**PROJECT REPORT**

**DONATION MANAGEMENT SYSTEM**



**Spring 2025**

**Database Management Lab**

Submitted by:

**Nouman Khan (22PWCSE2107)**

**Tariq Jamil (22PWCSE2184)**

**Mahran Khan (22PWCSE2185)**

ClassSection**: A**

Submitted to:

**Engr. Summayea Salahuddin**

July 08, 2025

**Department of Computer Systems Engineering**

**University of Engineering and Technology, Peshawar**

**Donation Management System**

**Introduction**

The Donation Management System is a web-based application developed using Laravel (PHP framework) with a MySQL backend. Its purpose is to streamline the process of managing donors, donations, and proof verification, providing administrators with an intuitive interface to track and report on charitable contributions. By automating core workflows—donor registration, donation submission with proof upload, TRXID verification, and donation record maintenance—the system aims to reduce manual errors, improve data integrity, and enhance transparency.

**Project Objectives**

To design and implement a robust, secure, and user-friendly Donation Management System that automates donor registration, donation submission, proof verification, and reporting, thereby facilitating efficient administration of charitable contributions.

**Significance & Benefits**

* **Operational Efficiency:** Automates workflows that are traditionally manual (e.g., paper-based proof collection), reducing administrative overhead.
* **Data Integrity:** Enforces schema constraints and validation rules to prevent duplicate or malformed entries.
* **Transparency:** Maintains an audit trail of donations and proof verifications, fostering trust among stakeholders.
* **Scalability:** Built on Laravel and MySQL, the system can handle growing donor bases and campaign portfolios.
* **Reporting:** Real-time dashboards and export functionality support informed decision-making and regulatory compliance.

**Technologies Used**

**1. Backend Framework**

**Laravel (PHP)**

* Modern PHP web application framework
* Handles routing, controllers, models, migrations, authentication, and more

**2.Programming Language (PHP)**

* Primary language for backend logic (Laravel)

**3. Frontend**

**Blade Templating Engine**

* Laravel’s built-in templating system for rendering HTML views

**HTML5, CSS3, JavaScript**

* Standard web technologies for UI

**Tailwind CSS**

* Utility-first CSS framework (see `tailwind.config.js`)

**4. Databases**

* MySQL for development
* SQLite for production, as seen in `database/database.sqlite’.
* Lightweight, file-based database

**System Architecture**

The Donation Management System is built on the Model-View-Controller (MVC) pattern, which cleanly separates concerns into three interconnected components:

1. **Model**

Responsible for all data-related logic and business rules:

* Stores and retrieves donor profiles, authentication credentials, and contact info.
* Persists each donation’s metadata: date, amount, TRXID, campaign (if any), and verification status.
* Manages file uploads (images/PDFs), including storage paths, file type/size validation, and cleanup.
* Ensures a donor cannot submit more than one application for the same campaign or donation drive.
* Validates that TRXIDs are unique and properly formatted before acceptance.

1. **View**

Renders the user interface using Blade templates, CSS, and JavaScript:

* Homepage and donation landing pages listing active campaigns and donation instructions.
* Forms for submitting new donations (with file-upload widget).
* History pages showing past donations and their verification status.
* Tables of pending donations awaiting TRXID verification.
* Summary widgets (total donated, donations per campaign, pending vs. approved).
* Login, registration, password-reset forms, all styled responsively for desktop and mobile.

1. **Controller**

Acts as the intermediary between Model and View, handling HTTP requests and orchestrating responses:

* Routes defined in web.php and api.php determine which controller method responds to each URL or API call.
* Fetches and validates input (e.g. donation amount, TRXID format).
* Calls Model methods (Eloquent ORM) to read or write data.
* Returns Blade views populated with data for GET requests.
* Redirects or returns JSON for POST requests, along with success/error messages.
* Catches validation failures, file-upload errors, and database exceptions, returning user-friendly feedback.

**Server for Deployment**

**1.Deployment Platform: Railway**

* Railway is a modern cloud platform that allows you to easily deploy web applications, databases, and services.

**2.GitHub Integration:**

* Our Laravel project is connected to a GitHub repository.
* Railway automatically deploys our application whenever we push changes to the main branch (or a specified branch) on GitHub.
* This enables continuous deployment (CI/CD) with minimal manual intervention.

**Database Design and Implemenatation**

**Entity Description:**

|  |  |
| --- | --- |
| **Entity** | **Description** |
| **DONOR** | A person or organization that makes one or more donations to the system. |
| **DONATION** | A record of a gift (money or items) given by a Donor, including amount, date, payment mode, and purpose. |
| **DONATION\_PROOF\_PENDING** | A record of the donations made by the donors which are not yet approved by the ADMIN\_USER |
| **FUNDRAISING\_EVENT** | An organized campaign or event held to collect donations toward a particular cause, with a target amount and date/location. |
| **EVENT\_DONATION** | A linking (junction) record that associates a specific Donation with a specific Fundraising Event. |
| **DISTRIBUTION\_REPORT** | A summary of how and when donated funds or items were disbursed or allocated to beneficiaries. |
| **VOLUNTEER\_DISTRIBUTE** | A person who is tasked up to help with distributing donations, tracked by VolunteerDistID, loginID contact info and role etc. |
| **VOLUNTEER\_TASK** | A person who is tasked up to help with events, tracked by VolunteerEvtID, loginID contact info and role etc. |
| **ADMIN\_USER** | A system user with elevated privileges to manage donors, events, donations, volunteers, and distribution reports. |

**Business Rules:**

**1. User Roles:**

* Admin: Manages donors, donations, events, volunteers, and verifies donations.
* Donor: Registers, logs in, donates, and submits proof of donation.
* Volunteer: Registers, logs in, and is assigned to events or distribution tasks.

**2. Donor Management:**

* Donor Registration: Donors can register and log in.
* Donor-Donation Relationship: One-to-Many One donor can make many donations.
* Admin-Donor Relationship: One-to-Many One admin can manage many donors.

**3. Donation Management**

* Donation Submission: Donors submit donations, each requiring proof.
* Donation-Donor Relationship: Many-to-One Many donations belong to one donor.
* Donation Proof: One-to-One: Each donation has one proof of donation (pending until verified).
* Admin-Donation Relationship: One-to-Many: One admin can verify many donations.

**4. Event Management**

* Event Creation: Admins create and manage fundraising events.
* Event-Volunteer Relationship: Many-to-Many: An event can have many volunteers as staff, and a volunteer can serve at many events (via `volunteer\_eventstaff` table).

**5. Volunteer Management**

* Volunteer Registration: Volunteers can be registered by admin and they can log in.
* Volunteer Assignment: Many-to-Many Volunteers can be assigned to many distribution tasks, and a distribution task can have many volunteers (via `volunteer\_distribute` table).
* Admin-Volunteer Relationship: One-to-Many One admin can manage many volunteers.

**6. Distribution Management**

* Distribution Task Assignment: Many-to-Many Distribution tasks can have many volunteers, and volunteers can be assigned to many tasks.
* Distribution Completion: Each assignment can be marked as completed.

**7. Authentication & Authorization**

* User-Profile Relationship: One-to-One Each user has one profile.
* Admin Authentication: Protected by middleware.

**8. Email Notifications**

* Triggered by Events: Donation approval, donor registration, volunteer creation, and task assignment trigger emails.

**9. Data Integrity & Security**

* Access Control: Only admins can manage (create, update, delete) donors, donations, events, and volunteers.
* Donors and volunteers can only access their own data.

**10. Reporting & Accountability**

* Admin-Report Relationship: One-to-Many Admin can view many reports (on distributions, donations, donors, volunteers).

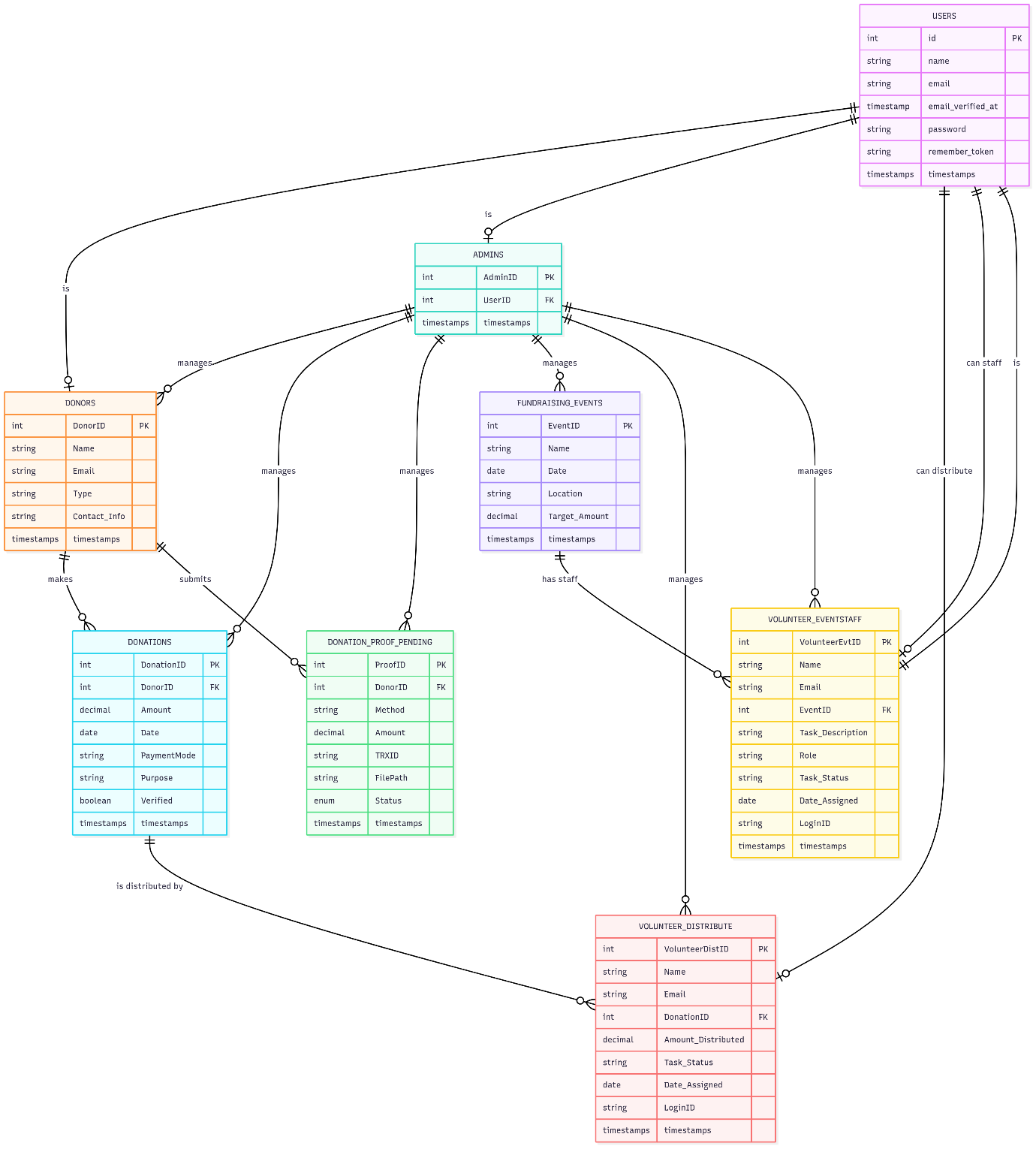
**11. Public Pages**

* Accessible to All: Home, About, Contact, and Welcome pages.

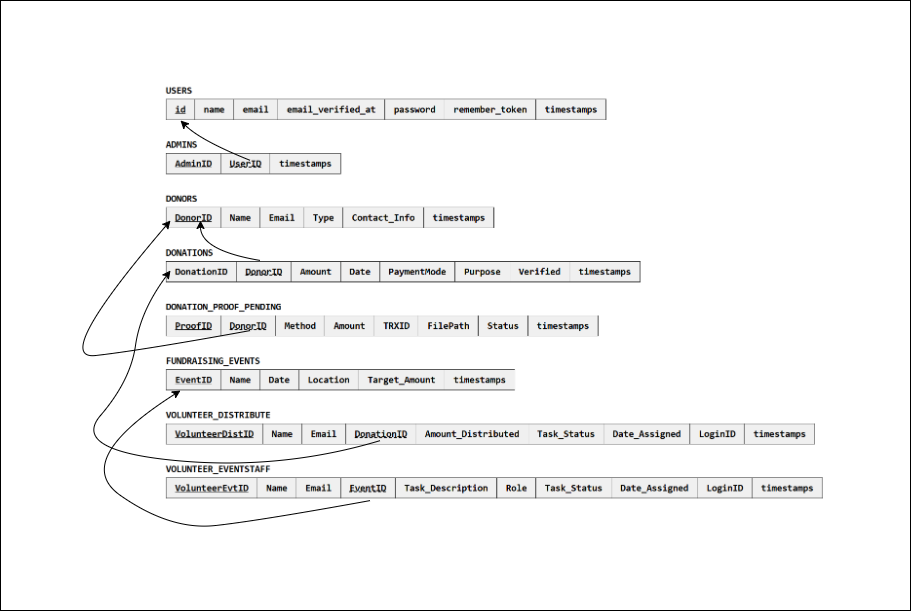
**13. Entity Relationship Cardinality Table**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Entity 1** | **Relationship** | **Entity 2** | **Cardinality** | **Description** |
| Donor | makes | Donation | 1-to-Many | One donor can make many donations |
| Donation | belongs to | Donor | Many-to-1 | Each donation is made by one donor |
| Donation | has | Proof | 1-to-1 | Each donation has one proof |
| Admin | manages | Donor | 1-to-Many | One admin manages many donors |
| Admin | verifies | Donation | 1-to-Many | One admin verifies many donations |
| Admin | creates | Event | 1-to-Many | One admin creates many events |
| Event | has staff | Volunteer | Many-to-Many | Events have many volunteers as staff |
| Volunteer | assigned to | Distribution Task | Many-to-Many | Volunteers assigned to many distribution tasks |
| User | has | Profile | 1-to-1 | Each user has one profile |

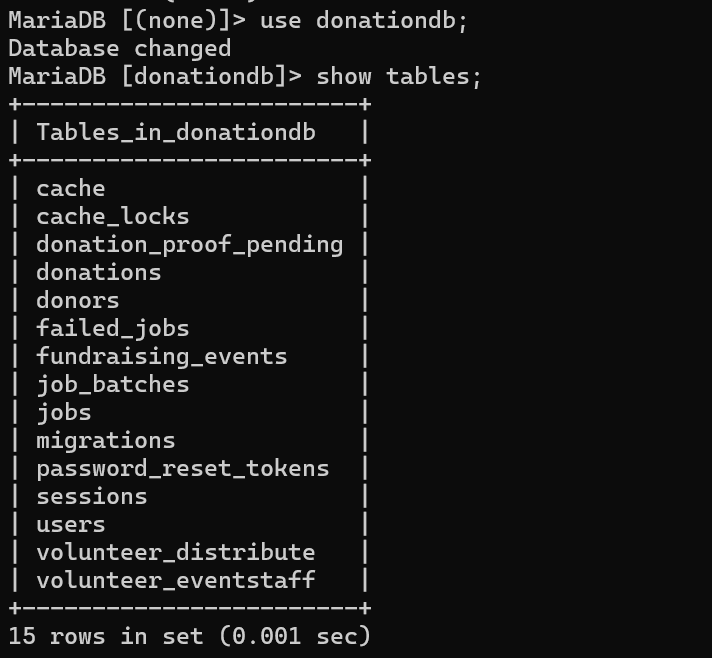
**Conceptual Schema (ERD):**

****

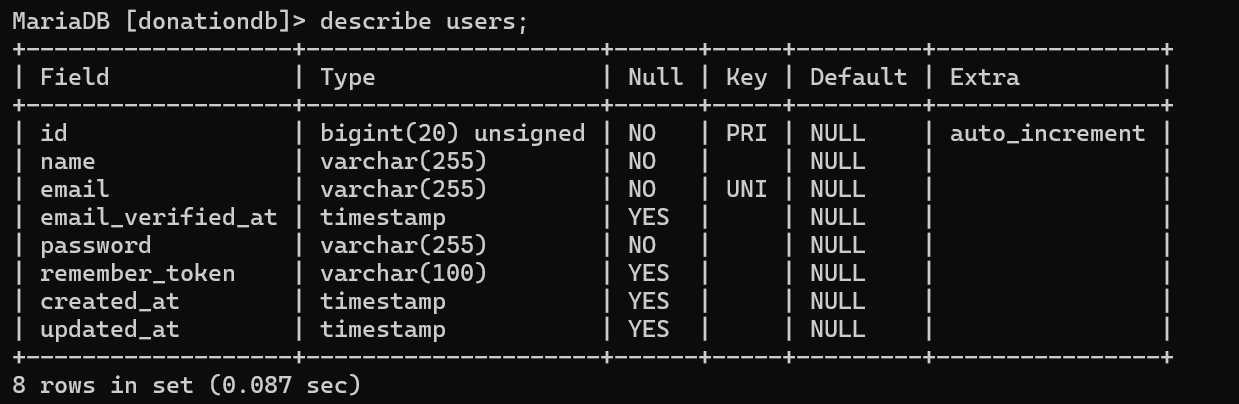
**Relational Schema:**

****

**MySQL database:**

****

**1. Users Table**

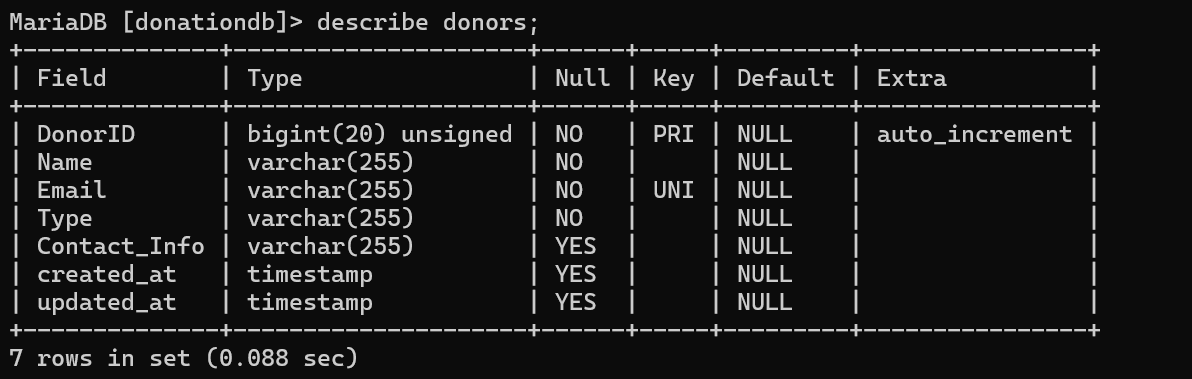
****

**Purpose:** Stores all user accounts (admins, donors, volunteers, etc.)

**Key Columns:**

* id (Primary Key)
* name
* email (unique)
* email\_verified\_at (nullable)
* password
* remember\_token
* created\_at
* updated\_at

**2. Donors Table**

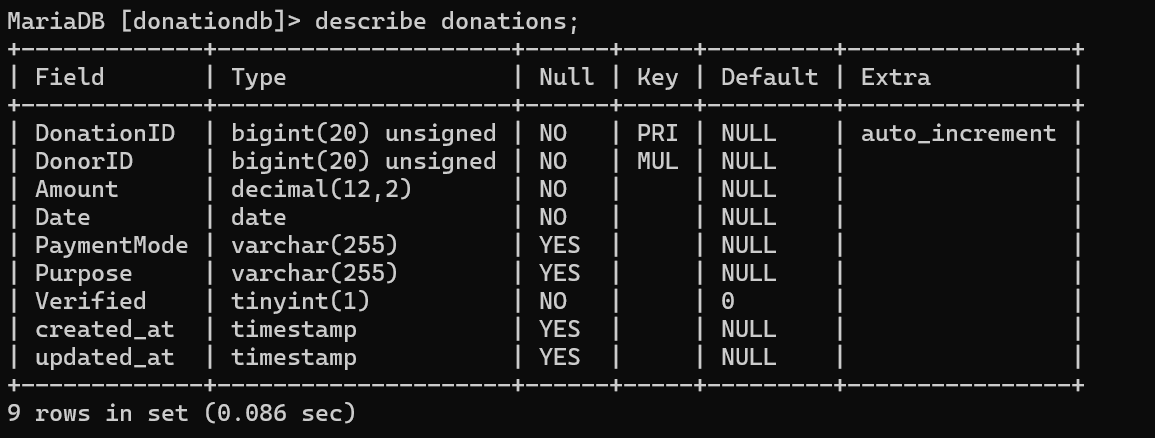
****

**Purpose:** Stores information about donors.

**Key Columns:**

* DonorID (Primary Key)
* Name
* Email (unique)
* Type (Individual or Organization)
* Contact\_Info (nullable)
* created\_at
* updated\_at

**3. Donations Table**

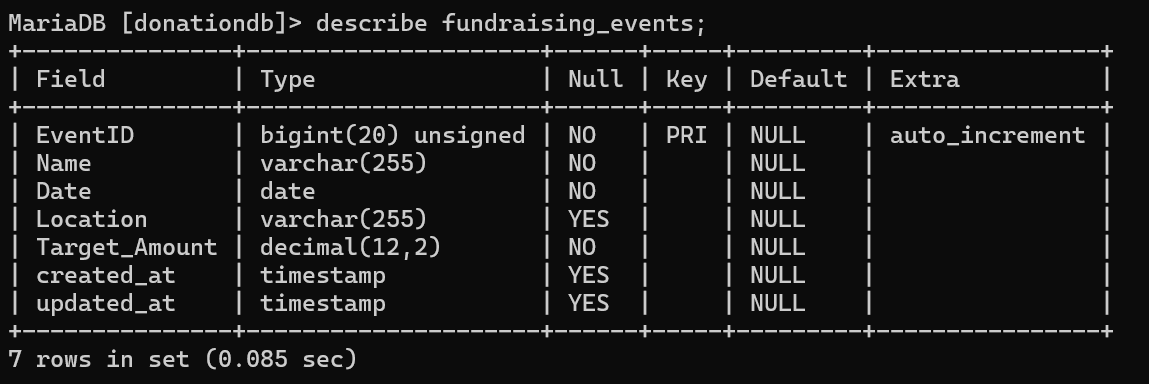
****

**Purpose:** Stores donation records.

**Key Columns:**

* DonationID (Primary Key)
* DonorID (Foreign Key to donors)
* Amount (decimal)
* Date
* PaymentMode (nullable)
* Purpose (nullable)
* Verified (boolean, default false)
* created\_at
* updated\_at

**4. Fundraising Events Table**

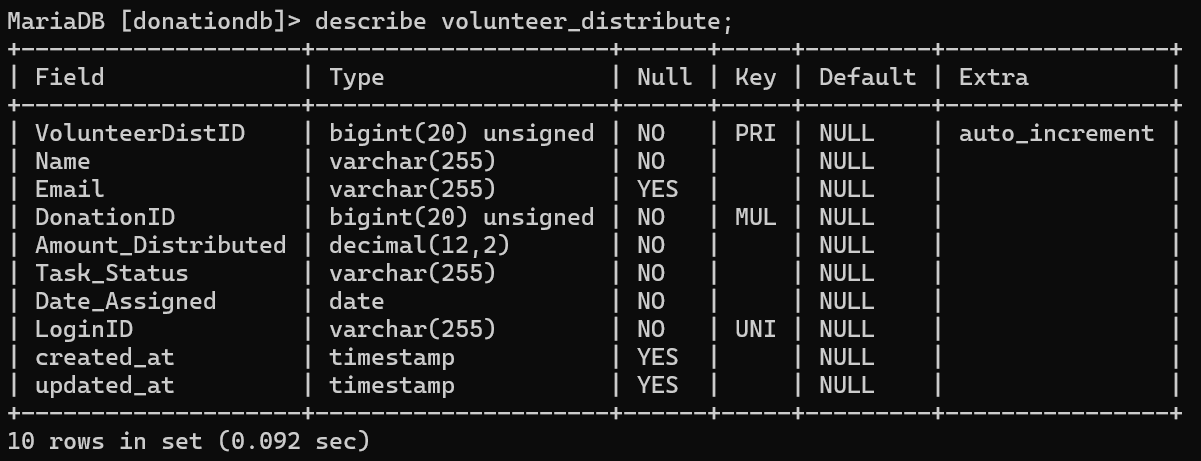
****

**Purpose:** Stores fundraising event details.

**Key Columns:**

* EventID (Primary Key)
* Name
* Date
* Location (nullable)
* Target\_Amount (decimal)
* created\_at
* updated\_at

**5. Volunteer Distribute Table**

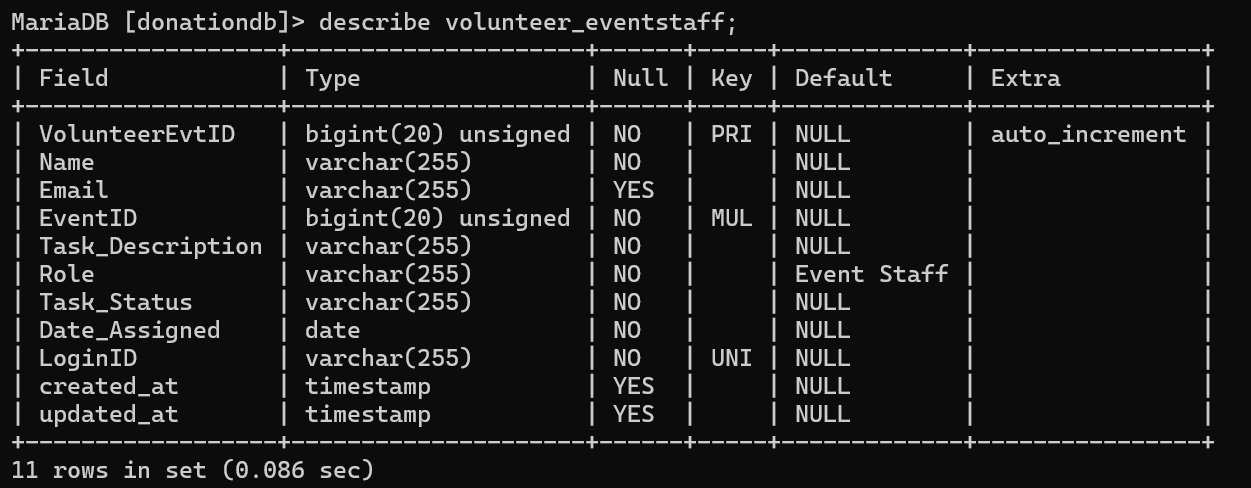
****

**Purpose:** Stores volunteers assigned to distribution tasks.

**Key Columns:**

* VolunteerDistID (Primary Key)
* Name
* Email (nullable)
* DonationID (Foreign Key to donations)
* Amount\_Distributed (decimal)
* Task\_Status
* Date\_Assigned
* LoginID (unique)
* created\_at
* updated\_at

**6. Volunteer Event Staff Table**

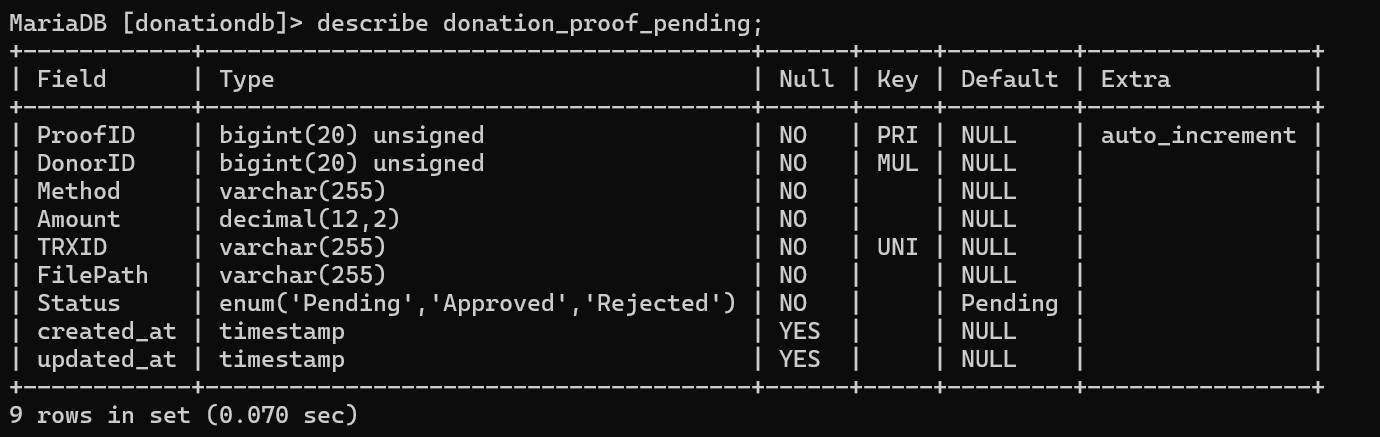
****

**Purpose:** Stores volunteers assigned as event staff.

**Key Columns:**

* VolunteerEvtID (Primary Key)
* Name
* Email (nullable)
* EventID (Foreign Key to fundraising\_events)
* Task\_Description
* Role (default 'Event Staff')
* Task\_Status
* Date\_Assigned
* LoginID (unique)
* created\_at
* updated\_at

**7. Donation Proof Pending Table**

****

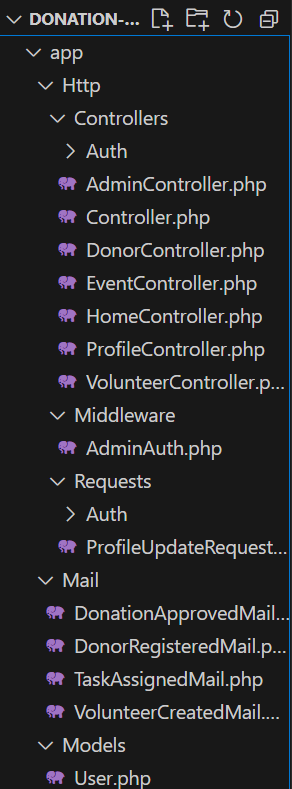
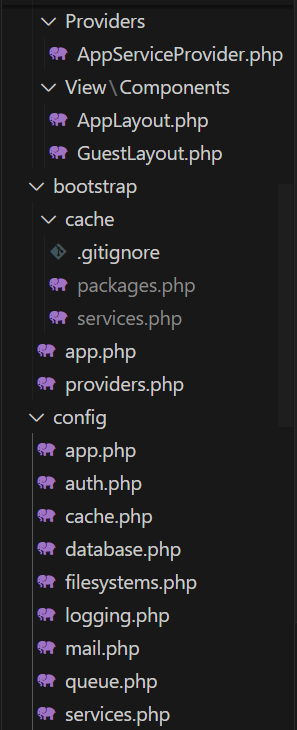
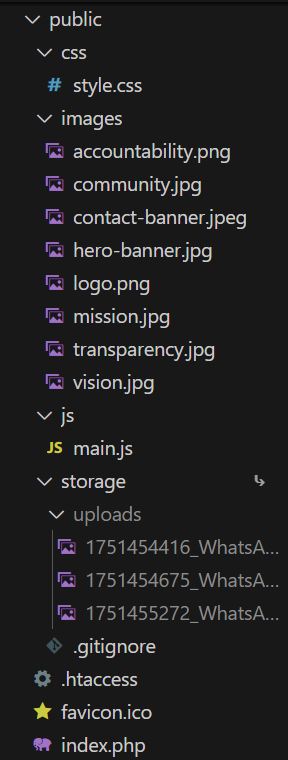
**Purpose:** Stores pending donation proofs awaiting verification.

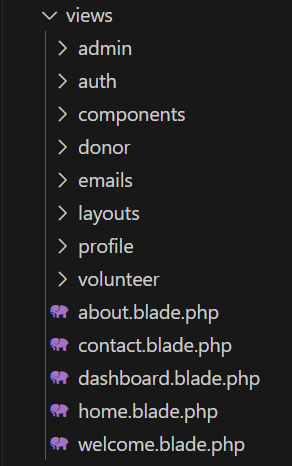
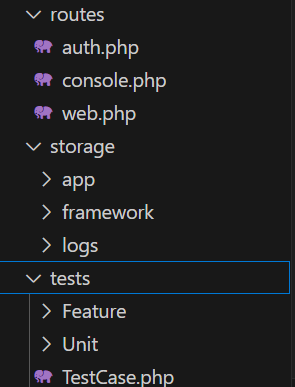
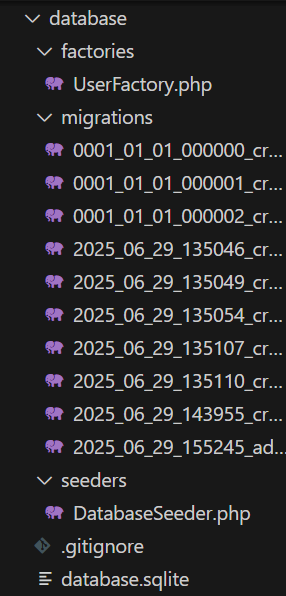
**Key Columns:**

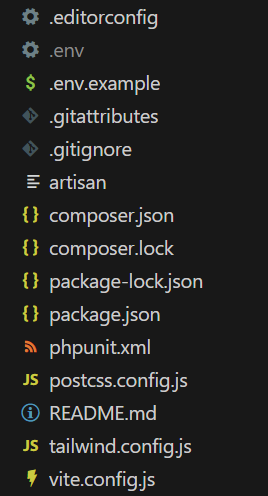
* ProofID (Primary Key)
* DonorID (Foreign Key to donors)
* Method
* Amount (decimal)
* TRXID (unique)
* FilePath
* Status (enum: Pending, Approved, Rejected; default 'Pending')
* created\_at
* updated\_at

**Laravel Implementation:**

**LARAVEL Project Structure:**

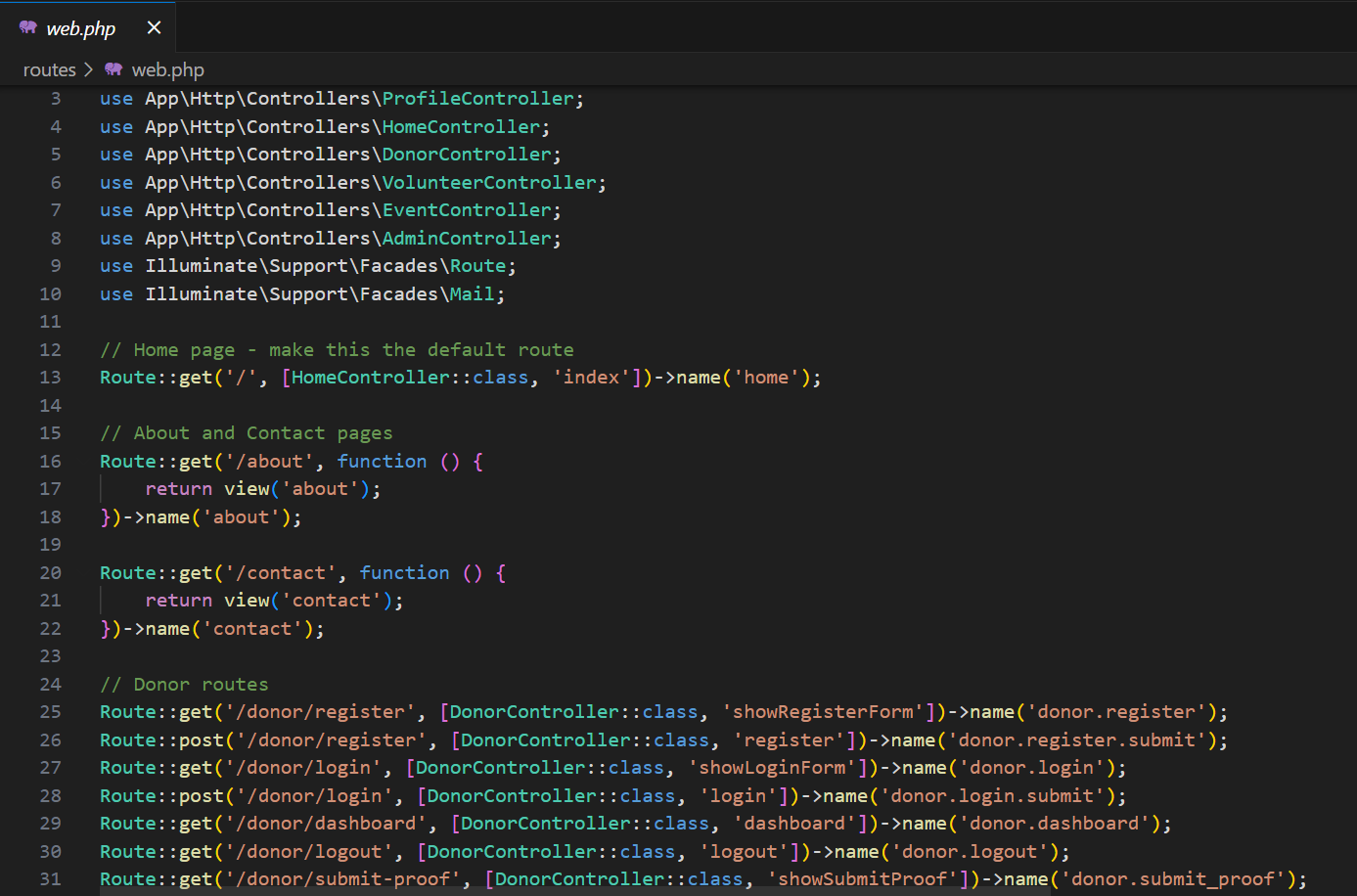
****

****

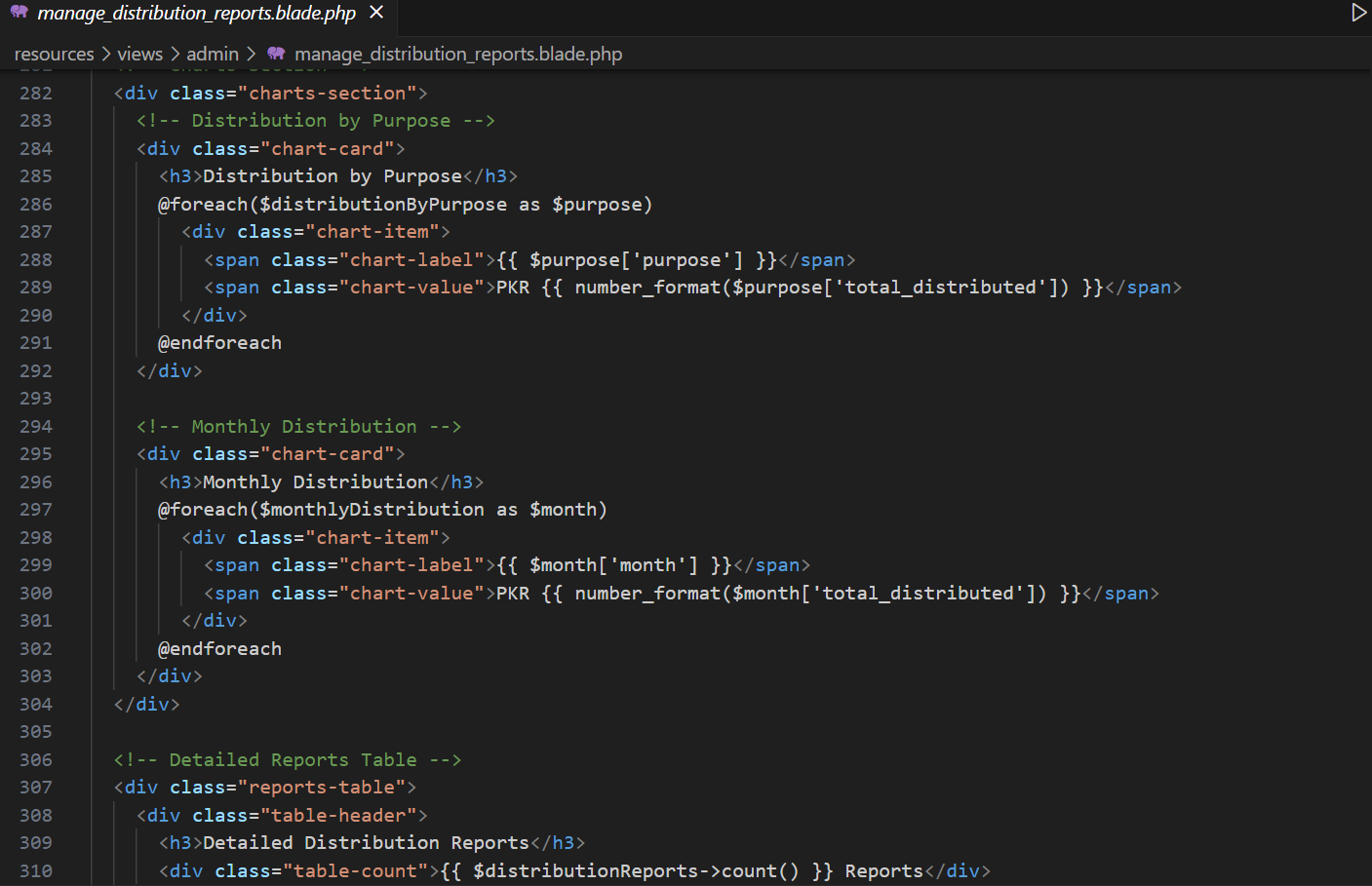
****

**Laravel MVC Architecture Implementation:**

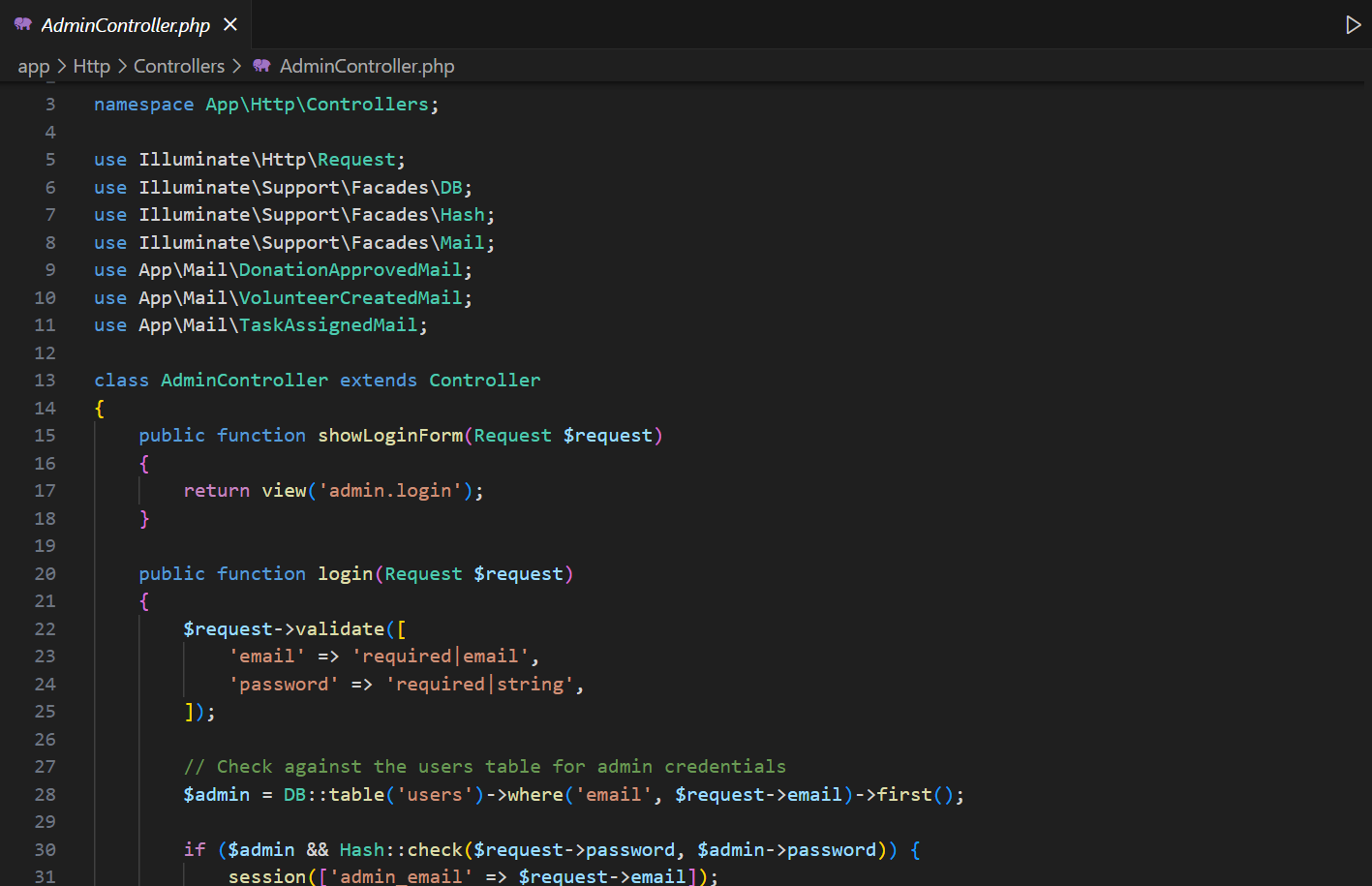
**Model:**

****

**View:**

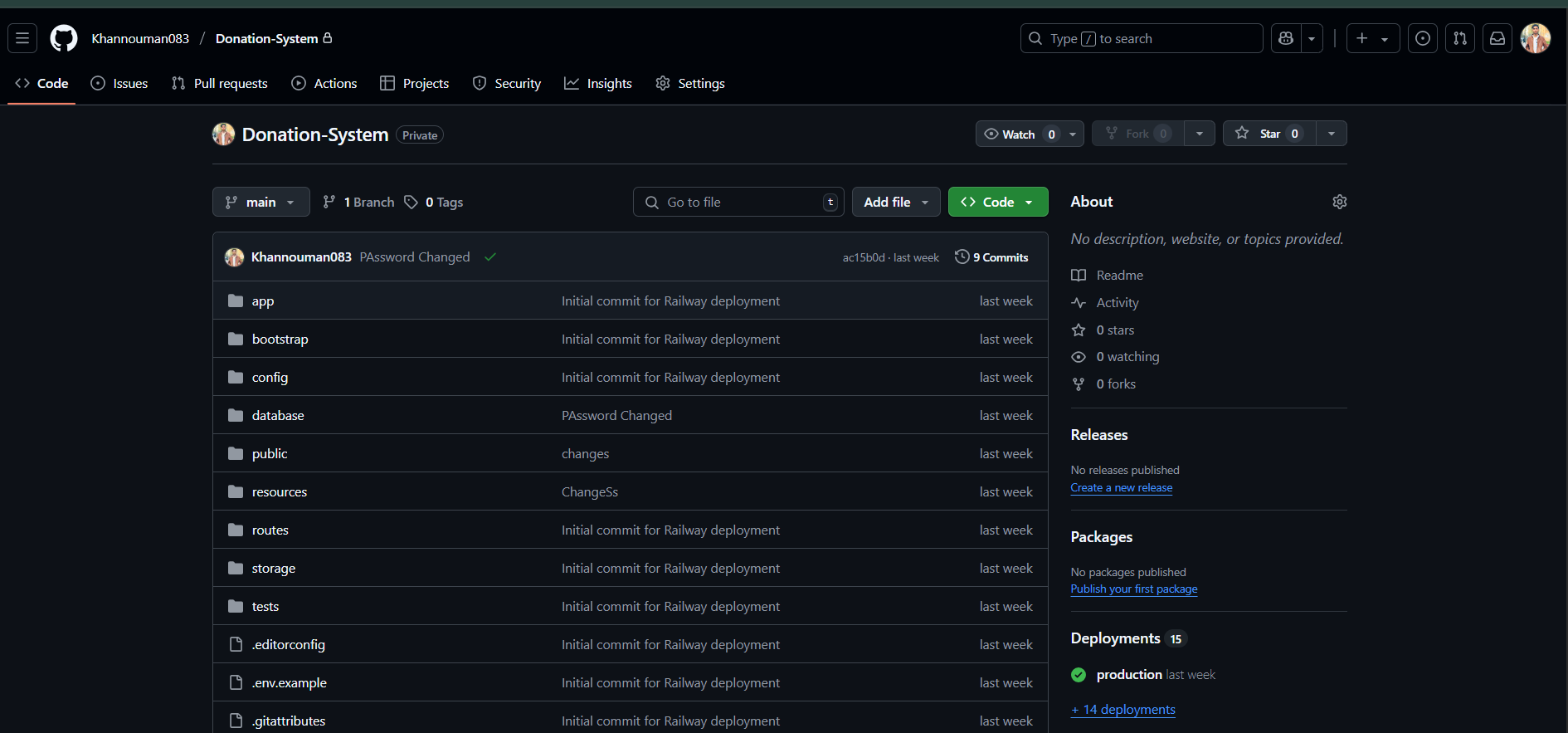
****

**Controller:**

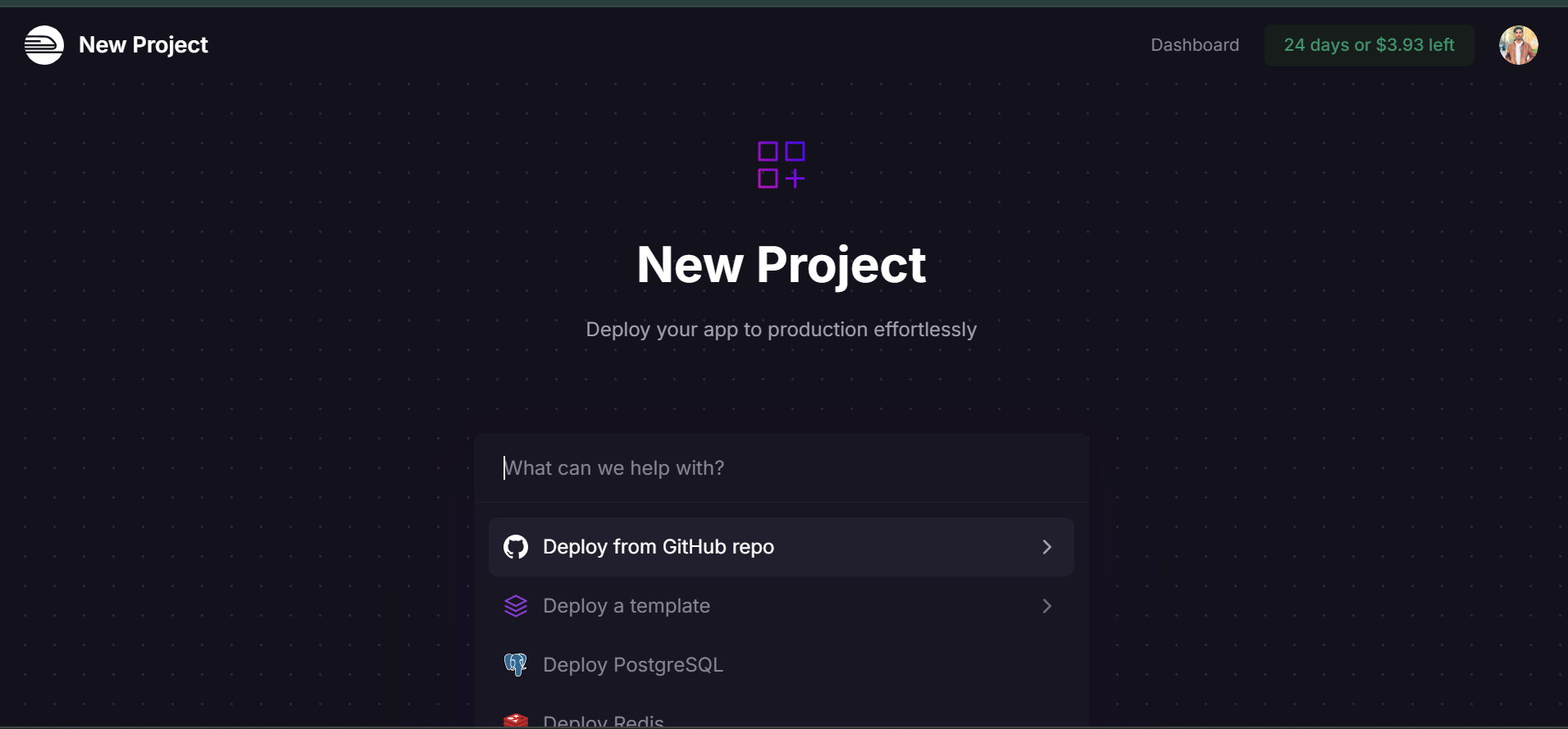
****

**Deployment:**

1. **Push your project to Github:**
   * Create a private github repo and pushed our code to it

****

1. **Create Railway Project from GitHub Repo**
   * In Railway, click New Project → Deploy from GitHub.
   * Authorize access and select your Laravel repository’s main branch.



1. **Provision the MySQL Database**
   * In your Railway project dashboard, add the MySQL plugin.
   * Railway will spin up a new database and display its connection details.

A screenshot of a computer

AI-generated content may be incorrect.

1. **Update Environment Variables**
   * Copy the Railway-generated DB credentials (DB\_HOST, DB\_NAME, DB\_USERNAME, DB\_PASSWORD) into your project’s Environment settings on Railway.
   * Ensure your Laravel .env file references these same variables (Railway injects them at runtime).

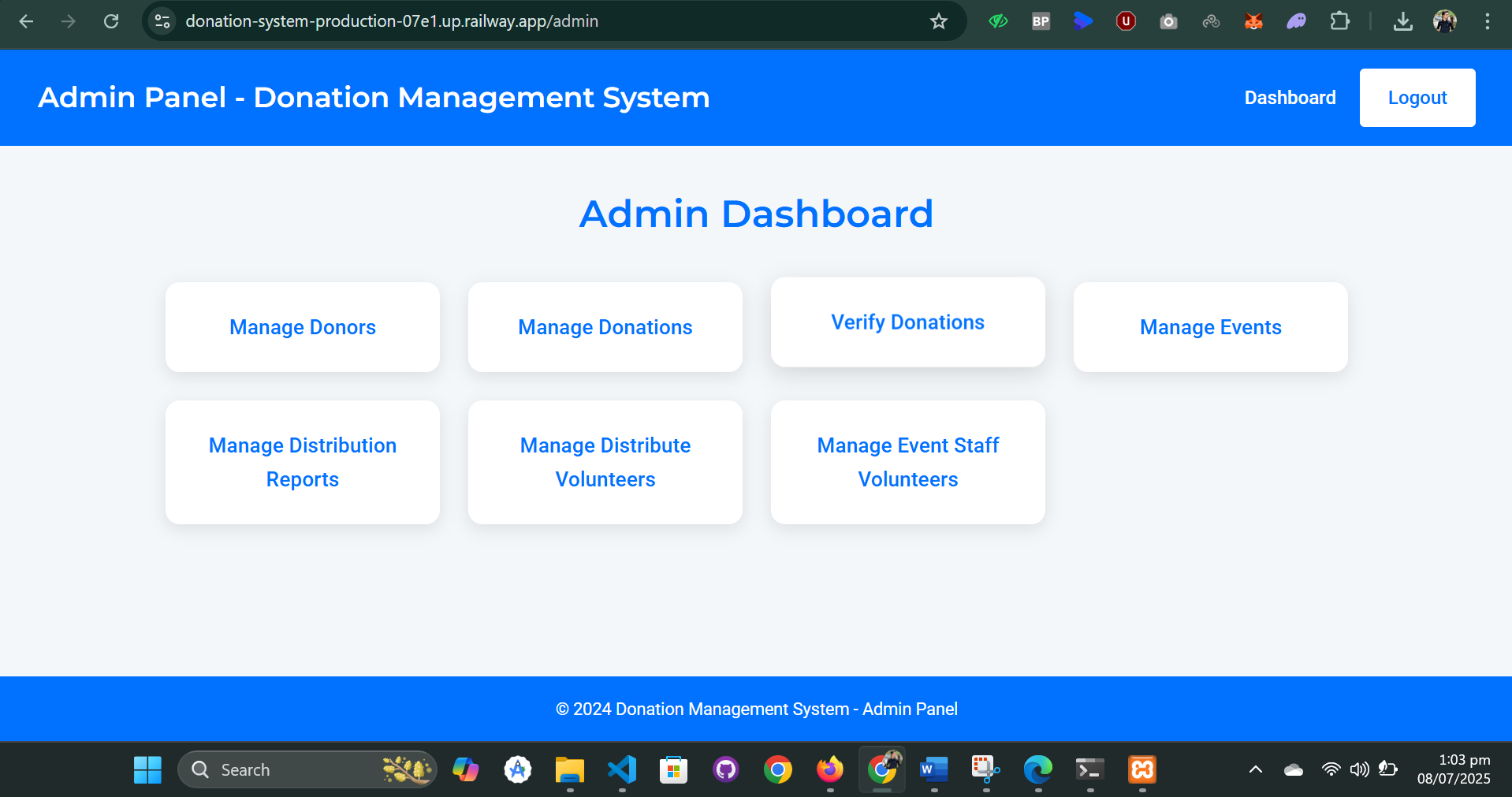
A screenshot of a computer

AI-generated content may be incorrect.

1. **Deploy the Application**
   * Back in Railway’s Deployments tab, trigger a deployment (or push to main on GitHub).
   * Railway will pull the latest code, build your app, run migrations, and launch your Donation Management System live.

A computer screen shot of a computer screen

AI-generated content may be incorrect.



**Deployment Links:**

* **Website Link:** <https://donation-system-production-07e1.up.railway.app>
* **MySQL Database:** Login to Railway using CLI and run this command  
  <mysql://root:chpjxDaNPNTzCubkMSGQXaJvjMHItuVl@yamabiko.proxy.rlwy.net:12604/railway>
* **Admin Panel:** <https://donation-system-production-07e1.up.railway.app/admin/login>
* **Admin Credentials:**
  + **Email:** [admin@donationapp.com](mailto:admin@donationapp.com)
  + **Password:** admin123

**References**

1. GitHub. *GitHub Actions Documentation*. Retrieved from <https://docs.github.com/actions>
2. Railway. *Railway Docs: Deploying from GitHub & Environment Variables*. Retrieved from [https://docs.railway.app](https://docs.railway.app/)
3. draw.io. *Diagrams.net – Online Diagramming*. Retrieved from [https://www.diagrams.net](https://www.diagrams.net/)
4. OpenAI. *ChatGPT* (GPT-4o Mini)
5. DBMS Theory Lecture Slides