EEET2582 – Software Engineering: Architecture and Design

Charitan - A Global Charity Donation Platform - Milestone 1

Panther B

Nguyen Huy Hoang – S3914298

Nguyen Tan Phat – S3911749

Pham Nguyen Minh Dang – S3922418

26 Nov 2024

School of Science, Engineering, and Technology

Table Of Contents

1. INTRODUCTION 3

2. PROJECT BACKGROUND 3

3. CONCEPT DATA MODEL 4

4. SYSTEM ARCHITECTURE 10

5. ARCHITECTURE ANALYSIS 25

6. CONCLUSION 30

REFERENCE 30

1. Introduction

The capacity for collective goodwill to influence communities and individuals in need is unprecedented in this linked world. The Charitan platform is a comprehensive and versatile solution intended to link contributors, volunteers, and philanthropic initiatives worldwide. Charitan aims to utilize advanced technology to facilitate a seamless and engaging experience, promoting significant connections and cultivating a culture of generosity.

Charitan functions with two primary objectives:

- To enable donors and volunteers to choose and endorse philanthropic activities that resonate with their values, both locally and globally, so ensuring their contributions yield optimal impact.
- To provide charities with comprehensive resources for the effective management of fundraising initiatives across several areas, including food, health, education, environment, and housing. This consolidated platform improves visibility, promotes collaboration, and efficiently mobilizes resources.

This research examines the platform's creative design, analyzing its features, user interfaces, and technological architecture to illustrate how Charitan achieves its objective of uniting hearts and hands for worldwide charity.

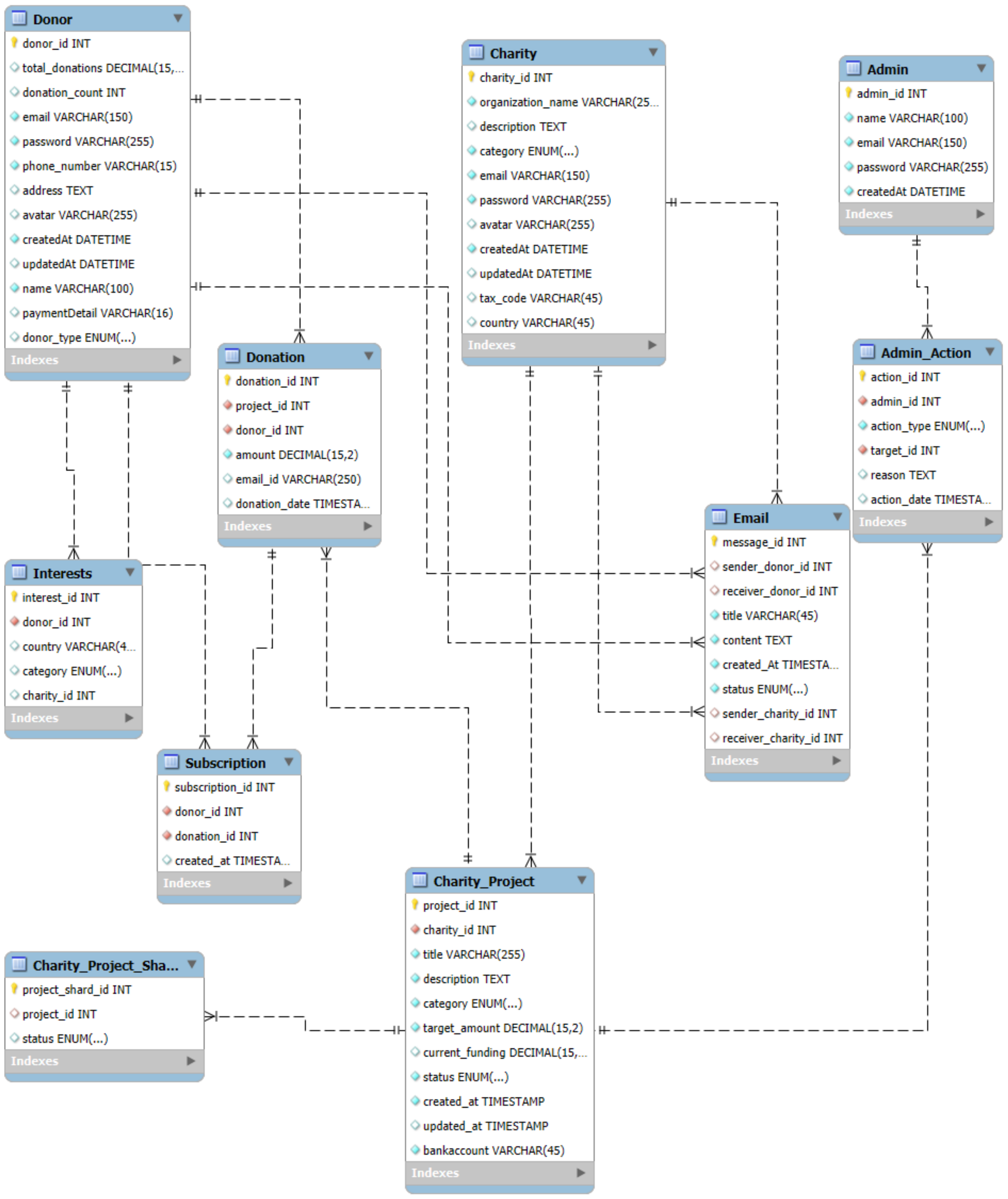
2. Project background

The Charitan platform is based on the acknowledgment of the increasing importance of the internet in charitable donations. Despite the growing prevalence of online donations, numerous charitable organizations have difficulties in enhancing their digital presence and obtaining significant money. Studies reveal that donor satisfaction with current charity websites is inadequate, highlighting the necessity for a more straightforward and efficient platform.

Charitan confronts these problems by offering an intuitive interface that enables donors to discover and endorse charities that resonate with their values. It provides charity with effective tools to optimize internet fundraising initiatives. The platform integrates the Elaboration Likelihood Model of persuasion with self-schema theory to augment its attractiveness and efficacy in fostering generosity.

Essential attributes comprise an intuitive interface, sophisticated search and discovery mechanisms, integrated crowdfunding functionalities, and systems that promote collaboration. Charitan seeks to enhance online donations, elevate donor satisfaction, and broaden support networks by connecting donors with organizations, so producing substantial community impact worldwide.

3. Concept data Model



1. **Charity:**

- **Attributes:**
  - charity_id (Primary Key): Unique identifier for each charity.
  - organization_name: Name of the charity organization.
  - description: Purpose of the charity.
  - category: Categorization of the charity.
  - email: Contact email for the charity.
  - password: Secured password for authentication.
  - avatar: Image or logo of the charity.
  - country: Country where the charity operates.
  - createdAt, updatedAt: Timestamps for creation and updates.
  - tax_code: Tax code of the charity
- **Description:** Represents registered charity, specifying their identification, classification, and contact details.

2. **Charity_Project:**
   - **Attributes:**
     - project_id (Primary Key): Unique identifier for each project.
     - charity_id (Foreign Key): Links to the associated charity.
     - title: Title of the project.
     - description: Goals and objectives of the project.
     - category: Project categorization
     - target_amount: Target fundraising amount for the project.
     - current_funding: Total funding received to date.
     - status: Current status of the project
     - created_at, updated_at: Timestamps for project creation and updates.
     - bankaccount: Bank account details for receiving donations.
   - **Description:** Denotes particular endeavors by charitable organizations, enabling fundraising and monitoring of success..
3. **Donor:**
   - **Attributes:**
     - donor_id (Primary Key): Unique identifier for each donor.
     - total_donations: Total amount donated by the donor.
     - donation_count: Number of donations made.
     - email: Donor’s contact email.

        - password: Secured password for donor account.
        - phone_number: Contact number of the donor.
        - avatar: Profile picture of the donor.
        - address: Address of the donor.
        - name: Full name of the donor.
        - paymentDetail: Payment method or details for processing donations.
        - donor_type: Type of donor (e.g., Individual, Organization).
        - createdAt, updatedAt: Timestamps for account creation and updates.
    - **Description:** Denotes individuals or collectives participating in projects, with their contributions being tracked.

4. **Donation:**
    - **Attributes:**
        - donation_id (Primary Key): Unique identifier for each donation.
        - project_id (Foreign Key): Links to the associated project.
        - donor_id (Foreign Key): Links to the donating individual.
        - amount: Amount donated.
        - donation_message: Message attached to the donation by the donor.
        - donation_date: Timestamp of the donation.
    - **Description:** Monitors contributions allocated to certain initiatives, guaranteeing transparency and traceability.
5. **Interests:**
    - **Attributes:**
        - interest_id (Primary Key): Unique identifier for each interest.
        - donor_id (Foreign Key): Links to the donor.
        - country: Geographical region of interest.
        - category: Specific area of interest (e.g., Environment, Education).
        - charity_id (Foreign Key): Links to a relevant charity.
    - **Description:** Advocates for donors' interests, facilitating the system's ability to suggest charities and initiatives aligned with their preferences.
6. **Subscription:**
    - **Attributes:**
        - subscription_id (Primary Key): Unique identifier for each subscription.
        - donor_id (Foreign Key): Links to the subscribing donor.

    - charity_id (Foreign Key): Links to the subscribed charity.
    - created_at: Timestamp when the subscription was created.
  - **Description:** Monitors contributors contributing to nonprofits, facilitating ongoing or sustained assistance.

7. **Email:**
   - **Attributes:**
     - message_id (Primary Key): Unique identifier for each email.
     - sender_donor_id (Foreign Key): Links to the sender donor.
     - receiver_donor_id (Foreign Key): Links to the recipient donor.
     - sender_charity_id (Foreign Key): Links to the sender charity.
     - receiver_charity_id (Foreign Key): Links to the recipient charity.
     - title: Title or subject of the email.
     - content: Email content.
     - created_at: Timestamp when the email was sent.
     - status: Email status (e.g., Sent, Received).
   - **Description:** Enhances communication between donors and charity or among donors for collaboration.
8. **Charity_Project_Share:**
   - **Attributes:**
     - project_shard_id (Primary Key): Unique identifier for shared project records.
     - project_id (Foreign Key): Links to the shared project.
     - status: Sharing status
   - **Description:** Facilitates the dissemination of projects, promoting collaboration among nonprofits.
9. **Admin:**
   - **Attributes:**
     - admin_id (Primary Key): Unique identifier for each admin.
     - name: Full name of the admin.
     - email: Contact email of the admin.
     - password: Secured password for admin account.
     - createdAt: Timestamp for admin account creation.

- **Description:** Represents system administrators tasked for supervising and administering charities, donors, and system operations.

10. **Admin_Action:**
    - **Attributes:**
        - action_id (Primary Key): Unique identifier for each admin action.
        - admin_id (Foreign Key): Links to the admin performing the action.
        - action_type: Type of action
        - target_id: ID of the target entity for the action.
        - reason: Reason for performing the action.
        - action_date: Timestamp of the action.
    - **Description:** Documents acts executed by administrators for accountability and auditing purposes.

Relationships and Their Descriptions

1. **Charity ↔ Charity_Project:**
    - Type: One-to-Many
    - Description: A charity may oversee numerous projects; nevertheless, each project is associated with just one charity.
2. **Donor ↔ Donation:**
    - Type: One-to-Many
    - Description: A donor may make several donations, however each donation is attributed to one specific donor.
3. **Donation ↔ Charity_Project:**
    - Type: Many-to-One
    - Description: Numerous donations can aggregate to support a singular project..
4. **Interests ↔ Donor:**
    - Type: One-to-Many
    - Description: A donor may possess several interests.
5. **Subscription ↔ Donor:**
    - Type: Many-to-One
    - Description: A contributor may contribute to many charities.
6. **Subscription ↔ Charity:**
    - Type: Many-to-One
    - Description: Charities may receive numerous subscriptions from donors.

7. **Email ↔ Donor:**
   - Type: Many-to-Many
   - Description: Enhances communication among donors.
8. **Email ↔ Charity:**
   - Type: Many-to-Many
   - Description: Facilitates communication between nonprofits and contributors.
9. **Charity_Project_Share ↔ Charity_Project:**
   - Type: One-to-Many
   - Description: A project may be disseminated repeatedly.
10. **Admin ↔ Admin_Action:**
    - Type: One-to-Many
    - Description: An administrator can execute many activities.

Data Model Design Rationale

Design Decisions and Justifications

1. **Entity Modularity:**
   - The segregation of entities guarantees modularity and facilitates data administration.
2. **Enhanced Personalization:**
   - Entities such as Interests and Subscription facilitate tailored experiences for donors.
3. **Streamlined Communication:**
   - The Email entity enables efficient communication between users and charities.
4. **Collaboration Support:**
   - The Charity_Project_Share entity facilitates sharing and collaborations.

Advantages of the Design

1. **Scalability:**
   - Modular components facilitate future growth without compromising the system.
2. **Transparency:**
   - Comprehensive monitoring of contributions, initiatives, and administrative activities bolsters user confidence.
3. **Personalized Engagement:**
   - Entities such as Interests and Subscription accommodate donor choices.

4. **Collaboration Opportunities:**
    - The capacity to share projects cultivates collaborations among nonprofits.

Potential Drawbacks and Mitigation Strategies

1. **Complexity of Queries:**
    - Drawback: Extracting data from numerous entities may yield intricate queries.
    - Mitigation: Enhance query architecture and utilize indexed fields.
2. **Data Privacy Concerns:**
    - Drawback: Confidential information such as passwords and email addresses is vulnerable.
    - Mitigation: Employ robust encryption and comply with data protection requirements.

4. System Architecture

4.1 Overall System Architecture

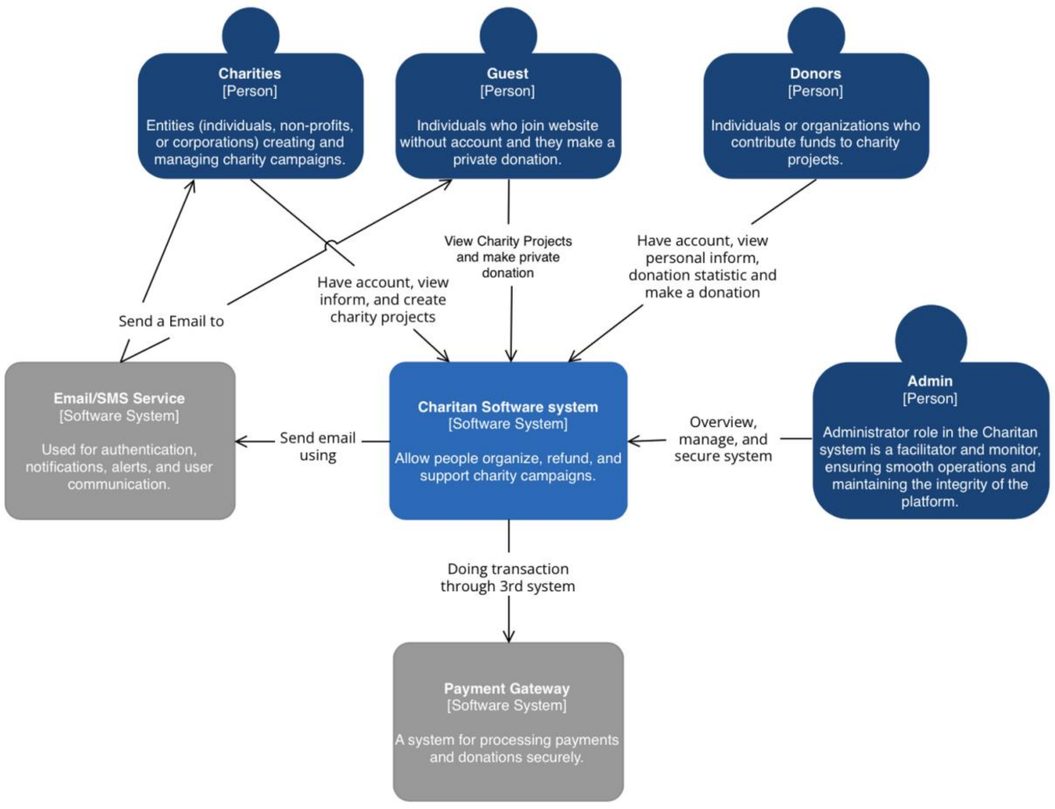

Charities
[Person]
Entities (individuals, non-profits, or corporations) creating and managing charity campaigns.
Guest
[Person]
Individuals who join website without account and they make a private donation.
Donors
[Person]
Individuals or organizations who contribute funds to charity projects.
Send a Email to
Have account, view inform, and create charity projects
View Charity Projects and make private donation
Have account, view personal inform, donation statistic and make a donation
Admin
[Person]
Administrator role in the Charitan system is a facilitator and monitor, ensuring smooth operations and maintaining the integrity of the platform.
Email/SMS Service
[Software System]
Used for authentication, notifications, alerts, and user communication.
Send email using
Charitan Software system
[Software System]
Allow people organize, refund, and support charity campaigns.
Overview, manage, and secure system
Doing transaction through 3rd system
Payment Gateway
[Software System]
A system for processing payments and donations securely.

Figure 1 Context diagram

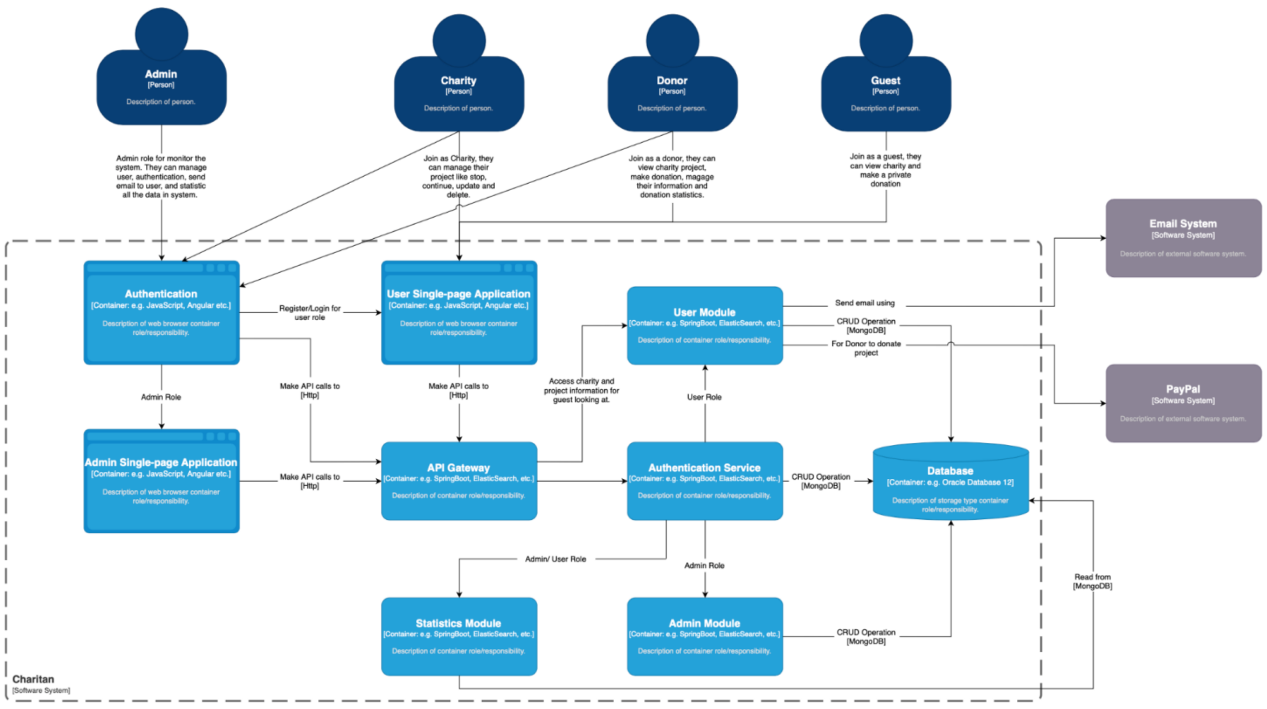

Admin
[Person]
Description of person.
Charity
[Person]
Description of person.
Donor
[Person]
Description of person.
Guest
[Person]
Description of person.
Admin role for monitor the system. They can manage user, authentication, send email to user, and statistic all the data in system.
Join as Charity, they can manage their project like stop, continue, update and delete.
Join as a donor, they can view charity project, make donation, magage their information and donation statistics.
Join as a guest, they can view charity and make a private donation
Email System
[Software System]
Description of external software system.
PayPal
[Software System]
Description of external software system.
Authentication
[Container: e.g. JavaScript, Angular etc.]
Description of web browser container role/responsibility.
Register/Login for user role
User Single-page Application
[Container: e.g. JavaScript, Angular etc.]
Description of web browser container role/responsibility.
User Module
[Container: e.g. SpringBoot, ElasticSearch, etc.]
Description of container role/responsibility.
Send email using
CRUD Operation [MongoDB]
For Donor to donate project
Admin Role
Make API calls to [Http]
Make API calls to [Http]
Access charity and project information for guest looking at.
User Role
Admin Single-page Application
[Container: e.g. JavaScript, Angular etc.]
Description of web browser container role/responsibility.
Make API calls to [Http]
API Gateway
[Container: e.g. SpringBoot, ElasticSearch, etc.]
Description of container role/responsibility.
Authentication Service
[Container: e.g. SpringBoot, ElasticSearch, etc.]
Description of container role/responsibility.
CRUD Operation [MongoDB]
Database
[Container: e.g. Oracle Database 12]
Description of storage type container role/responsibility.
Admin/ User Role
Admin Role
Read from [MongoDB]
Statistics Module
[Container: e.g. SpringBoot, ElasticSearch, etc.]
Description of container role/responsibility.
Admin Module
[Container: e.g. SpringBoot, ElasticSearch, etc.]
Description of container role/responsibility.
CRUD Operation [MongoDB]
Charitan
[Software System]

Figure 2 Container diagram

- Actor: Admin, Charity, Donor, Guest.
  - Admin: People who overview the system. They can manage user account, projects.
  - Charity: Can individuals, companies, non-profit organizations, who create and manage their charitable projects.
  - Donor: User who can manage their personal inform visit Charity page and join their charitable project to donate
  - Guest: Who can view the project and make private donation.

- Software system: Charitan Software System allow people to organize, refund and support charity campaigns.
  - Frontend: Authentication, Admin Single-page Application, User Single-page Application. They are the part of an application or website which is visible to the user when accessing the site. They are the outermost part of the system where the actions like clicking, typing or viewing are exercised by the user.
  - Backend: API Gateway, User Module, Admin Module, Statistics Module, Database. They describe the unseen part of an application or system where computation, storage and other dynamics that are masked from the

users take place. They are charged with the responsibility of ensuring the system computes properly.

- External system: Email System and PayPal.
  - Email System: Allow Charitan send message to users when they create charitable project, or donate a project successfully.
  - PayPal: Allow Donor and Guest donate projects.

4.2 Back-end Architecture:

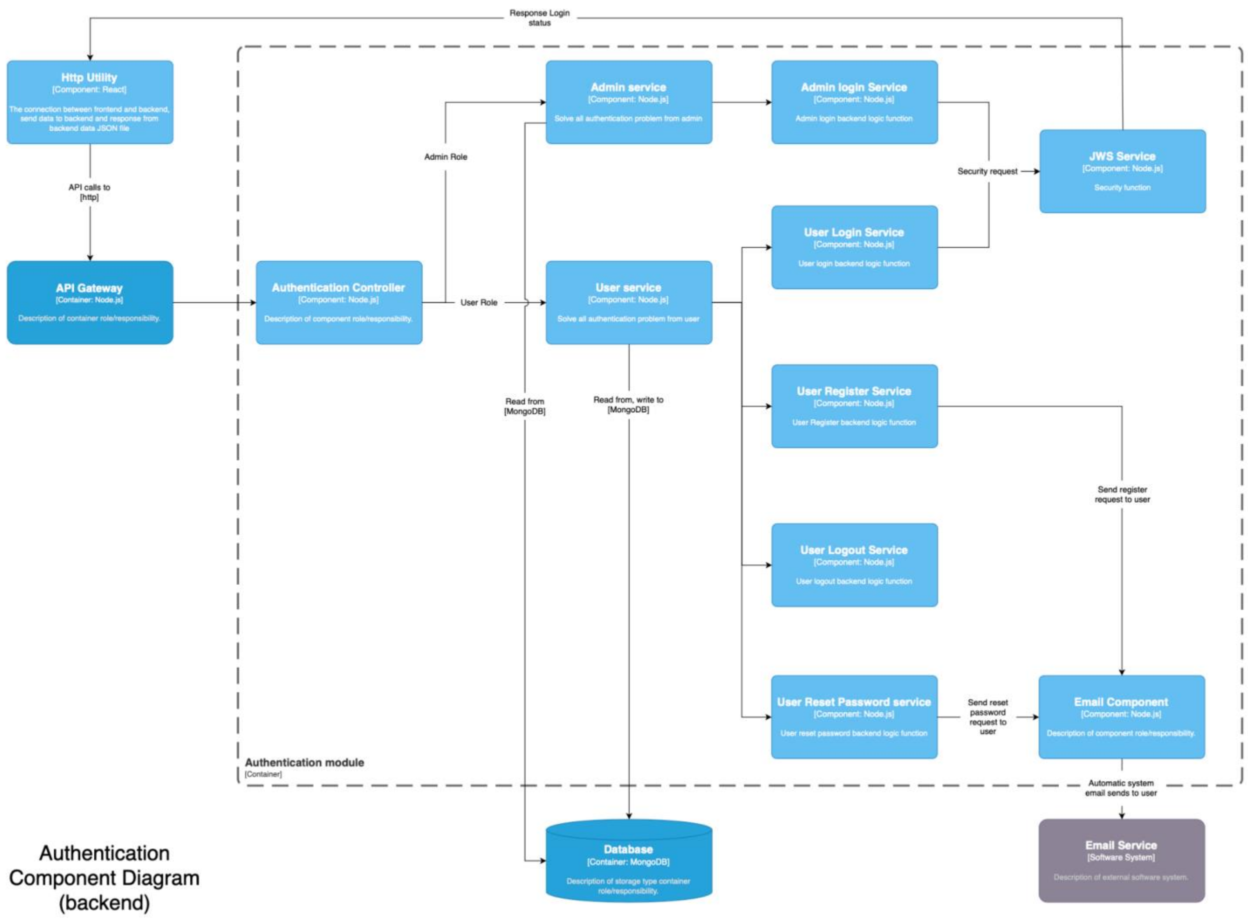

Response Login status
Http Utility
[Component: React]
The connection between frontend and backend, send data to backend and response from backend data JSON file
API calls to [http]
API Gateway
[Container: Node.js]
Description of container role/responsibility.
Authentication Controller
[Component: Node.js]
Description of component role/responsibility.
Admin Role
User Role
Admin service
[Component: Node.js]
Solve all authentication problem from admin
User service
[Component: Node.js]
Solve all authentication problem from user
Read from [MongoDB]
Read from, write to [MongoDB]
Admin login Service
[Component: Node.js]
Admin login backend logic function
Security request
JWS Service
[Component: Node.js]
Security function
User Login Service
[Component: Node.js]
User login backend logic function
User Register Service
[Component: Node.js]
User Register backend logic function
Send register request to user
User Logout Service
[Component: Node.js]
User logout backend logic function
User Reset Password service
[Component: Node.js]
User reset password backend logic function
Send reset password request to user
Email Component
[Component: Node.js]
Description of component role/responsibility.
Authentication module
[Container]
Automatic system email sends to user
Database
[Container: MongoDB]
Description of storage type container role/responsibility.
Email Service
[Software System]
Description of external software system.
Authentication Component Diagram (backend)

Figure 3 Authentication component diagram (backend)

This is authentication back-end component diagram.

- Component: Authentication Controller, Admin service, User service, Admin login service, User Login service, User Register service, User Logout service, User Reset Password service, JWS service, Email component.

When receive data through HTTP utility send to back-end, Authentication Controller will distinguish Admin or user role to read data from database, the services are the back-end logic functions solving authentication request sent from front-end. JWS service is a secure function to authorize account. Email component is used to send auto system email while register or reset password.

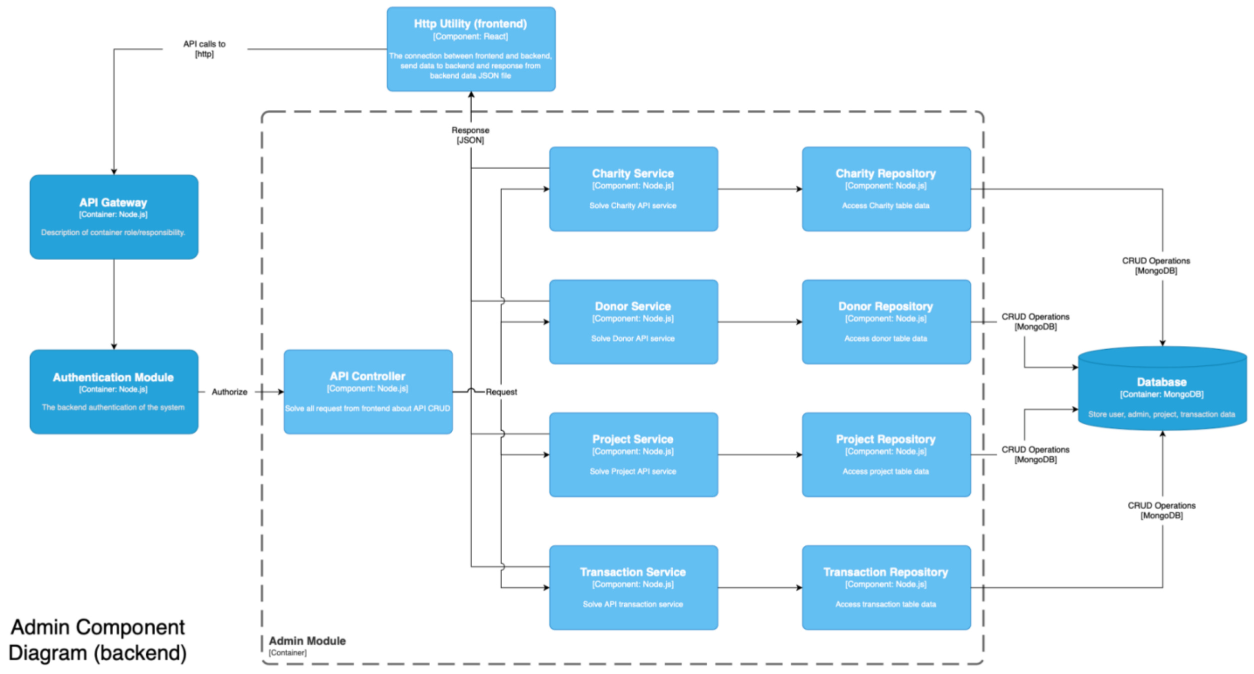


Figure 4 Admin component diagram (back-end)

This is the admin back-end component diagram, which show how admin system back-end work. Admin role can overview and manga all data in the system. To start the back-end, they need to authorize their admin role.

Component:

- API controller: control all CRUD API request.
- Charity, Donor, Project, Transaction Service: Function solve classified data CRUD request service, also response data like JSON file to front-end.
- Charity, Donor, Project, Transaction Repository: Access data table from database.

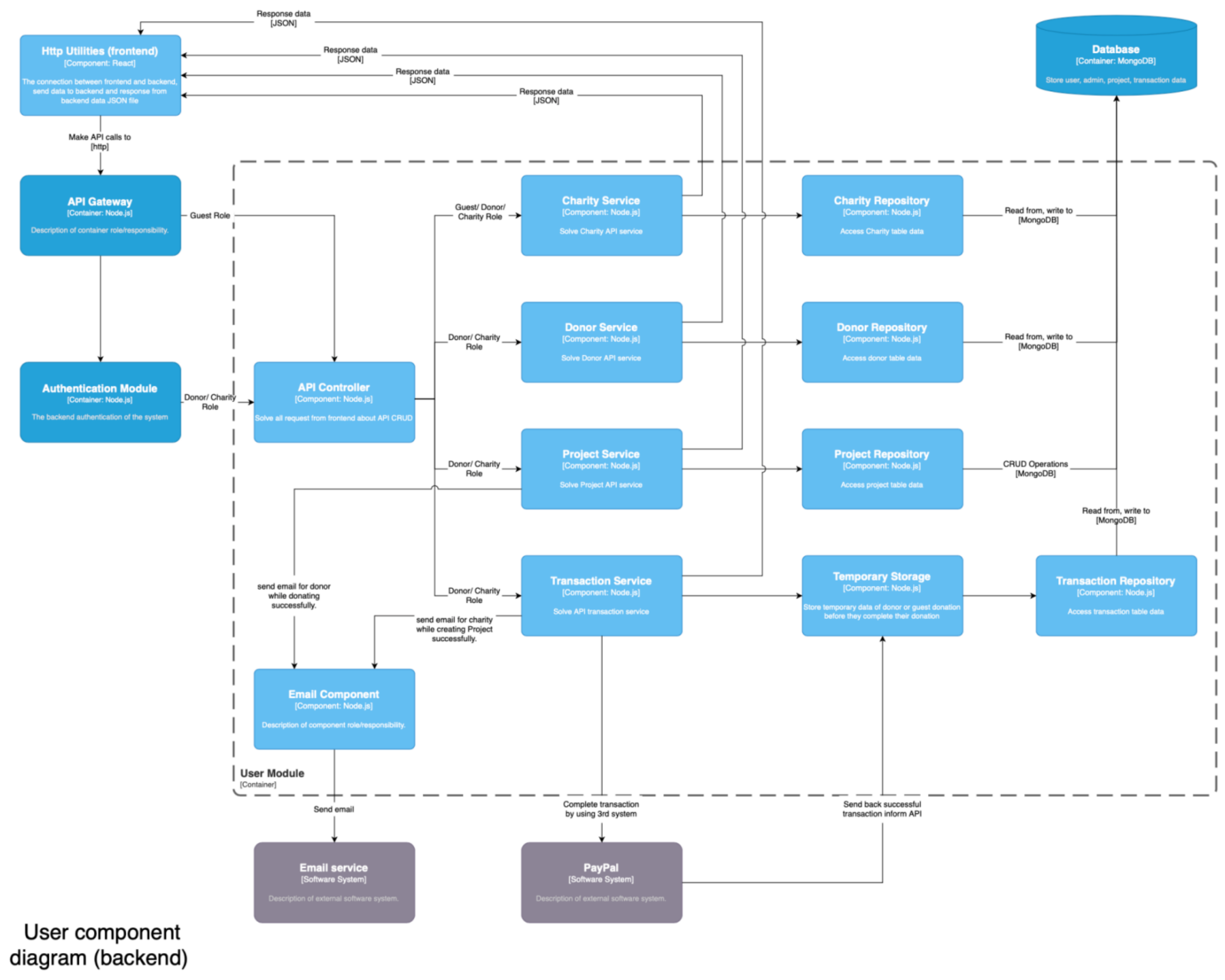

Response data [JSON]
Response data [JSON]
Response data [JSON]
Response data [JSON]
Http Utilities (frontend)
[Component: React]
The connection between frontend and backend, send data to backend and response from backend data JSON file
Make API calls to [http]
API Gateway
[Container: Node.js]
Description of container role/responsibility.
Guest Role
Authentication Module
[Container: Node.js]
The backend authentication of the system
Donor/ Charity Role
API Controller
[Component: Node.js]
Solve all request from frontend about API CRUD
Guest/ Donor/ Charity Role
Donor/ Charity Role
Donor/ Charity Role
Donor/ Charity Role
Charity Service
[Component: Node.js]
Solve Charity API service
Donor Service
[Component: Node.js]
Solve Donor API service
Project Service
[Component: Node.js]
Solve Project API service
Transaction Service
[Component: Node.js]
Solve API transaction service
Charity Repository
[Component: Node.js]
Access Charity table data
Donor Repository
[Component: Node.js]
Access donor table data
Project Repository
[Component: Node.js]
Access project table data
Temporary Storage
[Component: Node.js]
Store temporary data of donor or guest donation before they complete their donation
Transaction Repository
[Component: Node.js]
Access transaction table data
Read from, write to [MongoDB]
Read from, write to [MongoDB]
CRUD Operations [MongoDB]
Read from, write to [MongoDB]
Database
[Container: MongoDB]
Store user, admin, project, transaction data
send email for donor while donating successfully.
send email for charity while creating Project successfully.
Email Component
[Component: Node.js]
Description of component role/responsibility.
User Module
[Container]
Send email
Complete transaction by using 3rd system
Send back successful transaction inform API
Email service
[Software System]
Description of external software system.
PayPal
[Software System]
Description of external software system.
User component diagram (backend)

Figure 5 User component diagram (backend)

This is the user back-end component diagram, which shows how user role work on system back-end.

Charity, Donor, and Transaction data can be read and written. Project data can be read, written and deleted through the user interacting this application.

The back-end can send automatic email while charity creates successfully project, donor donates successfully a project.

Also, the system perform the transaction through 3rd system Paypal.

Component: API controller, Charity Service, Donor Service, Project Service, Transaction Service, Charity Repository, Donor Repository, Project Repository, Transaction Repository, Temporary Storage, Email Component.

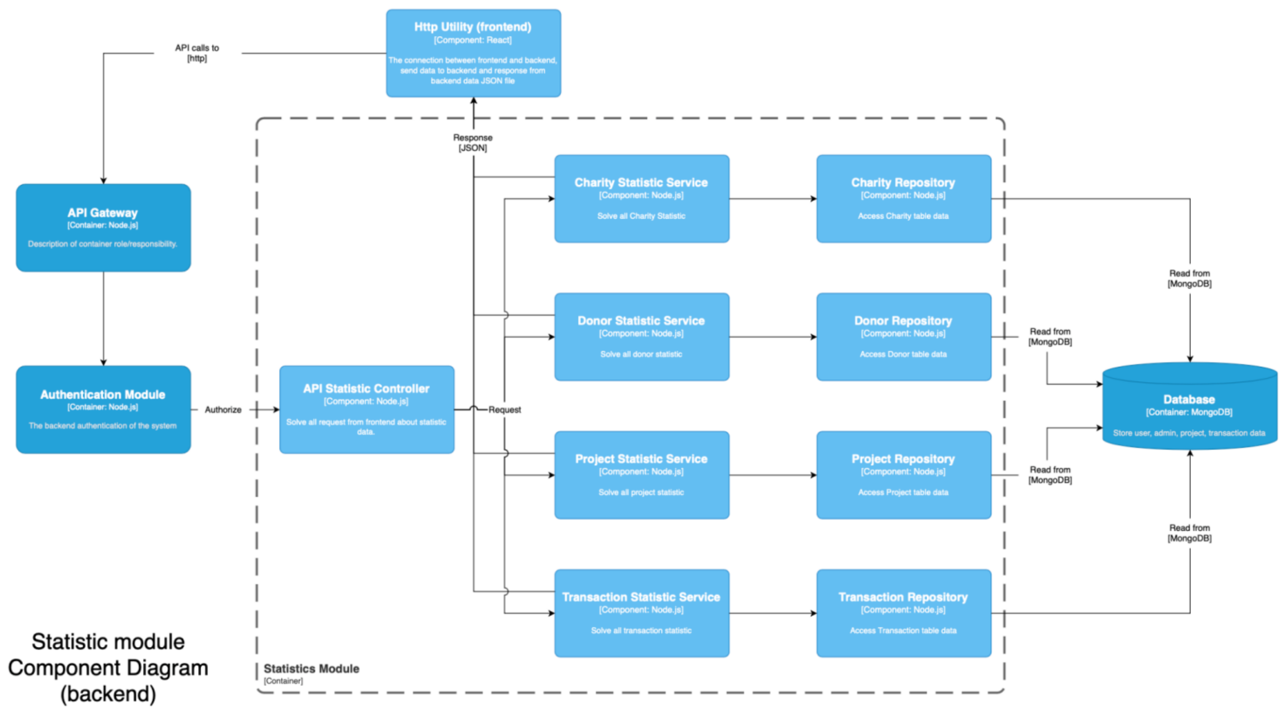


Figure 6 Statistic module component diagram (back-end)

This is the statistic module component back-end diagram, which show how the system analyse the data for both user and admin, the result will send back like a JSON file to the front-end through HTTP Utility.

Component: API Statistic Controller, Charity Statistic Service, Donor Statistic Service, Project Statistic Service, Transaction Statistic Service, Charity Repository, Donor Repository, Project Repository, Transaction Repository.

4.3 Front-end Architecture:

4.3.1 Authentication component diagram

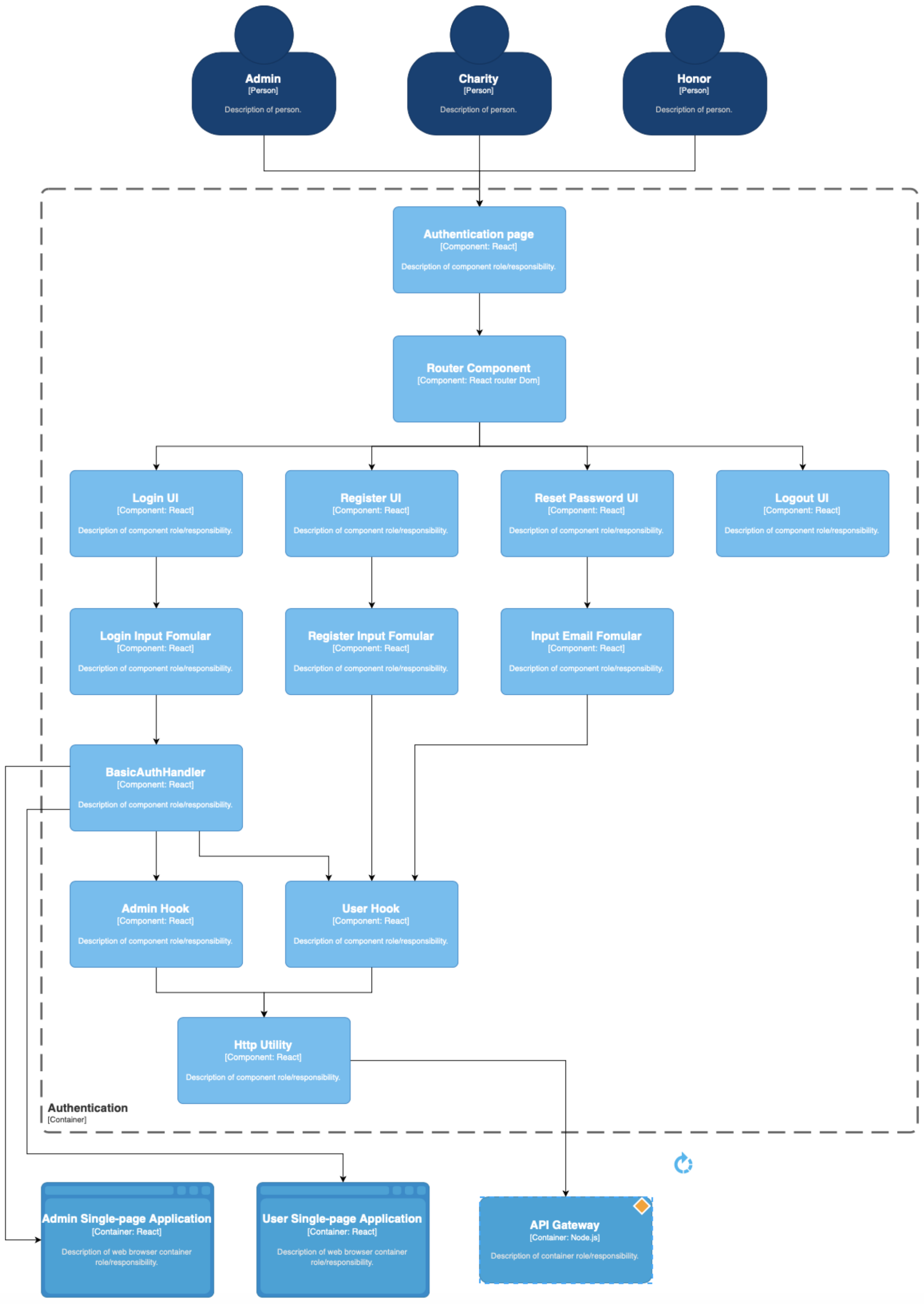

Admin
[Person]
Description of person.
Charity
[Person]
Description of person.
Honor
[Person]
Description of person.
Authentication page
[Component: React]
Description of component role/responsibility.
Router Component
[Component: React router Dom]
Login UI
[Component: React]
Description of component role/responsibility.
Register UI
[Component: React]
Description of component role/responsibility.
Reset Password UI
[Component: React]
Description of component role/responsibility.
Logout UI
[Component: React]
Description of component role/responsibility.
Login Input Fomular
[Component: React]
Description of component role/responsibility.
Register Input Fomular
[Component: React]
Description of component role/responsibility.
Input Email Fomular
[Component: React]
Description of component role/responsibility.
BasicAuthHandler
[Component: React]
Description of component role/responsibility.
Admin Hook
[Component: React]
Description of component role/responsibility.
User Hook
[Component: React]
Description of component role/responsibility.
Http Utility
[Component: React]
Description of component role/responsibility.
Authentication
[Container]
Admin Single-page Application
[Container: React]
Description of web browser container role/responsibility.
User Single-page Application
[Container: React]
Description of web browser container role/responsibility.
API Gateway
[Container: Node.js]
Description of container role/responsibility.

Figure 7 Authentication component diagram (front-end)

This is the authentication component front-end diagram. On website, it can be seen on nav bar and it combine login, register, logout, reset password UI.

Each UI has their input formula, Admin can just login into the system.

User can use all service in Authentication, system will send a confirmed email for user who register new account or reset password.

When login, front-end will modified username and password by basic auth and send it to back-end. Through the back-end verified the inform, they will join with admin role or user role.

4.3.2 Admin component diagram

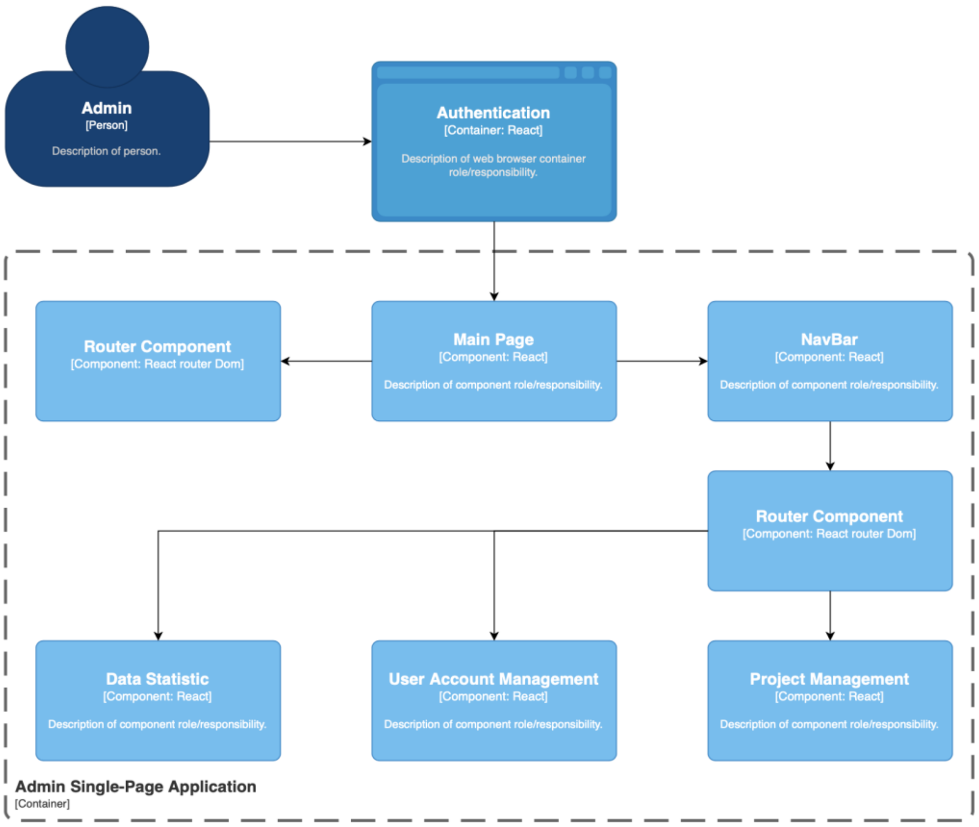

Admin
[Person]
Description of person.
Authentication
[Container: React]
Description of web browser container role/responsibility.
Router Component
[Component: React router Dom]
Main Page
[Component: React]
Description of component role/responsibility.
NavBar
[Component: React]
Description of component role/responsibility.
Router Component
[Component: React router Dom]
Data Statistic
[Component: React]
Description of component role/responsibility.
User Account Management
[Component: React]
Description of component role/responsibility.
Project Management
[Component: React]
Description of component role/responsibility.
Admin Single-Page Application
[Container]

Figure 8 Admin component diagram (front-end) (1)

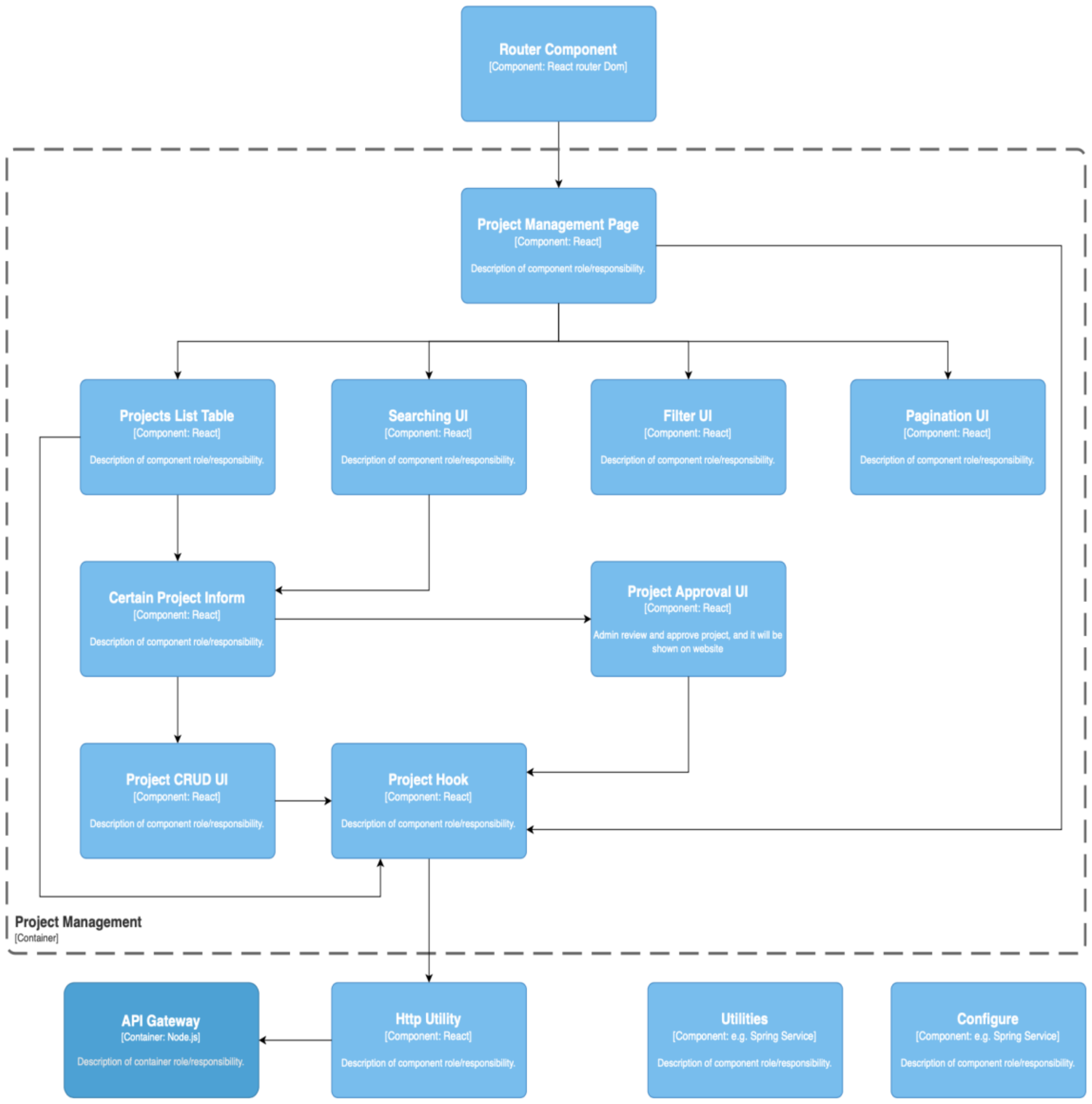

Router Component
[Component: React router Dom]
Project Management Page
[Component: React]
Description of component role/responsibility.
Projects List Table
[Component: React]
Description of component role/responsibility.
Searching UI
[Component: React]
Description of component role/responsibility.
Filter UI
[Component: React]
Description of component role/responsibility.
Pagination UI
[Component: React]
Description of component role/responsibility.
Certain Project Inform
[Component: React]
Description of component role/responsibility.
Project Approval UI
[Component: React]
Admin review and approve project, and it will be shown on website
Project CRUD UI
[Component: React]
Description of component role/responsibility.
Project Hook
[Component: React]
Description of component role/responsibility.
Project Management
[Container]
API Gateway
[Container: Node.js]
Description of container role/responsibility.
Http Utility
[Component: React]
Description of component role/responsibility.
Utilities
[Component: e.g. Spring Service]
Description of component role/responsibility.
Configure
[Component: e.g. Spring Service]
Description of component role/responsibility.

Figure 9 Admin component diagram (front-end) (2)

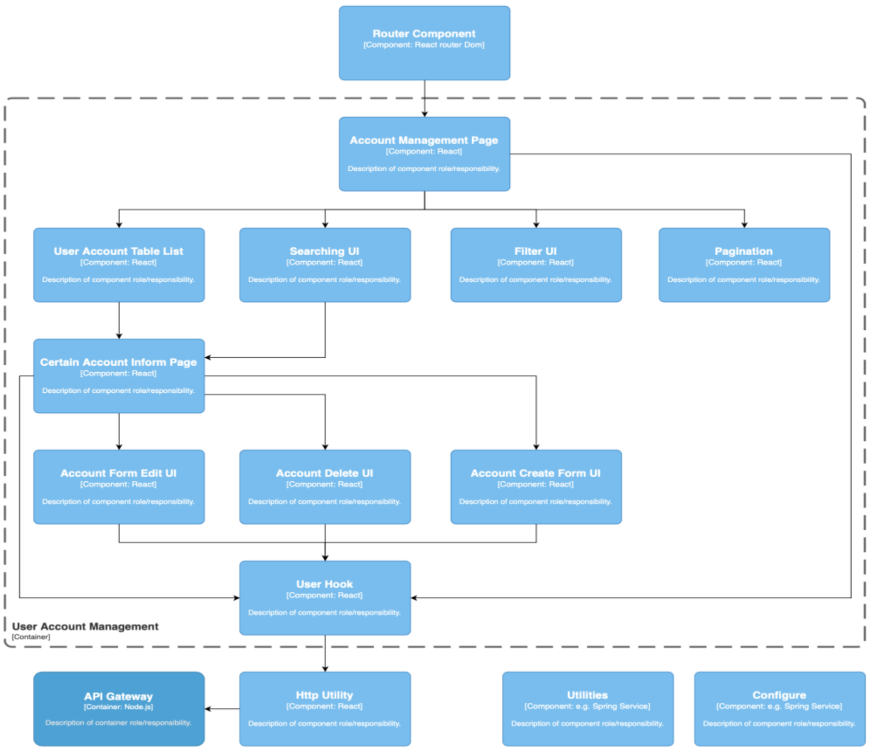

Router Component
[Component: React router Dom]
Account Management Page
[Component: React]
Description of component role/responsibility.
User Account Table List
[Component: React]
Description of component role/responsibility.
Searching UI
[Component: React]
Description of component role/responsibility.
Filter UI
[Component: React]
Description of component role/responsibility.
Pagination
[Component: React]
Description of component role/responsibility.
Certain Account Inform Page
[Component: React]
Description of component role/responsibility.
Account Form Edit UI
[Component: React]
Description of component role/responsibility.
Account Delete UI
[Component: React]
Description of component role/responsibility.
Account Create Form UI
[Component: React]
Description of component role/responsibility.
User Hook
[Component: React]
Description of component role/responsibility.
User Account Management
[Container]
API Gateway
[Container: Node.js]
Description of container role/responsibility.
Http Utility
[Component: React]
Description of component role/responsibility.
Utilities
[Component: e.g. Spring Service]
Description of component role/responsibility.
Configure
[Component: e.g. Spring Service]
Description of component role/responsibility.

Figure 10 Admin component diagram (front-end) (3)

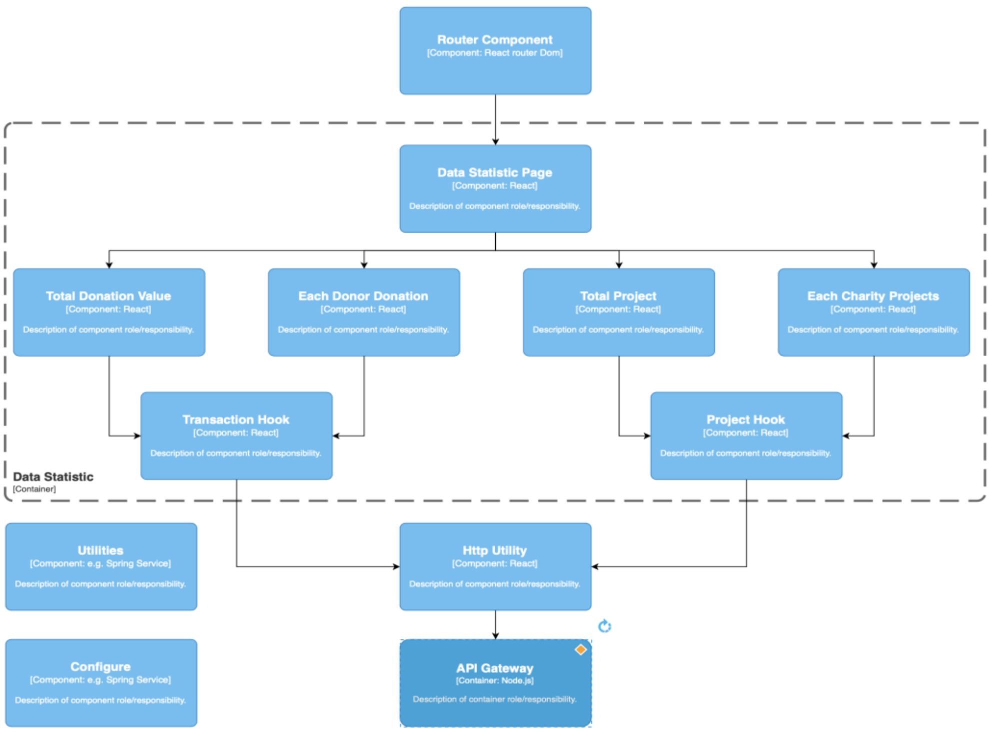

Router Component
[Component: React router Dom]
Data Statistic Page
[Component: React]
Description of component role/responsibility.
Total Donation Value
[Component: React]
Description of component role/responsibility.
Each Donor Donation
[Component: React]
Description of component role/responsibility.
Total Project
[Component: React]
Description of component role/responsibility.
Each Charity Projects
[Component: React]
Description of component role/responsibility.
Transaction Hook
[Component: React]
Description of component role/responsibility.
Project Hook
[Component: React]
Description of component role/responsibility.
Data Statistic
[Container]
Utilities
[Component: e.g. Spring Service]
Description of component role/responsibility.
Http Utility
[Component: React]
Description of component role/responsibility.
Configure
[Component: e.g. Spring Service]
Description of component role/responsibility.
API Gateway
[Container: Node.js]
Description of container role/responsibility.

Figure 11 Admin component diagram (front-end) (4)

To use the app with admin role, need login by admin account first. The admin system has project management page, user account management page, and data statistic page (figure 8)

Project management page has list table of charitable project, searching UI, filter UI and Pagination UI. It helps admin better to find project. When admin click on certain project, they can halt, edit, delete it. Also they can approve new projects. (figure 9)

Account management is the same. Moreover, admin can sent system email for new created account. (figure 10)

Data statistic page help admin better manage Donation and Project. (figure 11)

4.3.2 Charity component diagram

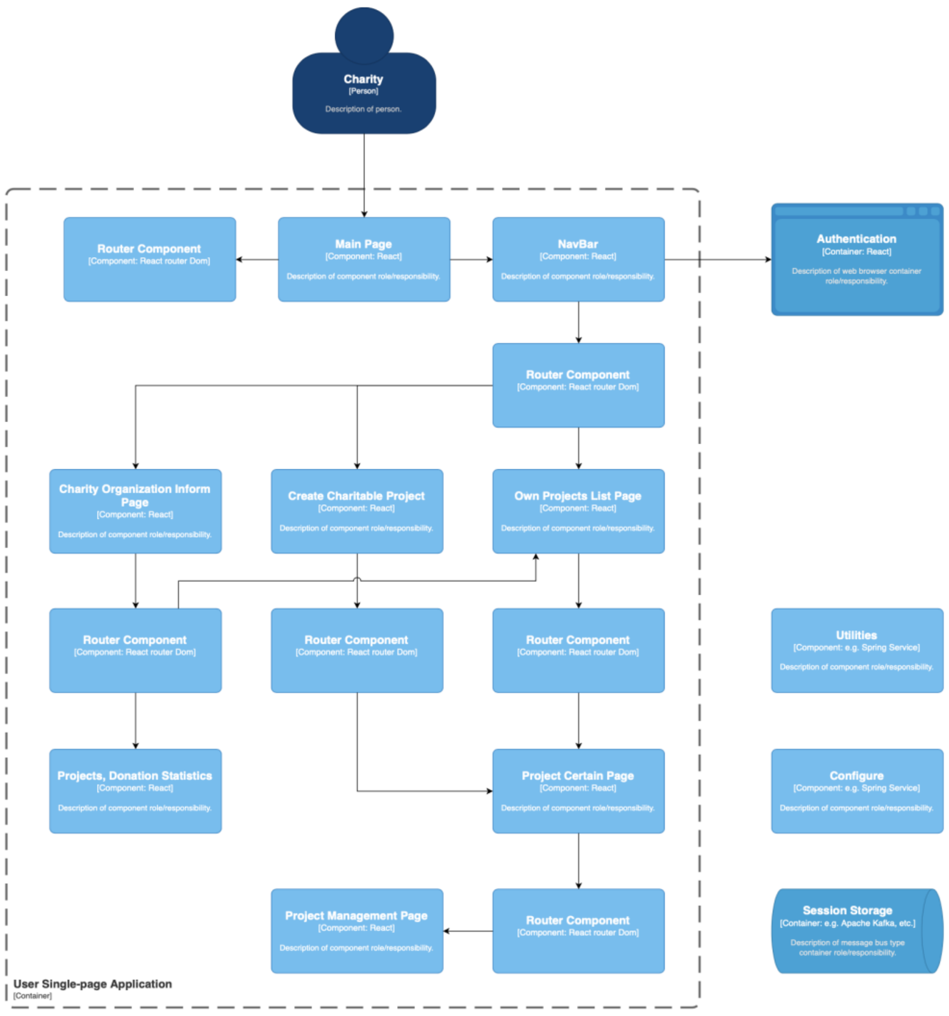

Charity
[Person]
Description of person.
Router Component
[Component: React router Dom]
Main Page
[Component: React]
Description of component role/responsibility.
NavBar
[Component: React]
Description of component role/responsibility.
Authentication
[Container: React]
Description of web browser container role/responsibility.
Router Component
[Component: React router Dom]
Charity Organization Inform Page
[Component: React]
Description of component role/responsibility.
Create Charitable Project
[Component: React]
Description of component role/responsibility.
Own Projects List Page
[Component: React]
Description of component role/responsibility.
Router Component
[Component: React router Dom]
Router Component
[Component: React router Dom]
Router Component
[Component: React router Dom]
Utilities
[Component: e.g. Spring Service]
Description of component role/responsibility.
Projects, Donation Statistics
[Component: React]
Description of component role/responsibility.
Project Certain Page
[Component: React]
Description of component role/responsibility.
Configure
[Component: e.g. Spring Service]
Description of component role/responsibility.
Project Management Page
[Component: React]
Description of component role/responsibility.
Router Component
[Component: React router Dom]
Session Storage
[Container: e.g. Apache Kafka, etc.]
Description of message bus type container role/responsibility.
User Single-page Application
[Container]

Figure 12 Charity component diagram (front-end) (1)

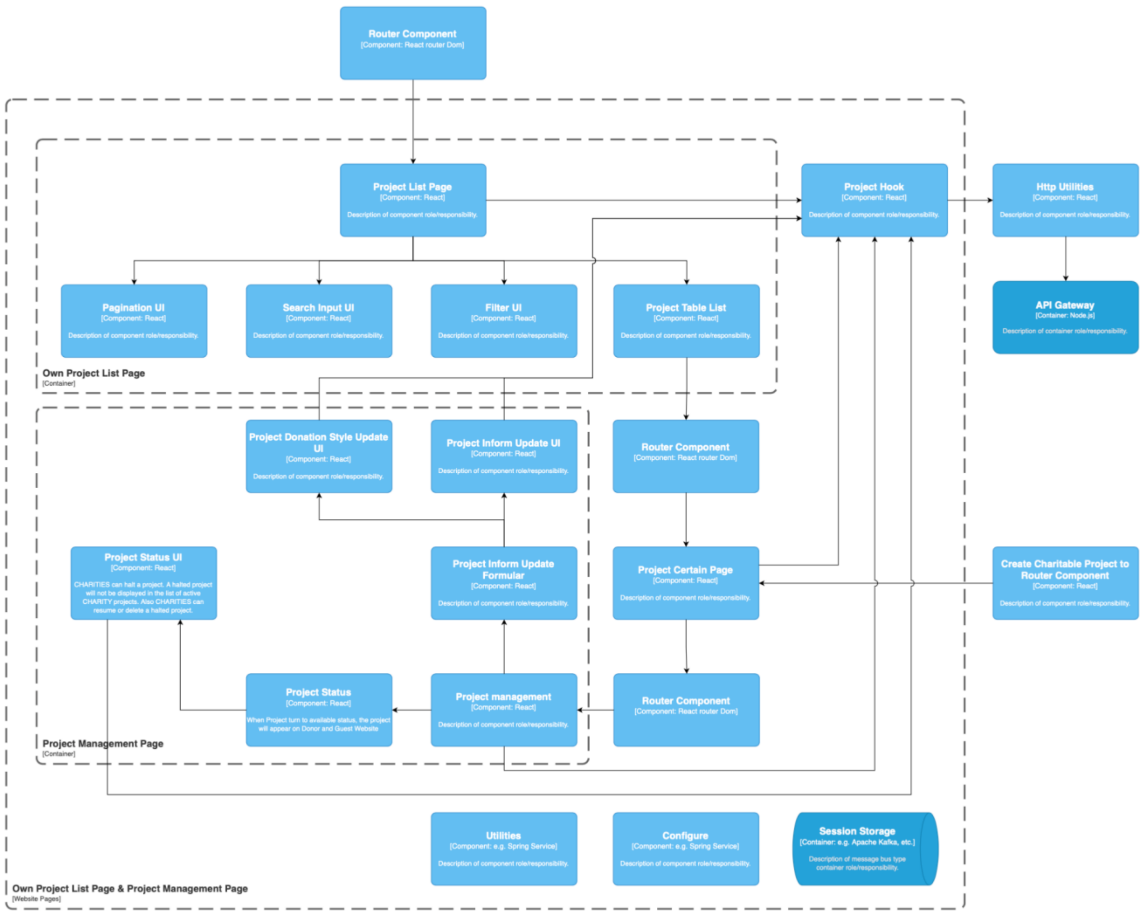

Router Component
[Component: React router Dom]
Project List Page
[Component: React]
Description of component role/responsibility.
Project Hook
[Component: React]
Description of component role/responsibility.
Http Utilities
[Component: React]
Description of component role/responsibility.
API Gateway
[Container: Node.js]
Description of container role/responsibility.
Pagination UI
[Component: React]
Description of component role/responsibility.
Search Input UI
[Component: React]
Description of component role/responsibility.
Filter UI
[Component: React]
Description of component role/responsibility.
Project Table List
[Component: React]
Description of component role/responsibility.
Own Project List Page
[Container]
Project Donation Style Update UI
[Component: React]
Description of component role/responsibility.
Project Inform Update UI
[Component: React]
Description of component role/responsibility.
Router Component
[Component: React router Dom]
Project Status UI
[Component: React]
CHARITIES can halt a project. A halted project will not be displayed in the list of active CHARITY projects. Also CHARITIES can resume or delete a halted project.
Project Inform Update Formular
[Component: React]
Description of component role/responsibility.
Project Certain Page
[Component: React]
Description of component role/responsibility.
Create Charitable Project to Router Component
[Component: React]
Description of component role/responsibility.
Project Status
[Component: React]
When Project turn to available status, the project will appear on Donor and Guest Website
Project management
[Component: React]
Description of component role/responsibility.
Router Component
[Component: React router Dom]
Project Management Page
[Container]
Utilities
[Component: e.g. Spring Service]
Description of component role/responsibility.
Configure
[Component: e.g. Spring Service]
Description of component role/responsibility.
Session Storage
[Container: e.g. Apache Kafka, etc.]
Description of message bus type container role/responsibility.
Own Project List Page & Project Management Page
[Website Pages]

Figure 13 Charity component diagram (front-end) (2)

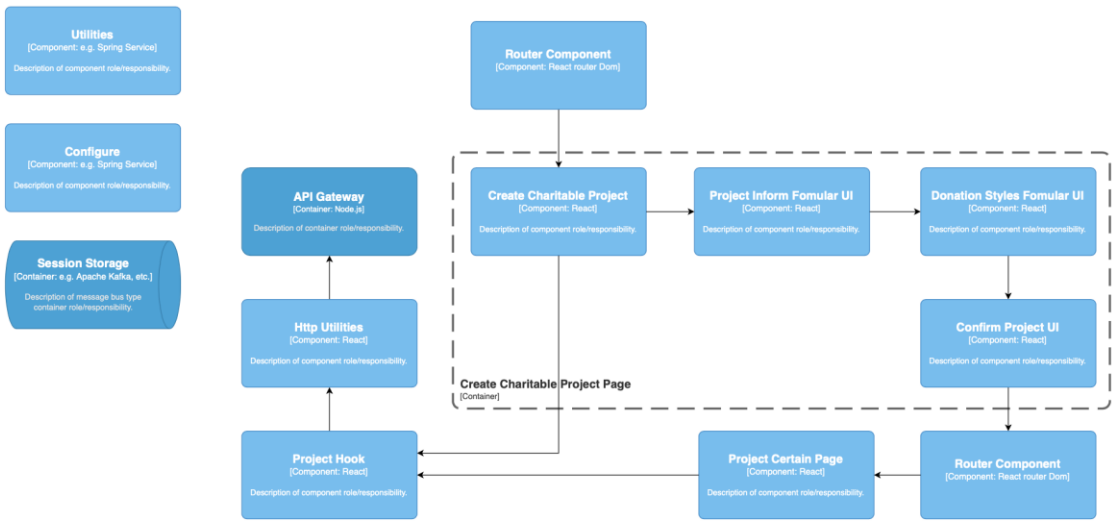

Utilities
[Component: e.g. Spring Service]
Description of component role/responsibility.
Configure
[Component: e.g. Spring Service]
Description of component role/responsibility.
Session Storage
[Container: e.g. Apache Kafka, etc.]
Description of message bus type container role/responsibility.
Router Component
[Component: React router Dom]
API Gateway
[Container: Node.js]
Description of container role/responsibility.
Create Charitable Project
[Component: React]
Description of component role/responsibility.
Project Inform Fomular UI
[Component: React]
Description of component role/responsibility.
Donation Styles Fomular UI
[Component: React]
Description of component role/responsibility.
Http Utilities
[Component: React]
Description of component role/responsibility.
Confirm Project UI
[Component: React]
Description of component role/responsibility.
Create Charitable Project Page
[Container]
Project Hook
[Component: React]
Description of component role/responsibility.
Project Certain Page
[Component: React]
Description of component role/responsibility.
Router Component
[Component: React router Dom]

Figure 14 Charity component diagram (front-end) (3)

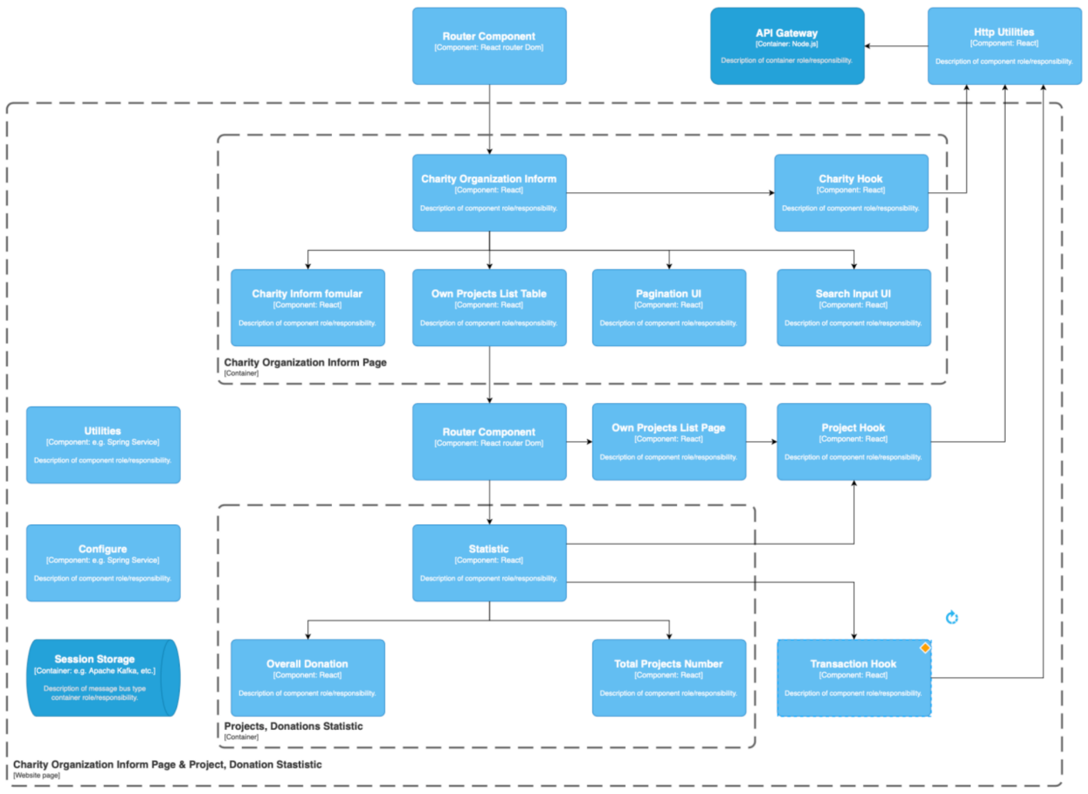

Router Component
[Component: React router Dom]
API Gateway
[Container: Node.js]
Description of container role/responsibility.
Http Utilities
[Component: React]
Description of component role/responsibility.
Charity Organization Inform
[Component: React]
Charity Hook
[Component: React]
Charity Inform fomular
Own Projects List Table
Pagination UI
Search Input UI
Charity Organization Inform Page
[Container]
Utilities
[Component: e.g. Spring Service]
Router Component
Own Projects List Page
Project Hook
Configure
[Component: e.g. Spring Service]
Statistic
Session Storage
[Container: e.g. Apache Kafka, etc.]
Description of message bus type container role/responsibility.
Overall Donation
Total Projects Number
Transaction Hook
Projects, Donations Statistic
[Container]
Charity Organization Inform Page & Project, Donation Stastistic
[Website page]

Figure 15 Charity component diagram (front-end) (4)

This diagram show the APP front-end in charity role. The main page compile Charity Organization inform page, create charitable project, and Own project list page. Open Charity Organization inform page also can go to Project, Donations Statistic or Own project list page and final go to project certain page. Create charitable project also can go to created project certain page. The project certain page go to project management page(figure 12).

Own project list page combines project table list, searching UI, filter UI, pagination UI. Project management page combines project inform update UI, donation inform update UI, and project status UI (Charity can halt and resume the project, halted project can be deleted.) (figure 13)

Create charitable project page start to input project inform, project donation style, and then click on confirm button. (figure 14)

Charity Organization inform page combine charity inform, project table list, pagination UI, and search UI. Projects, donations statistic page combine statistic overall donation and total project number. (figure 15)

4.3.3 Donor/ Guest component diagram

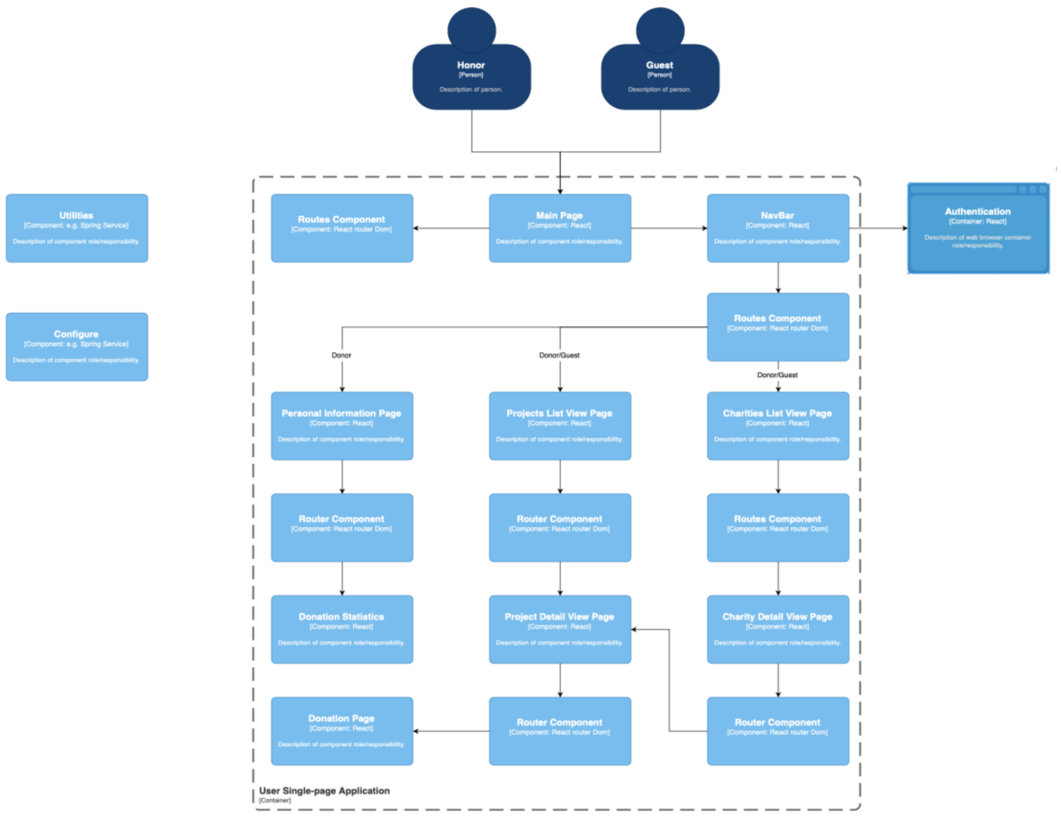


Figure 16 Donor/ Guest component diagram (frontend) (1)

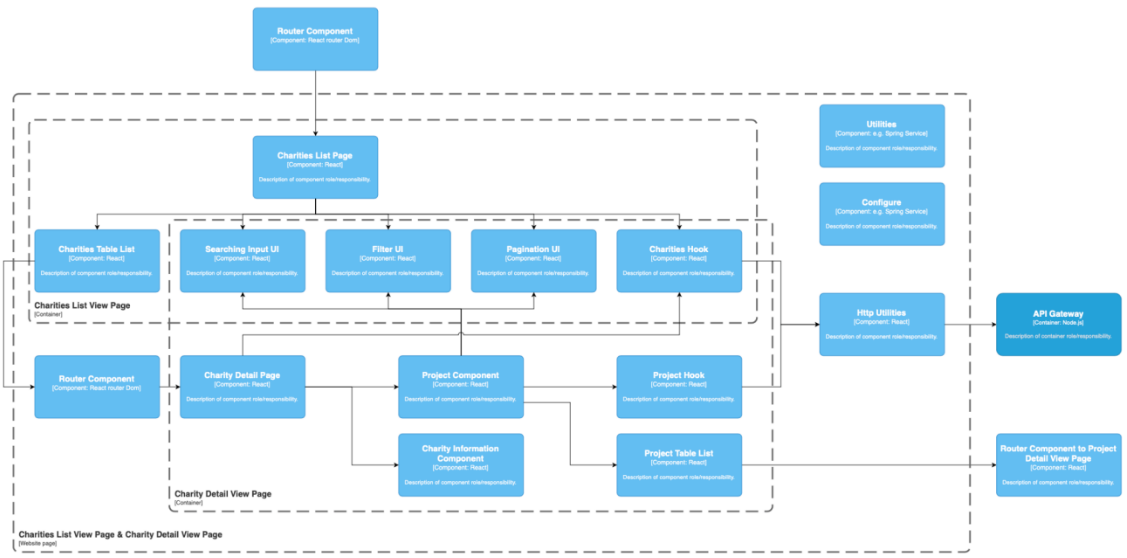


Figure 17 Donor/ Guest component diagram (frontend) (2)

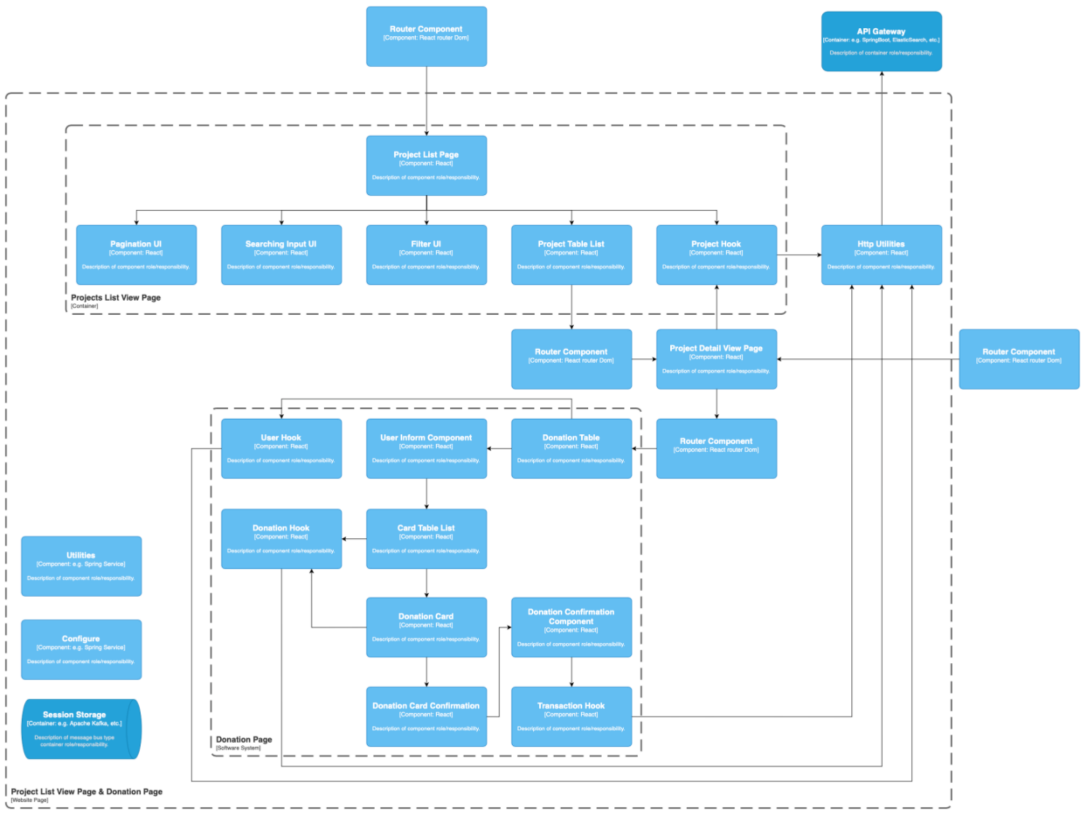

Router Component [Component: React router Dom]
API Gateway [Container: e.g. SpringBoot, ElasticSearch, etc.] Description of container role/responsibility.
Project List Page [Component: React] Description of component role/responsibility.
Pagination UI [Component: React] Description of component role/responsibility.
Searching Input UI [Component: React] Description of component role/responsibility.
Filter UI [Component: React] Description of component role/responsibility.
Project Table List [Component: React] Description of component role/responsibility.
Project Hook [Component: React] Description of component role/responsibility.
Http Utilities [Component: React] Description of component role/responsibility.
Projects List View Page [Container]
Router Component [Component: React router Dom]
Project Detail View Page [Component: React] Description of component role/responsibility.
Router Component [Component: React router Dom]
User Hook [Component: React] Description of component role/responsibility.
User Inform Component [Component: React] Description of component role/responsibility.
Donation Table [Component: React] Description of component role/responsibility.
Router Component [Component: React router Dom]
Donation Hook [Component: React] Description of component role/responsibility.
Card Table List [Component: React] Description of component role/responsibility.
Utilities [Component: e.g. Spring Service] Description of component role/responsibility.
Donation Card [Component: React] Description of component role/responsibility.
Donation Confirmation Component [Component: React] Description of component role/responsibility.
Configure [Component: e.g. Spring Service] Description of component role/responsibility.
Session Storage [Container: e.g. Apache Kafka, etc.] Description of message bus type container role/responsibility.
Donation Card Confirmation [Component: React] Description of component role/responsibility.
Transaction Hook [Component: React] Description of component role/responsibility.
Donation Page [Software System]
Project List View Page & Donation Page [Website Page]

Figure 18 Donor/ Guest component diagram (frontend) (3)

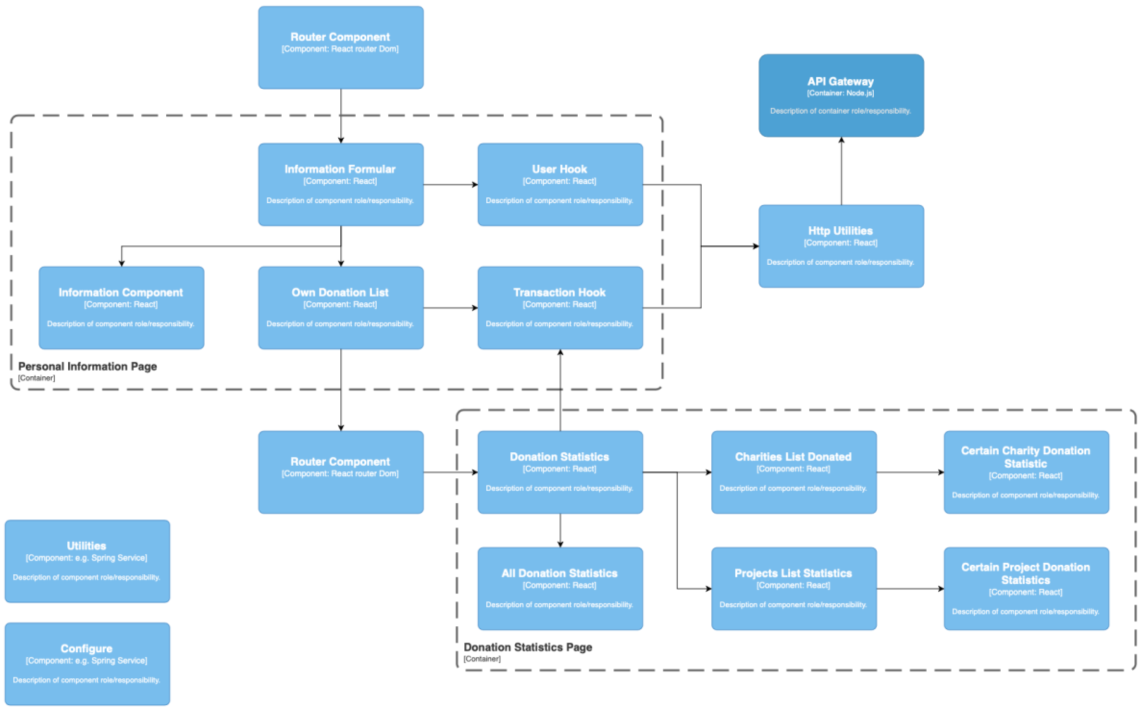

Router Component [Component: React router Dom]
API Gateway [Container: Node.js] Description of container role/responsibility.
Information Formular [Component: React] Description of component role/responsibility.
User Hook [Component: React] Description of component role/responsibility.
Http Utilities [Component: React] Description of component role/responsibility.
Information Component [Component: React] Description of component role/responsibility.
Own Donation List [Component: React] Description of component role/responsibility.
Transaction Hook [Component: React] Description of component role/responsibility.
Personal Information Page [Container]
Router Component [Component: React router Dom]
Donation Statistics [Component: React] Description of component role/responsibility.
Charities List Donated [Component: React] Description of component role/responsibility.
Certain Charity Donation Statistic [Component: React] Description of component role/responsibility.
Utilities [Component: e.g. Spring Service] Description of component role/responsibility.
All Donation Statistics [Component: React] Description of component role/responsibility.
Projects List Statistics [Component: React] Description of component role/responsibility.
Certain Project Donation Statistics [Component: React] Description of component role/responsibility.
Configure [Component: e.g. Spring Service] Description of component role/responsibility.
Donation Statistics Page [Container]

Figure 19 Donor/ Guest component diagram (frontend) (4)

This is Donor/ Guest component diagram. After users login, they will go to the main page. It has nav bar combined charities list view page, projects list view page, and personal inform page (just for Donor). Charities list view page goes to certain charity page, then goes to certain project page. Projects list view page goes to certain project page. So donor and guest can make a donation. Personal inform page (just for donor) shows their information and donation statistic. (figure 16)

Charities view page has charities list table, searching, filter and pagination UI. Open certain charity page, it also has its information and a projects list table. There also has searching filter and pagination UI for find project. (figure 17)

Projects list view page combines projects list, searching UI, filter UI, pagination UI to Donor and guest find a project. After view the project, they can make a donation. Donation page has Donor inform, if users are guest, they can add some basic information. Then there are a donation style list enable that they can choose. The donation card will be confirmed and the final state is completing the transaction. (figure 18)

Personal inform page that just for Donor, it show their information and total value they had donated. It all so has donation statistic for user that what they had paid for. (figure 19)

5. Architecture Analysis

1. Maintainability:

- Design choices for maintainability

- Architectural Approach:

  Level 1: So we use N - Tier Architecture that separates the Presentation, Business Logic, and Data Access layers.

  Level 2: Enforce strict API access rules by encapsulating business features into modules bound by contexts as a tactic of Modular Monolith.

  Level 3: Scalability and Independent Deployment Using Microservices.

  For scalability, our services will share their data across different MySQL databases, communicate using Kafka, etc.

- Microservices Evolution: Stick with a modular monolith for ease of use and then always start big and scale up to the microservices as you grow. The most important is that every service should contain a specific business logic so that all other services can keep on working during updates.

- Reusable Front-End Components: Reusable UI elements (buttons, forms, modals...) to drive consistency throughout the platform and make the development of new features faster.

  - Testing strategy

- Back-end: Unit Tests: Jest or Mocha to validate individual methods; for example, JWS generation, project approval, etc.
- Integration Tests: Verify inter-service communication: Kafka events and Redis caching.
- Database Tests: Make sure sharding works, and transactions roll back on errors.
- Front-end: Component Tests: You verify UI functionality in isolation.

  End-to-End Tests: Use Cypress to simulate workflows such as user registration, login, and donation.

2. Extensibility:

  - Component modularity

- Front-end: The main content is split into components that are self-contained, reusable, and have the presentation, event handlers, API calls, and styling its files.
- Back-end: Modular Monolith is a design for modularization of features like authentication and user management, to make it maintainable and potentially migrate it into microservices.

  - Interface design

- API: Secure external services like user registration APIs and internal APIs for inter-module communication, Restful APIs.

  JSON Web Tokens (JWT/JWE) provide lightweight secure authentication; responses contain as little data as possible and are always specific to that token.
- UI: Reusable components with consistent and responsive design.
- Caching: Redis makes money by caching common queries to donors or charities.

  - Future enhancement capabilities

- Extensibility: The API is versioned so you can freely add new features (e.g. blockchain, AI).

- Optimization: Database sharding, Redis caching, and Kafka messaging support high traffic and real-time updates.

3. Resilience:

- Error handling

- Front-end: Instead of checking component failure individually within child components themselves, you can set up centralized error boundaries (React ErrorBoundary) for UI fallback specifically. Clearly and user-friendly error messages are implemented for such actions as login failure or incorrect input fields.
- Back-end: Add middleware for global error handling things like (logging, and validation errors). Return error messages for consistency in your API and successfully return standardized error responses (HTTP status codes).

- Failure recovery

- Transaction Management: In case using database transactions to ensure rollback in case of failure (e.g. payment processing).
- Retry Mechanisms: Libraries such as Axios-retry were used to re-attempt the front-end retries of failed requests (network issues). When Kafka retries retry processes such as sending email notices, we have back-end queues that deal with critical processes.
- Data Recovery: Perform audit and debugging by logging all critical operations and errors. Restores from backups for a partial data loss.

- System redundancy

- Database Sharding: Just as partitioned databases can decrease failure occurrences with lower risk by partitioning high-traffic regions or projects.
- Message Broker: For events such as project halted or real-time donor notifications, Kafka does reliable, asynchronous messaging.
- Load Balancing: Load balancers distribute requests equally on the servers, and maintain uptime when server demand is high.

4. Scalability:

- Load handling

- Asynchronous Operations: Use Kafka message queues for non-blocking tasks (like email notifications, and project halt updates).
  Batch high-frequency operations such as real-time updates for a project.

- Caching: Use Redis for infrequently accessed data thus reducing database load for donor and charity information.
    - Resource management
- Efficient Database Use: Enforcing efficient queries by database sharding to enable the splitting of data by geographic regions or categories. Use connection pooling to pool the database connections across services.
- Resource Limits: Limit resource consumption for Docker containers by making sure they overload or starve. Prevent a user's misuse by monitoring and throttling API requests per user.
- Garbage Collection: Purge old data and stale cache entries at regular intervals. Take care to minimize memory usage in services by dealing with how long objects are persisted.
    - Performance optimization
- API Optimization: Add appropriate indexes to speed up common queries. Instead of returning full database models, APIs should return lightweight DTOs.
- Database Indexing: Speed common queries by adding appropriate indexes. Partitioned indexes are used for shared databases.
- Front-end Optimization: Bundles and compresses assets to minimize HTTP requests. Lazy load components such as user avatars and introduction videos.
- Monitoring: Track system performance metrics like response time and resource usage with tools like Prometheus and Grafana.

5. Security:

- Authentication/Authorization
- Secure Authentication Protocols: Get an OAuth 2.0 or OpenID Connect style authentication protocol up and running. Choose session-based or token-based authentication and use short-lived tokens as well.
- Role-Based Access Control: Define different roles like the donor, charity, and administrator with specific permissions to reduce risks of granting unauthorized access.
- Multi-Factor Authentication: Provide users with optional two factor authentication via e.g. SMS codes, authenticator app, or biometric validation for stronger security.
    - Data protection

- Data Masking and Anonymization: Users see partially masked information such as a user ID and the log looks to have only the partial credit card number, thus hiding sensitive information like full credit card numbers or user IDs.
- Secure File Storage: Upload files like charity documents or profile pictures to a safe place that keeps files in the cloud storage, but locks them with strong encryption and has limited access to these files.
  - Security measures
- Input Validation and Sanitization: All inputs are validated and sanitized to prevent injection attacks, like SQL injection or cross-site scripting (XSS), where can use frameworks that have built-in validation tools.
- API Security Enhancements: Limit API calls to protect our APIs from being brute-force attacked. Exclude IPs that should not interact with sensitive endpoints by IP whitelisting or restrictive endpoints via API keys.

6. Performance:

  - Response time optimization
- Efficient Query Design: Minimize execution time for database queries. Use query optimization techniques, and indexes and avoid unnecessary joins or sub queries.
- Load Balancing: Load balancers are used to distribute incoming traffic across multiple servers to make sure high traffic periods do not cause big differences in response times.
- Asynchronous Processing: Making non-essential operations asynchronous such as sending emails or generating rest email reports is the expected time reduction for user overload.
  - Resource utilization
- Containerization: Containerize as much as possible, packaging up as many resources as possible in lightweight containers such as Docker. When managing and deploying resources, keep within these lightweight containers to ensure isolation, also use resources more efficiently.
- Monitoring and Analytics: Keep an eye on CPU, memory, and network usage with Datadog or Prometheus, and fix any resource bottlenecks as they arise.
  - Caching strategies

- Data Caching: Keeping data about popular charity projects or donation statistics in tools like Redis or Memcached means caching frequently accessed data and loading the database-less often to decrease query time.
- Content Delivery Network: Serve static assets images, videos, and stylesheets for users, closer geographically, to reduce latency using a Content Delivery Network.
- Query Results Caching: Cache the results of common database operations for read-heavy operations.

6. Conclusion

The Charitan platform exemplifies a revolutionary method for connecting funders, volunteers, and philanthropic organizations with creative technological solutions. Charitan confronts the obstacles in online generosity, facilitating users in identifying and supporting significant causes, while allowing charities to enhance their outreach and effectiveness. The platform guarantees a seamless experience for all stakeholders with easy user interfaces, powerful search and filtering, integrated crowdfunding capabilities, and real-time notifications.

The project's stratified architecture, transitioning from N-Tier to Modular Monolith and ultimately to Microservices, exemplifies a resilient and scalable design that emphasizes usability, maintainability, and security. Utilizing advanced technologies like Kafka for real-time updates, Redis for caching, and Docker for microservice deployment, Charitan is strategically equipped to manage intricate operational demands and scale seamlessly with increasing demand.

Through the promotion of collaboration and the use of sophisticated design concepts, Charitan improves the efficacy of philanthropic endeavors while building a culture of generosity. The platform's dedication to harmonizing user requirements with technical proficiency guarantees its potential to create a significant worldwide influence, uniting individuals in pursuit of a better future.

Reference

[1] "The C4 model for visualising software architecture," C4 model, [Online]. Available: https://c4model.com. [Accessed 25 November 2024].

[2] A. Pliutau, "How to Create Software Architecture Diagrams Using the C4 Model," freeCodeCamp, 21 August 2024. [Online]. Available: https://www.freecodecamp.org/news/how-to-create-software-architecture-diagrams-using-the-c4-model/. [Accessed 1 December 2024].

[3] Maxi, "Visualizing Frontend Architecture," FRONTEND AT SCALE, 3 March 2024. [Online]. Available: https://frontendatscale.com/issues/17/. [Accessed 23 November 2024].