**SENG 244-Object Oriented Software Engineering**

**NGO Aid Operations Management System:**

**NGO-AOMSYS**

**Group Members:**

Doğa Ömrüuzun: 210201027

İrem Sena Alpak: 210201030

Melis Gedik: 220201027

Ayşe Asude Doğan: 220201044

Ayşe Aybala Girginol: 220204014

**SOFTWARE DESIGN DOCUMENT**

**Table Of Contents**

* **1.Introduction**

**1.1 Purpose of the system**

**1.2 Design goals**

**1.3 Definitions,acronyms and abbrevations**

**1.4 References**

**1.5 Overview**

* **2. Current software architecture**
* **3 Proposed software architecture**

**3.1 Overview**

**3.2 Subsystem decomposition**

**3.3 Hardware/software mapping**

**3.4 Persistent data management(Updated)**

**3.5 Access control and security**

**3.6 Boundary conditions**

**3.7 Subsystem services glossary**

* **4. Object design**

**4.1 Object design trade-offs**

**4.2 Interface documentation guidelines**

**4.3 Packages**

**4.4 Class interface glossary**

**1.1 Introduction**

Non-Governmental Organizations (NGOs) are important in managing aid and delivering aid to people in need.

NGO Aid Operations Management System (NGO-AOMSYS) was developed to meet these needs of NGOs. This project will help in better management of aid.

**1.1 Purpose of the system**

The purpose of this system is to manage and organize the aid provided in the most effective way and to help deliver it to people in need.

**1.2 Design Goals**

* Effectively managing the delivery of aid to those in need.
* Allowing donors to register, log in, and choose the type and amount of donation.
* Allowing volunteers to register,login and enter the information about availability details.
* Allowing individuals to register,log in and enter their informations.
* Allowing operation coordinators to schedule operations and manage the system.
* Analyzing the current state of donations ,requests and availability of the volunteers by using algorithms.

**1.3 Definitions, acronyms and abbrevations**

NGO: Non-Governmental Organization

AOMSYS: Aid Operations Management System

Donors: The people who provide aid

Volunteers: Aid volunteers

Indigents:The people who need aid

Administirator: The people who can manage the system

**1.4 References**

https://ase.in.tum.de/lehrstuhl\_1/component/content/article/266

**1.5 Overview**

This system is designed for managing the donations and providing the distributions of the donations to the indigents. The system provides communication between donors ,volunteers and coordinators.

**2. Current Software Architecture**

Turkish Red Crescent (Türk Kızılayı) : Turkish Red Crescent is one of the largest and most widespread humanitarian organizations in Turkey. It provides a wide range of humanitarian aid and social services including emergency relief, blood donation, shelter, food aid, education, and healthcare.

TEMA Foundation : TEMA Foundation is a prominent non-governmental organization in Turkey dedicated to raising environmental awareness and conserving natural resources. TEMA works on erosion control, afforestation, water resource conservation, and environmental education among other areas.

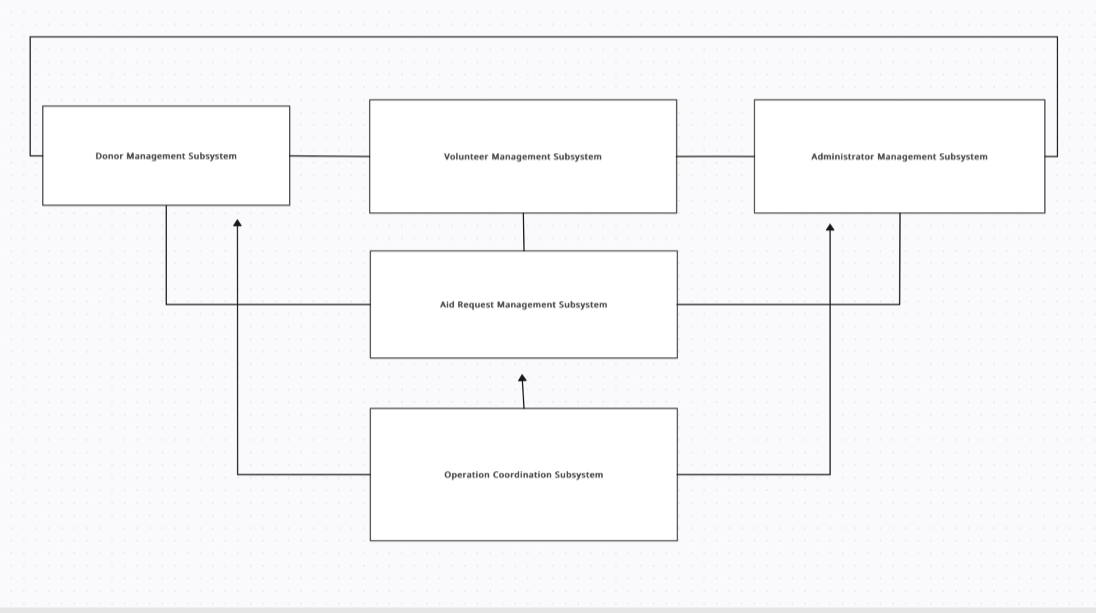
**3. Proposed Software Architecture**

**3.1 Overview**

The proposed software architecture for NGO-AOMSYS (Non-Governmental Organization Aid Operations Management System) entails a web-based application designed to streamline aid allocation and distribution. It features modules for user authentication and registration, donation management, volunteer coordination, aid request processing, operation coordination, reporting, and analytics.

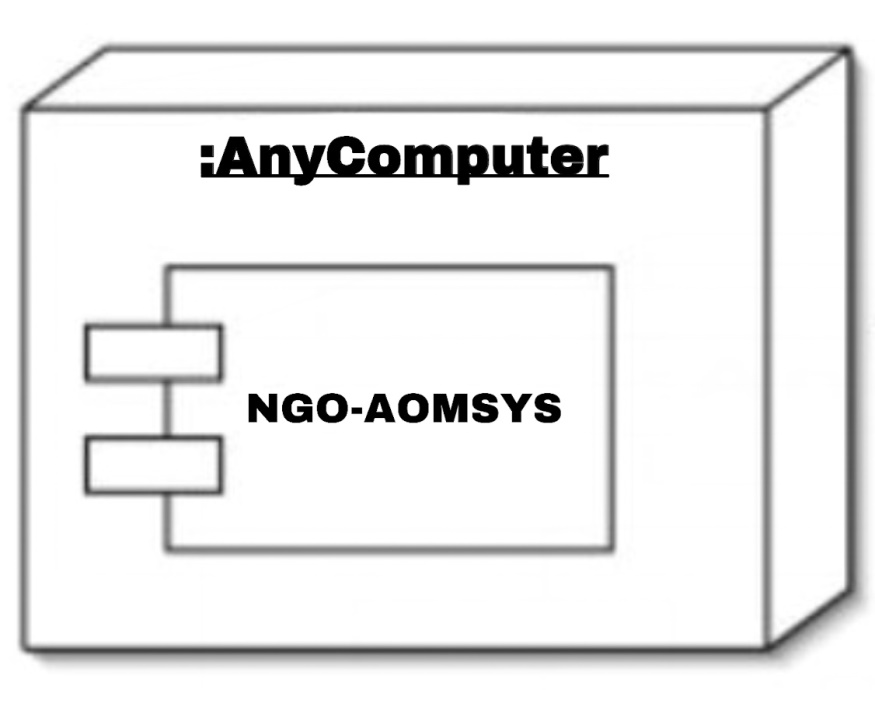
**3.2 Subsystem Decomposition**

The following UML class diagram gives an overview of the identified subsystems and their relations.



**3.3 Hardware/Software Mapping**

The following UML deployment diagram illustrates the hardware/software mapping for the system.



**3.4 Persistent Data Management (Updated)**

**Data Schemes**

The NGO-AOMSYS system stores several types of persistent data, including donor information, volunteer information, indigent aid applications, operational data, and system logs. Each type of data is stored in a separate SQL database table, with relationships established between tables where necessary.

* Donor Information: Stored in the ‘donor’ table, including fields such as ‘donorID’, ‘name’, ‘email’, ‘donationHistory’.
* Volunteer Information: Stored in the ‘volunteers’ table, including fields such as ‘volunteerID’, ‘name’, ‘profession’, ‘email’, ‘region’, and ‘availabilitySchedule’.
* Indigent Aid Applications: Stored in the ‘indigent\_aid\_applications’ table, including fields such as ‘applicationID’, ‘name’, ‘monthlyIncome’, ‘personID’.
* Operational Data: Stored in the ‘operations’ table, including fields such as ‘operationID’, ‘name’, ‘date’.

**Selection of Database**

The NGO-AOMSYS system uses SQL as its database management system to store its data. The system is designed to be database-agnostic, allowing for easy migration to a different SQL database if needed. Currently, the system is configured to use MySQL as its SQL database due to its reliability, performance, and compatibility with the system's requirements.

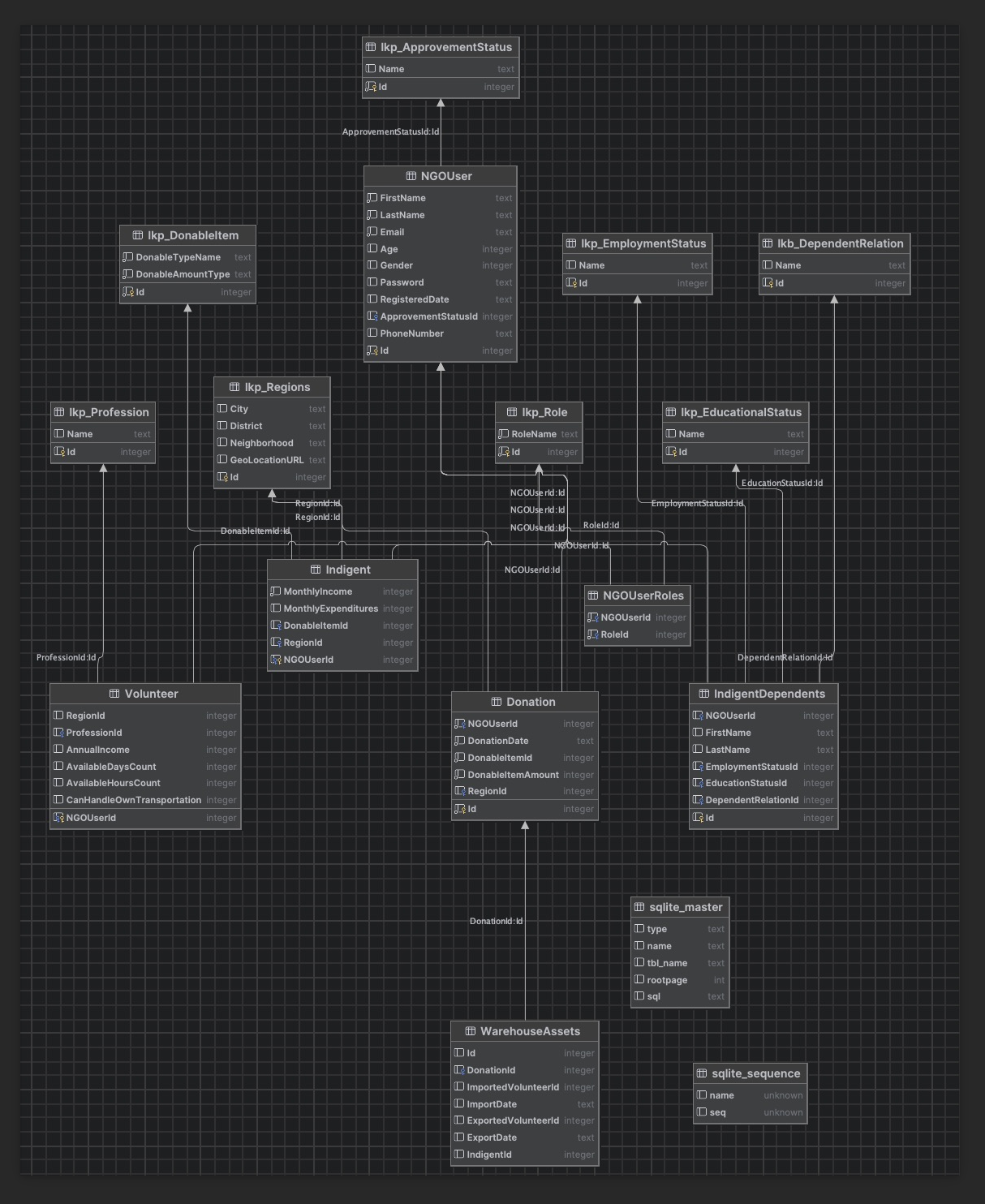
**Encapsulation of the Database**

To encapsulate the database functionality, the system uses a data access layer (DAL) that abstracts the SQL database operations from the rest of the system. This DAL provides an interface for interacting with the SQL database, including methods for querying, inserting, updating, and deleting data. The DAL also handles SQL database connection management, ensuring that connections are properly opened and closed to prevent resource leaks.

**Data Management Infrastructure**

The data management infrastructure includes backup and recovery mechanisms to ensure data integrity and availability. Regular SQL database backups are performed, and data recovery procedures are in place to restore data in the event of a system failure.

By following these practices, the NGO-AOMSYS system ensures that its persistent data is stored securely, efficiently, and in a way that is easy to manage and maintain.



**3.5 Access Control and Security**

Access control and security measures are crucial to safeguard sensitive information and ensure the integrity, confidentiality, and availability of the system. The NGO-AOMSYS project incorporates the following access control and security features:

1. **Authentication and Authorization:**
   * Users (donors, volunteers, administrators) will be required to authenticate themselves before accessing the system.
   * Different levels of access will be assigned based on user roles. For example, administrators will have access to all functionalities, while donors and volunteers will have limited access based on their permissions.
2. **Encryption:**
   * Data transmission between the client and the server will be encrypted using HTTPS protocol to prevent eavesdropping and ensure data confidentiality.
   * Sensitive data such as user credentials and personal information will be encrypted before storing in the database to prevent unauthorized access.
3. **Role-Based Access Control (RBAC):**
   * RBAC will be implemented to control access to system resources based on the roles of individual users.
   * Access permissions will be assigned to predefined roles (e.g., donor, volunteer, administrator), and users will be granted access based on their assigned roles.
4. **Data Validation and Sanitization:**
   * Input data will be validated and sanitized to prevent common security vulnerabilities such as SQL injection and cross-site scripting (XSS) attacks.
   * Client-side and server-side validation mechanisms will be implemented to ensure the integrity of the data.
5. **Audit Logging:**
   * All user interactions and system activities will be logged to maintain an audit trail.
   * Audit logs will include details such as user actions, timestamp, IP address, and outcome of the action for accountability and forensic analysis purposes.
6. **Session Management:**
   * Secure session management techniques, such as session tokens and session timeouts, will be employed to prevent session hijacking and unauthorized access.
   * Sessions will be securely maintained on the server side, and session identifiers will be regenerated upon authentication to mitigate session fixation attacks.
7. **Firewall and Intrusion Detection Systems (IDS):**
   * Firewall and IDS mechanisms will be implemented to monitor and filter incoming and outgoing network traffic.
   * Anomaly detection algorithms will be utilized to identify suspicious activities and potential security breaches.
8. **Regular Security Audits and Updates:**
   * Regular security audits will be conducted to identify and remediate potential vulnerabilities in the system.
   * Software and security patches will be applied promptly to mitigate known security risks and vulnerabilities.

By incorporating these access control and security measures, the NGO-AOMSYS project aims to ensure the confidentiality, integrity, and availability of sensitive information and protect the system from various security threats and attacks.

**3.6 Boundary Conditions**

The starting, stopping and installing of the NGO- AOMSYS website defines the boundary conditions. Here we must take care of the different operating systems NGO-AOMSYS is build for.

• Installing : For all operating systems, NGO-AOMSYS need no explicit installation execution. Copying the executable NGO-AOMSYS website into the local file system does the installation.

• Starting : The starting of NGO-AOMSYS depends on the operating system. If Windows or Mac OS X is used, the player starts NGO-AOMSYS by an double click on a ? NGO-AOMSYS bat? or a Mac OS X executable website file. For other Unix operating systems, we provide a Unix script for starting NGO-AOMSYS.

• Stopping : Pressing the ?Exit? menu item or closing the NGO-AOMSYS window stops the website.

**3.7 Subsystem Services Glossary**

At the moment, we provide no services in terms of operations for the subsystems.

**4. Object Design**

**4.1 Object Design Trade-Offs**

**Description**

The design of NGO-AOMSYS involved several key trade-offs to balance various considerations. One of the primary trade-offs was between system complexity and ease of use. While a more complex system could potentially offer more features and flexibility, it could also make the system harder to use for non-technical users. Therefore, we opted for a simpler design that focuses on usability and accessibility.

Another trade-off was between performance and scalability. We chose a design that prioritizes performance to ensure that the system can handle a large number of transactions efficiently. However, this design decision may limit the scalability of the system in the long run.

**Guidelines and Conventions**

* Naming Conventions: Classes are named with singular nouns, methods with verb phrases, and fields/parameters with noun phrases.
* Exception Handling: Error status is returned via an exception, not a return value.

**4.2 Interface Documentation Guidelines**

**Guidelines**

* Classes: Named with singular nouns.
* Methods: Named with verb phrases.
* Fields and Parameters: Named with noun phrases.
* Error Handling: Error status is returned via an exception, not a return value.
* Collections: Have an ‘elements()’ method returning an Enumeration.
* Enumerations: Returned by ‘elements()’ methods are robust to element removals.

**4.3 Packages**

**Decomposition**

The subsystems of NGO-AOMSYS are decomposed into the following packages:

* ‘DonorManagement’: Manages donor registration and donation tracking.
* ‘VolunteerManagement’: Manages volunteer registration and availability.
* ‘IndigentAidManagement’: Manages aid application and distribution to indigents.

**File Organization**

Each package is organized into a separate directory within the codebase, following a standard naming convention (e.g., ‘src/donor\_management’, ‘src/volunteer\_management’, ‘src/indigent\_aid\_management’).

**Dependencies**

* ‘DonorManagement’ depends on the ‘IndigentAidManagement’ package for tracking donation distributions.
* ‘VolunteerManagement’ depends on the 'IndigentAidManagement' package for coordinating volunteer efforts.

**4.4 Class Interface Glossary**

**Classes Overview**

* ‘Donor’: Manages donor information and donations.
* ‘Volunteer’: Manages volunteer information and availability.
* ‘Indigent’: Represents individuals in need of aid.
* ‘Administrator’: Manages the NGO-AOMSYS system.

**Dependencies**

* ‘Donor’ and ‘Volunteer’ classes depend on the ‘Indigent’ class for aid distribution information.
* ‘Administrator’ class depends on all other classes for system management.

**Public Attributes and Operations**

1. ‘Donor’ Class:

* Attributes: ‘donorID’, ‘name’, ‘email’.
* Operations: ‘register()’, ‘login()’, ‘selectDonation()’, ‘viewDonationHistory()’.

1. ‘Volunteer’ Class:

* Attributes: ‘volunteerID’, ‘name’, ‘profession’, ‘annualIncome’, ‘region’, ‘transportationAvailable’, ‘availability’.
* Operations: ‘login()’, ‘register()’, ‘fillProfileForm()’, ‘viewPendingStatus()’, ‘approveVolunteer()’.

1. ‘Indigent’ Class:

* Attributes: ‘personID’, ‘name’, ‘monthlyIncome’, ‘householdSize’, ‘children’, ‘employementStatus’, ‘monthlyExpenses’, ‘requestedSupport’.
* Operations: ‘login()’, ‘applyForAid()’, ‘fillRegistrationForm()’, ‘viewAidStatus()’.

1. ‘Administrator’ Class:

* Attributes: ‘adminID’, ‘name’.
* Operations: ‘login()’, ‘createNewUser()’, ‘queryUser()’, ‘editUser()’, ‘deleteUser()’, ‘viewDashboard()’.

**Exceptions**

* ‘DonorException’: Raised when a donor-related operation fails.
* ‘VolunteerException’: Raised when a volunteer-related operation fails.
* ‘IndigentException’: Raised when an indigent-related operation fails.
* ‘AdministratorException’: Raised when an administrator-related operation fails.

**Usage Examples**

1. **Donor class:**

* Example: A donor named "John Doe" registers on the NGO-AOMSYS platform using his email address "john.doe@example.com" and makes a donation of $100 to support a relief operation.

1. **Volunteer class:**

* Example: A volunteer named "Alice Smith" registers on the NGO-AOMSYS platform using her email address "alice.smith@example.com" and updates her availability to help with a disaster response operation.

1. **Indigent class:**

* Example: An indigent individual named "David Brown" registers on the NGO-AOMSYS platform using his email address "david.brown@example.com" and applies for aid to receive food assistance.

1. Administrator class:

* Example: An administrator named "Admin User" logs in to the NGO-AOMSYS system and manages user registrations, volunteer schedules, and aid distribution operations.

**Design Patterns**

1. Singleton Pattern:

* Used in the Administrator class to ensure that only one instance of the class exists throughout the system, allowing for centralized management of the system.

1. Factory Method Pattern:

* Used in the Donor class to create instances of donation objects, encapsulating the creation logic and providing a consistent way to create donations.

1. Observer Pattern:

* Used in the Volunteer class to notify the system when a volunteer's availability changes, allowing the system to update volunteer schedules accordingly.

1. Strategy Pattern:

* Used in the Indigent class to define different strategies for aid distribution based on the type of aid needed (e.g., food, shelter, medical).