System Requirements Specification (SRS) for Tanzeel Foundation Website

1. Introduction

1.1 Purpose

The purpose of this document is to define the requirements for developing a Tanzeel foundation which is non-profit website. The platform will enable the organization to share its mission, display projects, accept donations, and provide Islamic giving resources.

1.2 Scope

The website will offer:

- A responsive and visually appealing interface for public visitors.
- Secure online donation capabilities (credit card, PayPal, mobile money).
- A content management system (CMS) for managing content and media.
- Features to showcase projects, news, and events.
- Volunteer registration forms.
- Accessibility compliance for all users.

1.3 Definitions, Acronyms, and Abbreviations

- CMS – Content Management System
- UI – User Interface
- UX – User Experience
- API – Application Programming Interface

2. Overall Description

2.1 Product Perspective

The system will be a standalone web application with integrated third-party services for payments, email notifications, and analytics.

2.2 User Classes and Characteristics

- Visitor – Can browse the site without logging in.
- Donor – Can donate via multiple payment methods.
- Volunteer – Can apply through an online registration form.
- Admin – Can manage content, donations, and user information.

2.3 Operating Environment

- Frontend: HTML5, CSS3, JavaScript & J-query
- Backend: Python (Django)
- Database: SQLite or PostgreSQL
- Hosting: Cloud hosting (AWS, Namespace, DigitalOcean)
- Payment Integration: PayPal, Stripe, Mobile Money API
- Security: HTTPS, SSL encryption, database encryption

3. Functional Requirements

1. Homepage – Hero banner, mission statement, featured projects.
2. About Us – Vision, mission, team, and history.
3. Projects/Appeals – Display ongoing and completed projects.
4. Donation System – Secure payment processing, Zakat/Sadaqah/TFPDF options.
5. Volunteer Registration – Online application with document upload.
6. Blog/News – Articles, updates, and event details.
7. Contact Page – Contact form, map, and contact details.
8. Admin Dashboard – Manage site content, donations, and users.

4. Non-Functional Requirements

- Performance: Pages load in under 3 seconds.
- Scalability: Handle 100+ concurrent visitors.
- Security: SSL, CAPTCHA, input validation, and secure authentication.
- SEO Optimization: Search engine-friendly structure.
- Accessibility: WCAG 2.1 compliance.

5. External Interface Requirements

- Web Browsers: Chrome, Firefox, Edge, Safari.
- Payment Gateways: PayPal API, Stripe API, Mobile Money API.
- Email/SMS: SendGrid, Twilio, or equivalent.

6. Constraints

- Must avoid copying Tanzeel Foundation's proprietary content or images.
- Must comply with relevant local and international donation regulations.
- Limited budget for hosting and integration services.

7. Future Enhancements

- Mobile app version of the website.
- Multi-language support.
- AI-powered donation recommendations.
- Integration with blockchain for transparent donation tracking.

Proposed Django Apps:

1. **core**
    - **Functionality:** This app would handle essential functionalities that are central to the entire website and not specific to any particular module. This includes:
        - Custom user model (if needed beyond Django's default).
        - Site-wide settings (e.g., foundation name, contact email, social media links).
        - Utility functions or base templates used across multiple apps.
        - Error handling pages (404, 500).
    - **Why:** Provides a foundational layer, avoiding circular dependencies, and encapsulating generic site configurations.
2. **pages**
    - **Functionality:** Manages static content pages like the "Homepage," "About Us," and "Contact Page." This would include:
        - Models for pages with fields for title, content, meta descriptions.
        - Views to render these pages.
        - Integration with the CMS for content management as mentioned in the SRS.
    - **Why:** Groups all general content pages, separating them from dynamic features. This app would be highly reusable for any content-driven website.
3. **projects**
    - **Functionality:** Handles the display and management of "Projects/Appeals."
        - Models for projects (title, description, images, status, funding goal, etc.).
        - Views to list all projects and display individual project details.
        - Integration with the CMS for project creation and updates.
    - **Why:** Encapsulates all project-related logic, making it easy to manage and display the foundation's initiatives.
4. **donations**
    - **Functionality:** Manages the entire "Donation System." This is a critical and complex part, requiring its own app.
        - Models for donations (donor information, amount, payment method, date, associated project/appeal if any, Zakat/Sadaqah/TFPDF options).
        - Views for donation forms.

        - Integration with payment gateways (PayPal API, Stripe API, Mobile Money API).
        - Logic for secure payment processing and transaction recording.
        - Perhaps models for recurring donations if that's a future enhancement.
    - **Why:** High cohesion for all payment-related logic, which is sensitive and requires careful handling. It also allows for easier integration with various payment providers.
5. **volunteers**
    - **Functionality:** Handles "Volunteer Registration."
        - Models for volunteer applications (name, contact, skills, availability, document upload).
        - Forms for online application.
        - Views to process applications.
        - Admin interface for managing volunteer applications.
    - **Why:** Centralizes all volunteer management, keeping it distinct from other user types (donors, visitors).
6. **blog** (or news_events)
    - **Functionality:** Manages "Blog/News" content.
        - Models for articles/posts (title, content, author, publish date, categories, tags).
        - Views to list articles and display individual article details.
        - Integration with the CMS for managing blog content.
        - Could also include event details if they are tightly coupled with news.
    - **Why:** A common feature for many websites, this app can be highly reusable.
7. **dashboard** (or admin_panel)
    - **Functionality:** Provides the "Admin Dashboard" capabilities.
        - Custom views for administrative tasks not covered by Django's default admin.
        - Views to display summaries of donations, volunteer applications, content statistics.
        - User management interfaces for admins (beyond default Django admin).
    - **Why:** While Django's built-in admin is powerful, a dedicated dashboard app allows for custom admin-specific functionalities and reporting that might not fit perfectly into the individual feature apps. It acts as a central hub for administrative oversight.

This structure aims to keep the concerns separated, making development, testing, and maintenance more manageable. For example, if you need to overhaul the donation system, you primarily work within the donations app without heavily impacting other parts of the site.