SOFTWARE DESIGN DOCUMENT (SDD)

For

CharityNexus

V 1.0

Table of Contents

1. Introduction

1.1 Purpose of the Document

The purpose of this Software Design Document (SDD) is to provide a detailed overview of the design aspects, architecture, and implementation strategy for CharityNexus, a decentralized charity donation platform. This document serves as a guide for developers, designers, and stakeholders involved in the development and deployment of the platform.

1.2 Scope of the Design

The scope of this design document encompasses the architectural components, database structure, user interfaces, and integration with external systems necessary for the functionality of CharityNexus. It outlines the technical approach, design principles, and considerations for scalability, security, and usability.

1.3 Audience

The intended audience for this document includes developers, designers, project managers, quality assurance teams, and stakeholders involved in the CharityNexus project. It provides a technical blueprint and understanding of the system architecture for effective collaboration and implementation.

1.4 Definitions, Acronyms, and Abbreviations

- **SDD**: Software Design Document.
- **DApp**: Decentralized Application.
- **UI**: User Interface.
- **API**: Application Programming Interface.
- **DB**: Database.
- **UX**: User Experience.
- **CI/CD**: Continuous Integration/Continuous Deployment.

2. System Architecture

2.1 High-Level Architecture

The high-level architecture of CharityNexus comprises several key components that work together to provide the desired functionality:

- **Decentralized Application (DApp)**: The frontend interface accessible to users for browsing charities, making donations, and tracking contributions.
- **Backend Services**: The backend infrastructure handles user authentication, charity verification, donation processing, and smart contract interactions.
- **Blockchain Integration**: Integration with blockchain networks (e.g., Ethereum, Binance Smart Chain) for smart contract deployment, transaction processing, and data storage.
- **Database System**: The database stores user data, charity listings, donation records, and other essential information.
- **External APIs**: Integration with external APIs for payment processing, real-time data feeds, and verification services.

2.2 Detailed Architecture Components

2.2.1 Frontend Components

- **User Interface (UI)**: Responsive web design and mobile-friendly interfaces for user interactions.
- **UX Design**: Intuitive navigation, clear call-to-action elements, and engaging user experience.
- **Client-Side Technologies**: HTML, CSS, JavaScript frameworks (e.g., React, Vue.js) for frontend development.

2.2.2 Backend Components

- **Server-Side Technologies**: Node.js, Express.js for backend API development and routing.
- **Database Management**: MySQL, MongoDB for data storage, retrieval, and management.

- **Authentication and Authorization**: JWT (JSON Web Tokens) for user authentication and access control.
- **Smart Contract Integration**: Solidity for smart contract development and Ethereum Virtual Machine (EVM) deployment.

2.2.3 Blockchain Integration

- **Blockchain Network**: Selection of a suitable blockchain network (e.g., Ethereum, Binance Smart Chain) based on scalability, transaction fees, and community support.
- **Smart Contract Deployment**: Deployment of smart contracts for donation processing, charity verification, and transparency.
- **Blockchain Wallet Integration**: Integration with blockchain wallets for donation transactions and contract interactions.

2.3 Integration and Communication

The system architecture ensures seamless integration and communication between frontend, backend, blockchain, and external systems. APIs facilitate data exchange, transaction processing, and real-time updates across components.

3. Database Design

3.1 Database Schema

The database schema for CharityNexus includes the following tables and their relationships:

3.1.1 Users Table

- **UserID** (Primary Key): Unique identifier for users.
- **Username**: User's username for login and identification.
- **Email**: User's email address for communication and verification.
- **Password**: Encrypted password for user authentication.

3.1.2 Charities Table

- **CharityID** (Primary Key): Unique identifier for charities.
- **CharityName**: Name of the charity or organization.
- **Description**: Description of the charity's mission and activities.
- **Website**: Website URL for the charity.
- **VerificationStatus**: Status indicating if the charity is verified by administrators.

3.1.3 Donations Table

- **DonationID** (Primary Key): Unique identifier for donations.
- **UserID** (Foreign Key): References the user making the donation.
- **CharityID** (Foreign Key): References the charity receiving the donation.
- **Amount**: Donation amount.
- **Currency**: Currency type (e.g., USD, ETH).
- **Timestamp**: Timestamp of the donation transaction.

3.2 Database Relationships

The database relationships ensure data integrity and referential integrity between tables:

- Users can make multiple donations, linked by UserID in the Donations table.
- Charities can receive multiple donations, linked by CharityID in the Donations table.
- Administrators manage users and charities, maintaining verification status in the Charities table.

3.3 Database Management System (DBMS)

The DBMS for CharityNexus includes features for data storage, retrieval, indexing, and query optimization. Commonly used database technologies for this project may include MySQL, PostgreSQL, or MongoDB based on scalability and data modeling requirements.

4. User Interface Design

4.1 User Interface Overview

The user interface design of CharityNexus focuses on providing an intuitive and user-friendly experience for donors, charities, and administrators. It incorporates modern design principles, accessibility features, and responsive layouts for seamless interaction across devices.

4.2 Design Components

4.2.1 Donor Interface

- **Homepage**: Displays featured charities, donation opportunities, and user login options.
- **Charity Listings**: Allows donors to browse and select charities based on categories, causes, and impact areas.
- **Donation Form**: Provides a simple and secure form for making donations, selecting currency options, and adding donation notes.
- **Donation History**: Enables donors to track their donation history, view impact reports, and receive donation receipts.

4.2.2 Charity Interface

- **Dashboard**: Charities have access to a dashboard for managing listings, tracking donations, and providing updates.
- **Listing Management**: Allows charities to create, edit, and deactivate charity listings, update descriptions, and upload media.
- **Donation Tracking**: Provides real-time updates on donations received, donation goals achieved, and impact metrics.
- **Communication Tools**: Includes messaging features for communicating with donors, sending thank-you messages, and providing project updates.

4.2.3 Administrator Interface

- **Admin Dashboard**: Administrators have access to a centralized dashboard for managing users, verifying charities, and monitoring platform activity.
- **User Management**: Allows administrators to manage user accounts, handle account verification, and enforce security measures.
- **Charity Verification**: Enables administrators to verify charities, review documentation, and update verification status.
- **Reporting and Analytics**: Provides insights into platform usage, donation trends, and performance metrics through reporting tools.

4.3 Design Principles

The user interface design adheres to the following principles:

- **Simplicity**: Clean and uncluttered design for easy navigation and understanding.
- **Consistency**: Uniform design elements, color schemes, and typography across the platform.
- **Accessibility**: Compliance with accessibility standards, including screen reader compatibility and color contrast.
- **Mobile Responsiveness**: Responsive layouts and adaptive design for optimal viewing on mobile devices.
- **Feedback Mechanisms**: Interactive elements, progress indicators, and feedback messages for user guidance and feedback.

5. Backend Design

5.1 Overview

The backend design of CharityNexus encompasses the server-side logic, database interactions, API endpoints, and integration with external services. It is responsible for managing user authentication, charity verification, donation processing, and smart contract interactions.

5.2 Architecture

5.2.1 Server-Side Technologies

- **Node.js**: Backend runtime environment for JavaScript.
- **Express.js**: Web application framework for Node.js, handling routing and middleware.
- **JWT (JSON Web Tokens)**: Authentication mechanism for secure user sessions.
- **BCrypt**: Password hashing library for user authentication security.

5.2.2 Database Management

- **MySQL or MongoDB**: Selection of a suitable database management system based on scalability and data requirements.
- **ORM (Object-Relational Mapping)**: Sequelize for MySQL or Mongoose for MongoDB, providing abstraction for database operations.

5.2.3 API Design

- **RESTful API**: Designing API endpoints for CRUD (Create, Read, Update, Delete) operations.
- **Endpoint Authentication**: JWT-based authentication for accessing protected endpoints.
- **Error Handling**: Implementing error handling middleware for robust API responses.

5.3 Key Components

5.3.1 User Authentication

- Implementing user authentication endpoints for registration, login, and logout.
- JWT token generation and verification for secure user sessions.

5.3.2 Charity Verification

- Admin endpoints for verifying charities, updating verification status, and managing charity listings.

5.3.3 Donation Processing

- Donation endpoints for handling donation transactions, recording donations, and updating charity metrics.

5.3.4 Smart Contract Integration

- Integration with blockchain networks for deploying and interacting with smart contracts.
- API endpoints for executing smart contract functions, processing contract events, and retrieving contract data.

5.4 Scalability and Performance

- Designing scalable backend architecture with load balancing and clustering for high traffic scenarios.
- Performance optimization techniques such as caching, query optimization, and asynchronous processing.

5.5 Security Considerations

- Implementing HTTPS protocol for secure data transmission.
- Input validation, sanitization, and parameterized queries to prevent SQL injection attacks.
- Role-based access control (RBAC) for controlling user permissions and access levels.

6. Smart Contract Design

6.1 Overview

The Smart Contract Design section focuses on the design and implementation of smart contracts for CharityNexus. Smart contracts are self-executing contracts with predefined rules and conditions that facilitate secure and transparent transactions on the blockchain.

6.2 Smart Contract Components

6.2.1 Donation Contract

- **Purpose**: Facilitates donation transactions between donors and charities.
- **Functions**:
    - ***makeDonation***: Allows donors to make donations to specific charities.
    - ***getDonationAmount***: Retrieves the total donation amount received by a charity.
    - ***verifyCharity***: Verifies the authenticity and status of registered charities.

6.2.2 Charity Contract

- **Purpose**: Manages charity listings, verification status, and donation tracking.
- **Functions**:
    - ***createListing***: Enables charities to create and manage listings detailing their mission and funding needs.
    - ***updateVerificationStatus***: Updates the verification status of charities by administrators.
    - ***trackDonations***: Tracks and records donations received by charities, providing transparency to donors.

6.2.3 Access Control

- **Roles**: Designates roles such as admin, donor, and charity to control access and permissions.
- **Modifiers**: Implements modifiers to restrict access to certain functions based on user roles (e.g., only admins can verify charities).

6.3 Blockchain Integration

6.3.1 Network Selection

- **Blockchain Network**: Selects a suitable blockchain network (e.g., Ethereum, Binance Smart Chain) based on scalability, transaction fees, and contract support.
- **Deployment**: Deploys smart contracts on the chosen blockchain network using development tools (e.g., Truffle, Remix).

6.4 Testing and Deployment

6.4.1 Testing Environment

- **TestRPC/Ganache**: Sets up a local testing environment for smart contract development and testing.
- **Truffle Testing**: Conducts unit testing and integration testing for smart contracts using Truffle suite.

6.4.2 Deployment Strategy

- **Mainnet Deployment**: Deploys finalized and tested smart contracts on the main blockchain network for production use.
- **Gas Optimization**: Optimizes gas usage and contract efficiency to minimize transaction costs.

6.5 Security Considerations

- **Secure Coding Practices**: Adheres to secure coding practices for smart contract development, avoiding vulnerabilities such as reentrancy, overflow, and unauthorized access.

- **Code Review**: Conducts code review and auditing of smart contracts by experienced developers and auditors to identify and mitigate security risks.

7. Testing Strategy

7.1 Testing Approach

The testing strategy for CharityNexus includes a combination of manual testing, automated testing, and blockchain-specific testing to ensure the reliability, functionality, and security of the platform.

7.2 Testing Types

7.2.1 Unit Testing

- **Purpose**: Tests individual components, functions, and modules in isolation.
- **Tools**: Jest for JavaScript unit testing, Truffle for smart contract unit testing.

7.2.2 Integration Testing

- **Purpose**: Verifies interactions and integrations between different components and systems.
- **Tools**: Supertest for API integration testing, Truffle for smart contract integration testing.

7.2.3 End-to-End (E2E) Testing

- **Purpose**: Simulates user workflows and scenarios to test the entire system from end to end.
- **Tools**: Cypress for frontend E2E testing, Truffle for blockchain E2E testing.

7.3 Testing Environments

7.3.1 Development Environment

- **Description**: Local development environment for code writing, debugging, and initial testing.
- **Tools**: Local blockchain (TestRPC/Ganache), development servers (Node.js), local databases.

7.3.2 Staging Environment

- **Description**: Staging environment mimicking production environment for pre-release testing and validation.
- **Tools**: Staging blockchain network, staging servers, staging database.

7.3.3 Production Environment

- **Description**: Live production environment for final deployment and ongoing monitoring.
- **Tools**: Mainnet blockchain network, production servers, production database.

7.4 Continuous Integration/Continuous Deployment (CI/CD)

7.4.1 CI Pipeline

- **Purpose**: Automates build, test, and deployment processes for code changes.
- **Tools**: GitHub Actions, Travis CI for continuous integration.

7.4.2 CD Pipeline

- **Purpose**: Automates deployment of tested and approved code changes to production environment.
- **Tools**: Docker containers, Kubernetes for container orchestration, deployment scripts.

7.5 Blockchain-Specific Testing

7.5.1 Smart Contract Testing

- **Purpose**: Tests smart contracts for functionality, security, and gas optimization.
- **Tools**: Truffle suite for smart contract testing, Solidity unit testing libraries.

7.5.2 Blockchain Network Testing

- **Purpose**: Simulates blockchain network conditions and interactions for robustness testing.
- **Tools**: Truffle network simulator, Geth and Parity nodes for network testing.

8. Deployment Plan

8.1 Deployment Environment

CharityNexus will be deployed in multiple environments for development, testing, and production:

- **Development Environment**: Local development machines with development blockchain networks and servers.
- **Staging Environment**: Staging servers and staging blockchain networks for pre-release testing and validation.
- **Production Environment**: Live servers and mainnet blockchain network for final deployment and user access.

8.2 Deployment Steps

8.2.1 Development Deployment

1. Set up local development environment with necessary tools (Node.js, TestRPC/Ganache, MySQL/MongoDB).
2. Clone the repository from version control (e.g., GitHub).
3. Install dependencies using npm or yarn.
4. Configure environment variables for development settings.
5. Start development servers and blockchain network.
6. Run automated tests (unit tests, integration tests) in development environment.

8.2.2 Staging Deployment

1. Create staging servers and staging blockchain network.
2. Configure staging environment settings and database connections.
3. Deploy code changes from development branch to staging branch.
4. Perform manual and automated tests in staging environment (integration tests, E2E tests).
5. Conduct user acceptance testing (UAT) with stakeholders and testers.

6. Validate performance, scalability, and security in staging environment.

8.2.3 Production Deployment

1. Prepare production servers and mainnet blockchain network.
2. Update production environment settings and configurations.
3. Merge and deploy approved code changes from staging branch to production branch.
4. Perform final testing and validation in production environment.
5. Monitor deployment for any issues or errors.
6. Release platform to live users and stakeholders.

8.3 Rollback Plan

In case of deployment issues or unforeseen errors, the rollback plan includes:

1. Identify the issue or error causing deployment failure.
2. Roll back code changes to the previous stable version.
3. Address and fix the issue or error in development and staging environments.
4. Repeat testing and validation before attempting deployment again.
5. Communicate with stakeholders and users about the deployment status and resolution.

9. Maintenance and Support

9.1 Post-Deployment Responsibilities

9.1.1 Monitoring and Performance

- Continuous monitoring of server performance, database usage, and system metrics.
- Proactive identification and resolution of performance bottlenecks and scalability issues.

9.1.2 Security Updates

- Regular updates and patches for software dependencies, libraries, and frameworks.
- Security audits and vulnerability assessments to mitigate potential threats and risks.

9.1.3 Bug Fixing and Issue Resolution

- Monitoring and tracking of reported bugs, errors, and issues through a ticketing system.
- Timely resolution of bugs and issues based on severity and impact on users.

9.2 Support Channels

9.2.1 User Support

- Provide user support through email, chat support, and helpdesk tickets.
- Address user inquiries, feedback, and technical assistance requests promptly.

9.2.2 Charity Support

- Offer dedicated support channels for charities, including documentation, FAQs, and direct assistance.
- Assist charities with platform usage, donation tracking, and listing management.

9.2.3 Administrator Support

- Provide support and training for administrators on platform management, verification processes, and reporting tools.
- Ensure administrators have access to resources and documentation for effective platform governance.

9.3 Continuous Improvement

9.3.1 Feedback Collection

- Collect feedback from users, charities, and administrators through surveys, feedback forms, and user interviews.
- Analyze feedback to identify areas for improvement, feature requests, and usability enhancements.

9.3.2 Iterative Development

- Implement iterative development cycles for continuous improvements and feature enhancements.
- Prioritize and plan development sprints based on user feedback, market trends, and platform goals.

9.4 Versioning and Release Management

9.4.1 Version Control

- Use version control systems (e.g., Git) to manage code changes, branches, and releases.
- Follow semantic versioning conventions for version numbering and release management.

9.4.2 Release Schedule

- Define a release schedule for planned feature updates, bug fixes, and maintenance releases.
- Coordinate release cycles with development, testing, and deployment teams to ensure smooth updates.

9.5 Documentation and Knowledge Sharing

9.5.1 Internal Documentation

- Maintain internal documentation for codebase, APIs, configurations, and deployment procedures.
- Ensure developers, support teams, and administrators have access to up-to-date documentation.

9.5.2 User Documentation

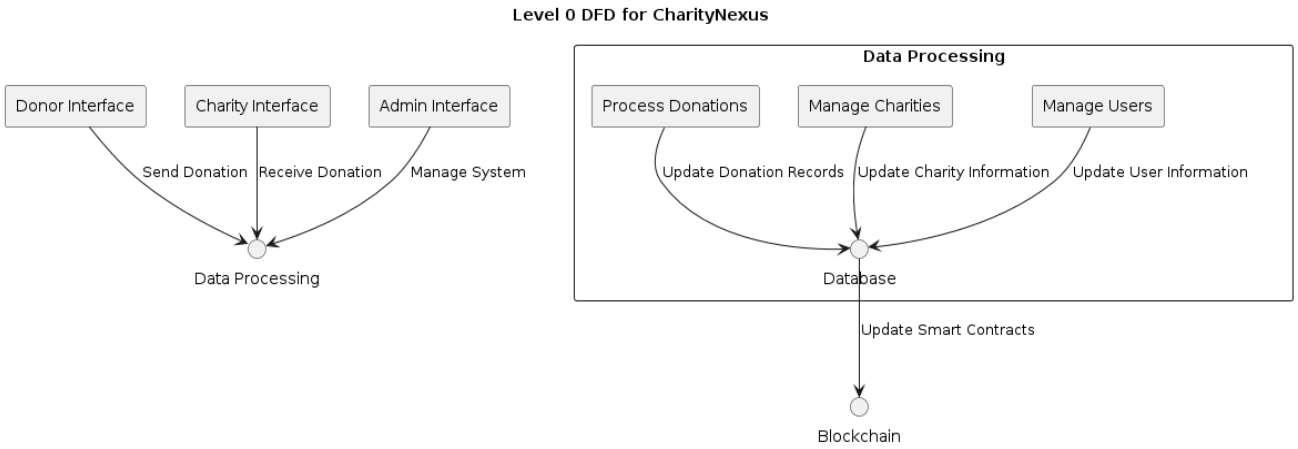
- Provide user documentation, guides, and tutorials for platform usage, features, and best practices.
- Create a knowledge base for users to find answers to common questions and troubleshoot issues.

10. Diagrams

10.1 Data Flow Diagram (DFD)

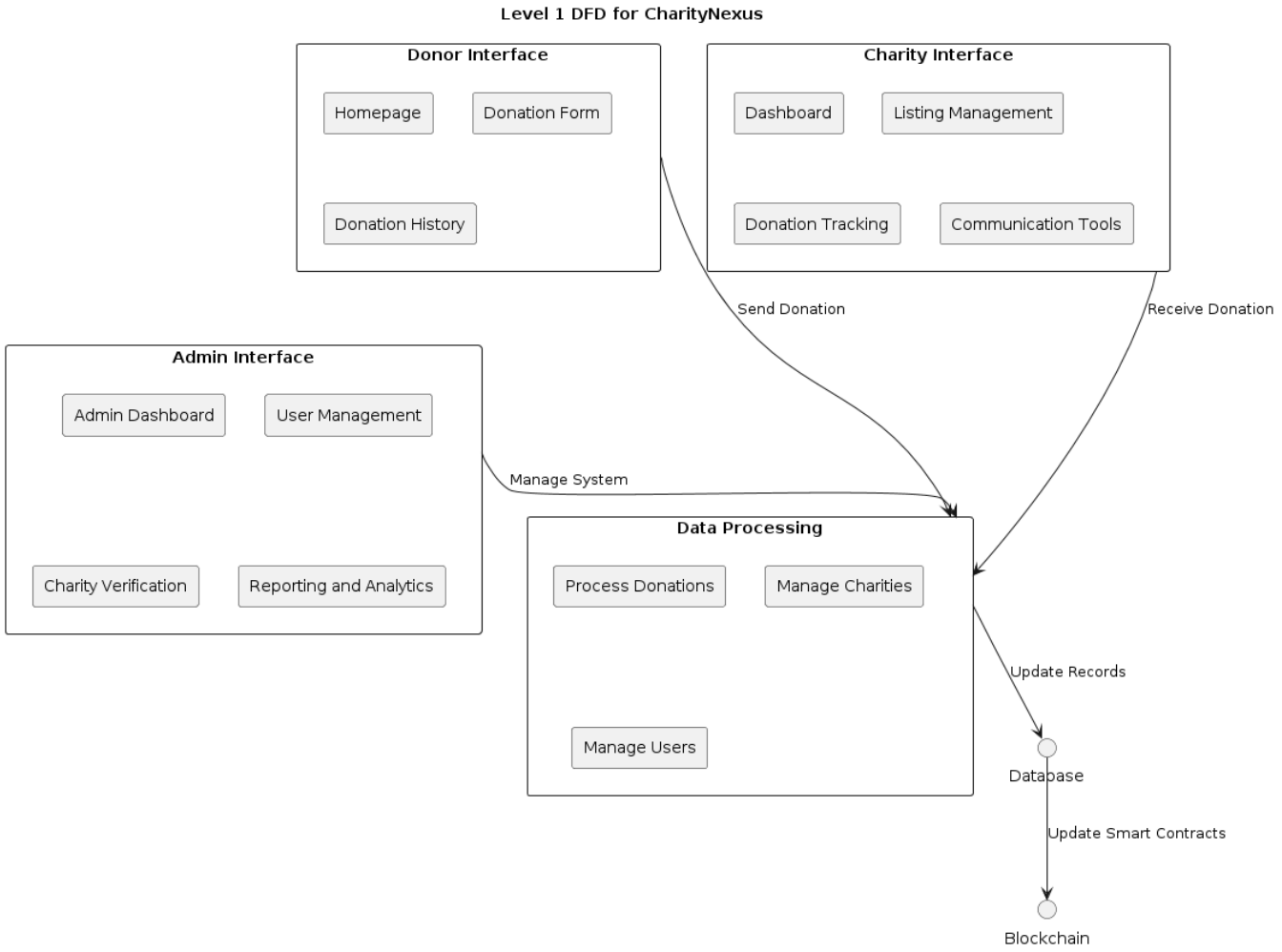
The Data Flow Diagram illustrates the flow of data within the CharityNexus platform, including data sources, processes, and data destinations.

Level 0 DFD



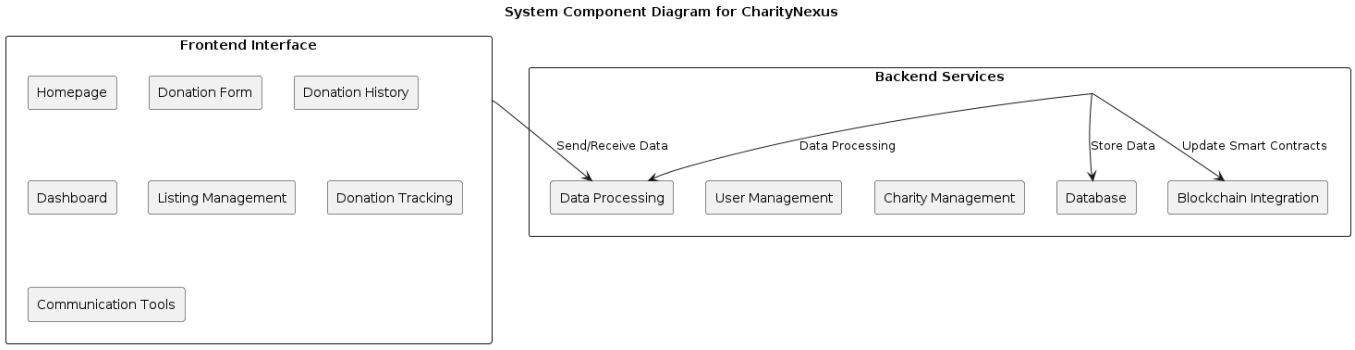
Level 1 DFD

- Donor Interface
- Charity Interface
- Admin Interface



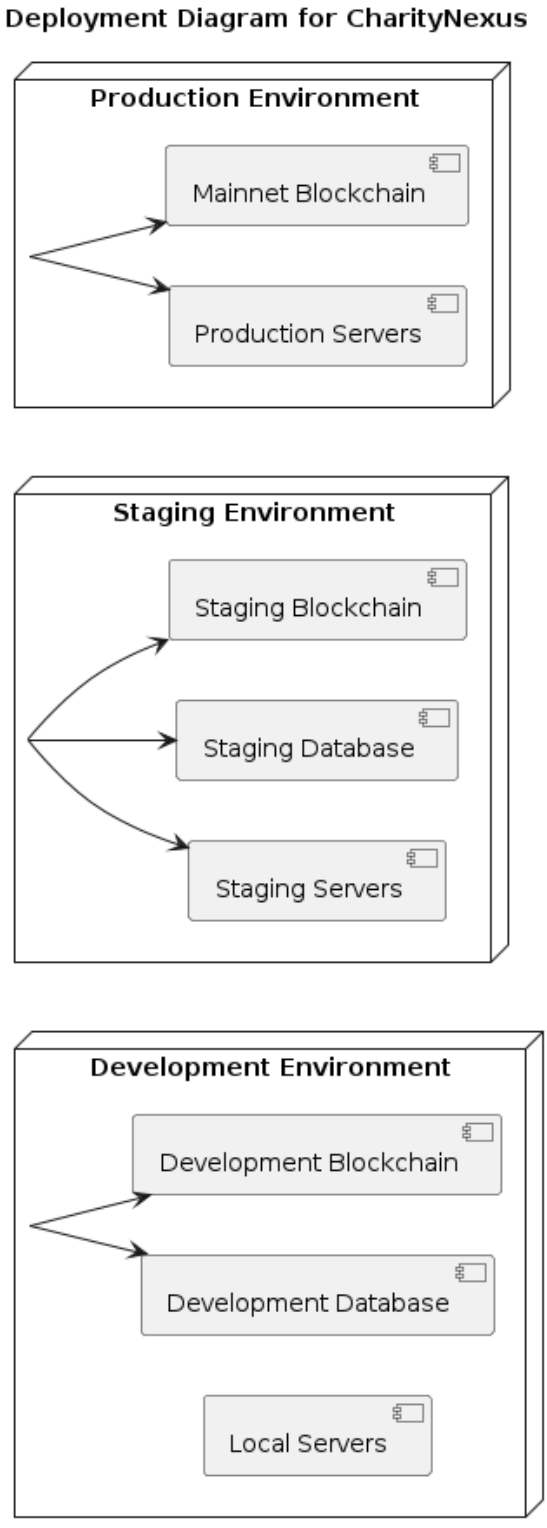
10.2 System Components Diagram

The System Components Diagram outlines the key components and modules of the CharityNexus platform, including frontend interfaces, backend services, database, and blockchain integration.



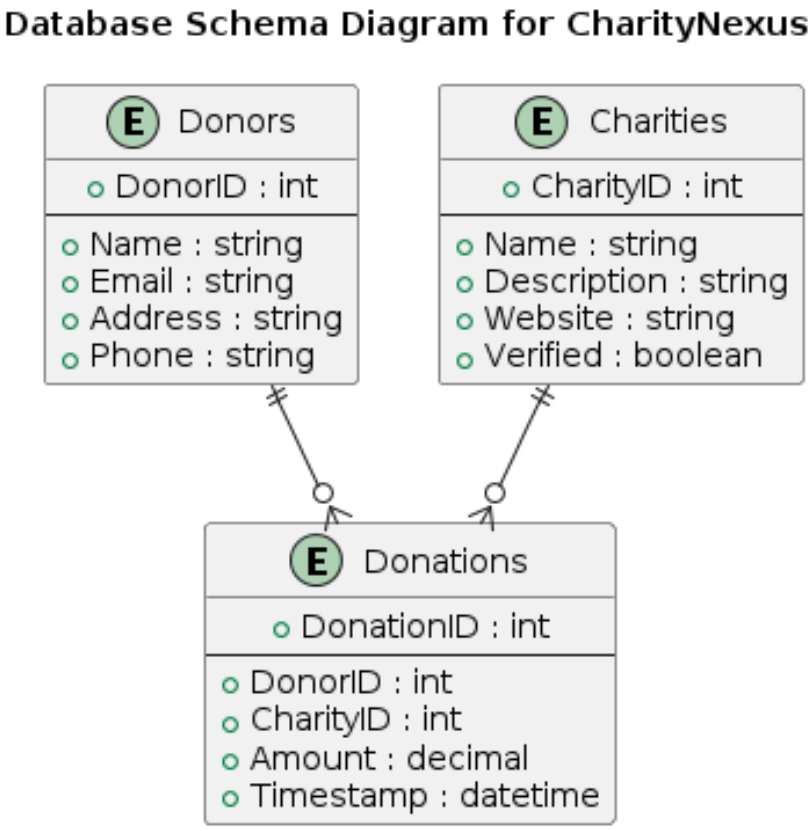
10.3 Deployment Diagram

The Deployment Diagram illustrates the deployment architecture of CharityNexus across development, staging, and production environments, including servers, databases, and blockchain networks.



10.4 Database Schema Diagram

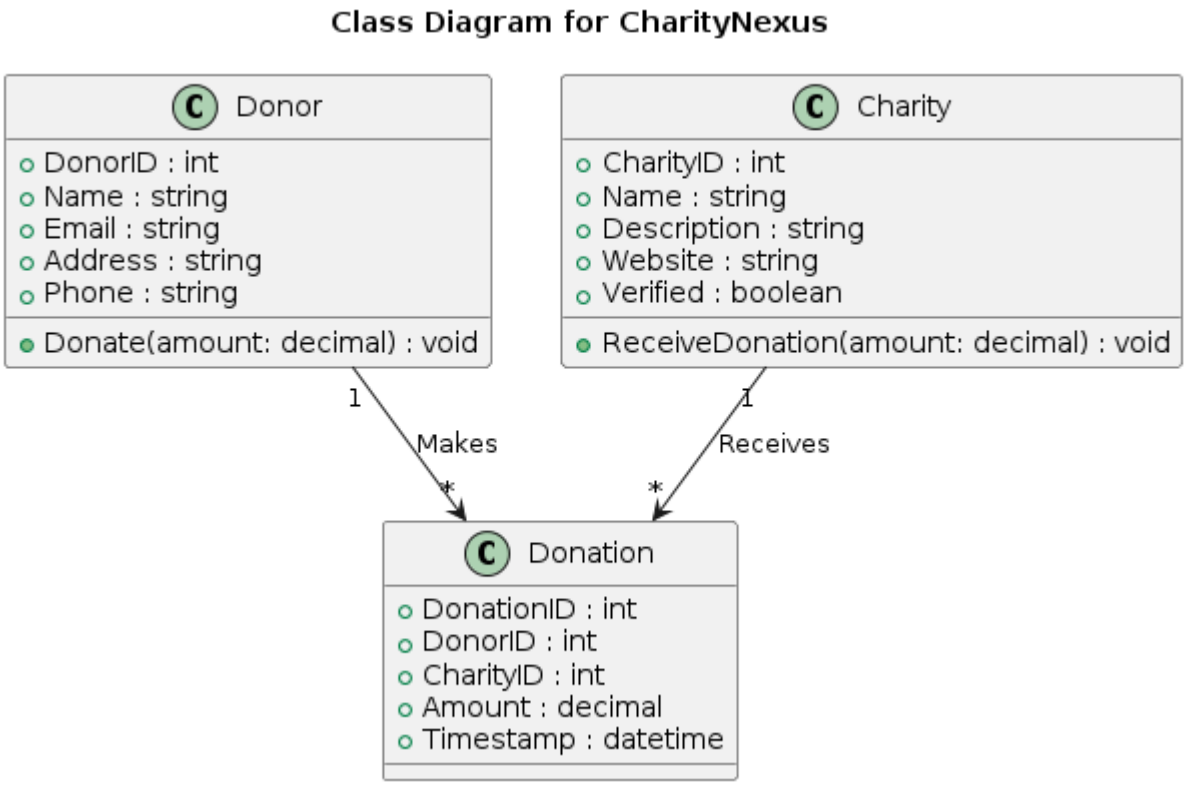
The Database Schema Diagram illustrates the structure of the database tables and relationships in the CharityNexus platform.



10.5 Interaction Diagrams

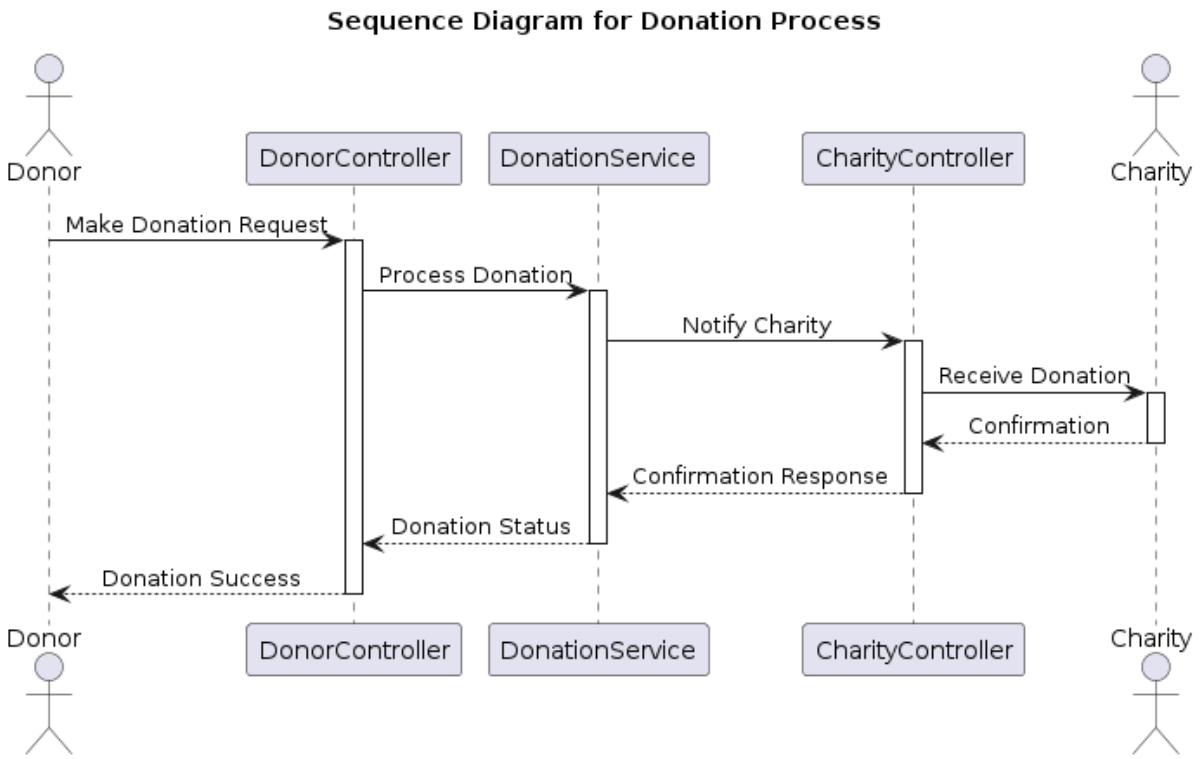
10.5.1 Class Diagram

The Class Diagram depicts the classes, attributes, methods, and relationships in the CharityNexus system.



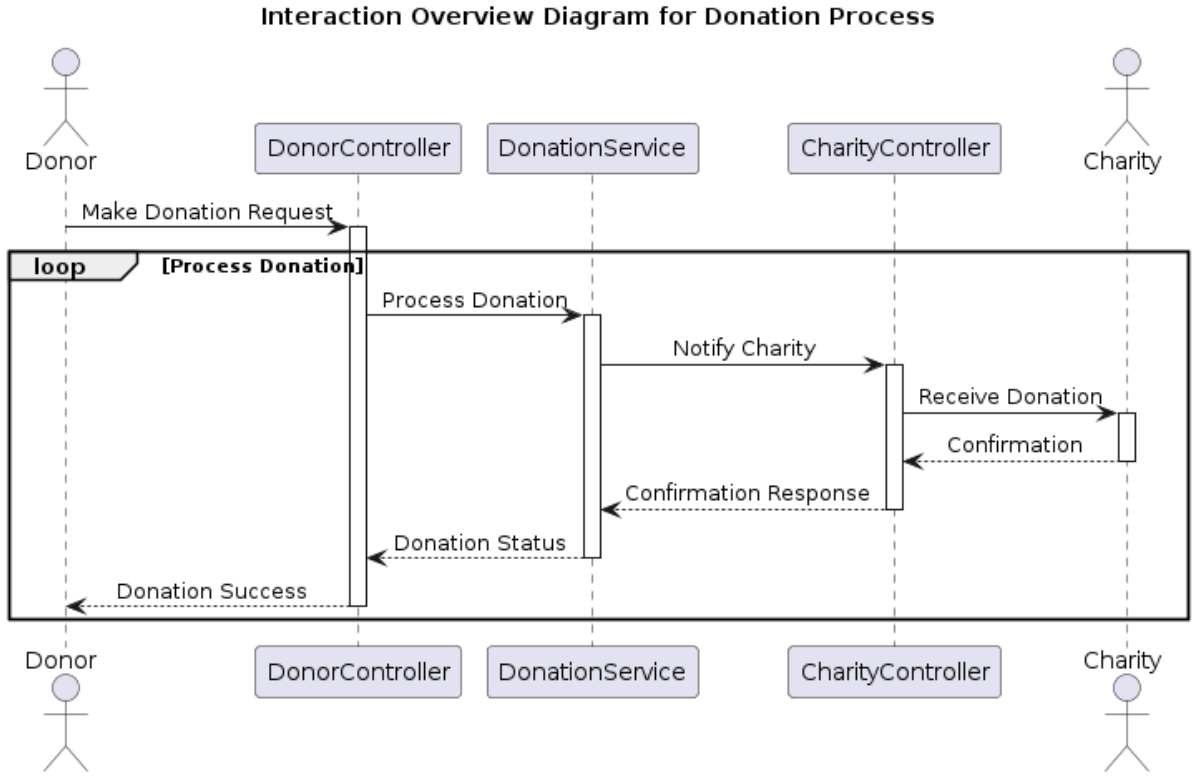
10.5.2 Sequence Diagram

The Sequence Diagram illustrates the interactions and message flows between system components during specific processes or use cases in CharityNexus.

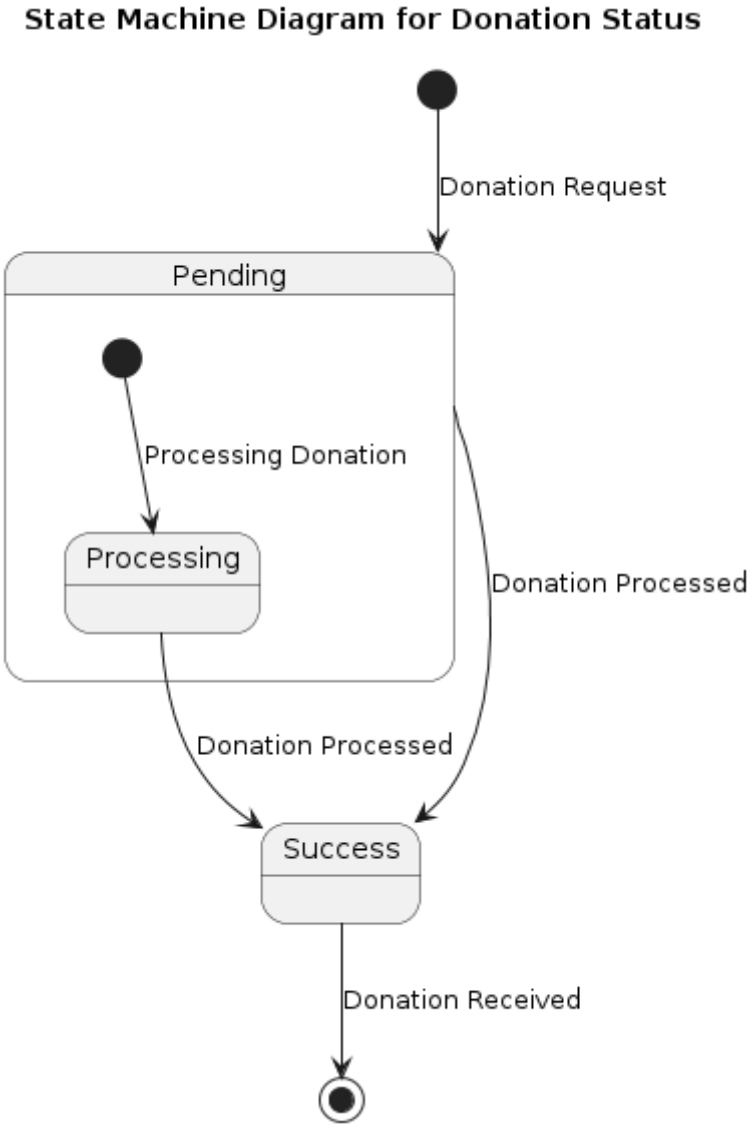


10.6 Other Diagrams

10.6.1 Interaction Overview Diagram



10.6.2 State Machine Diagram



11. Summary and Conclusion

The completion of the Software Design Document (SDD) for CharityNexus marks a comprehensive understanding of the design, architecture, and development plan for the platform. The key highlights of the SDD include:

1. **Purpose and Scope**: CharityNexus aims to provide a decentralized platform for transparent and secure donation transactions between donors and verified charities.
2. **System Architecture**: The system architecture comprises frontend interfaces, backend services, blockchain integration, and database management, ensuring a seamless and efficient platform operation.
3. **Smart Contract Design**: Smart contracts play a vital role in facilitating donation transactions, charity verification, and transparency on the blockchain network.
4. **Testing Strategy**: A robust testing strategy encompasses unit testing, integration testing, end-to-end testing, and blockchain-specific testing to ensure platform reliability and security.
5. **Deployment Plan**: The deployment plan outlines steps for deploying the platform in development, staging, and production environments, ensuring a smooth transition and minimal downtime.
6. **Maintenance and Support**: Post-deployment responsibilities include monitoring, security updates, bug fixing, user support, continuous improvement, versioning, release management, and documentation practices.
7. **Conclusion**: CharityNexus is poised to become a trusted platform for fostering charitable donations, enhancing transparency, and supporting charitable organizations worldwide.

Appendices

A. Glossary

- **Donor**: An individual or entity making donations to charities through the CharityNexus platform.
- **Charity**: An organization registered on CharityNexus to receive donations and provide services.
- **Donation**: The act of giving money or resources to a charity through the CharityNexus platform.
- **Blockchain**: A decentralized digital ledger technology used for secure and transparent transaction recording.
- **Smart Contracts**: Self-executing contracts with predefined rules and conditions implemented on the blockchain.

B. References

1. CharityNexus Project Proposal
2. CharityNexus System Requirements Specification (SRS)
3. CharityNexus User Interface Design
4. CharityNexus Database Schema
5. CharityNexus Deployment Plan

C. Acknowledgments

We would like to acknowledge the contributions and support of the following individuals and organizations during the development of CharityNexus:

- [Name/Organization]: [Description of Contribution/Support]
- [Name/Organization]: [Description of Contribution/Support]
- [Name/Organization]: [Description of Contribution/Support]

D. Revision History

Version	Date	Description	Author
1.0	[Date]	Initial Draft	[Author Name]
1.1	[Date]	Updated Class Diagram	[Author Name]
1.2	[Date]	Added Interaction Diagrams	[Author Name]
1.3	[Date]	Incorporated Feedback	[Author Name]
1.4	[Date]	Final Review and Approval	[Author Name]