

Project Report: NGO Registration and Donation Management System

1. Introduction & Project Scope

The primary objective of this project is to develop a robust, secure, and transparent MERN-based platform for NGOs to manage two distinct operational flows: [User Registration](#) and [Campaign-based Donations](#). By separating these flows, the system ensures that user data remains consistent even if a specific donation attempt fails or is abandoned.

2. System Architecture

The application utilizes a [Three-Tier MERN Stack Architecture](#):

- **Presentation Tier (Frontend):** Built with **React.js**, this tier handles all user interactions. It uses **React Router** for protected navigation between user and admin views and **CSS3** for a responsive, dashboard-style UI.
- **Application Tier (Backend):** Powered by **Node.js** and **Express.js**, this tier manages the business logic, including **JWT (JSON Web Token)** authentication, Stripe session orchestration, and administrative data aggregation.
- **Data Tier (Database):** **MongoDB Atlas** provides a scalable NoSQL document store. It handles complex data relationships through **Mongoose**, such as linking donation records to unique user IDs.

3. Database Schema

We defined two primary collections to manage the system's data integrity:

3.1 User Collection

- **Purpose:** Stores authentication and profile data.
- **Fields:** `name`, `email`, `password` (stored as a salted hash), `phone`, and `role`.
- **Role-Based Access:** The `role` field is the primary discriminator used by backend middleware to permit or deny access to the **Admin Control Panel**.

3.2 Donation Collection

- **Purpose:** Tracks every transaction attempt and its outcome.
- **Fields:** `userId` (reference), `campaign` (string), `amount` (number), `status` (Enum: PENDING, SUCCESS, FAILED), `paymentId` (Stripe ID), and `createdAt`.
- **Audit Trail:** The `paymentId` ensures that every record in our database can be cross-verified against the **Stripe Dashboard**.

4. Technical Flow

4.1 Secure Authentication Flow

1. **Registration:** User provides details; the backend hashes the password using **bcrypt** before saving to MongoDB.
2. **Login:** User provides credentials; the backend verifies the hash and signs a **JWT** containing the user's ID and role.
3. **Authorization:** For every protected request, the frontend sends the JWT in the **Authorization** header, which the backend verifies using a **protected** middleware.

4.2 Stripe Payment Gateway Handshake

1. **Request:** The user selects a campaign (e.g., "Child Education Support") and enters an amount.
2. **Initialization:** The backend creates a **Stripe Checkout Session** and returns a unique **payment_url** to the frontend.
3. **Redirection:** The user is redirected to the **Stripe Sandbox** to enter test card details.
4. **Verification & Update:** Upon return, the backend retrieves the session, verifies the **payment_status**, and updates the database record from **PENDING** to **SUCCESS**.

5. Key Design Decisions & Assumptions

- **Decoupled Workflows:** We decided to save a **PENDING** record in the database *before* redirecting to Stripe. This ensures we can track abandoned checkouts and fulfill the "**Track payment status and timestamps**" requirement.
- **Automated State Management:** A background cleanup task runs every 10 minutes to expire **PENDING** donations older than 30 minutes. This decision prevents the **Admin Dashboard** from showing inflated "Pending" counts for users who simply closed the tab.
- **Client-Side Data Export:** We implemented the **CSV Export** feature using the browser's **Blob API**. This decision offloads processing from the server to the client, ensuring the dashboard remains performant even with thousands of records.
- **Assumption - Sandbox Reliability:** The system assumes the **Stripe Sandbox API** is the final source of truth for payment verification. Manual status overrides are not permitted to ensure audit integrity.

6. Quality Assurance, Security, and Optimization

This project utilizes a multi-layered approach to ensure a secure, functional, and high-performing system.

6.1 Security & Data Integrity

- **Password Protection:** **Bcrypt** with high salt-rounds ensures passwords remain cryptographically secure against rainbow table attacks.
- **Session Management:** **JWTs** with limited lifespans mitigate token theft risks, while their stateless nature enables horizontal cloud scaling.
- **Injection Prevention:** **Mongoose validation** and backend sanitization prevent NoSQL injection, ensuring data integrity during registration.

6.2 Functional Testing Protocols

- **RBAC Verification:** Confirmed that users with the `role: "user"` are strictly denied access to administrative routes, receiving a `403 Forbidden` error.
- **Payment Lifecycle:** Successfully used the **Stripe 4242 test card** to verify the full flow: Sandbox redirection, processing, and status updates from `PENDING` to `SUCCESS`.
- **Real-time Accuracy:** Verified that Admin summary cards and campaign filters update instantly to provide precise financial oversight.

6.3 Performance & Accessibility

- **Stateless Architecture:** By eliminating server-side sessions, the application tier remains lightweight and optimized for containerized deployment.
- **Responsive Design:** **CSS3 media queries** and React components ensure the dashboard is fully accessible across mobile and desktop browsers.

7. Conclusion

The **NGO Registration and Donation Management System** effectively addresses the need for a transparent and secure digital platform for donor management. By leveraging the **MERN stack**, the project achieves a scalable and robust architecture that ensures operational integrity for both users and administrators. This project provides a reliable foundation for NGOs to streamline their donation processes and build trust with their donor base through technical transparency.