Design Documentation – ReMat: E-Waste Bin Scheduling System

1. Problem Statement

To develop a website for improper disposal of electronic waste (e-waste) leads to environmental pollution, health risks, and inefficient recycling processes due to:

- Inefficient Logistics: Collection trucks often visit empty bins or miss overflowing ones, leading to high fuel consumption and operational costs.
- Lack of Transparency: Users are unaware of what happens to their waste or where to dispose of it properly.
- Low Motivation: There is no immediate incentive for users to segregate and recycle e-waste responsibly.

2. Proposed Solution

ReMat is an end-to-end smart waste management ecosystem designed to bridge the gap between waste generators (users) and waste collectors (administrators).

- Users looking for nearby e-waste bins and rewards
- Smart bins that identify deposited items using AI
- Admins/municipal authorities who monitor bins and plan pickup routes

The system ensures efficient bin usage, optimized pickup scheduling, and user engagement through gamification.

3. User Flow

- Entry Points: Landing → public info and CTA; Auth → sign in / sign up; Deep Links → feature URLs (e.g., shared challenges).
- Authentication: Auth → email/OAuth → on success redirect to UserDashboard or AdminDashboard based on role.
- New User: Landing → Auth → Onboarding (optional tutorial) → UserDashboard.
- Returning User: Auth → UserDashboard (shows notifications, recent activity).
- User Core Flow: UserDashboard → access features:
    - Bin Finder: search by location → view map / directions → save favorites.
    - Waste Detection: upload/take image → analysis → receive disposal recommendations.
    - Scheduling System: create/manage pickups or reminders.
    - Community Challenges: view/join challenges → submit progress → earn badges.
    - Educational Content: browse topics → mark complete / bookmark.
    - Social Sharing / Notifications: share results or get push notifications.
- Admin Flow: AdminDashboard → UserManagement, Analytics, BinMonitoring, AlertsPanel → manage users, review metrics, monitor bins, send alerts.
- Settings & Profile: accessible from dashboards → update preferences, theme, notification settings, account.
- Offline Behavior: app shows OfflineBanner → queue actions (e.g., submissions) and sync when back online.
- Error & Edge Handling: clear error states, retry actions, explain permission requirements (camera, location).
- Permissions: request location for Bin Finder, camera/media for Waste Detection, notifications for push.

- Security & Privacy: role-based redirects, minimal data retained locally, consent for image uploads.
- Telemetry & Analytics: track feature usage, error rates; admin views surface actionable metrics.

4. Technical Approach

The system follows a client - server architecture:

- Client Layer:
  - User Mobile Application
  - Smart Bin Interface (AI simulation)
  - Admin Dashboard
- Server Layer:
  - Node.js backend handling APIs and business logic
- Data Layer:
  - MongoDB database for persistent storage

All clients communicate with the backend using RESTful APIs.

5. Technology Stack

Frontend

- User App: React Native (Expo)
- Admin Dashboard: React.js (Vite) + Tailwind CSS
- Bin Interface: React.js (Tablet-optimized)

Backend

- Node.js (API logic)
- MongoDB (users, bins, transactions)

AI / ML

- Google Teachable Machine (model training)
- TensorFlow.js (in-browser inference)

6. Database Design (MongoDB)

Collections

- users { uid, name, points, co2_saved, history[] }
- bins { binId, lat, lng, fillLevel, status, acceptedTypes[] }
- transactions { timestamp, userId, binId, itemType, weight }

7. Algorithms Used

A. Object Detection (Computer Vision)

We employed Transfer Learning using a pre-trained MobileNet architecture.

- Training: We finetuned the top layers of the network using the Teachable Machine platform on 5 distinct classes: Smartphone, Laptop, Battery, Cables, and Background.
- Execution: The model runs client-side using TensorFlow.js, ensuring privacy (images aren't sent to the server) and low latency.
- Trust Heuristic: We display the confidence score and detected label to the user to build trust in the AI's decision.

B. Route Optimization Logic

We used a Greedy Nearest-Neighbor Routing Algorithm for bin pickup scheduling.

- Filter all bins whose fill level is above a threshold:
  $$P = \{ b \in B \mid f(b) \geq 75\% \}$$
- Starting from the depot, the system repeatedly selects the nearest bin (distance calculated using the Haversine formula) and adds it to the pickup route.
- This continues until all selected bins are covered.

8. AI Model Simulation (Object Detection)

Model Used:

- COCO-SSD (@tensorflow-models/coco-ssd@2.2.2)
- A lightweight real-time object detection model that runs directly in the browser using TensorFlow.js.

Dataset Used:

- COCO Dataset (Common Objects in Context)
- Contains 80 object categories, including people, animals, vehicles, and common electronic/kitchen items.

Implementation in Our System:

- The model performs real-time detection through the device camera.
- Although COCO-SSD supports 80 classes, our application filters predictions using a predefined list of valid categories:
  $$TARGET\_CLASSES = \{ \text{cell phone, laptop, mouse, keyboard, etc.} \}$$
- Only objects belonging to TARGET_CLASSES are treated as valid e-waste items, ensuring the kiosk focuses on relevant detections.

9. Design Decisions

- **React Native + Expo** was chosen to enable rapid cross-platform mobile development and quick deployment during hackathon timelines.
- **React.js (Vite) + Tailwind CSS** was used for the Admin Dashboard and Bin Interface to ensure a clean UI, fast performance, and reusable components.
- **Node.js** backend was selected to efficiently handle real-time API requests, transaction logging, and scheduling logic with minimal overhead.
- **MongoDB** was used due to its flexible document-based schema, making it easy to store dynamic bin data, user reward history, and transaction logs without complex migrations.
- **Client-side AI inference** (TensorFlow.js) was implemented to reduce latency and improve privacy, since camera frames are processed locally without being sent to the server.
- **A Map-first interface** was prioritized for both user and admin flows, minimizing navigation steps and making bin discovery intuitive.
- **Gamification** (points, CO_2 saved, rewards) was added to encourage consistent and responsible e-waste disposal behavior.
- A **modular architecture** was followed, keeping user app, admin dashboard, bin interface, and backend loosely coupled for easier scaling and maintenance.