EEET2582 Software Engineering: Architecture and Design

Requirements Specification

Charitan Donation Platform

Introduction

In an increasingly interconnected world, the power of collective goodwill can make a profound impact on communities and individuals in need. **Charitan** emerges as a dynamic and robust platform designed to bridge the gap between **DONORS**, **VOLUNTEERS**, and **CHARITY** projects across the globe. By harnessing the potential of technology, **Charitan** aims to create a seamless and engaging experience for all users, facilitating meaningful connections and fostering a spirit of giving.

The platform is built on two primary goals:

First, to assist **DONORS** and **VOLUNTEERS** in discovering and contributing to **CHARITY** initiatives that resonate with their values, whether in their local communities or across international borders. **Charitan** empowers individuals to make informed choices about where their contributions can create the most significant impact.

Second, **Charitan** provides **CHARITIES,** whether individuals, corporations, or non-profit organizations, with the tools they need to effectively crowdfund projects across various sectors, including Food, Health, Education, Environment, Religion, Humanitarian efforts, and Housing. By offering a centralized space for these initiatives, **Charitan** not only enhances visibility but also encourages collaboration and support from a diverse pool of resources.

As we delve deeper into the specifications of this innovative platform, we will explore the functionalities, user interfaces, and technical architecture that will enable **Charitan** to fulfill its mission of connecting hearts and hands in the service of global **CHARITY**.

The functional requirements of the **Charitan** Application are divided into three levels:

- Level 1: Develop the Charitan services to a **minimally functional degree** using **N-Tier Architecture**
- Level 2: Provide the services with **enhanced features to improve usability**, **efficiency**, **responsiveness** (to mobile and PC devices) **maintainability**, and **security** using **Modular Monolith**
- Level 3: Provide services under a **highly maintainable, resilient,** and **extensible Microservice architecture**. The features are excellently optimized to bring **efficiency, convenience**, and **security** to both the users and **Charitan** system.

  You are highly recommended to **analyze the service requirements of all Levels 1, 2, and 3** to decide the **target complexity**, **capability**, **flexibility**, and **goals** of your system architecture and application.

Milestone 1 Requirements

Due Date: **11:00 AM, 4th December 2024**

In this section, every team is required to document the design decisions on the **Data Model**, **System Architecture** in a **30-page report**. The page count does NOT include your cover page and table of contents.

You should analyze which of the following technologies are suitable for the application, and design your architecture accordingly to leverage the potential of the chosen tech stack.

a/ Frontend

- **React-based framework**: e.g., React, Vite, Next.JS
- **Vanilla** (Plain) **HTML**, **CSS**, and **Javascript**

b/ Backend

- **MEN** Stack: MongoDB, ExpressJS, NodeJS
- **Spring Boot**
- **Flask**
- **Kafka (for Level 3)**

R01: Design Conceptual Data Model for Database

Develop an Entity Relationship Model (ER Model) to store information necessary for fulfilling the requirements. Include this ER model in your project report.

- **Conceptual Modeling:** Design a high-level representation of your database schema using appropriate concepts and entities relevant to your system's requirements.
- **Data Requirements:** Ensure the model accommodates all necessary data elements and relationships essential for system functionality.
- **Documentation:** Briefly explain the design ideas and purposes of entities and relationships depicted in your conceptual data model. **Justify the advantages and potential drawbacks of your design.**

R02: Present Architecture Diagram of the Entire System

You are required to separate the front-end and back-end systems in your deliverable. That is, the front-end system is not inside the back-end, and vice-versa.

Create an architecture diagram illustrating the entire system using either UML Component Diagram (Option 1) or C4 Model Container and Component Diagrams (Option 2).

Option 1: UML Component Diagram

Use UML Component Diagram to depict the architecture of your system.

System Level:

- **Architecture:** Illustrate the sub-systems (front-end, back-end, databases, etc.,) in your Charitan application, plus external systems that your front-end and back-end services use.

Back-end:

- **Architecture:** Illustrate how components are organized and interact to **deliver Charitan services**.
- **Utilities Classes and Configuration:** Show helper classes and configurations used in your back-end (e.g., Security, Token Generator, Cookie Configurer).

Front-end:

- **Architecture Breakdown:** Illustrate components, sub-components, and helper classes used in your web application.
- **Interaction with Back-end:** Express how front-end components communicate with the back-end (e.g., REST API calls), which backend components are the intermediate or final recipient of each front-end request.

Note:

- Depict the System Level in **one diagram.** Only specifies the sub-systems, not their modules, in your diagram.
- Depict the back-end and front-end architectures **in two diagrams**.
- In the latter option, only show the component on the other subsystem that directly interact with the currently focused subsystem. For example, if you are drawing the front-end architecture and express that the **User Authentication** Service in the front-end communicates with the **Authentication** module of the back-end, show only the AuthenticationController in the back-end subsystem as the recipient of the request.
- **Component Roles:** Describe the role and responsibilities of key components in the back-end and front-end of your system. Note that **database is considered a back-end element**.

Option 2: C4 Model Component Diagram

Use C4 Model Container and Component Diagrams to depict the architecture of your entire system.

Components Coverage:

- **Back-end and Front-end Components:** Represent components as described in Option 1.
- **Component Descriptions:** Include brief descriptions within each component regarding their purpose and functionality.
- **Include sub-components in your C4 Component Diagram**

General Requirements for Both Options:

- **Clarity and Conciseness:** Ensure the diagrams are clear, concise, and effectively communicate the system architecture.
- **Alignment with Requirements:** Demonstrate how your architecture meets the specified system requirements and design principles.
- **Architectural Rationale:** Concisely justify the architectural choices made for each component**. Justify the advantages and shortcomings of your design** and **state why do you accept the trade-offs in your system, when considering the following six aspects:**

  1/ **Maintainability:** Enables the system to be easily understood, updated, and tested independently by current and future Development Teams.

  2/ **Extensibility**: Enables new features to be added with minimal impact on existing components. This approach allows for future enhancements and adaptations to changing user needs or market demands.

  3/ **Resilience**: The system's ability to continue functioning correctly in the face of unexpected conditions, such as component failures or high loads.

  4/ **Scalability**: The capacity of a system to handle increased load or user demand without sacrificing performance

  5/ **Security**: The measures taken to protect a system and its users from unauthorized access, data breaches, and other threats.

  6/ **Performance**: Performance relates to how quickly and efficiently a system responds to user requests and processes data.

These tasks aim to evaluate your ability to design a robust data model and articulate a coherent system architecture, which are essential skills in software engineering.

R04: Frequent Usage of GitHub

Your team must use GitHub as the only tool to store your deliverables, including **diagrams**, **reports**, **database**, and **code base**. You are required to use GitHub in a professional, iterative, and active approach by frequently pushing your primitive, partially completed, or finalized deliverables into the repositories, and use appropriate communication language in your repository.

Additional Note:

1) NOT Using GitHub frequently during the project **results in up to 10 points penalty** of your assessment grade**.**

2) In **your report, submit the proof of GitHub contribution** throughout the project duration

3) In **Canvas, submit the zip file of your GitHub repository.**

Missing either 2) or 3) results in two points penalty. Missing both 2) and 3) results in four points penalty.

Milestone 2 Requirements

Due Date: **11:00 AM, 13th January 2025**

1/ You are required to implement the features as stated from Level 1 to Level 3 using the following technologies:

a/ Frontend

- React-based framework: e.g., React, Vite, Next.JS
- Vanilla (Plain) HTML, CSS, and Javascript

b/ Backend

- **MEN** Stack: MongoDB, ExpressJS, NodeJS
- **Spring Boot**
- **Flask**
- **Kafka (for Level 3)**

2/ Every Team is required to include a database script to insert the following initial data into the system:

a/ One **ADMINISTRATOR** who manages the system

b/ Six **CHARITY** organization accounts:

- Two individuals, one from Vietnam and one from USA
- Two companies, one from South Africa and one from Germany
- Two non-profit organizations, one from Ukraine and one from Israel

c/ 30 **DONORS**, every country below has 5 **DONORS:** Vietnam, Germany, Qatar, USA, and Cameroon

d/ Three **Global** Crisis **CHARITY PROJECT:**

- Middle East Crisis – organized by the Israeli non-profit organization in b/
- Ukraine – Russia War - organized by the Ukrainian non-profit organization in b/
- Food Program in South Africa (Lesotho, Malawi, Namibia, Zambia and Zimbabwe, Mozambique and Angola) – organized by any company in b/

e/ Four **Local** **CHARITY PROJECT**:

- Yagi Typhoon Support – Vietnam - Organized by one individual in b/
- Milton Hurricane Support – USA - Organized by one individual in b/
- Helping Ukrainian Refugee – Germany – organized by any company in b/
- Supporting SOS Children's Village – China – organized by any company in b/

Level 1

Overview

The Charitable Donation Platform aims to connect **DONORS** with charitable projects. It will provide a user-friendly interface for **DONORS** to contribute to projects, while allowing **CHARITIES** to create and manage their initiatives. An **ADMIN** will oversee the platform, ensuring smooth operations and maintaining data integrity.

You are strongly recommended to read through requirements of ALL 3 Levels and the rubric to decide your project goals and plans.

Role Specification

- **DONORS**: Individuals wishing to contribute to charitable causes.
- **CHARITIES**: Organizations seeking funding for their projects.
- **ADMIN**: Individuals responsible for overseeing the platform and managing users and projects.

General Architecture Requirements

- **Apply N-Tier (Layer-based) Architecture** to divide your **front-end AND back-end** into different layers. Each layer has a designated concern and stores all classes addressing that concern across different business features.

- A typical Layer Hierarchy is Presentation, Business Logic, and Data Access. **However,** to reuse query operations in the back-end and separate query execution from data access object (DAO)**, you are required to implement Repositories,** where queries statements are defined and executed.

- Any Feature that does not follow the Tier structure shall be organized separately. E.g., Cookie Configuration does not use any of the Tier.

Team A Requirements

1A.1 DONOR Requirements

1A.1.1 Browse Charitable Projects

- **DONORS** can browse a list of available charitable projects with details such as project descriptions, goals, and funding status.

- **DONORS** can search a project by name

- **DONORS** can filter projects by a **CHARITY** name

1A.1.2 Donate using Credit Card

- DONORS can securely make donations through a third-party credit card payment system.

- **Note: You do not have to use your card data, it is sufficient to use the sandbox card of a 3rd party API, such as Stripe or Paypal.**

1A.1.3 Allow Donate as Guest

- DONORS can opt to donate without creating an account, providing a simple guest donation option.

1A.1.4 Record Transaction

- Each donation will be recorded in the system with details such as amount, DONOR information, and project ID.

1A.1.5 Confirm Donation via Email

- DONORS will receive a confirmation email with transaction details upon successful donation.

1A.1.6 Include DONOR Message

- DONORS can include a personal message with their donation, which will be sent to the CHARITY.

1A.1.7 See Statistics on Total Number of Donated Projects & Donation Value

- DONORS can view statistics regarding the number of projects they have donated to, and the total amount contributed.

1A.2 CHARITY Requirements

1A.2.1 Create Charitable Projects

- CHARITIES can create new projects, providing necessary information such as title, description, funding goals, and duration.

1A.2.2 Configure Card Information

- CHARITIES can set up and manage their credit card information for receiving donations from DONORS.

1A.2.3 Manage CHARITY Project

- CHARITIES can update project details, track donation progress, and view donations and the donated DONORS.
- CHARITIES can halt a project. A halted project will not be displayed in the list of active CHARITY projects
- A halted project can be resumed by its CHARITY
- A halted project can be deleted by its CHARITY

1A.2.4 Confirm Project Creation via Email

- CHARITIES will receive an email confirmation upon successful project creation.

1A.2.5 See Statistics on total number of Projects & Donation Value

- CHARITIES can view statistics regarding the total number of their projects and the overall donation value received.

API Provision Requirements

Team A must provide the following Service API to Team B:

1. **CRUD of CHARITY Project**, only ADMIN can Delete a CHARITY Project
2. **Email Utility**

API Usage Requirements

Team A must NOT develop the following API. Instead, team A must apply the following Service API from Team B:

1. **Authentication/Authorization of DONORS and CHARITIES**
2. **Create, Read, and Update DONORS and CHARITIES**
3. **Statistics**

Team B Requirements

2B.1 Register and Authentication/Authorization Requirements

2B.1.1 Register

1. The system shall allow DONORS to register with their email, password, first name, last name and address (optional).
2. The system shall allow CHARITIES to register with their email, password, company name, address, and tax code. The address is a mandatory field.

3. By default, the System has a Master **ADMIN** account

2B.1.2 Login with BasicAuth

4. All Users in the system login with their email and password
5. The system shall apply **BasicAuth** on email and password before sending to the backend.

2B.2. ADMIN Features

2B.2.1 Create an ADMIN Account

- The system will allow the creation of **ADMIN** accounts for managing the platform.

2B.2.2 CRUD all CHARITY Project with basic Project Manager Info

- **ADMINS** can Create, Read, Update, and Delete charitable projects and manage associated information.

2B.2.3 Searching and Filtering Projects and Users

- **ADMINS** can search and filter through projects and user accounts for better management.

2B.2.4 New CHARITY Project Approval

- **ADMINS** will review and approve **CHARITY** projects before they go live on the platform.

2B.2.5 See Statistics:

- Total Number of Projects & Total Donation Value
- The number of Projects and Donation Values of one **DONOR** or **CHARITY**

API Provision Requirements

Team B must provide the following Service API to Team A:

1. **CRUD of DONORS and CHARITIES**
2. **Authentication/Authorization of DONORS and CHARITIES**
3. **Statistics**

API Usage Requirements

Team B must NOT develop the following API. Instead, team B must apply the following API from Team A:

1. **CRUD of CHARITY Project**, only **ADMIN** can Delete a **CHARITY** Project
2. **Email Utility**

Level 2

General Architecture Requirements

- **Apply Modular Monolith Architecture** to your **back-end**. Each module has a bounded context to handle a specific (group of) business features. A module stores all classes of the N-Tier Layers or another design pattern to provide the business feature.

- Within a Module, the access rules shall go from one layer to another. Specifically, the Presentation Layer shall communicate with the Business Logic Layer which calls Repositories method to execute queries. The Presentation Layer must NOT be able to access the functions provided by the Repository Layer.

- Each Module contains two APIs:

  1. **One external API** to expose public services for other modules in the system to invoke
  2. **One internal API** to be used by the Presentation Layer of the module. Services in the internal API are not accessible to other modules

- Each module exposes its internal and external API via interfaces. **The Presentation Layer or other modules shall NOT call the Business Logic Layer of the module to invoke the APIs' services.**

- The backend must **present only necessary data** when responding to a request from the front-end. For example, when showing the top **DONORS** of the month, their credit card number shall not be exposed.

- You must also organize your DTOs into **internal** and **external DTOs**. **External DTOs** are accessible by other modules, while internal DTOs are only accessible by classes within the module.

- **Each module store data on its own database(s).** In other words, there is **more than one database** in the modular monolith.

- You should define common display elements (e.g., Buttons, Labels) as configurable components in a separate folder, so they can be reused by different pages without redefining the style again.

- **Componentize your Front-end**, such that each component has a package containing the following elements:

  1. Component's presentation in HTML or JSX
  2. Sub-components
  3. Back-end service calls
  4. Event handlers on the component
  5. CSS styling, if applicable

All the five elements are NOT placed in the same Component JavaScript file. These five elements shall be placed in separate files under the same package.

- The front-end must contain a **REST HTTP Helper Class** to provide a base function for conducting REST Requests such as GET, POST, PUT, PATCH, and DELETE, along with parameters of request headers and request body.

Team A Requirements

2A.1 DONOR Requirements

2A.1.2 View Charitable Projects by Category

- The system shall display projects by eight categories: Food, Health, Education, Environment, Religion, Humanitarian, Housing, or Other.
- The system shall let **DONORS** filter Project by Country
- The system shall allow **DONORS** to filter projects by categories on the main dashboard.
- The system shall show project details such as description, goals, images, and current funding status.
- **The system must lazily load the charitable projects** when the number of projects is at least two times more than a PC view can load on a web page. E.g., if the system shows 5 projects per page, with 30 projects, you must demonstrate that the first 10 projects are loaded on the page, while the next 10 projects are loaded lazily. When the last 10 projects are loaded, it is not possible to see more projects when triggering the lazy load.

2A.1.3 Include Donation Message

- When donating, **DONORS** can attach a personal message to be displayed with the donation.
- The system shall allow **DONORS** to add a message up to 250 characters with each donation.
- The system shall display the donation message to the **CHARITY** associated with the project.

2A.1.4 Subscribe to New Projects by Regions or Categories

- **DONORS** can subscribe to notifications for new projects based on specific regions or categories.
- The system shall allow **DONORS** to select regions or categories for project notifications.
- The system shall send notifications via email when new projects matching the **DONOR**'s subscriptions are published.

2A.1.5 Provide Monthly Donation Option

- The system shall offer **DONORS** an option to make their donations monthly.
- This option can be cancelled anytime
- The system shall automatically process monthly donations on the 15th date of each month.

2A.1.6 View DONORS of the Month

- To encourage and appreciate **DONORS**, in the **DONOR** and **CHARITY** Dashboard, the system shall provide a list of top 10 individual **DONORS** of the month, and a list of top 10 organization **DONORS** of the month.
- For the list of top individual **DONORS**, each row shows the **avatar**, **first name**, **last name**, and **donation amount** of the **DONOR**.
- For the list of top individual **CHARITY**, each row shows the logo, name, and donation amount of the organization.

2A.2 CHARITY Requirements

2A.2.1 Create Charitable Projects with Categories and Images

- The system shall allow **CHARITIES** to create a new project with **title**, **description**, **category**, **target amount**, and **images**.
- The system shall allow up to 15 images per project.

2A.2.2 Delete Projects by Moving to Delete Shard

- The system shall allow **CHARITIES** to delete projects by marking them as inactive or hidden.
- The system shall move deleted projects to a 'Delete Shard' for future recovery or audit purposes.

2A.2.3 Publish New Project to Subscribers

- The system shall send notifications to **DONORS** subscribed to matching regions or categories.

2A.2.4 Cache Projects with Redis

- The system shall apply Redis to cache the content of charity projects to optimize the project loading process to both **DONORS** and **CHARITIES**.

API Provision Requirements

Team A must provide the following Service APIs to Team B:

1. **Email Module**
2. **CRUD of CHARITY Project**, only **ADMIN** can Delete a **CHARITY** Project
3. **Encryption/Decryption using Asymmetric Key Pair**

API Usage Requirements

Team A must NOT develop the following APIs. Instead, team A must apply the following Service APIs from Team B:

1. **Authentication/Authorization of DONORS and CHARITIES**
2. **Create, Read, and Update DONORS and CHARITIES**
3. **Statistics**

Team B Requirements

2B.1 Register and Authentication/Authorization Requirements

2B.1.1 Register

- The system shall allow **DONORS** and **CHARITIES** to register with their email, password, and address, avatar, or introduction video as three optional fields.
- For **CHARITY**, the user can specify whether they are a business or non-profit organization
- The system shall verify the **DONOR**/**CHARITY**'s email before activating their account.

2B.1.2 Login with JSON Web Signature (JWS)

- The system shall use JWS to manage authenticated sessions.
- The system shall require email and password for login and generate a JWS on successful authentication.
- The JWS shall be used by the frontend. This JWS is NOT stored in a local storage, session storage, or Non-HTTP-Only Cookie.
- The JWS must be used to authorize Users

2B.2 ADMIN Requirements

2B.2.1 Create DONOR or CHARITY Account for Non-Technical Users

- The system shall allow **ADMINS** to create accounts for **DONORS** or **CHARITIES**.
- The system shall support adding images, videos, and user types (Person, Company, Non-profit organization).

2B.2.2 Send Email to Newly Created Users

- The system shall automatically send an email notification to users when their accounts are created.

2B.2.3 Add Highlighted CHARITY Project per Region or World

- **ADMINS** can feature specific projects by region or World, highlighting them on the platform.
- The system shall allow **ADMINS** to select and highlight at most 3 regional projects.
- The system shall allow **ADMINS** to select and highlight at most 3 global projects
- When **DONORS** opens the homepage, The system shall display the highlighted global and regional projects

2B.2.4 Halt a Project and Notify Stakeholders

- The system shall allow **ADMINS** to mark a project as halted.
- The system shall notify the project's **CHARITY** and Monthly **DONORS** with the halt reason via email.

2B.2.5 Encrypt Credit Card Data

- The system shall remember credit card data so the users do not have to type the card information again in subsequent donations
- The data needed for credit card payment must be encrypted in the system

2B.2.6 View Statistics

- **ADMINS** can view statistics, such as project count and donation value, by continent, country, or category.
- The statistics allowed the **ADMIN** to compare donation value and projects by categories and continents. E.g., to compare the donation value of Food and Education projects in Africa and Vietnam.

2B.2.7 Cache Donors and Charities with Redis

The system shall apply Redis to cache the necessary data of **DONORS** and **CHARITY** to optimize the user loading process to **ADMINS**.

API Provision Requirements

Team B must provide the following Service API to Team A:

1. **Authentication/Authorization of DONORS and CHARITIES**
2. **Create, Read, and Update DONORS and CHARITIES**
3. **Statistics**

API Usage Requirements

Team B must NOT develop the following API. Instead, team B must apply the following Service API from Team A:

1. **Email Module**
2. **CRUD of CHARITY Project**, only **ADMIN** can Delete a **CHARITY** Project
3. **Encryption/Decryption using Asymmetric Key Pair**

Level 3

General Architecture Requirements

- **Apply Microservice Architecture** to your **back-end**. Each service has a bounded context to handle a specific (group of) business capability. To achieve this, a microservice typically stores all classes of the N-Tier Layers for a single or group of related business capabilities, a module of a business capability from the modular monolith architecture or apply a different design pattern to provide business capabilities.

- Every microservice must be Dockerized and independently deployable.

- Communication among microservices shall be conducted via **a Message Broker**, e.g., Kafka.

- **Every microservice manages its own database(s)**. Each microservice contains a specific connection String to its database.

- Your team must apply **database sharding techniques** to partition and query different shards. This requirement aims to test your design in terms of scalability and efficiency.

- Your database must **execute** transactions instead of queries, hence the database can rollback the transaction when an unexpected error occurs.

Team A Requirements

3A.1 DONOR Requirements

3A.1.1 View Charitable Projects by Category and by Their Residing Country

- **DONORS** should be able to view projects by category, with the system defaulting to show projects within the **DONOR**'s country
- To simulate the residence of the **DONOR**, the frontend shall use the **Accept-Language Header** to identify the **DONOR**'s language and country.
- The system shall automatically display projects located in the **DONOR**'s country on top, based on the identified country code in the **Accept-Language Header** (e.g., **en-US** indicates the **DONOR** is in the US, and **vi-VN** is in Vietnam).
- The system shall provide category filters for projects, allowing **DONORS** to narrow down results by interests such as Food, Education, Health, etc.

3A.1.2 Record Donation Transaction

- The platform should record every donation transaction made by **DONORS**.
- The system shall allow **DONORS** to make a donation to any active project.
- The system shall record each donation transaction with details, including the **DONOR**'s ID, Project ID, donation amount, date, and any message the **DONOR** includes.
- The system shall provide a summary of donation history for each **DONOR** on their dashboard.

3A.1.3 Real-time Subscribe to New Projects by Regions or Categories (Kafka)

- **DONORS** can subscribe to receive real-time updates on new projects published in specific regions or categories.
- The system must use **Kafka** to deliver real-time notifications to **DONORS** whenever a new project is published that matches their subscription requests.
- The system shall send notifications to subscribed **DONORS** within 7 seconds when a new project matches their subscription criteria is published.

3A.2 CHARITY Requirements

3A.2.1 Create Charitable Projects with Processed Images, and Videos

- The system shall allow **CHARITIES** to create new projects with fields for title, description, category, target amount, and multimedia (images and videos).
- The system shall support 3-15 images and less than or equal to 4 videos per project.
- The images shall be stored in original and thumbnail sizes. The thumbnail image is shown when displaying the list of projects.
- The system shall allow **CHARITIES** to categorize projects under the pre-defined 7 categories.

3A.2.2 Move Completed Projects to a Different Database or Shard

- The system shall detect when a project's target amount has been reached or when the **CHARITY** marks it as completed.
- When the target amount is reached, the system notifies and sends an email to the **CHARITY**
- The system shall move completed projects to a designated database or shard for archiving.
- The system shall retain all project information, including donation history and project updates, in the archive.

3A.2.3 Publish New Projects to Subscribers via Kafka

- The system shall use **Kafka** to publish notifications for new projects to **DONORS** who have subscribed to the region or category associated with the new project.
- The system shall process project publication and notification in real-time to ensure **DONORS** are promptly informed.

3A.3 Redis Redundancy

- The system must provide **two Redis instances** to serve the caching requirement **2A.2.4.**

3A.4 Image Compression (BONUS POINT)

- To save the bandwidth consumption and data storage, images uploaded by a **DONOR**, **CHARITY**, or **ADMIN** must be compressed **on the front-end side.**
- The compressed images are transferred to the back-end with a readable quality at 100% zoom on the browser, when the images are transferred back to the front-end.

API Provision Requirements

Team A must provide the following Service APIs to Team B:

1. **Email Module**
2. **CRUD of CHARITY Project**, only **ADMIN** can Delete a **CHARITY** Project
3. **Encryption/Decryption using Asymmetric Key Pair**

API Usage Requirements

Team A must NOT develop the following APIs. Instead, team A must apply the following Service APIs from Team B:

1. **Authentication/Authorization of DONORS and CHARITIES**
2. **Create, Read, and Update DONORS and CHARITIES**
3. **Statistics**

Team B Requirements

3B. ADMIN Requirements

3B.1 Register and Authentication/Authorization Requirements

3B.1.1 Register with JSON Web Encryption (JWE) and Thumbnail Images

- The system shall use JWE to store email, password of a **DONOR** or **CHARITY**. The address, avatar, or introduction video are still optional fields.
- The avatar shall be stored in a square dimension less than 300x300 pixels, and a square thumbnail of size less than 100x100 pixels. The thumbnail image is shown when displaying the list of Users.

3B.1.2 Login with encrypted credentials

- The front-end shall encrypt email and password for login OR encapsulate the email and password inside and generate a JSON Web Encryption (JWE), and then send the encrypted credentials to the backend.
- Once the user credential matches the authentication data, the backend generates a JWE containing authorization data of the user. This JWE shall be used by the frontend to authorize Users. This JWE is NOT stored in a local storage, session storage, or Non-HTTP-Only Cookie.

3B.2 Project Management

3B.2.1 Halt a Project and Notify Monthly DONORS via Kafka

- The system shall allow **ADMINS** to mark a project as “halted” and specify a reason.
- The system shall send an email to the **CHARITY**, stating the project name and halt reason.
- The system shall send notifications to all **Monthly DONORS** subscribed to the project **via Kafka**.
- The notification message should include the project name, halt status, and the reason for the halt. **The halt reason to the DONORS can be different than the reason sent to the CHARITY**
- Notifications shall be sent in real time to ensure timely updates for Monthly **DONORS**.

3B.3 Statistics and Reporting

3B.3.1 View Statistics by Year, Month, and Week

- The system shall provide a dashboard for **ADMINS** to view statistics such as:

  - Total number of active and completed projects
  - Donation values (total and by category)
  - The number of new **DONOR** registrations
- The system shall allow **ADMINS** to filter these statistics by year, month, and week.
- The system shall display statistics in various formats (e.g., charts, tables) for clarity.

3B.3.2 Provide Donation Statements for DONORS (BONUS POINT)

- The system shall allow **DONORS** to see and download their donation statements from a PDF file.
- The donation statement has a start date, end date, donation amount to which **CHARITY** project, the name and ID of the **CHARITY** that manages the project, and the total donation amount.

3A.3 Redis Redundancy

- The system must provide **two Redis instances** to serve the caching requirement **2B.2.7.**

API Provision Requirements

Team B must provide the following Service APIs to Team A:

1. **Authentication/Authorization of DONORS and CHARITIES**
2. **Create, Read, and Update DONORS and CHARITIES**
3. **Statistics**

API Usage Requirements

Team B must NOT develop the following APIs. Instead, team B must apply the following Service APIs from Team A:

1. **Email Module**
2. **CRUD of CHARITY Project**, only **ADMIN** can Delete a **CHARITY** Project
3. **Encryption/Decryption using Asymmetric Key Pair**

Contribution Requirements

Every team member is expected to contribute into **ALL** of the following tasks by producing recognizable deliverables or providing substantial feedback:

1. Design Data Model
2. Design System Architecture
3. UX/UI Design of **ALL** Requirements
4. Documentation

5. Database Construction

6. Specialized Front-end Development

   OR Specialized Back-end Development

   OR Full Stack Development (both Back-end and Front-end)

For development tasks, each member is expected to **contribute either in individual or pair-programming**. The amount of fair contribution is determined by the complexity of the supported features and average workload of the entire team.

If there has been a problem with group members not completing their allocated tasks, inform your Course Coordinator as soon as possible so staff can intervene before the problem escalates.

To verify unequal contribution towards group-based assignments, all group members have to formally acknowledge uneven contribution by submitting a document **detailing individual team member contribution** via a weight distribution metric for all group members. This document must be submitted with the assessment task and needs to be personally signed by all group members, with or without the unequal contributor, to be considered. **Final assessment and weight distribution of grades for group assignments will be made at the discretion of the course coordinator.**

Penalty

P1) NOT Using GitHub frequently during the project **results in up to 10 points penalty** of your assessment grade**.**

P2) In your report**, submit the proof of GitHub contribution** throughout the project duration

P3) In **Canvas,** submit the **zip file of your project's GitHub repository.**

P4) Did NOT submit the Project Charter **before Milestone 1,** which will result in **2 points penalty**

End of Requirement Specification