**PRESENTATION FOR ASSET MANAGEMENT SYSTEM**

**1. Project Structure & Stack**

* **Backend**: Django (with Django Rest Framework for APIs)
* **Database**: PostgreSQL
* **Frontend**: Django templates + Bootstrap (liquid glass style), with optional Vue.js/React for dynamic parts
* **QR/Barcode**: qrcode (Python) for generation, quaggaJS (JS) for scanning
* **PWA**: Service Worker + Manifest for offline support
* **File Storage**: Local (with abstraction for future cloud)
* **Reporting**: django-import-export, reportlab (PDF), openpyxl (Excel)
* **Theming**: Admin-configurable, using your provided palette/logo

**2. Key Apps & Features**

**a. Core Django Apps**

* users (custom user model, roles, dashboards)
* assets (asset CRUD, dynamic fields, QR, attachments)
* categories (admin-defined asset types/fields)
* audit (logging, activity tracking)
* reports (export, dashboard, scheduled jobs)
* settings (branding, theming, backup/restore)

**b. Models Overview**

* **User**: Extends Django user, with role field
* **AssetCategory**: Name, dynamic fields (JSON)
* **Asset**: Category, dynamic fields (JSON), status, QR, images, docs, assigned user, etc.
* **AuditLog**: User, action, asset, timestamp, details
* **Attachment**: Linked to Asset, file, type, size
* **ThemeSetting**: Colors, logo, etc.

**3. Implementation Plan**

**Step 1: Project Setup**

* Create Django project & apps
* Configure PostgreSQL, static/media, custom user model

**Step 2: User & Role Management**

* Implement roles: Admin, Manager, User
* Custom dashboards per role
* Secure login/logout, permissions

**Step 3: Asset & Category Management**

* Admin can create categories with dynamic fields (JSON schema)
* Asset model supports dynamic fields, images, docs, QR code
* Asset registration auto-generates QR code (PNG/SVG)
* Asset CRUD with permissions, approval workflow

**Step 4: QR Code Generation & Scanning**

* Generate QR on asset creation (backend)
* Print label option (PDF)
* Scanning interface (mobile-friendly, quaggaJS for browser camera)
* Manual search fallback

**Step 5: Attachments & Status**

* Image/document upload (with restrictions)
* Real-time status updates, editable descriptions

**Step 6: Audit Logging**

* Log all asset views, edits, moves, user activity, errors

**Step 7: Dashboard & Reporting**

* Real-time dashboard (asset counts, scan logs, movement, status)
* Export reports (PDF, Excel), on-demand and scheduled (future-ready)

**Step 8: Offline & PWA**

* Service Worker for offline asset lookup, sync logic
* Manifest for installable PWA
* Local storage for offline queue

**Step 9: Bulk Import/Export**

* CSV/Excel import for assets
* Export with field selection

**Step 10: Theming & Branding**

* Admin UI for color/logo upload (using your palette/logo)
* Dynamic Bootstrap theming

**Step 11: Backup & Restore**

* Admin-triggered DB backup/restore

**4. Branding & UI**

* Use your provided color palette and logo
* Bootstrap with glassmorphism (liquid glass) style
* Responsive, mobile-first design

**5. Codebase Generation**

I will now generate the **core codebase** (models, admin, views, templates, static, PWA, QR, scanning, theming, etc.) in a modular, ready-to-run format.**You’ll get:**

* All models, forms, and admin
* Key views and templates
* Static assets (logo/colors)
* JS for QR scanning (quaggaJS)
* PWA manifest & service worker
* Example dashboard/reporting logic
* Theming system