CentraLog

Varshesh Jitendrakumar Patel
Master of Applied Computing
University of Windsor
Windsor, ON, Canada
patel5p4@uwindsor.ca

Smitkumar Bhagyeshkumar Patel
Master of Applied Computing
University of Windsor
Windsor, ON, Canada
patel4f4@uwinndsor.ca

Parth Rameshbhai Patel
Master of Applied Computing
University of Windsor
Windsor, ON, Canada
patel7b5@uwindsor.ca

Ankit Kunwar
Master of Applied Computing
University of Windsor
Windsor, ON, Canada
kunwar1@uwinndsor.ca

***Abstract*— CentraLog is a centralized logging system that will support different languages, development environments, and production environments. It will collect all logs from the different applications using the Apache Kafka server and fetches that data into the Server application. The server application will store this data in the MongoDB Databases and will facilitate the feature of visualization of logs in the admin site to the developer. This allows users to store unstructured messages which can be used in the analytical processes.**

Keywords—MongoDB, Apache Kafka, Log Management, Distributed Databases, Angular.

I. Introduction & Motivation

In the world of Distributed Systems and vast development programing languages, Developers are having problems managing the user-based and system-based logs generated by the applications. These applications are deployed in different environments thus developers have to write specific features in every application they develop to manage the logs which in terms increases the development time and cost. There are also problems with the visualization of logs in any typical application. Developers must spend a lot of time finding any error indicating logs for debugging process.

Our team has a lot of experience with debugging and logging the information in our previous jobs. We had to use an individual-based logging system for every application which we developed and when we had to develop multiple applications concurrently, we faced many problems with debugging because of the different modes of logging for different languages and production environments. Which lead us to develop the CentraLog. We have developed the Centralog using .net core as the base framework while the Apache Kafka as a data streaming module. Which allows reliable and lossless transmission of the data from client libraries to the server.

II. Related Works

A. Logstash

Logstash is a Centralised server-side data processing system based on elastic search and Kibana that ingests data from different sources into the server[1]. Logstash stores log into Elasticsearch using its RESTful service. Elasticsearch uses Document based model to store data. Elasticsearch also provides an analytical engine to process data. While Kibana provides the visual interface to visualize log files. The major issue with Logstash is that the user needs to have a good understanding and analysis of the input of the log. Uses HTTPS protocol to collect logs that are not fast. Thus, we have implemented Apache Kafka-based log collection system which provides robust and fault-tolerant log ingestion and collection pipeline.

B. Graylog [2]

Graylog is the leading centralized logging software in the industry. It is dependent on the Elastic search component. Graylog provides a secure system for the security of log data which is collected through multiple devices and applications [2]. It collects logs from different devices using Beats and Syslog. Both services use HTTPS API calls to transfer data.

It has a great learning curve as one must learn all base components in their deployment stack. Thus, to reduce complexity in a system we have combined different stack installations into a single .net core application and Apache Kafka module. So, the user does not have to spend more time learning CentraLog and can start using it in direct production or development system.

III. Proposed Model

CentraLog follows microservice architecture to modularise the whole application to create fault-tolerant capabilities.

A. Architecture

1. Component Architecture
   The software can be modularized into Four main modules.

a) Producer Application: The producer Application contains user side library which will provide an interface to send logs from the user logic space to the Kafka server.

b) Kafka Server: The Kafka server is deployed in the docker system along with MongoDB and Redis.

c) Consumer Application: The consumer application is consisting of a logger server developed in .net core, MongoDB, and Redis Cache.

d) Admin Site: The admin site provides the viewing of logs and a project management system for users.

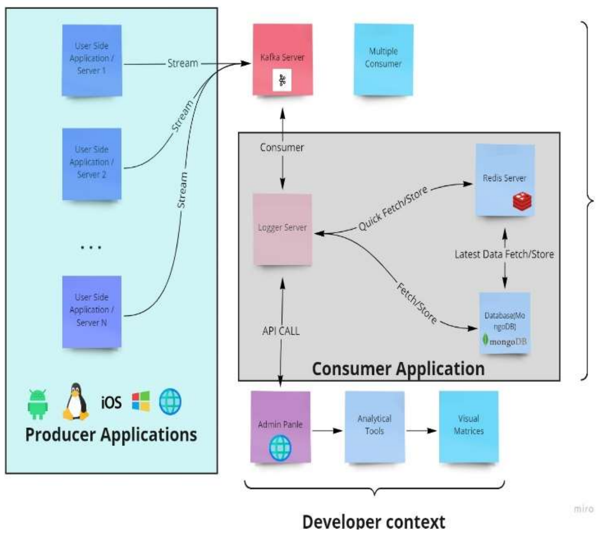
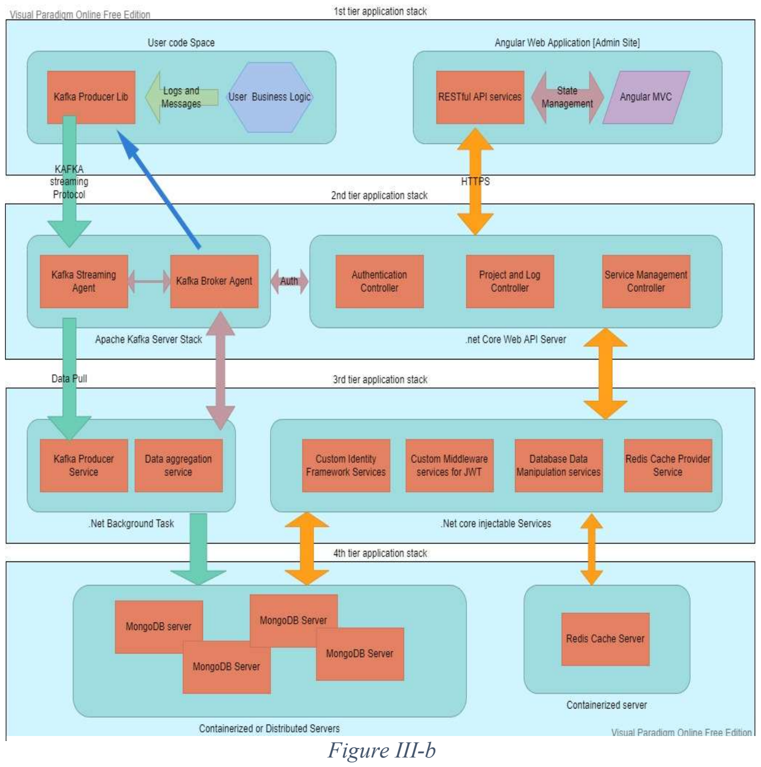


Figure III-a

2. 4-Tier Architecture

The software will follow 4-tier Architecture for the database and the business logic. The architecture of business logic and database are as follows.

- The angular application will be in the first tier
- RESTful APIs written in .net Core will be in the second tier as controllers and the Apache Kafka server also will be in the second tier.
- Different services such as authentication services, Authorization services, Database manipulation services, and Kafka Consumer services will be in the third tier.


Visual Paradigm Online Free Edition
1st tier application stack
User code Space
Kafka Producer Lib
Logs and Messages
User Business Logic
Angular Web Application [Admin Site]
RESTful API services
State Management
Angular MVC
KAFKA streaming Protocol
HTTPS
2nd tier application stack
Kafka Streaming Agent
Kafka Broker Agent
Auth
Authentication Controller
Project and Log Controller
Service Management Controller
Apache Kafka Server Stack
.net Core Web API Server
Data Pull
3rd tier application stack
Kafka Producer Service
Data aggregation service
Custom Identity Framework Services
Custom Middleware services for JWT
Database Data Manipulation services
Redis Cache Provider Service
.Net Background Task
.Net core injectable Services
4th tier application stack
MongoDB server
MongoDB Server
MongoDB Server
MongoDB Server
Redis Cache Server
Containerized or Distributed Servers
Containerized server
Visual Paradigm Online Free Edition
Figure III-b

- Redis Cache and MongoDB (Database) will be in the fourth tier.

3. Kafka Server Architecture

Apache Kafka is a real-time data streaming platform. There are lots of uses of Kafka, such as combining the data, storing the data, data-streaming across platforms, and many more. In this project, we used Kafka to collect the logs from the different applications and stream the log data to the developer's application. Apache Kafka can be able to scale production clusters up to a lot of brokers and can stream an infinite number of messages within a day[3].

Here, Kafka takes the logs from the numbers of applications from Application 1 to Application N and handed them to each Producer side. For every Application, there is a different Kafka Producer. Each Kafka Producer hands the Application logs to the Kafka Cluster of the particular bootstrap server. The Cluster Serializes the log data using JSON Serialization so that it can be transferred to the Consumer side securely. After that, the Cluster takes the help of the Kafka Zookeeper for the list of the Kafka Topic, and if matches then the Cluster hands Serialized log data to the various partitions, that are created on the Kafka topic. The concept of multiple partitions allows high message processing throughput. It improves the data access time from the consumer's point of view.

The Consumer connects from the other side to the particular Kafka Topics on the same bootstrap server as the Kafka Producer, to get access to the data. The Kafka Broker helps the consumers to fetch the serialized log data from the partitions. On the consumer's side, before displaying on the application, the serialized log data, is first, deserialized using the JSON deserialization. Then, it can be displayed on the developer's application.

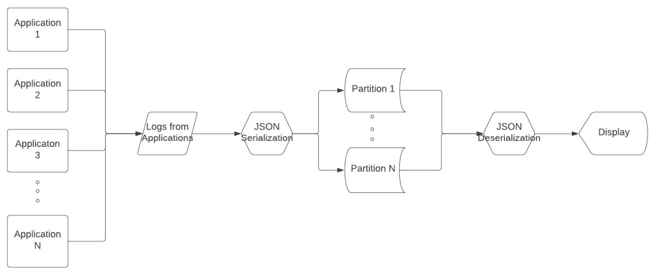


Figure III-c

4. Design Patterns [4]

The basic Pattern of a Web Server will be MSC (model service and controller), where the controller will have all the required and supportive data from the service passed in the form of models based on the command. While the Angular web application will follow MVC architecture patterns.

a) Creational Patterns

Factory Methods: This pattern will be applied to different services as they will be provided at the start of the application server

Abstract Factory: This pattern will be applied to different services to mask all business logic into the simple API methods.

Singleton and Scoped: This pattern will be applied to services and database contexts to create request-based shared resources which will use common objects for particular requests using dependency injection.

b) Structural Patterns

Faced: This pattern will be applied to all services and every tier of 4-tier architecture to simplify the chain of commands between tiers and services.

Decorator: Also known as the wrapper, will be used at all the tiers of the application to make it modularize.

c) Behavioral Patterns

Chain of Responsibility: This pattern will be applied to govern the flow of data and commands between tiers and services.

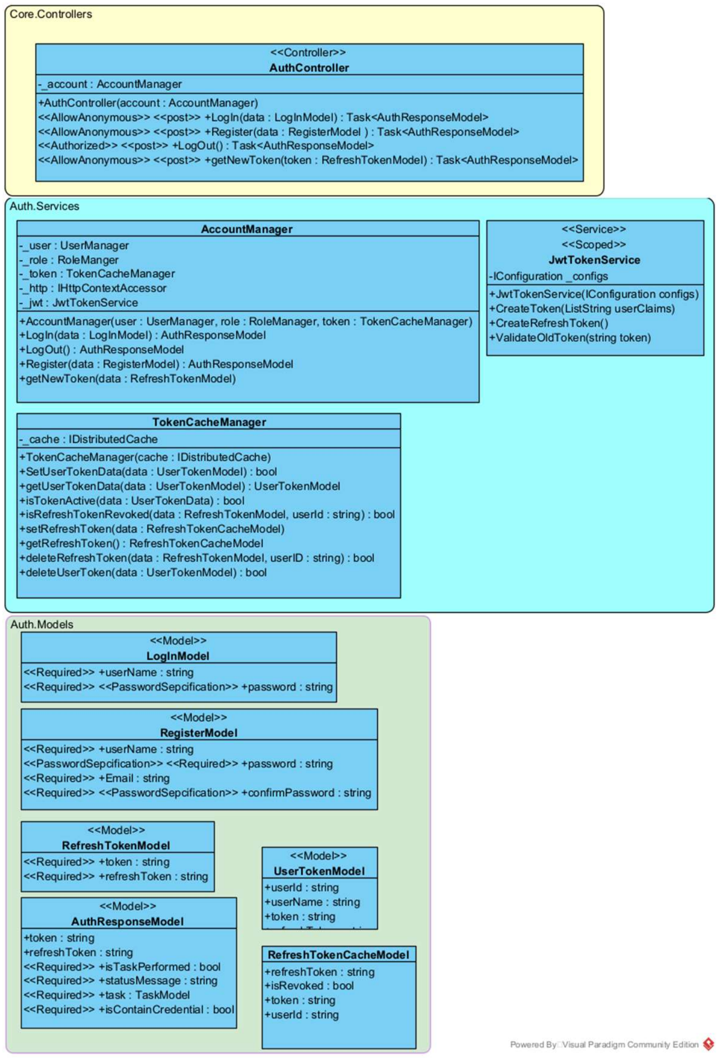

Core.Controllers
<<Controller>>
AuthController
-_account : AccountManager
+AuthController(account : AccountManager)
<<AllowAnonymous>> <<post>> +LogIn(data : LogInModel) : Task<AuthResponseModel>
<<AllowAnonymous>> <<post>> +Register(data : RegisterModel) : Task<AuthResponseModel>
<<Authorized>> <<post>> +LogOut() : Task<AuthResponseModel>
<<AllowAnonymous>> <<post>> +getNewToken(token : RefreshTokenModel) : Task<AuthResponseModel>
Auth.Services
AccountManager
-_user : UserManager
-_role : RoleManger
-_token : TokenCacheManager
-_http : IHttpContextAccessor
-_jwt : JwtTokenService
+AccountManager(user : UserManager, role : RoleManager, token : TokenCacheManager)
+LogIn(data : LogInModel) : AuthResponseModel
+LogOut() : AuthResponseModel
+Register(data : RegisterModel) : AuthResponseModel
+getNewToken(data : RefreshTokenModel)
<<Service>>
<<Scoped>>
JwtTokenService
-IConfiguration _configs
+JwtTokenService(IConfiguration configs)
+CreateToken(ListString userClaims)
+CreateRefreshToken()
+ValidateOldToken(string token)
TokenCacheManager
-_cache : IDistributedCache
+TokenCacheManager(cache : IDistributedCache)
+SetUserTokenData(data : UserTokenModel) : bool
+getUserTokenData(data : UserTokenModel) : UserTokenModel
+isTokenActive(data : UserTokenData) : bool
+isRefreshTokenRevoked(data : RefreshTokenModel, userId : string) : bool
+setRefreshToken(data : RefreshTokenCacheModel)
+getRefreshToken() : RefreshTokenCacheModel
+deleteRefreshToken(data : RefreshTokenModel, userID : string) : bool
+deleteUserToken(data : UserTokenModel) : bool
Auth.Models
<<Model>>
LogInModel
<<Required>> +userName : string
<<Required>> <<PasswordSepcification>> +password : string
<<Model>>
RegisterModel
<<Required>> +userName : string
<<PasswordSepcification>> <<Required>> +password : string
<<Required>> +Email : string
<<Required>> <<PasswordSepcification>> +confirmPassword : string
<<Model>>
RefreshTokenModel
<<Required>> +token : string
<<Required>> +refreshToken : string
<<Model>>
UserTokenModel
+userId : string
+userName : string
+token : string
<<Model>>
AuthResponseModel
+token : string
+refreshToken : string
<<Required>> +isTaskPerformed : bool
<<Required>> +statusMessage : string
<<Required>> +task : TaskModel
<<Required>> +isContainCredential : bool
RefreshTokenCacheModel
+refreshToken : string
+isRevoked : bool
+token : string
+userId : string
Powered By: Visual Paradigm Community Edition

Figure III-d

B. Features [5]

1. User: Developer

- Developers can view logs on the admin site of a project on which they are assigned.
- Developers can access Admin Site using their credential
- Developers can Export logs to files or can download logs of assigned project
- Developers can send logs to the Apache Kafka server using provided library for language.
- The developer can change the configuration of the provided library

2. User: Project Manager

- Project managers have all functionality of a Developer.
- Project managers can assign a developer to the project and can remove the developer.
- Project Managers can Create and Delete projects.
- Project Managers can remove logs from the database.
- Project managers can change the project configuration.

3. User: Administrator

- The administrator has all functionality of the Project Manager
- The administrator can add and remove the Project Manager.
- The Administrator can configure the number of instances of the web server and Kafka server for distributed services.
- The Administrator can change system architecture based on the environment

C. Tools and Technologies

1. Web Server

Web Server will be developed using ASP.Net core RESTful API server. ASP.net core uses C# language and is platform-independent technology because of this, the server can be deployed on different environments and platforms. The details regarding the framework are as follows [6].

- Version of Framework: .Net 6.0.0
- Language Version: C# 10.0
- Most Prominent Features: Microservice-oriented,

Very Fast compared to other existing Technologies, Custom Middleware and Identity Framework for Security, Parallel Task, and Background Task Support, Large Community.

2. Message streaming service

Message streaming service will be implemented using Apache Kafka. Apache Kafka is an open-source message streaming server that facilitates multiple senders (producers) and receivers (Consumers). Apache Kafka is a Unidirectional message streaming which means that only senders can send messages on the network [3].

- Version: Apache Kafka 3.2.0

3. Data storage and Inter-Process Communications

MongoDB will be used as the Main Data storage and Redis in Memory Cache will be used as Inter-process communication and caching. MongoDB provides NoSQL architecture which is beneficial for a large amount of unstructured data. Logs and Messages will be in the form of unstructured data[7].

- Database Version: MongoDB 4.4
- Most Prominent Features: High Performance in reading and writing
- NoSQL, Scalable, Platform Independent, Open Source
- Cache Version: Redis 7.0
- Most Prominent Features: In Memory Store, Scalable, High performance, Open source.

4. Web Application

Angular will be used to create a web application and interface for users to use the software. Angular is client-side and one-page rendering framework which will reduce the toll on web servers while providing the best features of the web application[8].

- Version: Angular 14 / 13
- Most Prominent Features: SAP (single page application), Client-side rendering, Bidirectional Dataflow, Large Community, Modularize

IV. Results

The load testing and Database average response testing has been done on Project and Log level database services. In the following table, we have described the average response time of query from restful API.

Table 1

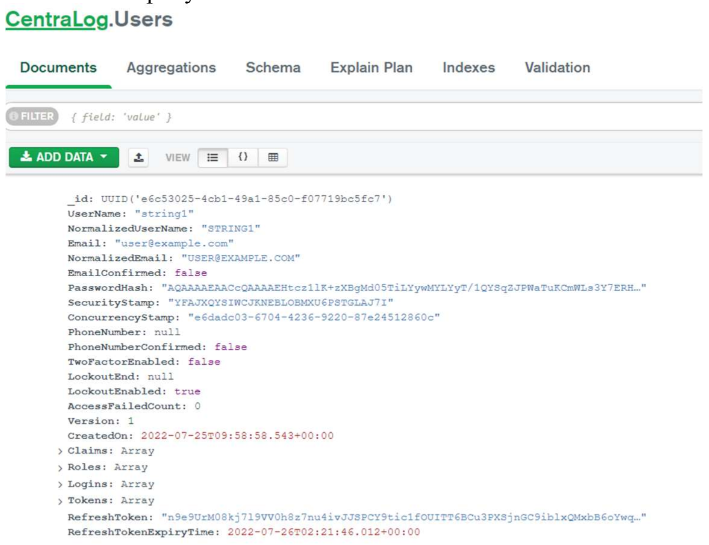
Query On	Query Type	Database	Response Time
Project	create	MongoDB	32.80 ms
Project	Get	MongoDB	28.60 ms
Project	Delete	MongoDB	36.80 ms
Log	Get	MongoDB	36.70 ms
Log	Insert	MongoDB (From Consumer)	37.80 ms

As we have used MongoDB for its NoSQL document-based storage system. Which has helped us in reducing the complexity of storing data in the document. Fig. IV-a and Fig. IV-b are Database Schema and data after insertion of data into MongoDB.

V. Limitation

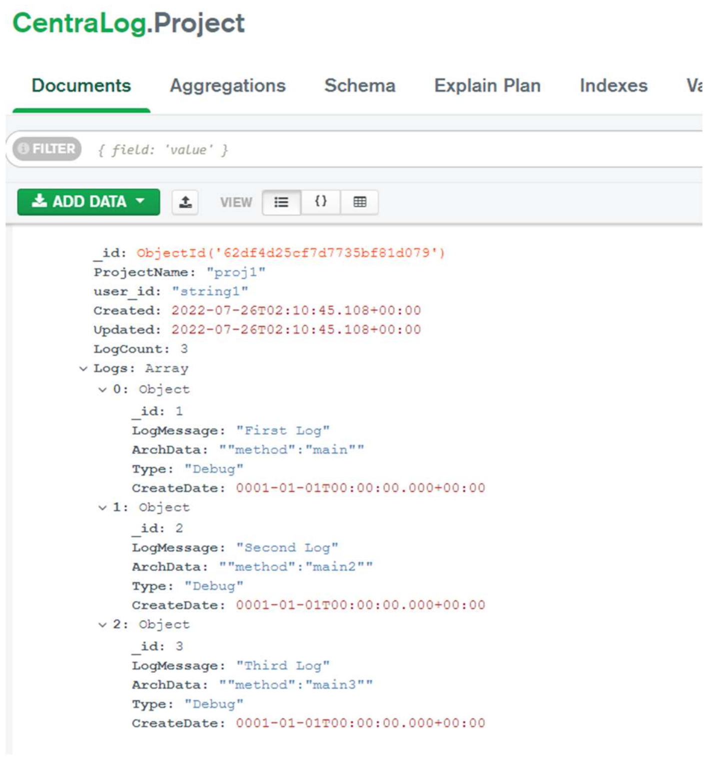
A. MongoDB Drivers

MongoDB drivers for the .Net Core application is still complicated compared to other providers. Because of that, multiple structural patterns cannot be created directly using MongoDB Drivers. To create them we still need to use old cursor-based query drivers.



```
_id: UUID('e6c53025-4cb1-49a1-85c0-f07719bc5fc7')
UserName: "string1"
NormalizedUserName: "STRING1"
Email: "user@example.com"
NormalizedEmail: "USER@EXAMPLE.COM"
EmailConfirmed: false
PasswordHash: "AQAAAAEAACcQAAAAEHtcz1lK+zXBgMd05TiLYywMYLYyT/1QYSqZJPWaTuKCmWLs3Y7ERH…"
SecurityStamp: "YFAJXQYSIWCJKNEBLOBMXU6PSTGLAJ7I"
ConcurrencyStamp: "e6dadc03-6704-4236-9220-87e24512860c"
PhoneNumber: null
PhoneNumberConfirmed: false
TwoFactorEnabled: false
LockoutEnd: null
LockoutEnabled: true
AccessFailedCount: 0
Version: 1
CreatedOn: 2022-07-25T09:58:58.543+00:00
> Claims: Array
> Roles: Array
> Logins: Array
> Tokens: Array
RefreshToken: "n9e9UrM08kj7l9VV0h8z7nu4ivJJSPCY9tic1fOUITT6BCu3PXSjnGC9iblxQMxbB6oYwq…"
RefreshTokenExpiryTime: 2022-07-26T02:21:46.012+00:00
```

Figure IV-a



```
_id: ObjectId('62df4d25cf7d7735bf81d079')
ProjectName: "proj1"
user_id: "string1"
Created: 2022-07-26T02:10:45.108+00:00
Updated: 2022-07-26T02:10:45.108+00:00
LogCount: 3
v Logs: Array
  v 0: Object
      _id: 1
      LogMessage: "First Log"
      ArchData: ""method":"main""
      Type: "Debug"
      CreateDate: 0001-01-01T00:00:00.000+00:00
  v 1: Object
      _id: 2
      LogMessage: "Second Log"
      ArchData: ""method":"main2""
      Type: "Debug"
      CreateDate: 0001-01-01T00:00:00.000+00:00
  v 2: Object
      _id: 3
      LogMessage: "Third Log"
      ArchData: ""method":"main3""
      Type: "Debug"
      CreateDate: 0001-01-01T00:00:00.000+00:00
```

Figure IV-b

B. Docker composer for Apache Kafka

Docker composer cannot be integrated into the .Net Core Installation setup as a direct component. Instead of that, we must use a bash script or batch script as the building pipeline component.

C. Library Wrapper For Different Languages

Each language has its execution procedure due to this all libraries for the different languages cannot be created with the same structure.

VI. Conclusion And Future Works

The CentraLog provides an efficient way to store and manage logs from different Application environments and languages. Due to the usage of an unstructured database, users can insert as much metadata into the log for future analysis regardless of the current provided models.

For future works, the following feature will be implemented.

- Analytical Engine: This feature will provide multiple analytical tools to analyze logs.
- Robust Server and Service Control: This feature will allow the user to manipulate the size of the server node and service life cycles.

VII. References

[1] https://github.com/elastic/logstash

[2] https://www.graylog.org/

[3] https://kafka.apache.org/documentation/

[4] https://refactoring.guru/design-patterns

[5] Kent, K. A., & Souppaya, M. (2006). Guide to Computer Security Log Management.

[6] https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-6.0

[7] https://www.mongodb.com/docs/drivers/csharp/

[8] https://angular.io/docs

[9] https://github.com/varsheshjp/CentraLog