Waste Food Management for Hostel Mess

Software Architecture Document

Prepared by :

20BCS038 Ajay Thakur

20BCS052 Davish Mehata

20BCS056 Shubham Thakur

20BCS057 Sahil Lagwal

Table of Contents

Software Architecture Document

Introduction

The purpose of this document is to outline the design and development of a Hostel Mess Waste Food Management Website. This website will serve as a tool for hostels to track and manage their food waste, with the goal of reducing waste and promoting sustainability.

1.1 Purpose

The purpose of the Waste Food Management website is to provide an efficient and sustainable solution for managing food waste and donations. The website will enable individuals and businesses to donate their excess food to organizations that can distribute it to those in need. The website will also provide tools for managing food waste, including tracking, reporting, and analytics.

1.2 Scope

The Waste Food Management website will be a web-based application that allows users to manage their food waste and donations. The website will consist of multiple microservices, which will be deployed in a cloud environment. The website will be designed using a microservices architecture, which will provide scalability, reliability, and easy maintenance. The website will be designed with security in mind, ensuring that user data is protected.

1.3 Definitions, Acronyms, and Abbreviations

- Waste Food Management website: The web-based application that allows users to manage their food waste and donations.
- Microservices: A software development technique that structures an application as a collection of loosely coupled services.
- Cloud Environment: A virtualized environment that delivers computing services over the internet.
- Kubernetes: An open-source container orchestration platform that automates the deployment, scaling, and management of containerized applications.
- CI/CD pipeline: A set of processes and tools that enable developers to build, test, and deploy code changes to production.
- IaC: Infrastructure as Code is the practice of managing infrastructure in a file-based declarative format.
- Terraform: A tool for building, changing, and versioning infrastructure safely and efficiently.

1.4 References

The following references were used in the development of this document:

- "Microservices: Flexible Software Architecture" by Eberhard Wolff
- "Kubernetes: Up and Running" by Kelsey Hightower, Brendan Burns, and Joe Beda
- "Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation" by Jez Humble and David Farley
- "Infrastructure as Code: Managing Servers in the Cloud" by Kief Morris

2. Architectural Representation

The Waste Food Management website is designed using a microservices architecture, which provides scalability, reliability, and easy maintenance. The website is deployed in a cloud environment using Kubernetes, which enables automated deployment, scaling, and management of the microservices. The microservices are developed using different programming languages and frameworks, and communicate with each other using APIs.

The website consists of the following microservices:

- **User Management:** This microservice is responsible for managing user accounts, including registration, login, and authentication.

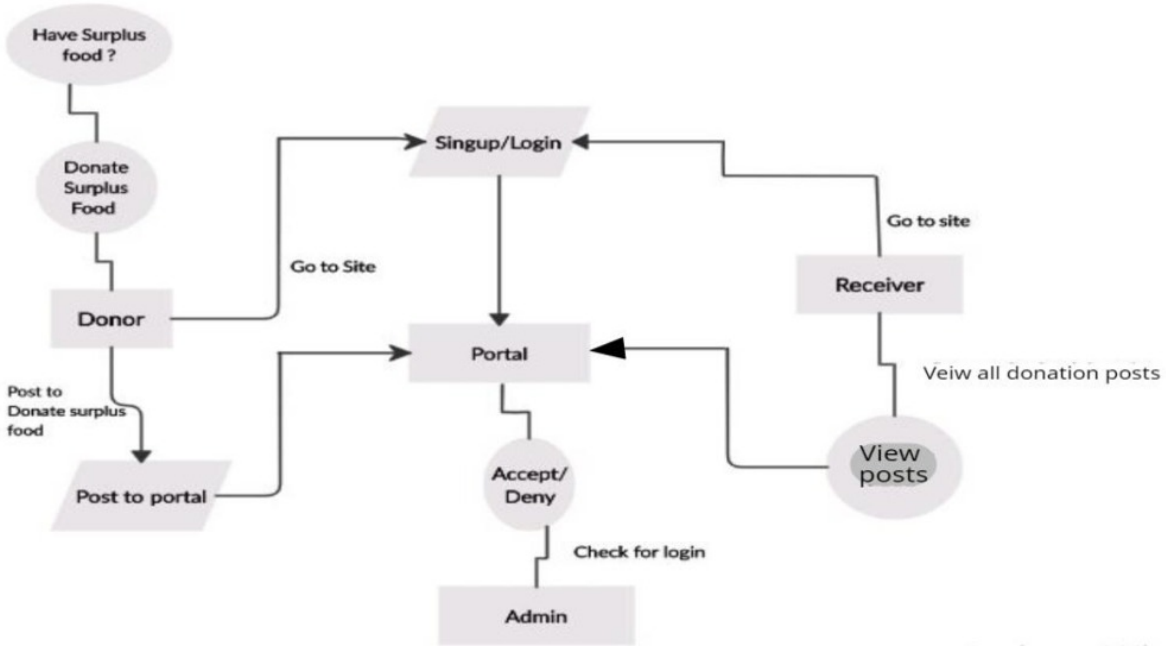
- **Donation Management:** This microservice is responsible for managing food donations, including creating, modifying, and deleting donations.

- **Organization Management:** This microservice is responsible for managing organization profiles, including creating and editing profiles.

- **Reporting and Analytics:** This microservice is responsible for generating reports and analytics on donation activities.

- **Payment Processing:** This microservice is responsible for processing payments for donations.

Project Workflow Diagram



3. Architectural Goals and Constraints

The architectural goals of the Waste Food Management website are as follows:

- **Scalability:** The website should be able to handle a large number of users and donations.
- **Reliability**: The website should be highly available and resilient to failures.
- **Security:** The website should be designed with security in mind, ensuring that user data is protected.
- **Performance:** The website should be optimized for performance, ensuring fast response times and minimal latency.
- **Maintainability**: The website should be easy to maintain and update, enabling rapid development and deployment of new features.

The following constraints apply to the architecture of the website:

- **Technology stack:** The website should be developed using a specific set of technologies and frameworks.
- **Compliance:** The website should comply with relevant regulatory requirements and standards, such as data privacy regulations.

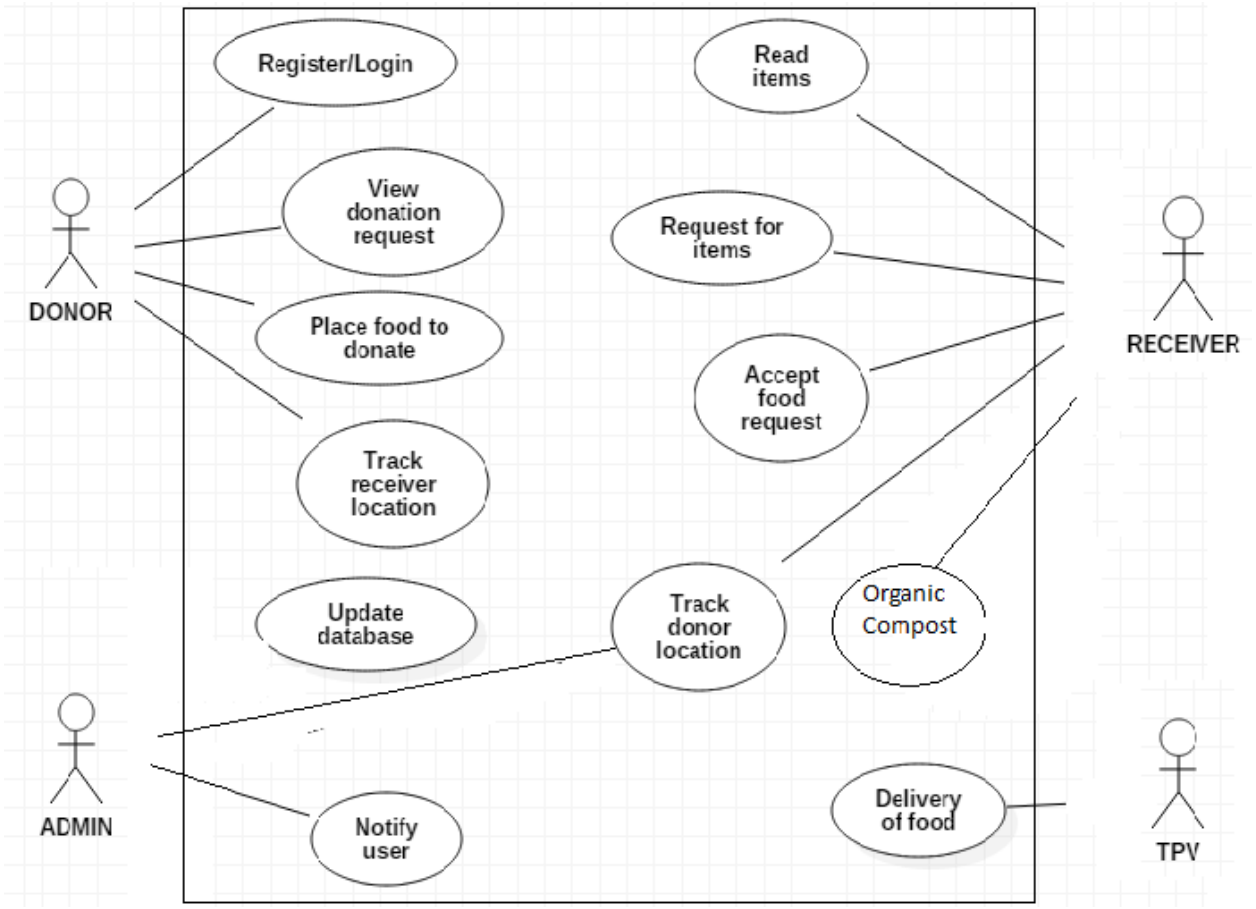
- **Integration:** The website should be able to integrate with other systems and services, such as payment gateways and notification systems.
- **Time Constraints:** The architecture of the website should be designed based on the timeline and deadlines set for the project, ensuring timely completion of the website.

The architectural goals and constraints provide a framework for designing and developing the Waste Food Management website. By keeping these goals and constraints in mind, the architecture of the website can be optimized for performance, scalability, reliability, and security while meeting the specific needs of the project.

All performance and loading requirements, as stipulated in the Vision Document and the Supplementary Specification , must be taken into consideration as the architecture is being developed.

4. Use-Case View

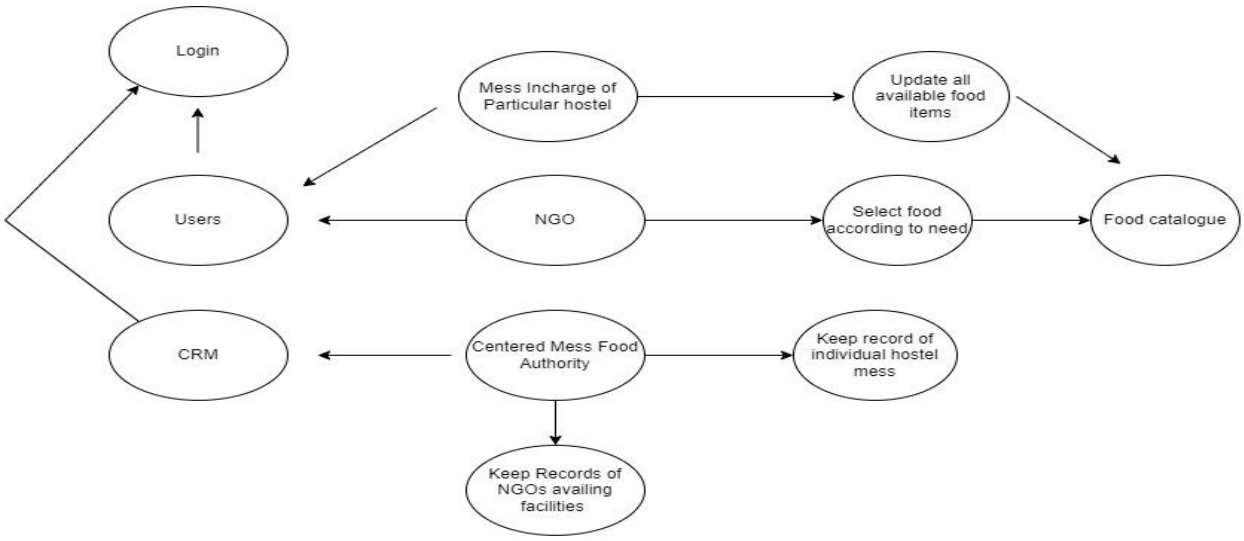
The use-case view describes the interactions between the users of the Waste Food Management website and the system. This view provides an overview of the key functionalities of the system and their interactions with the users.



4.1 Architecturally-Significant Use Cases

The following use cases are architecturally significant and represent the key functionalities of the system:

- **User Registration and Login:** This use case allows users to register and create an account on the website. Users can then log in to their account to access the website's features.
- **Donation Management:** This use case allows users to manage their food donations. Users can create, modify, and delete donations, as well as view their donation history.
- **Organization Management:** This use case allows organizations to manage their profile on the website. Organizations can create a profile, edit their profile information, and view their donation history.
- **Reporting and Analytics:** This use case allows users to generate reports and analytics on their donation activities. Users can view statistics such as the amount of food donated, the number of donations, and the types of food donated.
- **Search and Filter:** This use case allows users to search and filter donations and organizations based on specific criteria, such as location, type of food, and donation history.
- **Notification Management:** This use case allows users to manage their notification settings. Users can choose to receive notifications about new donations, updates to their donations, and other relevant information.
- **Payment Processing:** This use case allows users to make payments for their donations. Users can choose to make a one-time donation or set up recurring donations. The website will process payments securely using a third-party payment gateway.



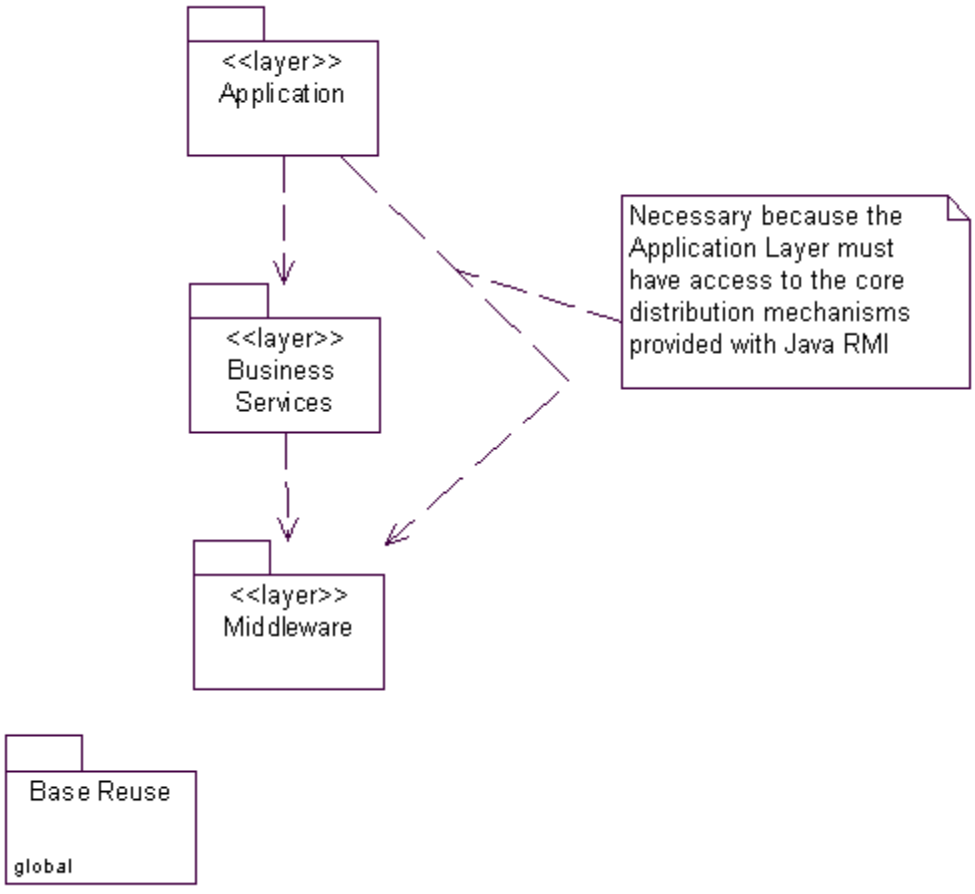
- **User Feedback:** This use case allows users to provide feedback on the website and its features. Users can submit feedback on their experience using the website, report issues, and suggest improvements.

These architecturally significant use cases represent the core functionalities of the Waste Food Management website and are critical to the overall success of the system.

5. Logical View

In the logical view of the Waste Food Management Website, the system is organized into packages and subsystems, which are layered to provide a clear separation of concerns and promote modularity.

5.1 Architecture Overview – Package and Subsystem Layering



5.1.1 Application layer
This layer contains the order management, payment gateway, and feedback management modules, which are responsible for the core business logic of the system. They interact with the other layers to perform their respective functions.

5.1.2 Business Services layer
The Business Services process layer has all the controller classes that represent the use case managers that drive the application behavior. This layer represents the client-to-mid-tier border. The Business Services layer depends upon the Process Objects layer; that straddles the separation of the client from mid-tier.

5.1.3 Middleware layer
The Middleware layer includes message queuing, service bus, authentication & Authorization components.

5.1.4 Base Reuse The Base Reuse package includes classes to support list functions and patterns.

6. Process View

A description of the process view of the architecture. Describes the tasks (processes and threads) involved in the system's execution, their interactions and configurations. Also describes the allocation of objects and classes to tasks.

6.1 Processes

6.1.1 NGO Registration Process:
This process involves registering NGOs that are interested in providing food to the hostel mess. The NGOs can register through a dedicated registration form or by contacting the hostel administration. The registration process is managed by the registration controller.

6.1.2 NGO Registration Form:
The NGO registration form is a web-based form that NGOs can use to provide their details, such as their name, contact information, and the type of food they can provide. The form is accessible through the website's presentation layer and is processed by the registration controller.

6.1.3 Mess Registration Process:
The mess registration process involves registering hostels that require food supplies. The hostel administration can register the hostel through a dedicated registration form or by contacting the food management system administrator. The registration process is managed by the registration controller.

6.1.4 Registration Controller:
The registration controller is responsible for managing the NGO and mess registration processes. It receives the registration information from the registration forms and manages the registration details in the database layer.

6.1.5 Close Registration Process:
The close registration process involves closing the registration for NGOs or hostels when the required number of registrations has been received or the registration period has ended. The process is triggered by the registration controller.

6.1.6 Close Registration Controller:
The close registration controller is responsible for managing the close registration process. It receives the trigger to close the registration and manages the status of the registration process in the database layer.

6.1.7 Mess Catalog System Access:
The mess catalog system access process allows the registered NGOs to access the food catalog system. The access is granted by the food management system administrator, and the access details are stored in the database layer.

6.1.8 Food Catalog System:
The food catalog system is a web-based system that allows the NGOs to view the available food items and place orders. The system is managed by the application layer, which communicates with the database layer to retrieve the food catalog data and process the orders.

6.2 Process to Design Elements

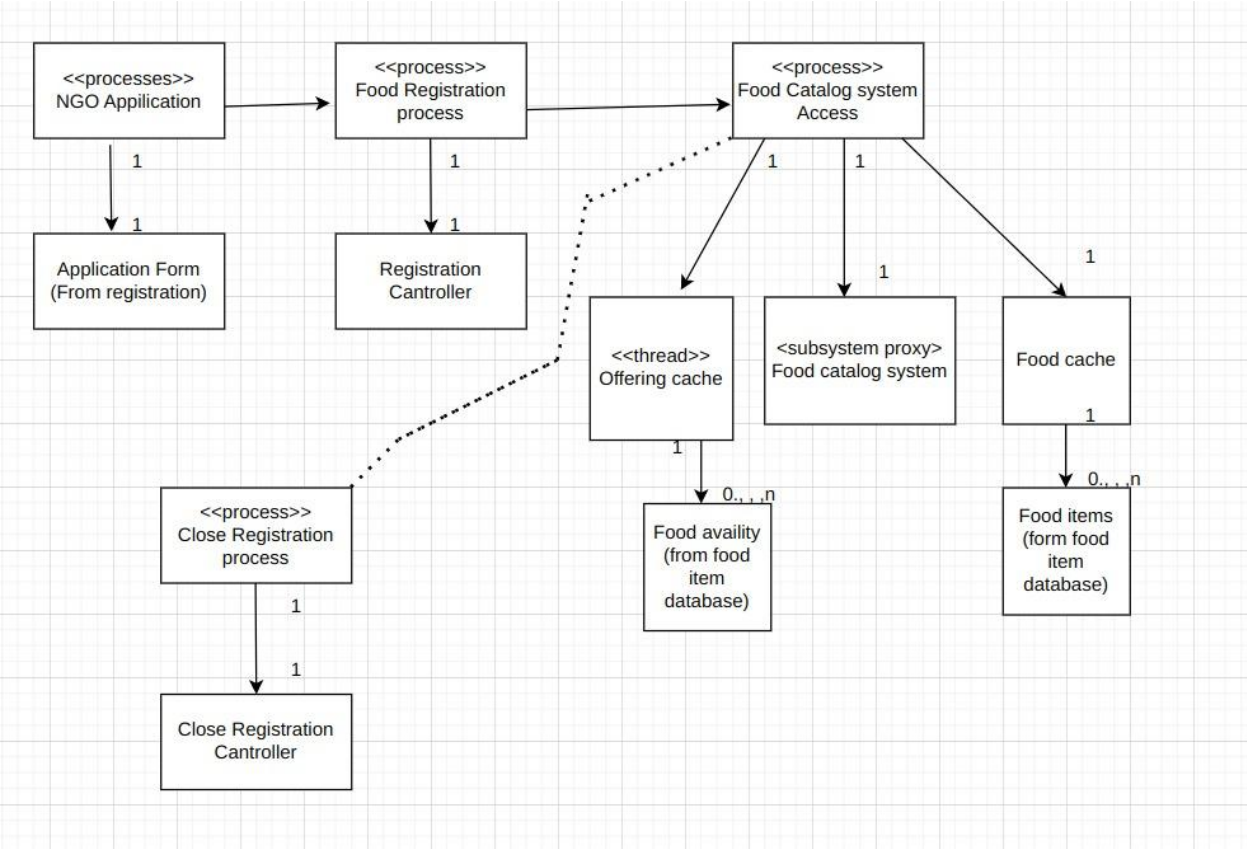


Diagram Name: Process to Design Elements

6.3 Process Model to Design Model Dependencies

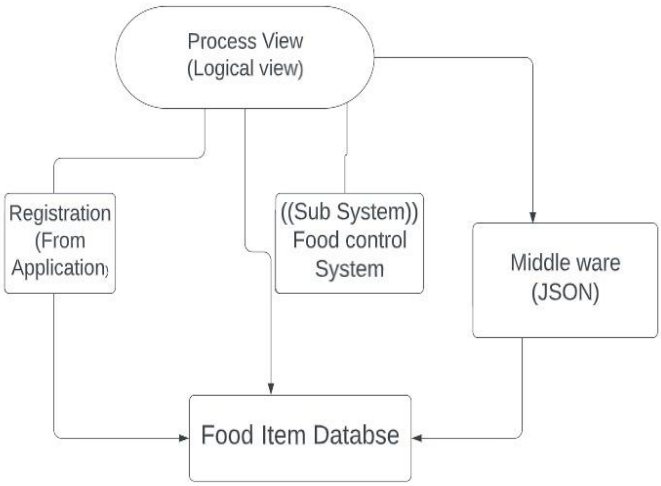


Diagram Name: Process Model to Design Model Dependencies

6.4 Processes to the Implementation

6.4.1 Remote

* The Remote interface serves to identify all remote objects. Any object that is a remote object must directly or indirectly implement this interface. Only those methods specified in a remote interface are available remotely.
* Implementation classes can implement any number of remote interfaces and can extend other remote implementation classes.

6.4.2 Runnable

* The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. The class must define a method of no arguments called run.
* This interface is designed to provide a common protocol for objects that wish to execute code while they are active. For example, Runnable is implemented by class Thread.
* Being active simply means that a thread has been started and has not yet been stopped.

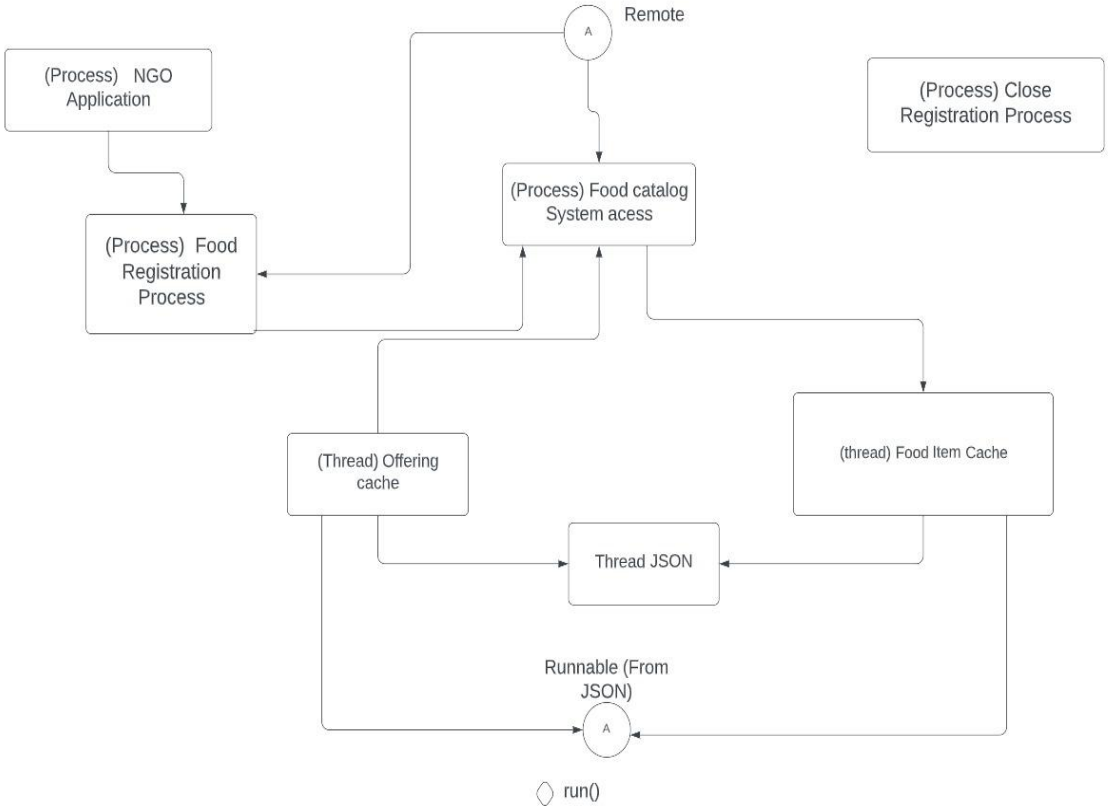


Diagram Name: Processes to the Implementation

6.4.3 Thread

* A *thread* is a thread of execution in a program. The Java Virtual Machine allows an application to have multiple threads of execution running concurrently.
* Every thread has a priority. Threads with higher priority are executed in preference to threads with lower priority. Each thread may or may not also be marked as a daemon. When code running in some thread creates a new Thread object, the new thread has its

priority initially set equal to the priority of the creating thread, and is a daemon thread if and only if the creating thread is a daemon.

7. Size and Performance

The chosen software architecture supports the key sizing and timing requirements:

1. The system shall support up to 2000 simultaneous users against the central database at any given time, and up to 500 simultaneous users against the local servers at any one time.
2. The system shall provide access to the legacy course catalog database with no more than a 10 second latency.
3. The system must be able to complete 80% of all transactions within 2 minutes.
4. The client portion shall require less than 256 MB disk space and 512 MB RAM.

The selected architecture supports the sizing and timing requirements through the implementation of a client-server architecture. The client portion is implemented on local campus PCs or remote dial up PCs. The components have been designed to ensure that minimal disk and memory requirements are needed on the PC client portion.

8. Quality

The software architecture supports the quality requirements:

1. The desktop user-interface shall be Windows 7 compliant.
2. The user interface of the registration system shall be designed for ease-of-use and shall be appropriate for a computer-literate user community with no additional training on the System.
3. Each feature of the registration system shall have built-in online help for the user. Online help shall include step by step instructions on using the system. Online Help shall include definitions for terms and acronyms.
4. Mean Time Between Failures shall exceed 300 hours.
5. The registration System shall be available 24 hours a day, 7 days a week. There shall be no more than 4% down time.