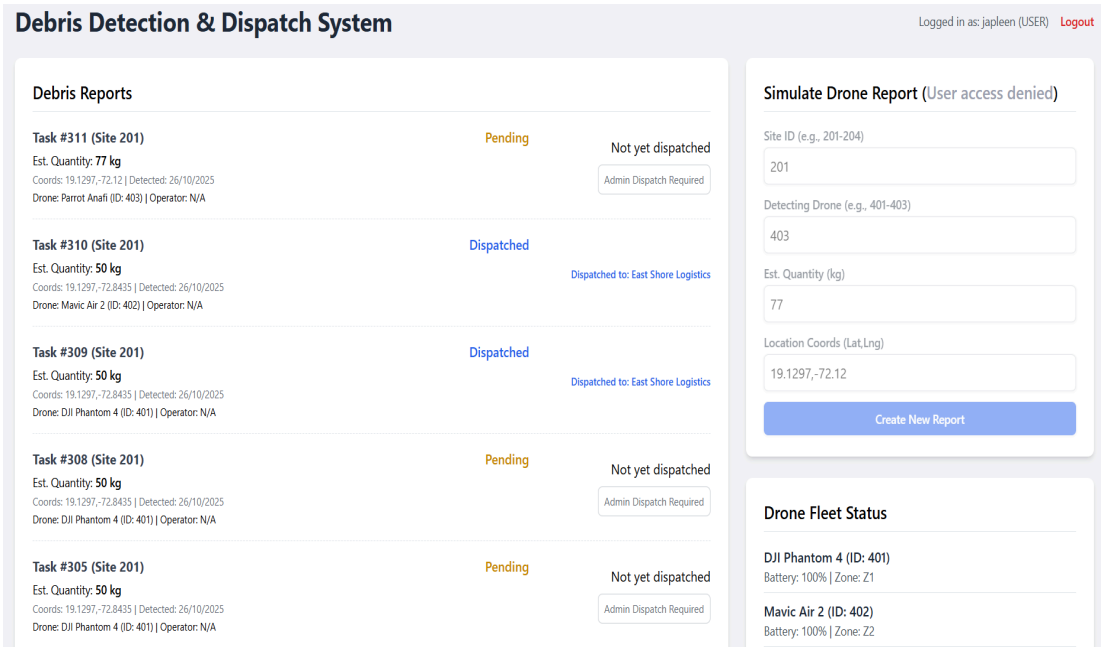
## A. The Frontend-

- index.html: This file defines the structure of the website – the login page and the main dashboard layout (using HTML tags). We used Tailwind CSS to make it look clean and modern quickly.
- app.js: This is the JavaScript code that makes the website interactive.
    - It handles showing/hiding the login form vs. the dashboard.
    - It manages the Login/Register process by sending username/password to the backend.
    - It saves the security token (JWT) in the browser's localStorage after login.
    - It uses the fetch command to make secure API calls to the backend (like getting reports or dispatching trucks), always sending the saved token for verification.
    - It takes the data received from the backend (in JSON format) and dynamically creates the report cards and drone list you see on the page.
    - It handles button clicks (Login, Register, Logout, Dispatch, Create Report).






## B. The Backend-

- server.js: This Node.js file is the core brain of the application.
    - It uses the Express.js framework to create a web server and define API routes (the URLs the frontend calls, like /api/reports or /api/dispatch/:taskId).
    - It connects securely to the MySQL database using the mysql2/promise library and credentials stored safely in the .env file.
    - Authentication: It handles user registration (hashing passwords with bcryptjs) and login (checking passwords, creating. It includes special checkAuth and checkAdmin functions (middleware) that act like security guards, protecting routes so only logged-in users or admins can access them.
    - The Algorithm: It contains the POST /api/dispatch/:taskId route where the Nearest Neighbor logic runs. This includes the parseCoords and getEuclideanDistance helper functions to calculate distances between the debris location and truck depots.
    - Database Interaction: It runs all the necessary SQL queries (SELECT, INSERT, UPDATE) to fetch data, create reports, and update statuses when a truck is dispatched. It also includes validation logic to check if Site IDs and Drone IDs exist before creating reports.

## C. The Database-

- MySQL: We used MySQL to store all the project's information in tables.
- Key Tables:

    - User: Stores login usernames, hashed passwords, and roles ('admin' or 'user').
    - DebrisTask: Stores each reported debris incident (now acts as the "Report"), including its location, estimated quantity, status ('Pending', 'Dispatched', 'Completed'), and which depot it was assigned to.
    - TruckDepot: Stores the locations of the truck depots and how many trucks are currently available.
    - Other tables (Zone, CollectionSite, Drone, Operator, Supervisor, etc.) store related information needed for the reports.
- debris_db_setup.sql: This script contains all the commands to create the database, build the tables correctly, and add sample data to start with.

## 6. Results and Analysis

The system was tested thoroughly, confirming all objectives were met.

- Successful Login & Role Separation: Users logging in as 'admin' could access all features, including the dispatch button and report creation form. Users logging in with a standard 'user' role could only view data; the dispatch button was replaced with text, and the report form was visibly disabled.
- Real-time Data Display: The dashboard correctly loaded and displayed data from the database upon login, including pending, dispatched, and completed reports, along with drone statuses.
- Dispatch Algorithm Works: Clicking "Find & Dispatch Truck" on a pending task successfully triggered the backend algorithm. Testing confirmed:
    - The task status updated to 'Dispatched' on the dashboard.
    - The correct (closest) TruckDepot name was displayed.
    - Checking the database directly showed the DebrisTask.Status and DebrisTask.AssignedDepotID were updated correctly.
    - Checking the TruckDepot table confirmed the TrucksAvailable count decreased for the assigned depot. [Image showing a task before and after dispatch]
- Report Creation & Validation Works: Using the "Simulate Drone Report" form (as Admin) successfully created new tasks that appeared on the dashboard. Entering invalid SiteIDs or DroneIDs correctly resulted in error messages from the backend validation, preventing bad data from entering the system. [Image showing the success alert after creating a report] [Image showing the error alert after trying to create a report with invalid ID]

**Analysis:** The tests demonstrate that the full-stack integration is successful. The authentication system correctly enforces roles, the backend API provides data securely, and the core dispatch algorithm functions as designed by selecting the nearest available resource and updating the database state. The system provides a working proof-of-concept for managing debris cleanup dispatch.

## 7. Conclusion

This project successfully developed a Debris Detection & Dispatch System as a full-stack web application. It addresses the need for a centralized platform to manage debris reports and efficiently assign cleanup resources using a location-based algorithm. The implementation of secure authentication and role-based access control ensures that system operations are managed appropriately.

The use of Node.js, Express, MySQL, and vanilla JavaScript provides a robust and scalable foundation. The Nearest Neighbor algorithm, while simple, provides an effective solution for the defined problem of dispatching the closest available truck.

**Future Enhancements:**

- Integrate with real drone APIs for automatic report generation.
- Implement a more sophisticated routing algorithm (considering traffic, road networks).
- Add features for tracking truck progress and marking tasks as 'Completed'.
- Develop a map interface to visualize debris locations and depot positions.

## 8. References


- MySQL: Official Documentation - https://dev.mysql.com/doc/
- GeeksforGeeks: Node.js Authentication using JWT - https://www.geeksforgeeks.org/node-js-jwt-authentication/
- Full Stack Web Development Tutorials-https://www.youtube.com
- GeeksforGeeks: Node.js Tutorial - https://www.geeksforgeeks.org/nodejs/
- GeeksforGeeks: MySQL Tutorial https://www.geeksforgeeks.org/mysql/
- Google Gemini: https://gemini.google.com
- Chat GPT:https://chatgpt.com/