



UNIVERSITY OF NAIROBI

SCHOOL OF COMPUTING AND INFORMATICS

A Microservices Based Student Industrial Attachment Information System Model  
for Technical and Vocational Education and Training Institutions in Kenya

Student Name: Esadia Benard

Registration Number: P53/11300/2018

Supervisor: Professor Robert Oboko

A research project report submitted to the School of Computing and Informatics in  
partial fulfillment of the requirements for a ward of the Degree of Master of  
Science in Distributed Computing Technology of University of Nairobi, Nairobi,  
Kenya

August 2021

**Declaration**

This research report is my original work and has not been presented for any award in any other University.

Signature .....  ..... Date 20/08/2021 .....

Student Name: Esadia Benard

Registration Number: P53/11300/2018

This research project report has been submitted for examination with my approval as University of Nairobi Supervisor

Signature .....  ..... Date 27/08/2021 .....

Professor Robert Oboko

School of Computing and Informatics

University of Nairobi

## **Abstract**

Microservices is a software development technique—a sub type of the service oriented architecture style that structures an application as a collection of autonomous, independent and loosely coupled services that communicate using network lightweight communication protocols. Lack of a system to automate Industrial Attachment activities in TVET institutions was the driving force towards coming up with a microservices system that has the benefits of being more resilient, fault tolerant and scalable. The study used descriptive research design to identify user needs of the model. A questionnaire; an easy to use tool to widely reach the identified sample was used in data collection. Data analysis was by simple tabulation. Agile software development methodology was used to develop and implement the model because of its iterative nature and customer engagement approach. The Security service was implemented as SaaS using Okta Oauth 2.0 protocol, User Interface was implemented as a microservice using React JavaScript library, the Attachment microservice and Reports microservice were implemented using Python Django framework and Django REST framework. The model was deployed using Docker compose containers where users tested and validated it by performing assigned tasks. Locust load testing tool was used to test the performance of the system when it was scaled horizontally. The study used texts, and line charts to communicate and display the analyzed data. Testing results shows the system implemented the user functional requirements and was easy to learn & easy to use. The results confirmed that container platforms help in effectively deploying microservices to achieve scalability. Properly scaled components leads to improved system performance. The study recommends adoption of the system by TVET institutions and further study to be done on API's management in microservices.

## Table of Contents

DECLARATION .....	I
ABSTRACT .....	II
TABLE OF CONTENTS .....	III
LIST OF FIGURES .....	IV
LIST OF TABLES .....	V
LIST OF ABBREVIATIONS .....	VI
DEFINITION OF TERMS .....	VII
<b>CHAPTER ONE: INTRODUCTION.....</b>	<b>1</b>
1.1 BACKGROUND OF THE STUDY .....	1
1.2 PROBLEM STATEMENT .....	3
1.3 OBJECTIVES .....	4
1.4 SIGNIFICANCE OF THE STUDY.....	4
1.5 SCOPE OF THE STUDY .....	5
1.6 THE ASSUMPTIONS OF THE STUDY .....	5
<b>CHAPTER TWO: LITERATURE REVIEW.....</b>	<b>6</b>
2.1 INTRODUCTION .....	6
2.2 INDUSTRIAL ATTACHMENT IN TVET INSTITUTIONS .....	6
2.2.1 <i>Industrial Attachments Activities in TVET Institutions</i> .....	6
2.2.2 <i>Challenges associated with management of industrial attachment in TVET institutions</i> .....	7
2.3 MICROSERVICE ARCHITECTURE.....	8
2.3.1 <i>Monolith Application versus Microservices Based Application</i> .....	8
2.3.2 <i>Service Oriented Architecture Application versus Microservices Application</i> .....	10
2.3.3 <i>Microservices Design Principles</i> .....	11
2.3.4 <i>Microservices Decomposition Patterns</i> .....	11
2.3.5 <i>Microservices Integration Patterns</i> .....	12
2.3.6 <i>Security in Microservices</i> .....	12
2.3.7 <i>Microservices and Containers</i> .....	13
2.3.8 <i>Scaling in Microservices</i> .....	13
2.3.9 <i>Challenges faced while using Microservices</i> .....	13
2.4 REVIEW OF RELATED SYSTEMS.....	15
2.4.1 <i>Monolith Systems</i> .....	15
2.4.2 <i>SOA Systems</i> .....	16
2.4.3 <i>Microservices Systems</i> .....	16
2.5 THE PROPOSED MODEL.....	17
<b>CHAPTER THREE: RESEARCH METHODOLOGY .....</b>	<b>18</b>
3.1 INTRODUCTION .....	18
3.2 RESEARCH DESIGN.....	18
3.3 DEVELOPING THE MODEL .....	19
3.3.1 <i>Software Development Methodology</i> .....	19
3.4 SYSTEM REQUIREMENTS GATHERING .....	20
3.4.1 <i>Target Population</i> .....	20
3.4.2 <i>Sampling Procedure</i> .....	20
3.4.3 <i>Sample Size</i> .....	20
3.5 DATA COLLECTION INSTRUMENTS.....	21
3.5.1 <i>Questionnaire</i> .....	21
3.5.2 <i>Secondary Data</i> .....	22
3.5.3 <i>Validity and Reliability</i> .....	22

3.6 DATA COLLECTION .....	22
3.6.1 Administration of Questionnaires .....	22
3.6.2 Documents Review .....	23
3.7 DATA ANALYSIS .....	23
3.7.1 Responses from Students: .....	23
3.7.2 Responses from Lecturers .....	24
3.7.3 Document Reviews .....	25
3.8 SYSTEM REQUIREMENTS SPECIFICATION .....	26
3.8.1 System Feasibility .....	26
3.8.2 System Analysis .....	26
3.8.3 Use Case Diagram .....	29
3.8.4 Data Flow Diagram .....	30
3.9 SYSTEM DESIGN .....	32
3.9.1 Architectural Design .....	32
3.9.2 Sequence Diagrams .....	33
3.9.3 Database Design .....	34
3.9.4 Program Design .....	36
3.9.5 User Interface Design (Forms) .....	38
3.10 SYSTEM IMPLEMENTATION .....	40
3.10.1. Hardware Resources .....	40
3.10.2 Software Resources .....	40
3.10.3 Programming Tools .....	41
3.10.4 Security Service Implementation .....	42
3.11 SYSTEM TESTING .....	42
3.11.1 System Scalability Testing .....	42
3.11.2 System Validation Testing .....	44
<b>CHAPTER FOUR .....</b>	<b>46</b>
<b>4.0 RESULTS AND DISCUSSIONS .....</b>	<b>46</b>
4.1 MODEL SCALABILITY AND LOAD TESTING .....	46
4.2 MODEL VALIDATION TESTING RESULTS .....	49
<b>CHAPTER FIVE .....</b>	<b>52</b>
<b>5.0 FINDINGS, CONCLUSIONS AND RECOMMENDATIONS .....</b>	<b>52</b>
5.1 FINDINGS .....	52
5.2 LIMITATIONS OF THE RESEARCH .....	53
5.3 CONCLUSIONS .....	53
5.4 RECOMMENDATIONS FOR PRACTICE .....	54
5.5 RECOMMENDATIONS FOR FURTHER RESEARCH .....	54
<b>APPENDICES .....</b>	<b>55</b>
I) REFERENCES .....	55
II) RESEARCH QUESTIONNAIRES .....	60
III) PROJECT SCHEDULE .....	63
IV) PROJECT BUDGET .....	63
V) SAMPLE USER INTERFACE .....	64
VI) SAMPLE CODE .....	66

## List of Figures

Figure 1: Monolithic Architecture (Kanjilal , 2020) .....	8
Figure 2: Microservices Architecture (Kanjilal , 2020) .....	8

Figure 3: SOA vs Microservices Communication (Wittmer, 2021).....	10
Figure 4: Agile software development cycle (Brush & Silverthorne, 2021).....	19
Figure 5: Use Case Diagram.....	29
Figure 6: Context Diagram.....	30
Figure 7: Data Flow Diagram Level 1.....	31
Figure 8: System Architectural Design .....	32
Figure 9: Authentication with Okta Sequence Diagram (Okta, 2021) .....	33
Figure 10: Attachment Microservice Sequence Diagram.....	33
Figure 11: Integration Sequence Diagram.....	34
Figure 12: Entity Relational Diagram .....	35
Figure 13: System Flowchart.....	36
Figure 14: Design of ILO’s Page User Interface.....	38
Figure 15: Design of Admin’s Add User Page Interface.....	39
Figure 16: Design of Admin Attachment Opportunities Page User Interface.....	40
Figure 17: Django, Django Rest Framework, React Interactions.....	41
Figure 18: Dockerized Load Balancing Architecture (Maximilian , 2021).....	42
Figure 19: Locust System Load Testing Interface.....	46
Figure 20: Locust Load Test Results. Test 1 (system scaled at 3 instances).....	47
Figure 21: Locust Load Testing Results. Test 2 (system scaled at 6 instances).....	47
Figure 22: Locust Load Testing Tests1 and Test 2 Results. Test Charts .....	48

**List of Tables**

Table 1: Requirement before proceeding for Industrial Attachment.....	23
Table 2: Challenges faced while sourcing for Industrial Attachment .....	23
Table 3: Challenges faced by students on industrial attachment.....	23
Table 4: Industrial Attachment Activities .....	24
Table 5: Documents Awarded on Completing Industrial Attachment .....	24
Table 6: Trainers Industrial Attachment Challenges .....	24
Table 7: Documents used by the Industrial Attachment.....	25
Table 8: Industrial attachment Assessment Parameters .....	25
Table 9: Technologies used by Industrial Attachment Docket.....	25
Table 10: Measures to improve Industrial Attachment Operations.....	25
Table 11: Validation Test Tasks and Results for Admin.....	49
Table 12: Validation Test Tasks and Results for ILO .....	50
Table 13: Validation Test Tasks and Results for Students.....	51
Table 14 Validation Test Tasks and Results for Lecturers.....	51

## **List of Abbreviations**

API	-	Application Programming Interface
ESB	-	Enterprise Service Bus
HTTP	-	Hypertext Transfer Protocol
ILO	-	Industrial Liaison Officer
JSON	-	JavaScript Object Notation
REST	-	Representative State Transfer
SaaS	-	Software as a Service
SOA	-	Service Oriented Architecture
SOAP	-	Simple Object Access Protocol
SSO	-	Single Sign-On
TVET	-	Technical and Vocational Education and Training
WSDL	-	Web Service Definition language
WSGI	-	Web Server Gateway Interface
XML	-	Extensible Markup Language
YAML	-	Ain't Markup Language

## **Definition of Terms**

**Docker:** an open platform for packaging and running software applications in a loosely isolated and secure computing environment called a container. A container is a runnable instance of an image. An image is a script that packages an application stating the applications pre-configured server environments.

**Microservice:** a lightweight application, which provides a narrowed list of features with a well-defined contract. It's a component with a single responsibility, which can be developed and deployed independently and interacts with other components using network communication mechanism.

**Model:** a sample, prototype, or a mock-up of a product built to experiment an idea.

**SaaS:** a cloud computing software delivery model that allows users to connect to and use cloud-based applications over the internet.

**Scalability:** the ability of a device to adjust to the variations in the environment and meet the impending varying needs mostly the growing amount of work.

**Service:** a module that supports a specific task or business goal and uses a simple, well-defined interface, such as an application programming interface (API), to communicate with other sets of services.

**SOA:** is a term that represents a model in which software automation logic is structured into smaller, distinct components (units of logic). Together, these components make up an entire software system.

**SSO:** A web session and web user verification service that authorises a user to use one set of web login credentials (example is a name and password) to access numerous web applications.

**WSGI:** Python Common Gateway Interface (CGI) standard that simplifies how to write a python application in order to serve HTTP requests.

**YAML:** is a digestible, human-readable data-oriented / serialization language that is often used for writing container configuration files.



# CHAPTER ONE: INTRODUCTION

## 1.1 Background of the Study

The ever advancing Internet technology is continually calling for the need to find improved ways of designing and implementing software systems. Customers' needs that are always changing are forcing organizations to adopt software applications that are quick to deploy, easy to maintain and always available. While traditional architectures can still handle a lot of this, as the code size grows, it reaches a point where, a more dynamic, scalable style of application development is needed. One such approach is the microservices architecture (Jeremy, 2021).

Microservices is a subset of Service Oriented Architecture (SOA) style of software development that structures an application as a collection of loosely attached services. These services are fine-grained, are independently deployed and use lightweight protocols to communicate (Hardik, 2020). The services are smaller in scope, they have a single responsibility and work together to form the entire system.

Microservices are the opposite of monolithic applications. A Monolith is made up of software modules or components that are tightly coupled and cannot be executed independently (Ziade, 2017). This makes it difficult to scale, maintain and reuse the components. Microservices support use of various technology stack where different platforms, programming languages and technologies can be used to develop different services of the same application. Mordo (2021) explains that microservice writes only to its own database, supports stateless interaction and can be deployed as multiple instances mostly using containers or virtual machines in a data center.

Microservices approach allows building distributed applications that comprises of small, autonomous components or services which interact with each other using Application Programming Interfaces (APIs). The APIs have various message formats including Extensible Markup Language (XML), JavaScript Object Notation (JSON) among others (Jeremy, 2021). A number of communication protocols like Hypertext Transfer Protocol (HTTP), Hypertext Transfer Protocol Secure (HTTPS), Simple Object Access Protocol

(SOAP), Representational State Transfer (REST), Remote Method Invocation (RMI) and Advanced Message Queuing Protocol (AMQP) aid API's interaction. These advantages makes microservices one of the most searched for architectures that promises scalability and agility of enterprise software (Hardik, 2020). Thus the need to design an Information System using Microservices; a case of Student Industrial Attachment for Technical and Vocational Education and Training (TVET) institutions in Kenya.

Industrial Attachment Training is a “work-based experience programme” that exposes students to real-life organizational context. The work-based session has the main aim of allowing students acquire industry knowledge, skills and attitudes under the supervision of a professional who is a mentor and a coach. This formal placement of students in the workplace is a mandatory academic requirement that takes approximately three to six months to supplement classroom training (Mutiso, 2021).

TVET institutions have Industrial Liaison Office that coordinate industrial attachment operations before, during and after the attachment ensuring that students are placed appropriately, are assessed and appraised (Korir, 2021). Students are expected to keep a daily log or portfolio of all the activities and lessons learnt during the attachment (Mutiso, 2021). The student is evaluated by both the industrial attachment supervisor and the academic department assessor (KTTC, 2021). The exercise ends when the student submits a written report detailing learnt experience to the academic department which compiles and awards the student an attachment score.

## 1.2 Problem Statement

Implementing software system using Microservices Architecture has been trending as a better option compared to SOA approach or even monolith architecture (Baboi, Iftene, & Gîfu, 2019). Many institutions have monolith software systems that are struggling to keep pace with the user expectation for interactive, rich and dynamic systems that are highly available, scalable and easy to execute (Familiar, 2015). Baboi, Iftene, & Gîfu (2019) observes that monolith applications have a large code base that keeps on growing and becoming complex, are difficult to scale, are more susceptible to failures as debugging is not easy and are harder to maintain or add new features thus the need to adopt the trending microservices approach. SOA lost momentum to microservices because of it being heavyweight, complex and having multiple processes that can reduce speed. While SOA offers Enterprise Service Bus to manage communications, microservices offers a better approach of smart endpoints and dumb pipes. To have systems that performs as per users expectations, organizations are structuring their applications into Microservices (Pachghare, 2016).

The management of Industrial Attachment docket in TVET Institutions in Kenya has not fully embraced technological advancements (Yannuar , Hasan, Abdullah, Hakim, & Wahyudin, 2018). This is despite the institutions having other information systems like Students Management Information Systems, Fees Collection Systems and Library Management Systems. The industrial attachment docket relies on manual reports like student's fees status and academic progress from these systems in order to execute its activities. Student's attachment records like Log Book and Attachment Report are manually stored. The process, even with some form of automation makes most of the institutions information disintegrated thus introduction of Microservices provides well organized, comprehensive and handy information to monitor and supervise students on industrial attachment.

Hymet & Arban (2020) cites the lack of a distributed industrial attachment management system has created a communication gap among the stakeholders. For instance, businesses / companies used different sources to find students or announce industrial attachment posts,

training institutions didn't have automated systems to aid students in applying for industrial attachment, monitoring students learning experiences and student's assessment was not automated. The available systems used Ms Word, Excel and Access which have less user functions for capturing, recording, finding, sharing, analyzing and producing output.

### **1.3 Objectives**

The overall objective of this project was to investigate and evaluate how to use microservices architecture in information systems for the management of students' industrial attachment programme in TVET Institutions in Kenya.

The study specific objectives included:

- 1) To find out requirements of the new industrial attachment management system in TVET institutions.
- 2) To provide a scalable means of monitoring and supervising students on industrial attachment using microservices architecture.
- 3) To test and validate the performance of the developed microservices model.

### **1.4 Significance of the Study**

This project report adds knowledge and understanding to the already existing literature on how to design software systems using microservices architecture. This is a source of literature for those intending to do further research on microservices architecture.

The study also generates insights and information to administrators, industrial attachment organizations and training institutions on how to offer the best hands-on experience to students undergoing industrial attachment exercise. This has resulted into improved productivity, efficiency and proper timely decision making on matters concerning industrial attachment.

The new system creates opportunities to expose students to the practical world to practice the theory and technical skills learnt in the classroom. The system strengthens the student's supervision and assessment process on industrial attachment. This improves students'

competence, their qualifications and facilitates them to enter the job market after graduating.

The system provides a foundation to integrate other information systems in TVET institution to provide unified sharing of information. Prospective systems for integration include student's management system and KNEC student's registration system.

### **1.5 Scope of the Study**

The microservice based model for managing Industrial Attachment information in TVET institutions in Kenya handles the following:

- Helps students secure and be placed on industrial attachment.
- Monitors and helps in supervising students' daily learning activities during the industrial attachment training period.
- Links TVET institutions and the industry for the purpose of developing competent manpower.
- Maintains student's industrial attachment records including award of scores.

### **1.6 The Assumptions of the study**

The literature reviewed while doing the project are assumed to have been adequate to be the groundwork of the arguments about the research project. Plagiarism as an issue was adequately handled by citation and reference list.

It is also assumed that the sample population used answered the questionnaires correctly, the data collected was analyzed accurately and with appropriate summaries, and research tools and techniques were accurate for the study. Thus the study's findings and conclusions are accurate and correctly stated.

The study observed ethical issues before, during and after data collection. For instance the researcher ensured confidentiality of the respondents over the information gathered. The sampled institutions were notified of the exercise during data collection.

## **CHAPTER TWO: LITERATURE REVIEW**

### **2.1 Introduction**

An overview of industrial attachment as a compulsory academic exercise and the difficulty experienced by handling manual disintegrated data in the industrial attachment docket are discussed by this chapter. The inefficiencies of Monolith and SOA architectures are conversed and the benefits of adopting Microservice-based system are also discussed.

### **2.2 Industrial Attachment in TVET Institutions**

Industrial attachment is a compulsory academic requirement for students learning in TVET institutions whose objective is to promote acquisition of practical work knowledge, ethics and skills. Kiplagat, et al. (2016) and KTTC (2021) notes that most students gain a unique experience to technology, work place ethics, morals & principles, and the organogram or organizational chart through Industrial Attachment.

#### **2.2.1 Industrial Attachments Activities in TVET Institutions**

Sourcing for attachment places, placing students on industrial training, assessment of students on industrial attachment plus linking institutions to workplaces are the major activities of Industrial Attachment department in TVET institutions (Mutiso, 2021). The department/section comprises of the Industrial Liaison Officer (ILO) as the head, members who are representatives of various academic departments and the office administrative staffs (KTTC, 2021). A student after having covered stipulated academic units in a given area of study, he / she is required to apply for an industrial attachment place under the guidance of the industrial liaison officer (NITA, 2021).

Once the student secures an attachment place he/she is given a posting letter (created using MS Word) to be presented to the administration of the organisation offering the attachment. The organisation should also do a letter to confirm the arrival of the student to the organisation and commencement of the industrial training (KTTC, 2021).

During the training, students are required to update their logbooks with the learning experiences acquired in their daily routines. The progress of the training is continually monitored and supervised by the supervisor from the industry who makes comments in the

student's logbooks (KTTC, 2021). An Assessor from the academic institution makes visits to evaluate the student's progress.

The Assessor is expected to compile the student's final score using supervisor's scores, student's logbook recordings, student's industrial attachment report and his / her own assessment comments (Korir, 2021). The ILO can award an Industrial Attachment Certificate to signify completion of the training.

### **2.2.2 Challenges associated with management of industrial attachment in TVET institutions**

According to Aineah (2019), industrial attachment has a number of challenges affecting its key participants. Students, hosting organisations and training institutions are victims of this challenges. Placement of students to the industry is not that easy (KTTC, 2021). About 10% of the students usually end up being attached to organisations which are under capacitated where they are exposed to competencies far from the area of specialization (Aineah, 2019).

Most of the records generated in the process of attaching students are manually kept in files by the ILO (Hasti, Lesari, & Gustiana, 2019). These includes student's industrial attachment qualification documents like copies of fees payment, academic progress and insurance cover documents. Additionally, copies of attachment request letters, attachment posting & confirmation letters and recommendation letters are also filed in cabinets. Finally, student's logbooks, student's attachment reports, assessor's reports and students' scores are also kept in student's individual folders. This makes it difficult to track the events in the entire process leading to manually searching for files to come up with information. Communication between the attachment organisation and training institution is largely unstructured (e.g. through e-mails, letters, phone calls) characterised with manual way of record keeping.

## 2.3 Microservice Architecture

Baboi, Iftene, and Gîfu, (2019) notes that the difficulties experienced by using monolithic applications or SOA systems are paving way for software professionals to embrace microservices as an approach to develop internet based applications.

### 2.3.1 Monolith Application versus Microservices Based Application

Kanjilal (2020) defines a monolith as an application that has a single-code base consisting of tightly coupled units that are installed as a single component. Ideally it is made up of a client user interface, Business unit and a Database unit as illustrated in the diagram below;

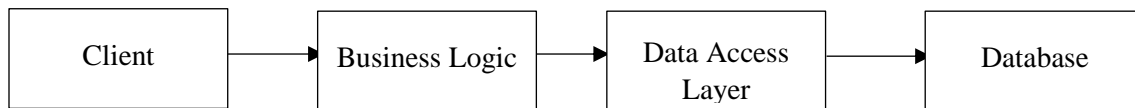


Figure 1: Monolithic Architecture (Kanjilal , 2020)

On the other hand Kanjilal (2020) explains that microservices delivers an application as a collection of small, independent, loosely coupled services. Microservices divides an application into small components that are independently functioning on their own and interact with each other to make up the entire system as shown in the diagram below;

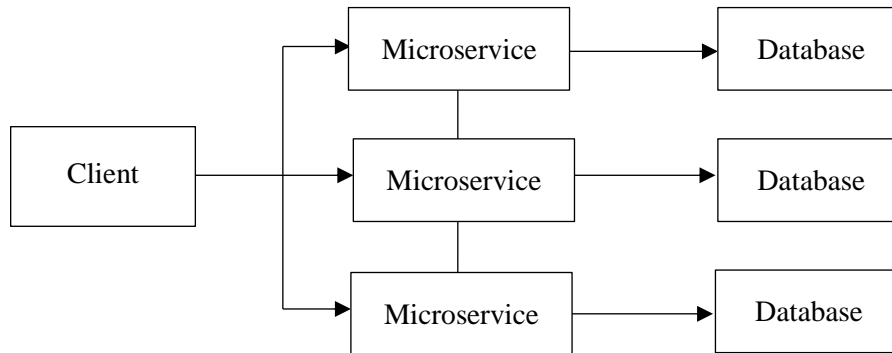


Figure 2: Microservices Architecture (Kanjilal , 2020)

Baboi, Iftene and Gîfu (2019) prefers microservices applications whose services are isolated from one another and execute independently of each other compared to monolith applications whose modules are executed in a tightly coupled manner. This give an edge to microservices as Wittmer (2021) terms them as more resilient and fault tolerant. Because a monolith is a single unit with many interdependencies, one bug can bring down



the entire application. Equally it is difficult to isolate the root cause of any problems that might emerge, and also difficult to recover from failures since it means the entire monolith application needs to be rebuilt. In contrast, a bug found in one microservice might not affect other services in the application because the services are autonomous thus it is easy to isolate a service, fix bugs in the service and redeploy the service.

In terms of agility, Yadav (2019) routes for Microservices as they support speedy development, speedy feature enhancement and redeployment because the development teams are usually divided into smaller teams probably each focusing on a single service. On the other hand, Monolith applications are characterised with slow feature enhancements and slow redeployment as mostly they have a huge code base, and the entire team can have difficulty in understanding the single code base.

Microservices application allows heterogeneous technologies to be used in implementing the individual services (Chandramouli, 2019). For instance each of the multiple services can individually be developed using a unique programming language and a unique data storage technology, whereas most of the monoliths are built using a single technology stack. Microservices also accommodates different team member's expertise in developing applications unlike monoliths single technology stack.

Concerning data storage, Swathi and Rashmi (2020) expounds that most of the microservices are designed to have their own databases as compared to a monolith application which share a single database. This makes the application more open to different database technologies. Database per service also allows faster development, enhances loose coupling and forms a basis for building agile and scalable systems.

Chandramouli (2019) adds that monoliths have different interaction styles compared to microservices. In monolith, each component is developed as procedure or function that communicates using call statements. Microservices have services running in their own distinct network nodes that communicate using APIs. Most of them use REST over HTTP for asynchronous and synchronous communication (Pachghare, 2016). For payloads they use XML or JSON with JSON being smaller in size and faster than XML. Both XML and JSON support integration between various languages and heterogeneous systems.

### 2.3.2 Service Oriented Architecture Application versus Microservices Application

SOA applications and Microservices applications have similarities. Both architectures consists of services which are smaller in scope and focuses on performing one particular business process. Wittmer (2021) agrees that the two architectures are open as developers have the leeway to use a platform and a programming language of their choice. They all allow interoperability where SOA services interact using Enterprise Service Bus (ESB) while Microservices communicate by Remote Procedure Call (RPC) across a network (Chandramouli, 2019).

The two architecture also differ in certain aspects. Waseem, Liang and Márquez (2020), Richards (2021) and Ghahrai (2017) in their articles agrees that the two architectures contrast in service granularity and component sharing. For instance, microservices have extremely smaller services that are constantly evolving thus fine-grained while SOA has large services that are more stable and coarse grained. Microservices are more resilient, easily deployed and are agile as compared to SOA services. Services in microservices are autonomous and don't share a common runtime environment as might be the case with SOA. Wittmer (2021) explains the differences using the diagram below;

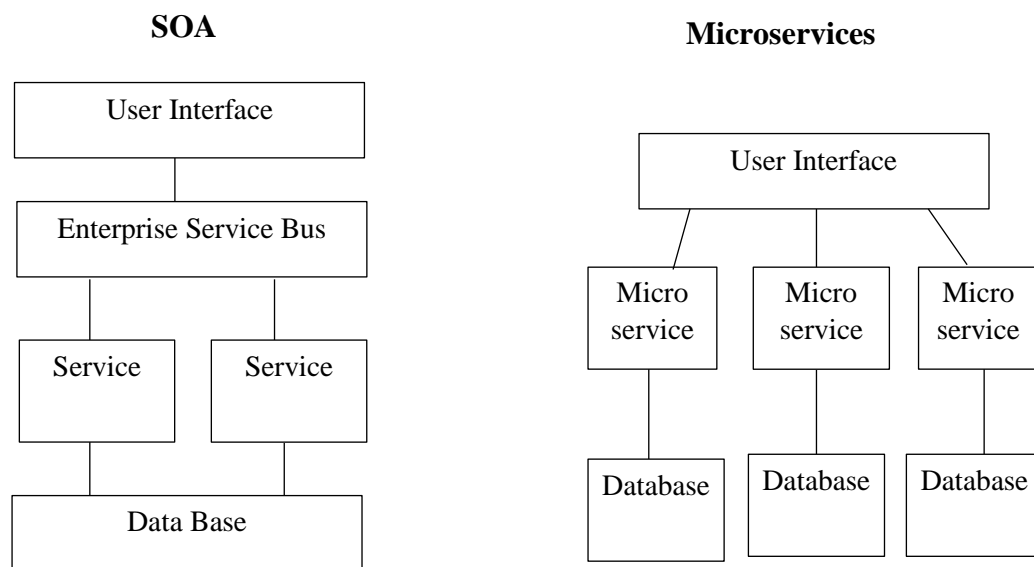


Figure 3: SOA vs Microservices Communication (Wittmer, 2021)

Wittmer (2021) further explains that microservices and SOA also differ in the ways they communicate. Microservices uses smart endpoints (communication logic is part of the

service) and dumb pipes (a messaging system that doesn't have any business logic). Conversely, SOA communicates via API (specified by Web Service Description Language (WSDL)) and or Enterprise Service Bus (ESB). XML based WSDL is considered as a heavy weight that embraces Simple Object Access protocol (SOAP) for communication and has a directory (UDDI) for available services. ESB provides a single point where communication messages are orchestrated which on the flipside can become a single point where communication fails or happens slowly (Indrasiri & Siriwardena, 2018).

### **2.3.3 Microservices Design Principles**

According to Chandramouli (2019), Ziade (2017), Pachghare (2016) and Newman (2015), microservices have a number of design principles; i) an individual microservice should allow scaling, upgrading, replication and deployment independent of other services, ii) A microservice should have a single limited responsibility, iii) The microservices should be stateless and fault tolerant, iv) Modularity: Each microservice represents a logically cohesive, lightweight and independent business functionality with well-defined boundaries. By design, microservices are highly granular, and independently built and deployed, v) Stateless: Microservices provide stateless communication between the client and the server and vii) Microservices should communicate with each other using network calls known as smart endpoints and dumb pipes with preference to REST over HTTP.

### **2.3.4 Microservices Decomposition Patterns**

According to Shivakumar (2019) in decomposing an application into microservices, loose coupling, high functionality coherence and optimum granularity of services is required. A number of ways of decomposing exists, some of them are;

**Decomposition based on business capability:** Create microservices based on business capabilities. For instance, in an e-commerce solution, we can create services based on business capabilities like service management, product promotions and order management

**Decomposition based on sub-domain:** The sub-domains of the core domain (business) are identified and microservices created based on that. For instance, the order management domain has sub-domains such as product catalog and inventory management.

**Decomposition based on transaction:** develop microservices based on the main transactions of the application. For instance, the main transactions of an e-commerce application are check-in/login, backup, checkout and search; we can create microservices for these transactions (Indrasiri & Siriwardena, 2018).

**Decomposition based on resources:** We can create microservices based on nouns or resources and define the operations. For instance, in an e-commerce solution, ‘products’ is a resource and we can define and then list all products (GET /products), query particular product (GET /product/{1}), delete a product (DELETE / product/{}), insert product (PUT /product/{}) (Shivakumar K. S., 2021).

### 2.3.5 Microservices Integration Patterns

Integration patterns describes the optimal ways to invoke multiple microservices, microservice invocation sequence, data and resource security, data transformation and responses for different clients (Lewis & Fowler, 2014). Newman (2015), notes that database integration should be avoided, choreography should be preferred over orchestration and REST should be considered over RPC for request/response integration. A number of integration patterns are in use currently including;

**API gateway pattern:** An API gateway provides a centralized access point for invoking a microservice handling security (such as authentication, authorization), governance (such as logging service, monitoring service), request routing, load balancing, protocol transformation, performance management, data transformation and the aggregation of responses from multiple services (Pachghare, 2016) and (Shivakumar K. S., 2021).

**Aggregation pattern:** When a single microservice needs responses from multiple microservices, a composite service can take the responsibility of aggregating the response.

**User Interface composition pattern:** The end user interface layer is laid out into various sections, which individually invokes the corresponding microservice asynchronously.

### 2.3.6 Security in Microservices

In monolithic web applications, authentication happens with a login form, and once the user is recognized, a cookie is set and used for all succeeding requests (Ziade, 2017). A

paper written by Shaik and Mane (2017) discloses Oauth 2.0 as the most popular protocol for microservices verification and approval mechanism. The core idea of Oauth 2.0 is that a centralized service is in charge of authenticating a caller, and can grant some access in the form of codes or tokens called keys. The tokens can be used by users or services to access a resource, as long as the service providing that resource accepts that token.

### **2.3.7 Microservices and Containers**

According to Douglis and Nieh (2019), container technologies like Docker and Orchestration systems like Kubernetes are the common infrastructure that supports microservices. As Operating System virtualization platform, containers allow multiple services to run in a single operating system and dynamically provision runtime resources. Each microservice can be packed as a docker container (2019) using a docker file.

Docker has Docker Swarm an equivalent of Kubernetes container orchestration technology. Both platforms have embraced each other and are used to schedule, automate, deploy and scale containerized applications. They use YAML; a human-readable digestible data-serialization language to configure and deploy multiple containers.

### **2.3.8 Scaling in Microservices**

Microservices systems can be scaled horizontally or vertically. An application whose architecture can be scale by adding more runtime instances of its processes is said to be horizontally scaled. This is usually achieved by virtual machines or containers which can be configured to autonomously add more application instances to address system performance needs. Vertical scalability comes in where a systems' hardware resources are added e.g. adding more memory and storage space, adding CPUs, adding input and output devices in the name of improving system performance (Bradley, 2021). Vertical scalability tends to be more expensive and has a finite scope of improvement compared to horizontal scalability that has a lower risk of system downtime and hardware failures.

### **2.3.9 Challenges faced while using Microservices**

Many organizations are used to the old-fashioned three-tier monolithic applications that comprises of application tier, business tier and database/ storage tier. Most of them are

transitioning to the popular microservices architecture and are facing difficulties in tearing down a large software application or even running microservices systems (Sengupta, 2021). They are struggling to determine: each microservices size, optimal boundaries and connection points between each microservice and the framework to integrate the services.

Security is a challenge as deployment of microservices is often in distributed environments which are security vulnerable points. This can lead to increased risk and loss of control and visibility of application components. Each microservice communicates with others via various API layers, making it even harder to test for these vulnerabilities in cloud environments. Due to its distributed framework, setting up access controls plus permissions and administering secured authentication to individual services poses not only a technical challenge but also increases the attack surface greatly (Chandramouli, 2019).

The testing phase of any software development lifecycle (SDLC) is increasingly complex for microservices-based applications. Given the standalone nature of each microservice, you have to test individual services independently. Worsening this complexity, development teams also have to factor in integrating services and their interdependencies in test plans when they do some end-to-end tests while deploying the application (Ziade, 2017). Luckily, there are now many tools to facilitate deployments of applications that are built with several components to help in the success and adoption of microservices.

Communication is also an issue as independently deployed microservices act as tiny standalone applications that must interconnect with each other (Ziade, 2017). Inter Process Communication (IPC) is applied for the communication among the services. Deciding a better communication mechanism and messaging protocols remains a challenge.

Data storage and sharing amongst services also possess a challenge. As a principle, services should be loosely coupled and independent of each other thus the emphasis on the principle of each service should have its own database. Avoiding duplication of data while achieving isolated microservices emerges as one of the challenges experienced in designing microservices-based applications (Ziade, 2017).

## **2.4 Review of Related Systems**

### **2.4.1 Monolith Systems**

Juhana, et al (2017) designed an easily accessible Electronic Portfolio application that automates students internship operations. The application documents students activities while on internship and enhances students, trainer and coach online consultations. The system is considered to be interactive, user-friendly and aids in monitoring/supervising students on Internship. Deploying the application as a monolith that contains tightly coupled components makes the system harder to scale and difficult to maintain and accommodate the ever changing user requirements.

Hasti, Lesari and Gustiana (2019) developed a web-based internship application that offers automated solutions to the difficulties experienced at each stage of the internship process which includes student's registration, applying for internship opportunities, selecting a supervisor, internship assessment process and appraisal of student's internship reports. Object oriented approach was used to develop the prototype. User's evaluation of the system indicated that it minimized errors in the process of managing internship and provided timely information. The authors proposed future studies to be done to integrate the system with other systems so that the process of making reports is simplified.

A virtual conference paper written by Hymet and Arbana (2020) about the role of Internship management systems in improving the relationship between stakeholders, recommended a web-based internship dynamic platform based on the Model View Controller (MVC) architecture and framework. The argument was that the MVC framework provides components separation, reusability, storage and independence. The new system offered simplified code development process, easier maintenance and more resilience in terms of fault tolerance. The application aimed at reducing manual activities and improving communication amongst stakeholders. The resultant system incorporated storage of intern's records in a database, checking of students internship progress and created synergy between students, industry and educational institutions. Although MVC framework divided the code into three components; Model, View, and Controller, it was still a monolith with a single run time environment (Hoffman, 2021).

### **2.4.2 SOA Systems**

Girsang, Jafar and Fajar (2018), designed a SOA based project management system that integrated the software development stages of requirements elicitation, requirements analysis, system design and implementation plus the operations of a company engaged in Information Technology Consulting. SOA was the preferred approach as it has the benefits of establishing reusable services with good integration mechanism, efficient application development process and increasing collaboration between the developers and the stakeholders of the application. The report recommends development of the remaining business services and be integrated to the new system by bypassing the ESB to make the company reap benefits of software scalability and agility.

The need for a flexible reusable system to adjust to continuous customer needs in a Toyota automotive company production system drove Nugroho and Fajar (2019) to design a system based on SOA. This architecture integrated two systems i.e. production system and production support system that were based on different technology platforms, using RESTful web services. XML and JSON were used in data sharing. This solved communication problem between the two platforms, reusability of code was achieved and system maintenance become easier too. Centralizing of data storage was seen as a hindrance to achieving loosely coupled and independent services.

### **2.4.3 Microservices Systems**

Shaik, Ramkumar, Hameed and Mohammed (2019), proposed a microservice based architecture model for Oman's National Health Record system that was initially designed as a monolith. They acknowledge that microservices is a solution for a monolith system that has a growing code base. To solve the complexity of large code base, loosely coupled services were introduced to tap into distributed architecture benefits like high availability, easy scalability, heterogeneity, interoperability, openness and reusability. The new system was built as a cluster of microservices organized around business capabilities that were deployed independently. The system decentralized the old systems data and the team realized reduced deployment time for each service. The new model services were designed to be consumed through APIs using REST protocol. An API gateway was implemented between clients and the microservices cluster to act as a single entry point into the services



provided by the application. The API gateway was adopted by the team because its implementation structures enhanced functions such as authentication, authorization, load balancing, cache management and service level agreement (SLA) management.

Wu, Wan and Zhu (2018) presented a journal article detailing how they designed a Microservices based Students WeChat system for Huaiyin Institute of Technology. The article outlined shortcomings of monolith system as having a large code base with single runtime environment, reliance on single technology stack and difficult to adapt and incorporate new customer needs as the motivation to use microservices. They also argued that Microservices were selected because they used lightweight communication protocols like REST and HTTP unlike SOA which largely uses web services protocols like SAOP and WSDL to interact. SOA was viewed as an avenue to achieving multiple systems integration whereas microservices decomposed an application into multiple loosely coupled, distributed and autonomous services and allowed systems integration. The new system was divided into four independent service to facilitate maintainability and scalability. The services were exposed as RESTful API interfaces that communicated using JSON messaging format. Docker containers were incorporated in deploying the individual services to achieve horizontal scalability autonomic computer system.

## **2.5 The Proposed Model**

The proposed model handles industrial attachment operations by automating the processes of attachment placement, monitoring & supervision, assessment & appraisal and data storage. The model provides a coordinated way of students accessing the available attachment opportunities, applying their preferred attachment places and allows admission of students to their attachment places. Admitted students can record their daily learning activities electronically and upload reports. The system also acts as a communication platform between various stakeholders. The system allows the Industry trainer, coach & mentor, to track and supervise the progress of the student at the work environment.

The new microservices system also allows the Industrial Liaison Officer to assess the students on attachment by allocating assessors to evaluate the students in the field and grade the final report of the students. The system has a report generation service to print students' industrial attachment nominal roll and a student can also print his/ her final score.

## **CHAPTER THREE: RESEARCH METHODOLOGY**

### **3.1 Introduction**

Descriptive research design was employed in coming up with the requirements of the new system. Convenience sampling technique was used to select KTTC and NYSEI to represent TVET institutions where stratified sample was used to draw a sample from which data was collected. Data collection was by questionnaires & documents review methods and the collected data was represented and analyzed using statistical methods. The microservices model was developed by agile application development methodology using python frameworks and JavaScript library. Docker containers were used to deploy and scale the model. Nginx was configured as load balancer/reverse proxy and Locust load testing tool was used to test system performance.

### **3.2 Research Design**

Research design created a blue print within which data collection, data measurement and data analysis was conducted (Kothari, 1985). Descriptive research design which systematically described the state of affairs of the current system was used. The design answered the what, when, where and how questions concerning the study. Descriptive research design allowed gathering of in-depth data in a participant's natural environment.

To successfully achieve the goals of research, a quantitative research approach was used where numerical data was collected and analyzed (Bhandari, 2021). Quantitative research allowed standardized data collection and generalizing of research findings. The data produced was numerical and was analyzed using mathematical and statistical methods.

### 3.3 Developing the Model

#### 3.3.1 Software Development Methodology

Agile Software Development Methodology was adopted to develop the microservice based model. Agile methodology is fast, flexible, error-proof development methodology that focuses on iteration to come up with software products (Ihor, 2021).

Agile has the advantage of creating a collaborative culture amongst teams allowing them to work together and releases high-quality products making it suitable for microservices development. The methodology has an ongoing testing process that allows continuous change to improve the quality of the software product. The development cycle was split into six stages; starting from concept identification, followed by inception and analysis, then iteration/construction phase, release phase and finally production and retirement phase (Brush & Silverthorne, 2021).

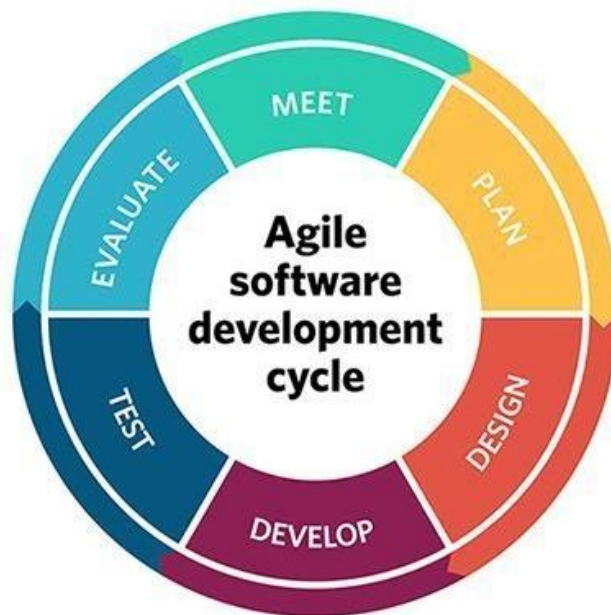


Figure 4: Agile software development cycle (Brush & Silverthorne, 2021).

At the iteration/construction phase is where user requirements were transformed into a working software artefact which was a basis for customer feedback. Multiple iterations came up after another which led to improved quality software model that captured user's expectations.

Inside an iteration flow, the following steps were followed; i) user requirements were defined based on product backlog (list of items to be done), ii) the requirements were implemented, iii) testing which involved quality assurance and user training was done, iv) the working product was delivered and integrated to the entire system, v) stakeholders and customers experience was recorded to be part of the next print requirements.

### **3.4 System Requirements Gathering**

#### **3.4.1 Target Population**

Kumar R (2011), defines the study population as the people (individuals, groups and communities) from whom information is collected. The targeted population included students, trainers and industry representatives involved in industrial attachment training docket of TVET institutions. A representative sample was drawn from the population to take part in the investigation.

#### **3.4.2 Sampling Procedure**

Convenience sampling and stratified sampling technique were used. With convenience sampling the researcher simply selected respondents who were convenient for the study (Oates, 2006). Whilst the technique was treated with caution, convenience sampling is easy to use, has fewer rules to follow and the study achieved the sample size needed in a relatively fast and inexpensive way. Kenya Technical Trainers College (KTTC) and NYS Engineering Institute (NYSEI) were selected to represent TVET institutions.

Using stratified random sampling method, members of the two institutions who had experience with industrial attachment were divided into two groups; i) students and ii) teaching staff. This was to remove biasness and ensure every subgroup is well presented in the sample to achieve precise conclusions (Kothari, 1985).

#### **3.4.3 Sample Size**

Kothari (1985) defines sample size as the number of items to be nominated from the universe to make a sample. The sample size was calculated using the formula:

$$n = \frac{z^2 P(1-P)}{d^2} \text{ (Ali , 2014),}$$

Where:

n: was the desired sample size

z: the standard normal deviate usually set at 1.96 (which corresponds to the 95% confidence level)

p: the proportion in the target population to have specific characteristic. In this case 50% (or 0.50) was used there being no estimates of the target population

d: was the absolute precision or accuracy, normally set at 0.05

on calculation:  $n = \frac{1.96^2 \cdot 0.5(1-0.5)}{0.05^2}$

$$n = 384, \text{ the desired sample size}$$

### **3.5 Data Collection Instruments**

#### **3.5.1 Questionnaire**

A questionnaire consists of a set of questions/prompts whose aim is to collect information from the respondents (Oates, 2006). The reply from the questionnaires provided information that was analysed and interpreted as results. Generalisations about the actions or views of the larger population were made based on sample results.

The choice of the questionnaire was guided by the fact that the instrument could be mailed through the internet and collectively administered or administered in public platform or online platform (Kumar R. , 2011). This meant the instrument was more convenient and less costly to administer since the intended respondents could easily be reached.

The questionnaire, which was largely closed ended for easier analysis of the collected data was administered on online platform in line with COVID-19 protocols and also in person to the respondents. The respondents were briefed and guided well in filling. Closed questions which are factual in eliciting information were designed to be clear and easy to enable respondents to take a shorter time in filling (Kumar R. , 2011).

### **3.5.2 Secondary Data**

Oates (2006) acknowledges that documents or existing literature is a source of data just like interviews, questionnaires, observations and other data collection instruments. Organizational documents like sample attachment requests letters, sample attachment posting letters, attachment log books, assessment forms and recommendation letters were obtained from the two TVET institutions for the purpose of data collection.

Publications from academic literature like project reports, journals, conference papers, web documents and software guides about designing and implementing microservices based systems also came in handy. Other documents were obtained by visiting the library or using the web (Oates, 2006). In text citations and reference list were used to acknowledge the contribution of these documents in designing the proposed model.

According to Oates (2006), document based data has a number of advantages, for instance, much of the documents can be obtained quickly, cheaply & conveniently. Documents are always permanent in that other researchers can easily check and scrutinize them thus giving credibility to the data collected.

### **3.5.3 Validity and Reliability**

According to Middleton (2020), how accurate a method measures what it is intended to measure is called validity. High validity means results corresponds to real characteristics in the universe. Reliability indicates the consistency a method measures something. The questionnaires were pre-tested first for validity and reliability with six colleagues at workplace and 18 students before being used to collect data. Errors were identified, corrected and thereafter used for data collection.

## **3.6 Data Collection**

### **3.6.1 Administration of Questionnaires**

The study administered 384 questionnaires to 192 respondents from KTTC and 192 respondents from NYSEI. The response constituted 176 filled questionnaire which were returned of which 122 were face to face and 54 were online questionnaires giving a 46%

response rate. This was done from a sample of teachers, and students from KTTC and NYSEI between 7<sup>th</sup> May and 28<sup>th</sup> May 2021.

### 3.6.2 Documents Review

Organizational documents which included sample attachment requests letters, sample attachment posting letters, attachment log books, assessment forms and recommendation letters were obtained from the 2 TVET institutions for the purpose of data collection between 7<sup>th</sup> May and 28<sup>th</sup> May 2021.

### 3.7 Data Analysis

#### 3.7.1 Responses from Students:

The study sampled students who had completed their industrial training exercise. A total of 142 responses were received and their data was analyzed as follows.

- i) The students listed the following as the requirement for them to proceed for Industrial attachment

Requirement	Frequency	Percentage
Recommendation from the Head of Department	134	94%

Table 1: Requirement before proceeding for Industrial Attachment

- ii) Challenges faced while sourcing for industrial attachment places

Challenge	Frequency	Percentage
Difficulty in identifying Attachment Places	109	75%
Poor tracking of outcomes from the applied sources	118	82%

Table 2: Challenges faced while sourcing for Industrial Attachment

- iii) Challenges faced by students on industrial attachment

Challenge	Frequency	Percentage
Lack of official Communication link	123	86%
Manual way of record Keeping	138	97%

Table 3: Challenges faced by students on industrial attachment

iv) Industrial Attachment activities

The following was identified by the students as activities done during industrial attachment:

<b>Activity</b>	<b>Frequency</b>	<b>Percentage</b>
Daily recording of learning activities	<b>142</b>	<b>100%</b>
Supervision/Mentoring by Industrial organization staff	<b>141</b>	<b>100%</b>
Assessment by industry supervisor and college assessors	<b>139</b>	<b>98%</b>

Table 4: Industrial Attachment Activities

v) The documents awarded on completing industrial attachment

<b>Document</b>	<b>Frequency</b>	<b>Percentage</b>
Clearance from the establishment	<b>142</b>	<b>100%</b>
Recommendation letter	<b>139</b>	<b>98%</b>

Table 5: Documents Awarded on Completing Industrial Attachment

### 3.7.2 Responses from Lecturers

The study received 34 questionnaires as responses from the ILO and teachers in order to incorporate their views in the development of the model.

i) Challenges faced by Trainers while assessing students on industrial attachment

<b>Challenge</b>	<b>Frequency</b>	<b>Percentage</b>
Manual Keeping of Documents	34	100%
Poor means of communication between trainers, students and the attached institutions	34	100%

Table 6: Trainers Industrial Attachment Challenges

ii) Documents used by the industrial attachment department

<b>Document</b>	<b>Frequency</b>	<b>Percentage</b>
Attachment Request Letter	34	100%
Log Book	34	100%



<b>Document</b>	<b>Frequency</b>	<b>Percentage</b>
Recommendation Letter	34	100%
Assessment Form	34	100%

Table 7: Documents used by the Industrial Attachment

iii) Industrial Attachment Assessment Parameters

<b>Item</b>	<b>Weighting</b>	<b>Frequency</b>	<b>Percentage</b>
Up to date comprehensive well organized Log book	10	34	100%
Industry Supervisors Score	40	34	100%
Academic Institutions Assessor Score	20	34	100%
Final Attachment Report Score	20	34	100%

Table 8: Industrial attachment Assessment Parameters

vi) Technologies used by the industrial attachment department in its operations

<b>Technology</b>	<b>Description</b>	<b>Frequency</b>	<b>Percentage</b>
Email	Communication	<b>33</b>	<b>99%</b>
Ms Office	Creating Documents	<b>34</b>	<b>100%</b>
WhatsApp/Instagram	Communication	<b>33</b>	<b>100%</b>

Table 9: Technologies used by Industrial Attachment Docket

iv) Measures to improve operations of Industrial Attachment Docket

<b>Item</b>	<b>Frequency</b>	<b>Percentage</b>
Introduction of an interactive, high available system	34	100%

Table 10: Measures to improve Industrial Attachment Operations

### 3.7.3 Document Reviews

The study evaluated and gathered requirements from existing document's that are used in the attachment process. Sample of Attachment request letters, attachment placement letters, log books, assessment forms and student log books provided important information in coming up with the system requirements.

## **3.8 System Requirements Specification**

### **3.8.1 System Feasibility**

Feasibility study was done to measure how beneficial or practical the system will be to TVET institutions once developed. This confirmed that the project was worth to develop and implement and the benefits outweighed the expenses;

#### **i) Operational Feasibility**

This clearly revealed that the new system would solve the problems outlined and users were ready and willing to embrace the new technology. Users of the system were literate and already using other systems in place like student management information system. The institutions have information technology staff ready to run and maintain the system.

#### **ii) Technical Feasibility**

It was established that the hardware and software resources for developing and deploying the new model were readily available. TVET institutions have the necessary hardware and software needed. The institutions have internet connection to acquire open source software from the internet. Supporting infrastructure for the system is sufficient for deployment in Kenya because of good internet connectivity.

#### **iii) Economic Feasibility**

This was conducted to determine the cost-effectiveness and cost benefit analysis of the project. Development costs, installation costs and operational costs were estimated and the benefits projected. It was concluded that once the system is implemented and operational, more benefits were to be realized.

### **3.8.2 System Analysis**

The data collected was then transformed into system requirements and the proposed features were refined into data flow diagrams to best capture the functional requirements of the system.

## **i) Functional Requirements**

This is a level where system subtasks were identified and the relationship among them defined. This ensured the subtasks were transformed into subsystems which collectively acted to form the system. The following user's functions were identified:

### Admin

- Admit students due for attachment
- Add Users of the system
- View progress of students on industrial Attachment
- Create Attachment Sessions

### Students

- Register for industrial attachment
- Check available training opportunities
- Apply for industrial attachment
- Record daily industrial training activities
- Upload an attachment report
- Check the industrial attachment score

### Institutions Industrial Liaison Officer (ILO)

- Admit students due for attachment
- Maintain database of training partners
- Add Users of the system
- View progress of students on industrial Attachment
- Create Attachment Sessions

### Industry Supervisor

- Register in the system
- Review students' progress
- Score attachees/students

The Assessor / Lecturer

- Register into the system
- View student's allocation details
- Add evaluation details
- Remark on students daily logs

## ii) Non- Functional Requirements

Security

- The system must allow only authorized users to access services.
- The system is able to grant access to authenticated users based on their defined roles and permissions.

Efficiency

- System activities will be ready to be viewed in time. The tasks will execute within optimal time limits.

Usability

- Easy to use system with aesthetically pleasing, clear and consistent user interface

Reliability

- Up-to-date information will be provided to users
- Users will do decision making effectively and produce valuable results

Scalability

- The system should be scalable, highly available and fault tolerant
- Supports loose coupling

Usability

- The user interface should be designed such that screens are similar therefore it will be easy to use, easy to learn and easy to work on.

### 3.8.3 Use Case Diagram

The diagrams are made of actors and use cases enclosed by a system boundary (Dennis, Wixom, & Roth, 2012). They aided in identifying and splitting the functionality of the system. An actor represents various external people or entities that interact with the system.

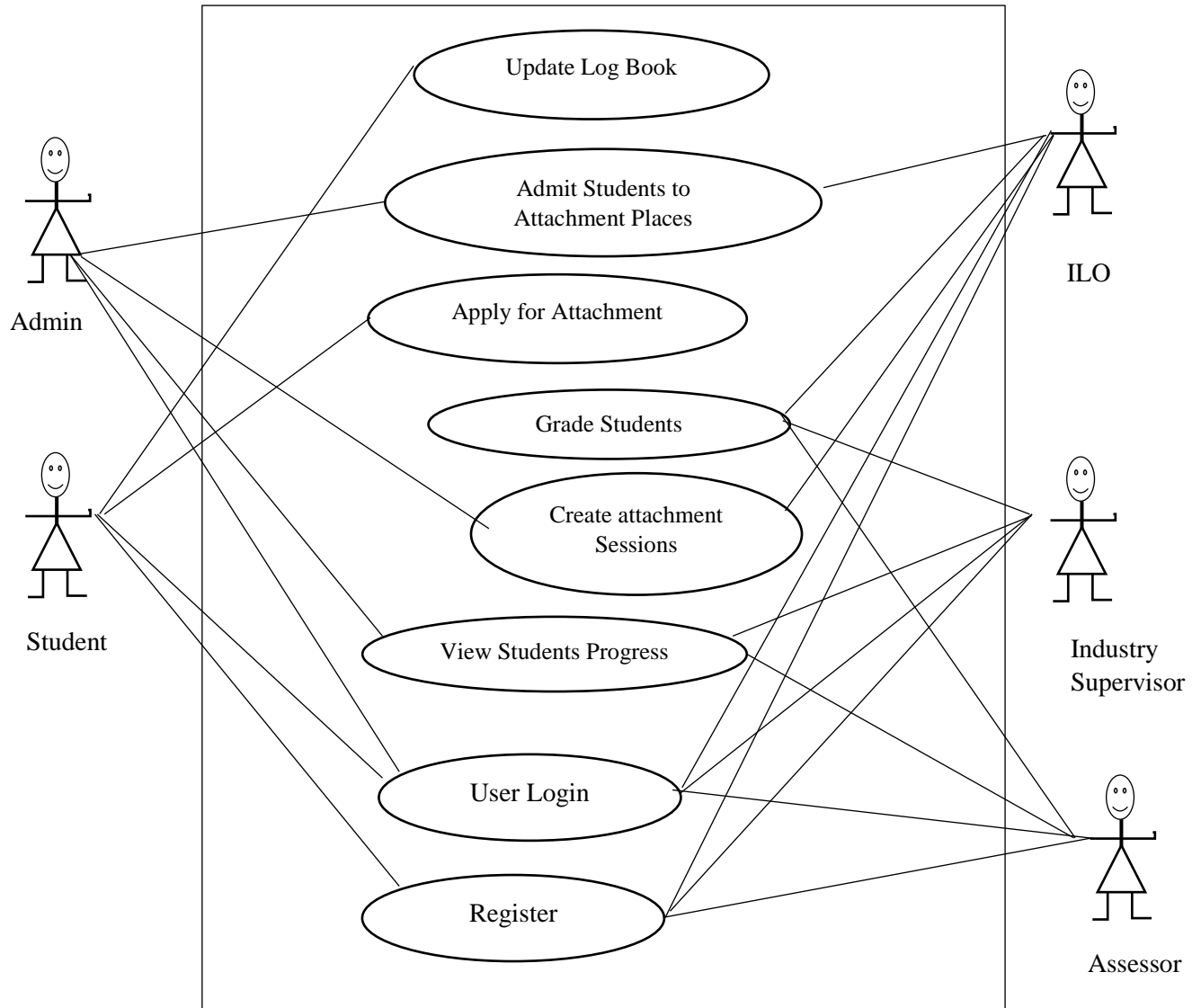


Figure 5: Use Case Diagram

### 3.8.4 Data Flow Diagram

#### i) Context Diagram

A context diagram shows the external agents interacting with the system and the data flowing in and out of the system and the interactions

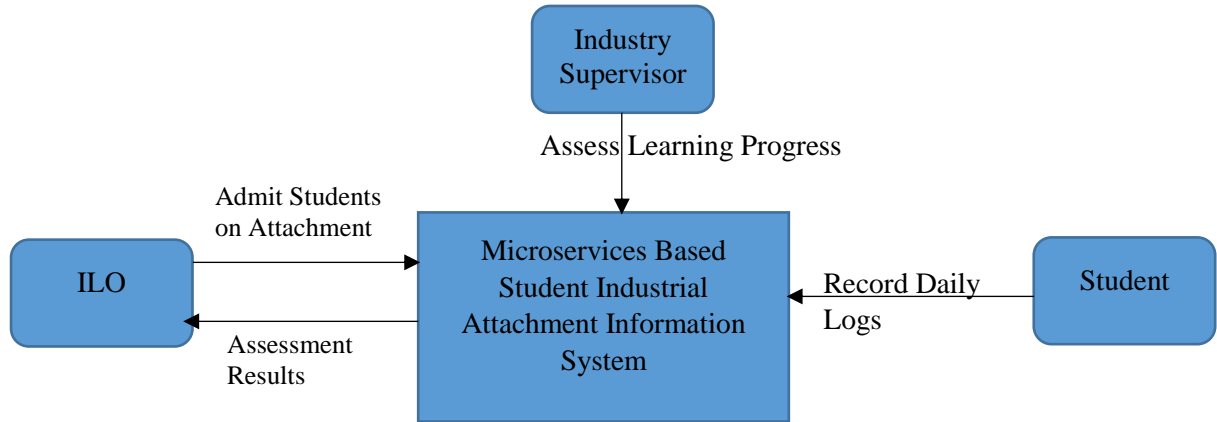


Figure 6: Context Diagram

## ii) Data Flow Diagram Level 1

Data flow diagram level 1 enabled visualizing how data or information is passed between elements of a system. The diagram clearly showed the main sub-processes and stores of data that make up the system.

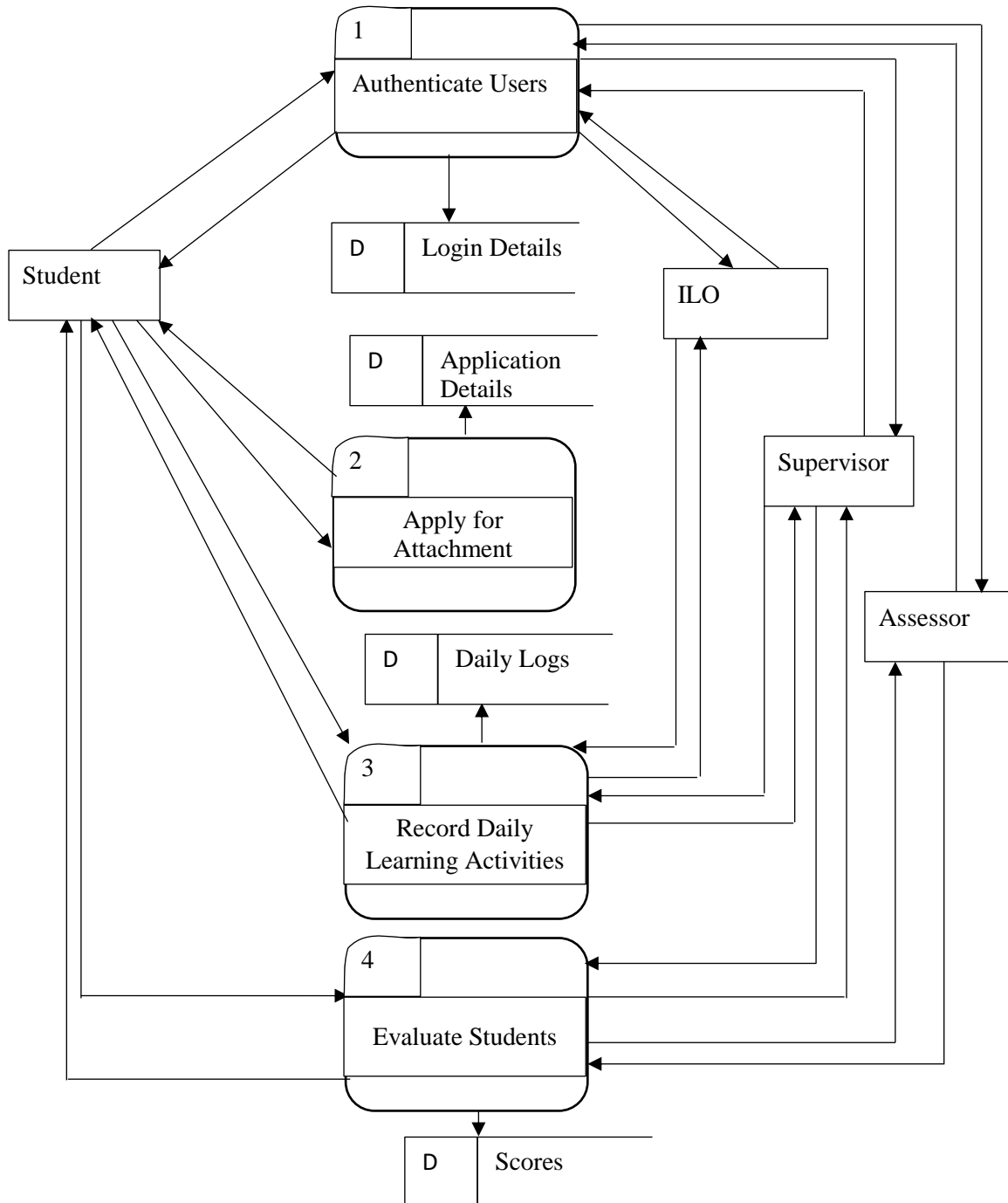


Figure 7: Data Flow Diagram Level 1

### 3.9 System Design

This phase conceived system elements that realized system operation in terms of hardware, software, and network infrastructure (Dennis, Wixom, & Roth, 2012). The different interfaces of the components and the data that goes through the system was also designed. The design activities included coming up with system architecture, data storage design, process design, network design and user interface design.

#### 3.9.1 Architectural Design

This provided system subsystems and depicted how various services interact as shown in figure 8. The system has User interface Microservice implemented as docker image, Security Service, Attachment Microservice implemented as docker image and Reports Microservice implemented as docker image. Security service is implemented as Software as a Service (SaaS) single sign on option from Okta.

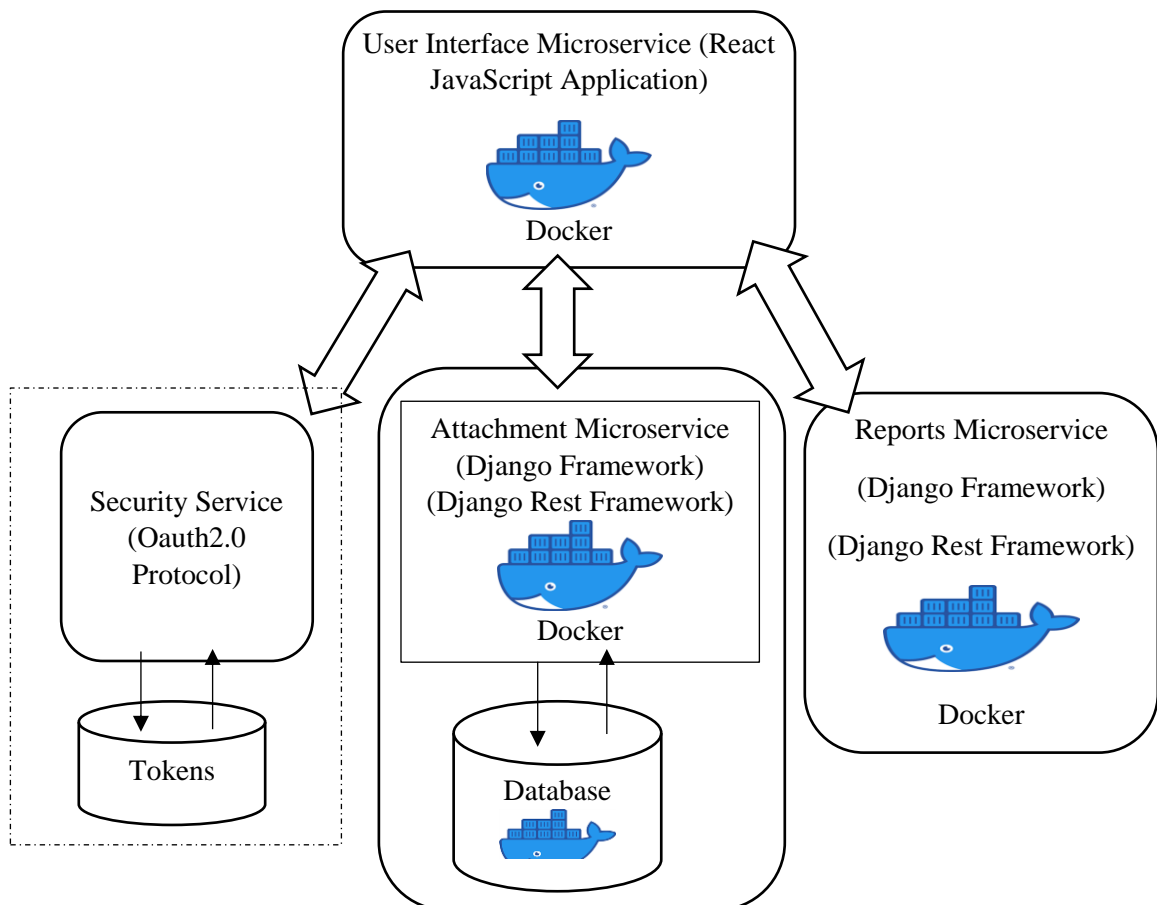


Figure 8: System Architectural Design



### 3.9.2 Sequence Diagrams

#### i) Security Service Sequence Diagram

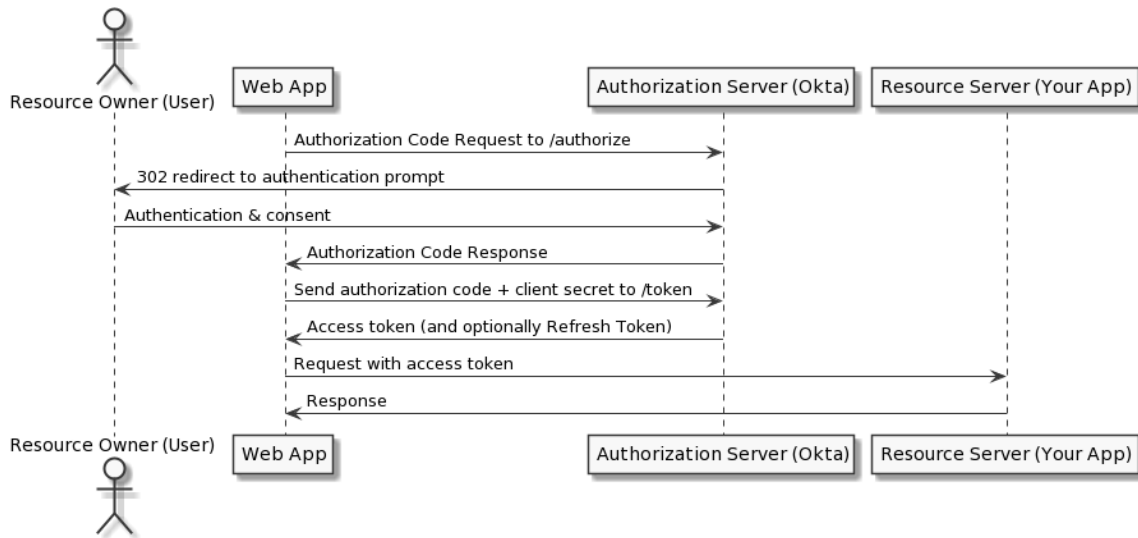


Figure 9: Authentication with Okta Sequence Diagram (Okta, 2021)

#### ii) Attachment Microservice Sequence Diagram

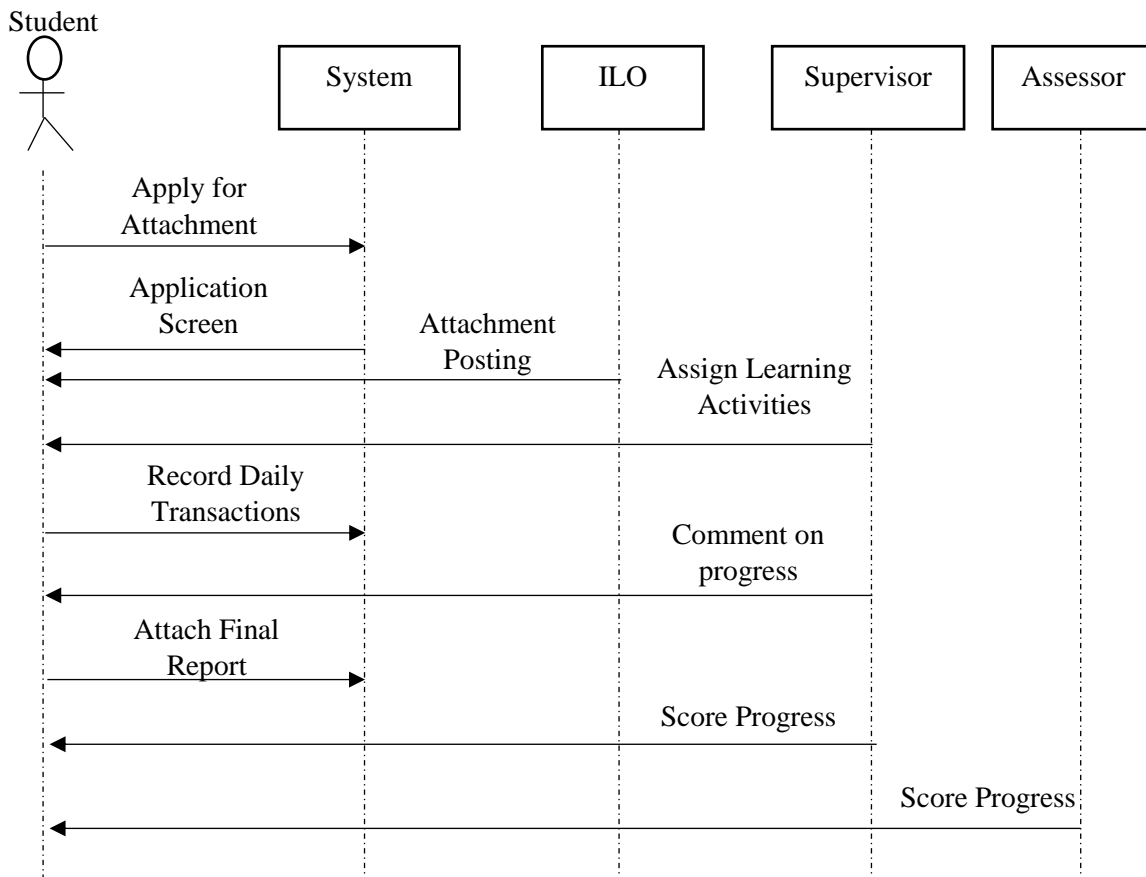


Figure 10: Attachment Microservice Sequence Diagram

### iii) Integration Sequence Diagram

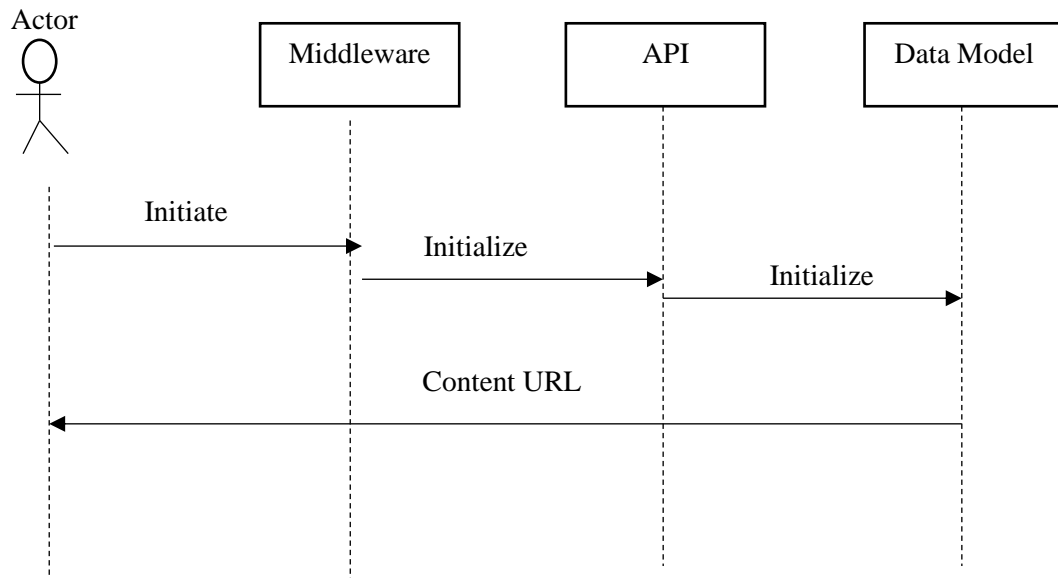


Figure 11: Integration Sequence Diagram

### 3.9.3 Database Design

Database Design included the steps that helped with creating, developing, updating, deleting and maintaining organizations data storage management system (Naeem, 2021). The design produced physical schema and logical schema of the database.

Logical schema employed Entity Relationship modelling that transformed the database requirement through normalization process. Physical model took Object Relational Mapping (ORM) programming technique to come up with a functional database management system.

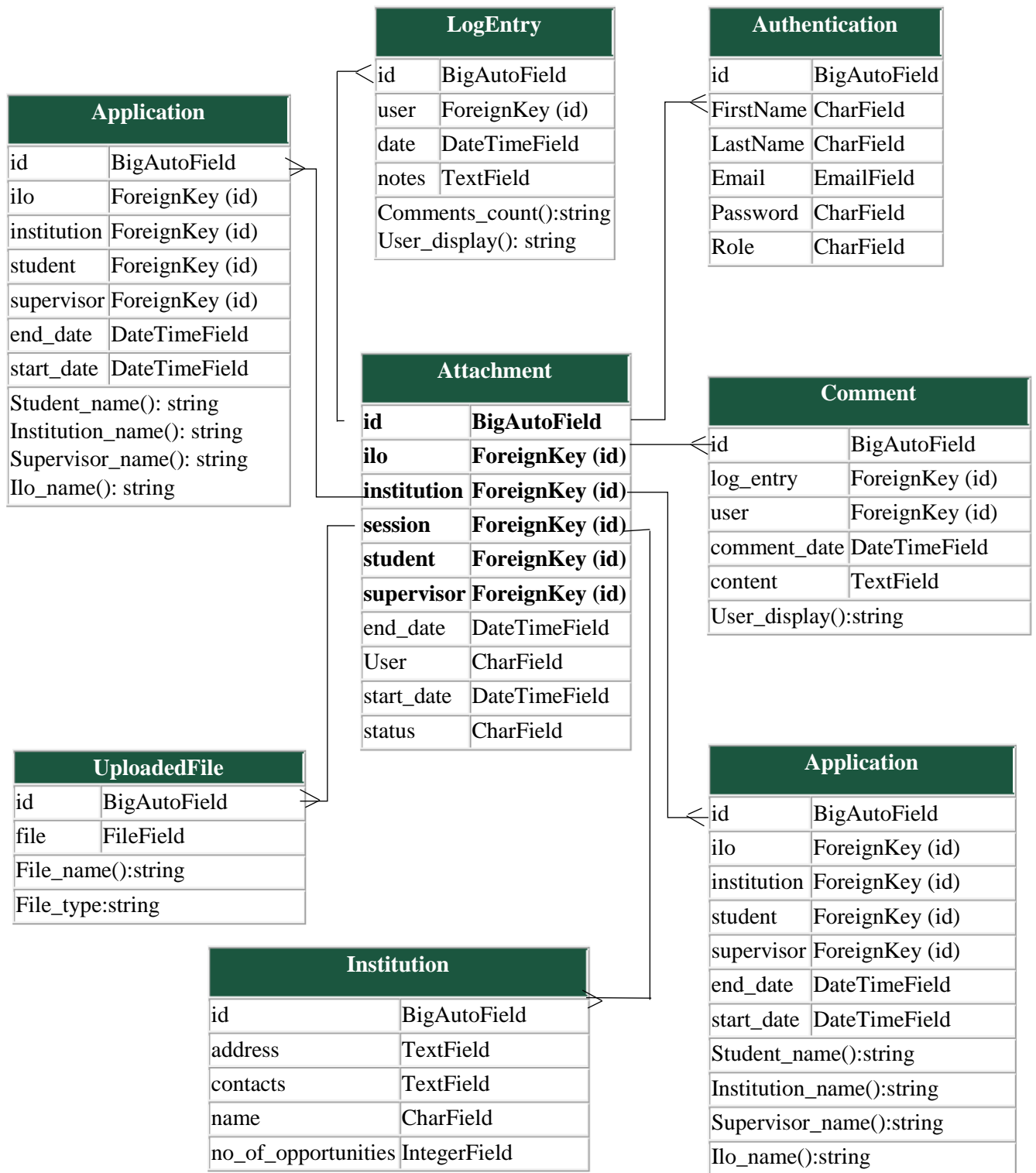


Figure 12: Entity Relational Diagram

### 3.9.4 Program Design

Flowcharts and pseudocodes were used to guide in coming up with clear and concise programming statements. Flowcharts showed pictorial representation of algorithms (Rai, 2021).

#### i) Overall System flowchart

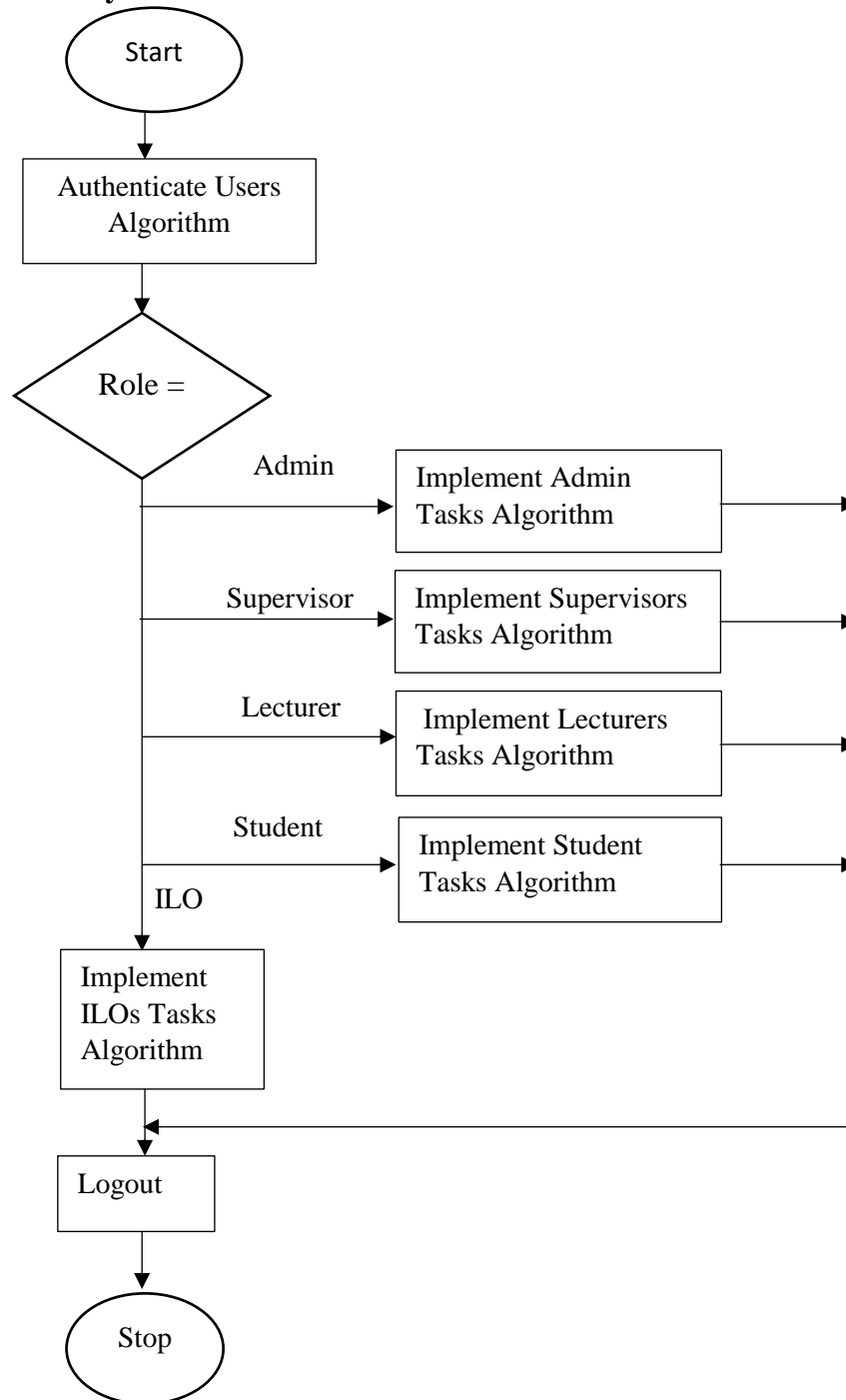


Figure 13: System Flowchart

Pseudocodes as a program design tool were used to describe the flow of logic in the programs' algorithms. Sample of the Pseudocodes used are as below;

**ii) Authenticate Users Pseudocode**

```
Begin
  Display Login Form
  If REGISTERED Then
    Fill Username and Password
    Click Login Button
    If CORRECT username And password Then
      Authorize user based on Role
      Login Successful
      Exit Login Page
      Display User Page
    Else
      Display wrong username and or password
      Display Login Form
  Else
    Display registration Form
    Fill Registration Details
    Save Details
    Display Login Form
  End if
  Exit Login Form
End
```

**iii) Admin Page Pseudocode**

```
Begin
  Display Admin Page
  Display dashboard commands
  Display Logout button / command
  Display Add User Command
  Display Add Attachment Institutions Command
  Display View Attachees progress Command
  Display Attachment Summary Report Command
End
```

**iv) Logout Pseudocode**

```
Begin
  Confirm logout
  Exit all processes
  Exit page
  Display login page
End
```

**v) Student Page Pseudocode**

Begin  
    Display student Page  
        Display Dashboard Commands  
        Display Logout Button / Command  
        Display Apply for Attachment Command  
        Display Add Daily Logs  
        Display View supervisor’s comments  
        Display Add Attachment Report  
End

**3.9.5 User Interface Design (Forms)**

This described how users interact with the system; allowing the users to work thus the need for it to have attractive layout design.

**ILO’s Page**

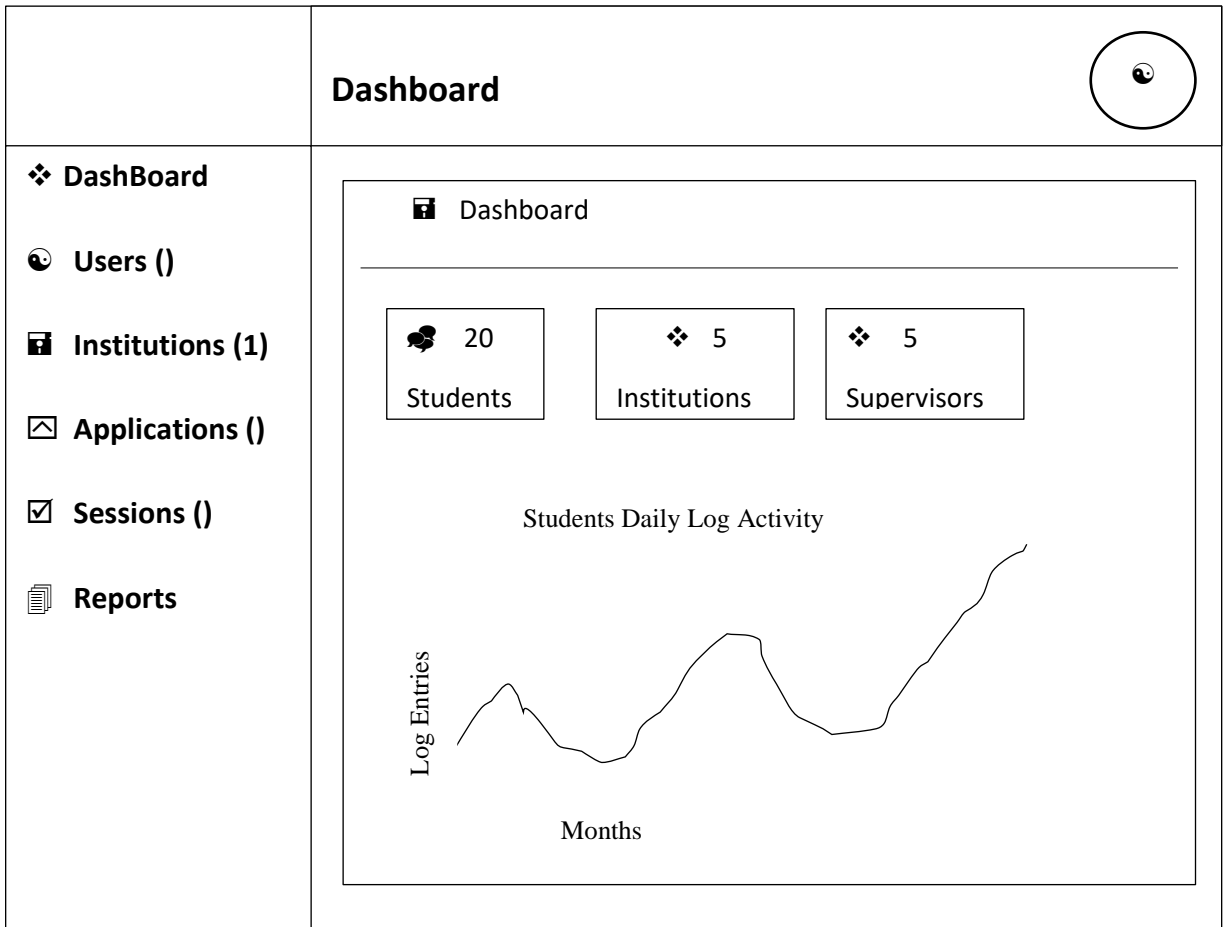


Figure 14: Design of ILO’s Page User Interface

## Admin Add Users Page

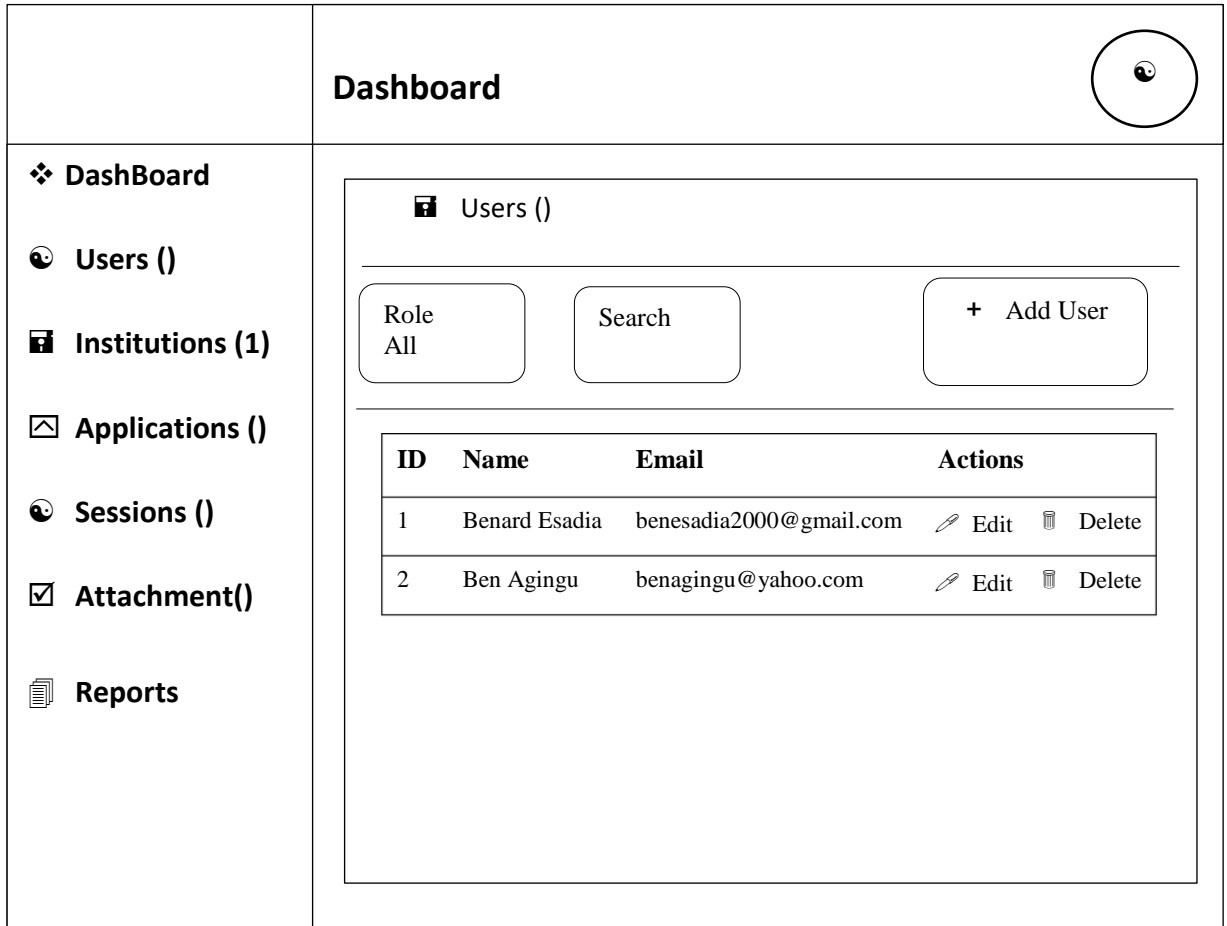


Figure 15: Design of Admin's Add User Page Interface

## Admin Attachment Opportunities

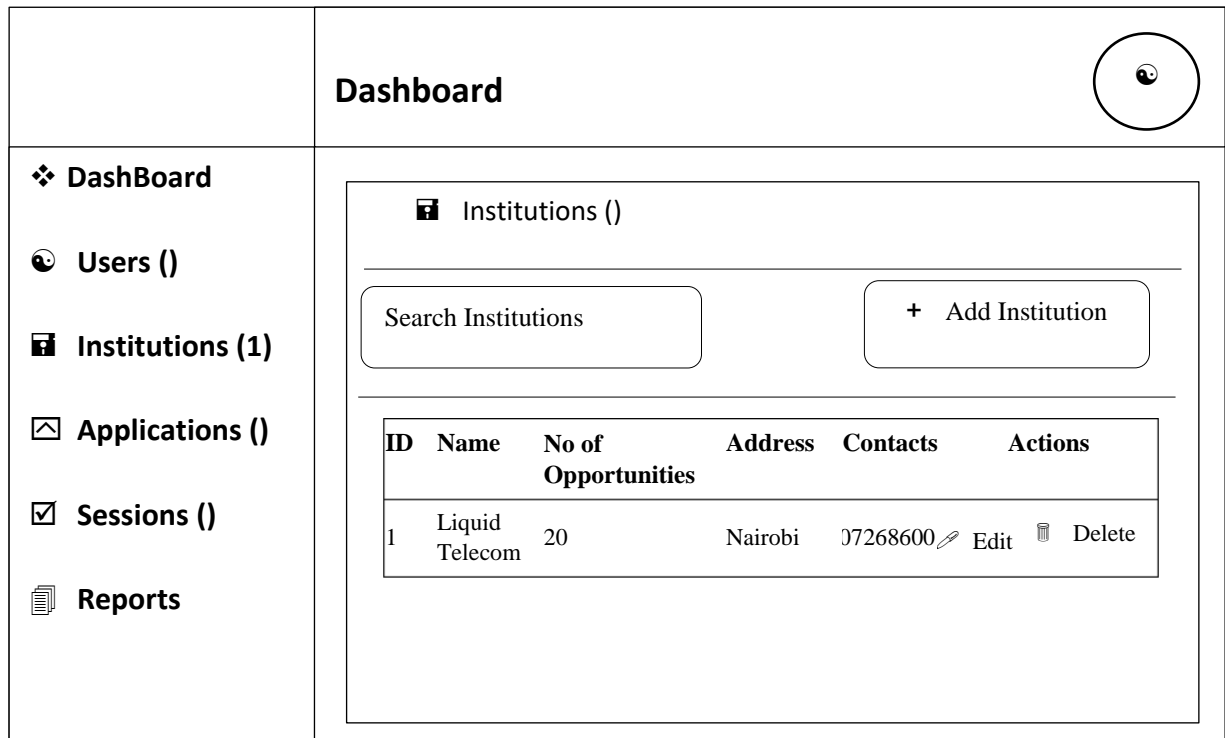


Figure 16: Design of Admin Attachment Opportunities Page User Interface

### 3.10 System Implementation

#### 3.10.1. Hardware Resources

The hardware resources that were used during the implementation are listed below:

- A computer with 8 GB RAM
- Internet Connection

#### 3.10.2 Software Resources

The software requirements are as follows:

- Windows Operating System
- Pycharm JetBrains Programming Integrated Development Environment
- Visual Studio Code as a code editor
- Docker and Docker Compose as container management tools
- Python Django Framework



- Python Django Rest Framework
- React JavaScript Library

### 3.10.3 Programming Tools

#### i) React JavaScript Framework

This is a JavaScript library for building interactive, responsive and lightweight user interface frontends for REST API backend. The user interface consists of classes with methods which are called by the engine when the page is created or updated. ReactJS is based on the idea that the page can be decomposed into basic components, which are called for rendering parts of the page (React, 2021).

#### ii) Python Django Rest Framework

The framework was used to build Web RESTful APIs. An API is a tool that allows one piece of software to interact with another, but not users (Django, 2021). The application endpoints were designed such that when accessed, they return JSON Data.

Serializers converted model instances to Python dictionaries, which are then rendered in various API appropriate formats - like JSON or XML. Figure 17 below explains further.

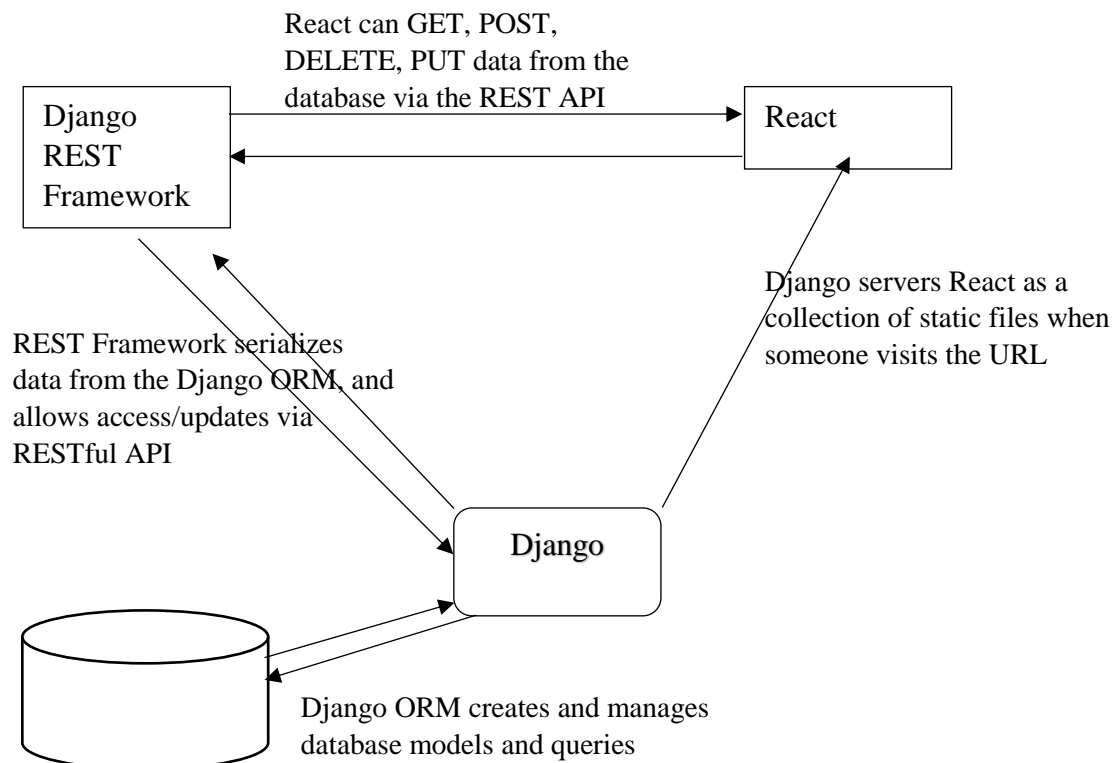


Figure 17: Django, Django Rest Framework, React Interactions

### 3.10.4 Security Service Implementation

Okta as cloud SaaS identity provider was used to provide Single Sign-on to the system microservices (Okta, 2021). The service provided three major kinds of authentication including; i) the authentication API that controls access to other Microservices. The API has user profile registration and password recovery function, ii) The Oauth 2.0 protocol that has authorization rules and permission while accessing an API service and iii) The OpenID Connect that conveys or passes information about verified users to a given web application.

### 3.11 System Testing

#### 3.11.1 System Scalability Testing

Locust as a load testing tool was configured to test the attachment microservice Application Programming Interface as shown in figure 18. The tool supported executing tasks distributed over multiple applications and simulated system users behavior simultaneously. User's behavior was defined in regular python code.

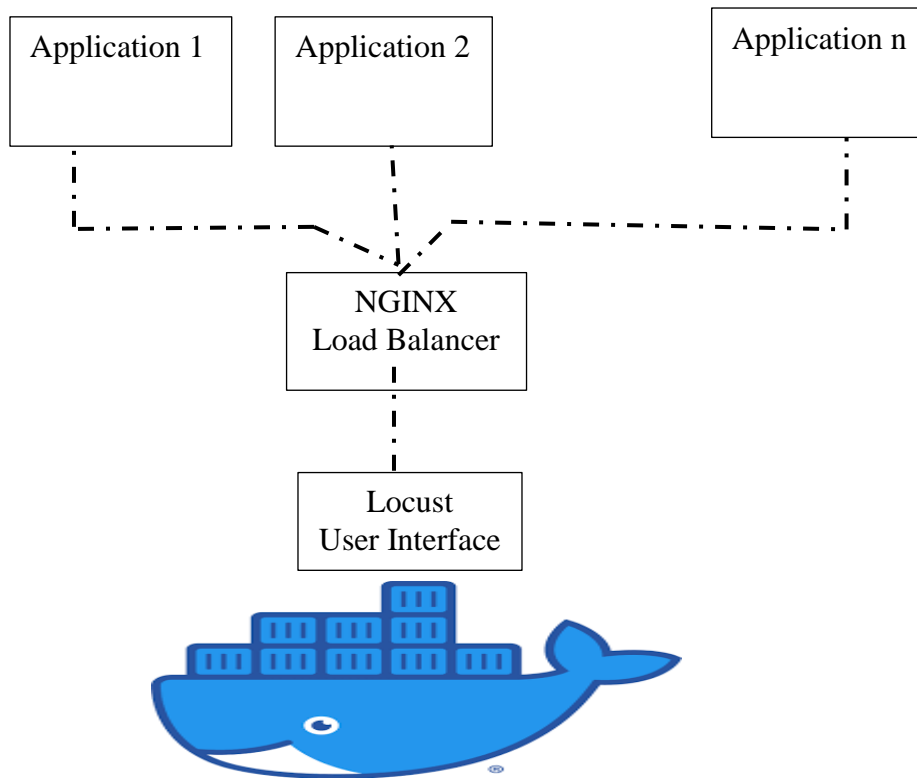


Figure 18: Dockerized Load Balancing Architecture (Maximilian , 2021)

Docker provided an open isolated environment to implement the services as containers (Docker , 2021). Docker's lightweight feature made it easier to manage workloads during runtime by scaling up or tearing down microservices to achieve the expected system performance as simulated by locust load testing tool.

Docker compose allowed exposing the ports for services to an external load balancer called Nginx (reverse proxy). As a load balancer, Nginx used a round-robin setup, wherein workloads were equally passed among the nodes, one after the next. The configuration can be visualized as shown in figure 18 above and the docker-compose YAML file below.

### Docker Compose YAML file

```
version: '3'
services:
  ui:
    build:
      context: ./ui
    depends_on:
      - internsyst-api
    links:
      - internsyst-api:internsyst-api
    volumes:
      - './ui:/usr/src/app'
    command: bash -c 'npm start'
    ports:
      - 3001:3000

  internsyst-api:
    build:
      context: ./internsyst-api
    depends_on:
      - db
    links:
      - db:db
    volumes:
      - './internsyst-api:/usr/src/app'
    command: bash -c
    environment:
      - DB_ENGINE=django.db.backends.postgresql_psycpg2
      - DB_NAME=internsyst
      - DB_USER=internsyst
      - DB_PASS=32e3289u92u3JIJ8E29909II0JH
      - DB_HOST=db
      - DB_PORT=5432
    ports:
      - "8001"

  reports:
    build:
      context: ./reports
    depends_on:
```

```

- db
- internsys-api
links:
- db:db
volumes:
- ./reports:/usr/src/app
command: bash -c
environment:
- DB_ENGINE=django.db.backends.postgresql_psycopg2
- DB_NAME=internsys
- DB_USER=internsys
- DB_PASS=32e3289u92u3JIJ8E29909II0JH
- DB_HOST=db
- DB_PORT=5432
ports:
- "8002:8002"

locust:
build:
context: ./internsys-api
depends_on:
- nginx
links:
- nginx:nginx
volumes:
- ./internsys-api:/usr/src/app
command: bash -c 'locust -f locustfile.py'
ports:
- 8089:8089

db:
image: postgres
environment:
- POSTGRES_DB=internsys
- POSTGRES_PASSWORD=32e3289u92u3JIJ8E29909II0JH
- POSTGRES_USER=internsys
volumes:
- ./postgres-data:/var/lib/postgresql/data

nginx:
image: nginx:latest
volumes:
- ./nginx.conf:/etc/nginx/nginx.conf:ro
depends_on:
- internsys-api
ports:
- "8001:80"

```

### 3.11.2 System Validation Testing

Software Validation is a system testing activity of a given application to ascertain that the intended user requirements and business requirements have been achieved ( O'Donnell,

2020). This is done by verifying system behavior and general features are being exposed as per user expectations.

Using convenience sampling method, seventy-seven actual system users were identified and worked on the system to test and confirm that it meets user expectations. The model was hosted on NYSEI server on Monday 21<sup>st</sup> June 2021 to enable validation testing. Using the institutes office computers and computer laboratory computers the users of the system (Admin, ILO, Students, and Lecturers) were assigned specific tasks to perform using the system. The sample constituted 2 Admins, 1 ILO, 8 Lecturers and sixty-six students.

# CHAPTER FOUR

## 4.0 RESULTS AND DISCUSSIONS

### 4.1 Model Scalability and Load Testing

The first test was done by scaling the system to 3 instances using the command `docker-compose up --scale internsys-api=3`. The test involved allowing the locust interface to simulate 500 users using the system, then followed by simulation of 1000 users using the system, 1500 users using the system, 2000 users using the system, and finally 2500 users also using the system. NGINX; the load balancer/ reverse proxy spawned 10 users per second as shown in figure 19.

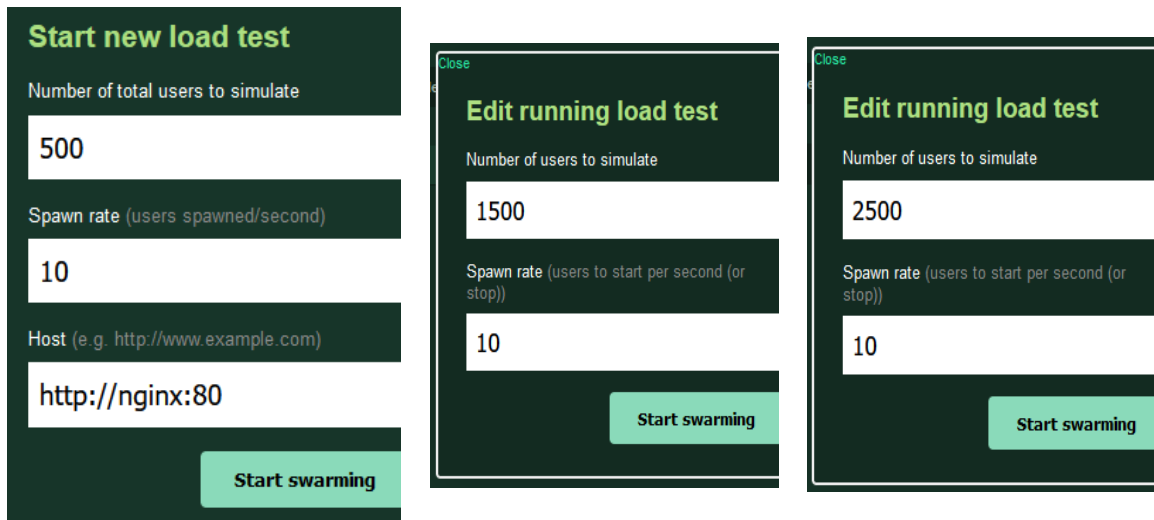


Figure 19: Locust System Load Testing Interface

The second test was done by scaling the system to 6 instances using the command `docker-compose up --scale internsys-api=6`. The locust interface also simulated 500 users with NGINX spawning 10 users at a second. The number of users were also incrementally changed to 1000, 1500, 2000, and 2500 during the test.

The 2 tests results were recorded as shown in the figure 20, figure 21 and figure 22 below;

## 4.1.2 Test 1 Results

Locust Test Report									
During: 2021-07-22 12:09:50 - 2021-07-22 12:48:14									
Target Host: http://nginx:80									
Request Statistics									
Method	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
GET	/api/v1/Inp/	305597	53276	10050	6	39968	14	132.7	23.1
	Aggregated	305597	53276	10050	6	39968	14	132.7	23.1
Response Time Statistics									
Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
GET	/api/v1/Inp/	6700	7900	13000	19000	24000	27000	33000	40000
	Aggregated	6700	7900	13000	19000	24000	27000	33000	40000
Failures Statistics									
Method	Name	Error	Occurrences						
GET	/api/v1/Inp/	('Connection aborted.', ConnectionResetError(104, 'Connection reset by peer'))	856						
GET	/api/v1/Inp/	('Connection aborted.', RemoteDisconnected('Remote end closed connection without response'))	52420						

Figure 20: Locust Load Test Results. Test 1 (system scaled at 3 instances)

## 4.1.2 Test 2 Results

Locust Test Report									
During: 2021-07-22 12:09:50 - 2021-07-22 12:46:36									
Target Host: http://nginx:80									
Request Statistics									
Method	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
GET	/api/v1/Inp/	288221	47451	9949	6	39968	14	130.7	21.5
	Aggregated	288221	47451	9949	6	39968	14	130.7	21.5
Response Time Statistics									
Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
GET	/api/v1/Inp/	6600	7800	13000	18000	24000	28000	34000	40000
	Aggregated	6600	7800	13000	18000	24000	28000	34000	40000
Failures Statistics									
Method	Name	Error	Occurrences						
GET	/api/v1/Inp/	('Connection aborted.', ConnectionResetError(104, 'Connection reset by peer'))	808						
GET	/api/v1/Inp/	('Connection aborted.', RemoteDisconnected('Remote end closed connection without response'))	46643						

Figure 21: Locust Load Testing Results. Test 2 (system scaled at 6 instances)

### 4.1.3 Locust Load Testing Test1 and Test 2 Results Charts



Figure 22: Locust Load Testing Tests1 and Test 2 Results. Test Charts



## Discussion

When the service was scaled to 3 instances Test1 results recorded 23.1 failure rate with connection aborted error having 856 occurrences. When the service was scaled to 6 instances Test2 results captured 21.5 failure rate with connection aborted error having 808 occurrences. This shows the service failure rate is minimized by running more instances of a service. This is in agreement with Baboi, Iftene, & Gifu (2019) findings that microservices creates scalable and fault tolerant information systems.

### 4.2 Model Validation Testing Results

#### i) Validation Testing Tasks and Results for Admin

The exercise was carried out in the ICT Technician office at NYSEI on Monday 21<sup>st</sup> June 2021. Two technicians worked on the system as Admins and performed the following tasks.

Task	Instruction	Expected Results	Actual Results
1.	Setting Up/Installing the System on the Server's	Success	Success
2.	Registering as first user	Success	Success
3.	Create Attachment Sessions	Success	Success
4.	Confirm attachment status	Success	Success
5.	View students daily logs	Success	Success
6.	Add users	Success	Success
7.	Assign users roles	Success	Success
8.	Logout	Success	Success

Table 11: Validation Test Tasks and Results for Admin

## ii) Validation Testing Tasks and Results for ILO

A lecturer in charge of Industrial Attachment docket worked on the system as user role ILO and performed the following tasks.

<b>Task</b>	<b>Instruction</b>	<b>Expected Results</b>	<b>Actual Results</b>
1.	Registering as first user	Success	Success
2.	Create Attachment Sessions	Success	Success
3.	Confirm attachment status	Success	Success
4.	View students daily logs	Success	Success
5.	Add users	Success	Success
6.	Logout	Success	Success

Table 12: Validation Test Tasks and Results for ILO

## iii) Validation Testing Tasks for students:

This was done on Monday 21st June 2021 from 2:00 PM to 4:00PM at NYSEI ICT Computer Laboratory. A total of sixty-six students who had completed their industrial attachment were randomly selected. Having their manual industrial attachment logbooks, the students used the system by completing the following tasks.

<b>Task</b>	<b>Instruction</b>	<b>Expected Results</b>	<b>Actual Results</b>
1.	Register as a first user	Success	Success
2.	Log into the system	Success	Success
3.	Apply for industrial attachment	Success	Success
4.	Confirm attachment status	Success	Success
5.	Record at least 3 daily learning activities (Guided by their Manual log books)	Success	Success

<b>Task</b>	<b>Instruction</b>	<b>Expected Results</b>	<b>Actual Results</b>
6.	Upload a file (Scan any sketch in their log book)	Success	Success
7.	View attachment grade	Success	Success
8.	Logout	Success	Success

Table 13: Validation Test Tasks and Results for Students

**iv) Validation Testing Tasks for lecturers**

<b>Task</b>	<b>Instruction</b>	<b>Expected Results</b>	<b>Actual Results</b>
1.	Register as a first user	Success	Success
2.	Log into the system	Success	Success
3.	View Students daily logs	Success	Success
4.	Comment on students logs	Success	Success
5.	Score students	Success	success
6.	Logout	Success	Success

Table 14 Validation Test Tasks and Results for Lecturers

## **CHAPTER FIVE**

### **5.0 FINDINGS, CONCLUSIONS AND RECOMMENDATIONS**

#### **5.1 Findings**

The first objective was to find out the requirements of the new industrial attachment management system in TVET institutions. This objective was achieved and the requirements of the new system which formed the basis for building the Microservices model were identified using questionnaires and review of existing documents. The fact finding process both from students and lecturers of TVET institutions revealed that the business process was manual.

The second objective was to provide a scalable means of monitoring and supervising students on industrial attachment using microservices architecture. Literature review identified detailed coverage on how to build Microservices. Using agile software development methodology, the microservices model was designed and implemented with OAuth 2.0 protocol being adopted as security service, React JavaScript library built the user interface microservice while python Django & Django Rest framework implemented the attachment microservice and reports microservice. The study concurs with Swathi & Rashmi (2020) that container platforms like Docker help in effectively deploying microservices to achieve scalability.

Docker compose tool was used to deploy the application on a single host by running the services as multiple containers (Douglis & Nieh, 2019). NGINX was configured as a load balancer and locust load testing tool was used to figure out system performance as the number of users simultaneously accessed and transacted from the attachment microservice. The testing results revealed that each microservice can be scaled by running several instances in order to accommodate the increasing number of users. The results further shows that more instances of the microservices improves system performance as more instances take less time to respond to requests. The study is also in agreement with Wu, Wan, & Zhu (2018) results developing information systems as that loosely coupled microservices improves system scalability, availability and maintainability.

The third objective was to test and validate the performance of the developed model. The new model was configured and users allowed to run a number of tasks. Based on the analysis, all the users successfully accomplished the tasks. They managed to understand the flow of the system without any guidelines. Most of them agreed that the system user interface was friendly and interactive.

## **5.2 Limitations of the Research**

The COVID-19 pandemic Ministry of Health protocols affected the time frame and costs associated with the research. The limited time frame of six months affected aspects like sample size, data analysis, data interpretation, prototype development and implementation. The researcher being a student was tied to spend within the proposed budget which impacted negatively during the actual study.

The study adopted descriptive research design which had its own weaknesses, it couldn't answer the question "why". The design was used because of its ease of use and its capability of describing a situation as it is in its natural environment. The research also incorporated quantitative techniques to analyse data statistically.

The questionnaire as a data collection instrument had its limitations of responses being biased based on users/respondents understanding. Reliability and validity testing of the questionnaire could not completely eliminate respondent's ignorance's. While Locust load testing tool provided statistics about successful and failed connections, it didn't capture some data such as lost and duplicate messages from the API calls.

## **5.3 Conclusions**

The project findings reveals that currently, industrial attachment operations and processes are largely manual, hence the need for a web based system that can be implemented at the National level. The study also reveals that the introduction of microservices based model improves the industrial attachment docket operations. Microservices are easily scaled using containers, they can easily be replicated as demand increases. Scalability improves system performance by increasing the system availability and resilience in handling its operations.

From the analysis of validation testing, it can be concluded that the system functions as envisaged by users. Users were able to complete the task given successfully. The new system is easy to navigate, interactive, fast loading and easy to understand.

#### **5.4 Recommendations for Practice**

It is recommended that the system be implemented at national level to handle industrial attachment services in TVET institutions in Kenya.

The other existing information systems in TVET institutions like students management information systems, Higher Education Loans Board management system and Kenya National Examinations Exams registration systems to be converted into microservices and provide an application programming interface to the Industrial attachment system.

#### **5.5 Recommendations for Further Research**

As highlighted by Bradley (2021), Microservices put a lot of pressure on APIs, further studies are recommended to come up with a strong API management technology that can be adopted by Microservices.

While designing the Microservices model, complexity was experienced in decomposing a software application into smaller autonomous independent services. It was hard to determine: Each microservices size, optimal boundaries and database sharing policy. The study makes a second recommendation of further research be done on how to decompose a system into microservices particularly database sharing options between services while achieving the loose coupled, fine grained services.

The study used only docker compose container technology to scale the microservices to multiple instances and Locust Load testing to test system performance with simulated user behavior. It is recommended other container technology to be used with other open source load testing tools to be used and compare the results.

# APPENDICES

## i) References

- O'Donnell, C. (2020, 4 28). *FDA Software Validation: Guide, Methods, Tools and Template*. Retrieved from The Dacor Blog: <https://www.dacor.com/the-dacor-blog/fda-software-validation>
- Aineah, A. (2019, 09 15). *Challenges Students face in search for Instrial Attachment*. Retrieved from The Standard: <https://www.standardmedia.co.ke > ... > Money & Careers>
- Ali , M. (2014). Sampling & Sample Size Estimation. *Geneva Foundation for Medical Education and Research (GFMER)*.
- Baboi, M., Iftene, A., & Gifu, D. (2019). Dynamic Microservices to Create Scalable and Fault Tolerance Architecture. *23rd International Conference on Knowledge-Based and Intelligent Information & Engineering Systems* (pp. 1036-1044). Elsevier.
- Bhandari, P. (2021, 15 2). *An introduction to quantitative research*. Retrieved from Scribbr: <https://www.scribbr.com/methodology/quantitative-research/>
- Bradley, T. (2021, 6 26). *The challenges of scaling microservices*. Retrieved from TechBeacon: <https://techbeacon.com/app-dev-testing/challenges-scaling-microservices>
- Brush, K., & Silverthorne, V. (2021, 04 26). *TechTarget*. Retrieved from Agile Software Development: <https://searchsoftwarequality.techtarget.com/definition/agile-software-development>
- Casey, K. (2020, 2 10). *Popular microservices testing tools developers should know about*. Retrieved from TechTarget: <https://searcharchitecture.techtarget.com/feature/Popular-microservices-testing-tools-developers-should-know-about>
- Chandramouli, R. (2019). Security Strategies for Microservices-based Application Systems. Draft NIST Special Publication 800-204.
- Coelho, N. R. (2018). Security in Microservices Architectures. *Centeris 2020*. Vilamoura.
- Dennis, A., Wixom, B. W., & Roth, R. M. (2012). *System Analysis and Design*. John Wiley & Sons, Inc.
- Django. (2021, 05 24). *Django Rest Framework*. Retrieved from <https://www.django-rest-framework.org/>
- Docker . (2021, 6 26). *Docker Documentation*. Retrieved from Docker overview: <https://docs.docker.com/get-started/overview/>
- Douglis, F., & Nieh, J. (2019). Microservices and Containers. *IEEE Computer Society*, 5 - 6.
- Familiar, B. (2015). *Microservicees, IoT, and Azure Leveraging DevOps and Microservice Architecture to deliver SaaS Solutions*. Apress.
- Fatima , F., Javed , M., Amjad, F., & Ghanni, K. U. (2018). An Approach to Enhance Quality of Rapid Application Development (RAD). *Journal of American Science*, pp 47-56.

- Ghahrai, A. (2017, 4 19). Retrieved from Testing Microservices - A Beginner's Guide: <https://devqa.io/testing-microservices-beginners-guide/>
- Girsang, A. S., Jafar, F., & Fajar, A. N. (2018). Design Project Management System Based on SOA Approach. *International Conference on Computation in Science and Engineering* (pp. 1-10). IOP Publishing.
- Hardik, S. (2020, 12 24). *Best of 2020: When To Use – and Not To Use – Microservices*. Retrieved 03 12, 2021, from <https://containerjournal.com/topics/container-ecosystems/when-to-use-and-not-to-use-microservices/>
- Hasti, N., Lesari, S., & Gustiana, I. (2019). Web-Based Internship Information System. *Web-Based Internship Information System* (pp. 1-6). IOP Publishing.
- Hipschman, D. (2021, FEB 10). *Python Microservices With gRPC*. Retrieved 03 16, 2021, from <https://realpython.com/python-microservices-grpc/>
- Hoffman, J. (2021, 3 27). *MVC vs. Microservices: Understanding their architecture*. Retrieved from WisdomPlexus: <https://wisdomplexus.com/blogs/mvc-vs-microservices/>
- Hymet, M., & Arbana, K. (2020). Using Internship Management System to Improve the the relationship between Internship Seekers, Employers and Educational Institutions. *Procededings of ENTRENOVA, Virtual Conference*, (pp. 97-104). Zagreb.
- Ihor, F. (2021, 4 26). *Agile Software Development Lifecycle Phases Explained*. Retrieved from Relevant Software LLC: <https://relevant.software/blog/agile-software-development-lifecycle-phases-explained/>
- Indrasiri, K., & Siriwardena, P. (2018). *Microservices for the Enterprise Designing, Developing, and Deploying*. San Jose: Apress.
- Jaafar, A. N., Rohafauzi, S. b., Enzai, N., Fauzi, b. F., & Amron, M. ( 2018 ). Development of internship monitoring and supervising web-based system. *2017 IEEE 15th Student Conference on Research and Development (SCOReD)*, 193-197.
- Jeremy, H. (2021, 3 26). *Microservices vs Web Services* . Retrieved from Dream Factory: <https://blog.dreamfactory.com/microservices-vs-web-services/#The-Web-Services-Application-Architecture>
- Juhana, A., Abdullah, A. G., Somantri, M., Aryadi, S., Zakaria, D., & Arasid, W. (2017). E-Portfolio Web-based for Students' Internship Program Activites. *IOP Conf. Series: Materials Science and Engineering 306 (2018)*, (pp. 1-9).
- Kamate, G., & Prasad, R. (2018). Docker & Containers, The Future of Microservices. *International Journal of Science and Research (IJSR)*, 23-25.
- Kanjilal, J. (2020, 1 24). *TechTarget*. Retrieved from Pros and cons of monolithic vs. microservices architecture: <https://searchapparchitecture.techtarget.com/tip/Pros-and-cons-of-monolithic-vs-microservices-architecture>
- Kiplagat, H., Khamasi, J. W., & Kareri, R. (2016). Students' Experience of Industrial Attachment: A Case of a Public University. *Journal of African Studies in Educational Management and Leadership*, 7, 82-94.



- Korir, j. (2021, 3 24). *Industrial Liaisons Office*. Retrieved from Kaiboi Technical Training Institute: <https://www.kaiboitech.ac.ke/index.php?title=Industrial%20Liaison%20Office>
- Kothari, R. C. (1985). *Research Methodology Methods & Techniques* (Second Revised Edition ed.). New Delli: New Age International (P) Limited, Publishers.
- KTTC. (2021, 3 19). *Kenya Technical Trainers College- Industrial Liaison*. Retrieved from Kenya Technical Trainers College: <https://www.kttc.ac.ke/non-academic/ilo>
- Kumar, R. (2011). *Research Methodology a step-by-step guide for beginners* (3rd Edition ed.). London: Sage Publications.
- Kumar, R. (2018, 8 13). *Selecting the Right Database for Your Microservices*. Retrieved 3 12, 2021, from <https://thenewstack.io/selecting-the-right-database-for-your-microservices/>
- Lewis, J., & Fowler, M. (2014, 3 24). *Microservices*. Retrieved 3 2021, 2021, from <https://martinfowler.com/articles/microservices.html>
- Locust. (2021, 6 26). *Locust Documentation*. Retrieved from Locust: <https://locust.io/>
- Maximilian , S. (2021, 6 28). *How to Use Docker Compose to Run Multiple Instances of a Service in Development*. Retrieved from PSPDFKit GmbH: <https://pspdfkit.com>
- Middleton, F. (2020, 6 26). *Reliability vs validity: what's the difference?* Retrieved from Scribbr: <https://www.scribbr.com/methodology/reliability-vs-validity/>
- Mordo, A. (2021, 3 12). *Best practices for scaling with DevOps and microservices*. Retrieved from <https://techbeacon.com/app-dev-testing/best-practices-scaling-devops-microservices>
- Mugenda, O. M., & Mugenda, A. G. (1999). *Research Methods: Quantitative and Qualitative Approaches*. Nairobi: Acts Press.
- Mutiso, P. N. (2021, 3 25). *Industrial Liaison Coordination Center*. Retrieved from Rift Valley Technical Training Institute – Eldoret: <https://rvti.ac.ke/industrial-liaison-office/>
- Naeem, T. (2021, 05 26). *All You Need to Know About Database Design*. Retrieved from Astera: <https://www.astera.com/type/blog/all-you-need-to-know-about-database-design/>
- Newman, S. (2015). *Building Microservices*. O'reilly.
- NITA. (2021, 3 19). *NITA* . Retrieved from Industrial Attachment: <https://www.nita.go.ke/our-services/industrial-attachment.html>
- Njoroge. (2002). *Relationship between Mathematical Language and Students Republic of Kenya, Central Bureau. Ministry of Planning and National Development: The population and Housing*. Nairobi. : Government Printer.
- Nugroho, R. A., & Fajar, A. N. (2019). *DEsigning Production SYstem Using Service Oriented Architecture case study in Automotive Manufacturing*. *Journal of Theoretical and Applied Information Technology*, 2322-2333.
- Oates, B. J. (2006). *Researching Information Systems and Computing*. London: Sage Publications.

- Okta. (2021, 04 29). *Authenticatio / Okta Developer*. Retrieved from Okta Developer: <https://developer.okta.com/docs/concepts/authentication/>
- Pachghare, V. K. (2016). Microservices Architecture for Cloud Computing. *Journal of Information Technology and Sciences*, 1-13.
- Rai, P. (2021, 05 12). *Program Design Tools*. Retrieved from <https://prajwalrai.com.np/algorithm-and-flowchart-introduction/>
- React. (2021, 05 18). *A JavaScript library for building user interfaces*. Retrieved from React: <https://reactjs.org/>
- Richards, M. (2021, 3 12). *Microservices vs. Service-Oriented Architecture*. Retrieved from <https://www.oreilly.com/library/view/microservices-vs-service-oriented/9781491975657/ch01.html>
- Sarlan, A., Ahmad, F. W., Jolonius, J. N., & Samsudin, N. (n.d.). Online Web-based Industrial Internship System. *1st International Malaysian Educational Technology Convention*, (pp. 194-200).
- Schroeder, D. A. (n.d.). *Microservice Architectures*. Elsenheinmerstr 55A. Retrieved from [www.codecentric.de](http://www.codecentric.de)
- Sengupta, S. (2021, 1 20). *Challenges of Microservices & When To Avoid Them*. Retrieved from DEVOPs Blog: <https://www.bmc.com/blogs/microservices-challenges-when-to-avoid/>
- Shaik, K. S., Ramkumar, Hameed, S. S., & Mohammed, S. (2019). Microservice Based Architecture Model for EHR Systems of MOH Hospitals - Sultanate of Oman. *International Journal of Contemporary Research in Computer Science and Technology (IJCRCST)*, 8-13.
- Shaik, N. S., & Mane, B. S. (2017). Authentic Techniques Of Authentication In Microservices. *International Journal of Current Advanced Research*, 3342-3345.
- Shivakumar, K. S. (2021, 3 19). *Microservices Architecture for Modern Digital Platforms*. Retrieved from Mindtree: <https://www.mindtree.com/about/resources/microservices-architecture-solutions-modern-digital-platforms>
- Shivakumar, S. K. (2019). *Microservices rchitecture for Modern Digital Platforms. A white paper*. Mindtree Limited.
- Singh, A. (2019, 12 6). *What Is Rapid Application Development (RAD)?* Retrieved from Capterra: <https://blog.capterra.com/what-is-rapid-application-development/>
- Sommerville, I. (2016). *Software Engineering* (Tenth Edition ed.). Edinburgh Gate: Pearson Education Limited.
- surendra, s. (n.d.). *Analysis and Design of Internship Report and Thesis Mentoring Management System*. Retrieved from [https://www.academia.edu/5254885/Analysis\\_and\\_Design\\_of\\_Internship\\_Report\\_and\\_Thesis\\_Mentoring\\_Management\\_System](https://www.academia.edu/5254885/Analysis_and_Design_of_Internship_Report_and_Thesis_Mentoring_Management_System)

- Swathi, & Rashmi, R. (2020). Microservice Architectural Style. *International Research Journal of Engineering and Technology (IRJET)*, 8.
- Tanenbaum, A. S., & Steen, v. M. (2016). A brief introduction to distributed systems. In *"Distributed Systems, Principles and (pp. 967-1009)*. Springerlink.com.
- Velez, G. (2019). Kubernetes vs. Docker: A Primer. *Container Journal*.
- Waseem, M., Liang, P., & Márquez, G. (2020). Testing Microservices Architecture-Based Applications: A Systematic Mapping Study.
- Wasson, C. S. (2006). *System Analysis, Design, and Development Concepts, Principles, and Practices*. Hoboken, New Jersey: A John Wiley & Sons, Inc.,.
- Wittmer, P. (2021, 3 29). *Service-oriented architectures are like microservices in that they're both a collection of services focused on performing one specific function*. Retrieved from Tiempo Development: <https://www.tiempodev.com/blog/microservices-vs-soa/>
- Wu, S., Wan, X., & Zhu, Q. (2018). Design of WeChat Service System Based on Microservice Architecture. *IOP Conf. Series: Journal of Physics: Conf. Series 1069 (2018)* (pp. 1-8). IOP Publishing.
- Yadav, S. (2019, 5 21). *Introduction to Microservices Architecture vs SOA vs Monolithic*. Retrieved from The Geek Stuff: <https://www.thegeekstuff.com/2019/05/intro-to-microservices/>
- Yannuar , Y., Hasan, B., Abdullah, A., Hakim, D. L., & Wahyudin, D. (2018). Design and implementation of web-based internship information system at vocational school. *IOP Conference Series: Materials Science and Engineering*. IOP.
- Ziade, T. (2017). *Python Microservices Development*. Birmingham: Packt Publishing Ltd.

**ii) Research Questionnaires**

**a) Students Questionnaire**

This is a research questionnaire on a Microservices based Student Industrial Attachment Information System Model for TVET institutions in Kenya. The findings will help improve the current industrial attachment system by developing a new system.

For closed questions, kindly put a check (√) in the appropriate box as applicable to you. Please write your answers in the space provided for open questions.

Department ..... Year of Study..... Date .....

1. What are the requirements one needs before proceeding for industrial attachment?

Item	Yes	No
Studied and Passed given academic units		
Recommendation from Head of Department		
Insurance Cover		
Any Other		

2. What are some of the challenges faced while sourcing for industrial attachment places?

Item	Yes	No
Difficulty in identifying industrial attachment places		
Lack of common place to coordinate applications		
Poor tracking of outcomes from the applied sources		
Any Other		

3. What are some of the challenges faced while on industrial attachment?

Item	Yes	No
Lack of common communication link		
Manual way of keeping records		
Any Other		

4. What are the daily activities followed while on Industrial Attachment?

Item	Yes	No
Adherence to the attachment institutions policies		
Daily recording of learning activities		
Supervision by organization Staff		
Assessment by college assessors		
Any Other		

5. What are the documents awarded by the organization on completion of the attachment?

Item	Yes	No
Clearance from the establishment		
Receiving a recommendation letter		
Any Other		

Thanks in advance.

### b) TVET Institutions Trainer Questionnaire

This is a research questionnaire on a Microservices based Student Industrial Attachment Information System Model for TVET institutions in Kenya. The findings will help improve the current industrial attachment system by developing a new system.

For closed questions, kindly put a check (√) in the appropriate box as applicable to you. Please write your answers in the space provided for open questions.

Department .....Position/Designation..... Date .....

1. What are some of the challenges faced by Trainers while assessing students in various attachment institutions?

Item	Yes	No
Poor record keeping of students documents		
Poor record keeping of assessment document		
Poor means of communication between Attached institutions		
Any Other		

2. What are some of the important documents used in the industrial attachment docket?

Item	Yes	No
Registration form		
Attachment Request letter		
Posting Letter		
Log Book		
Recommendation Letter		
Assessment Form		
Any other		

6. What technologies are used while on industrial attachment? What is the purpose of each technology?

Item	Yes	No	Purpose
Email			
Ms Word/Excel/Access			
WhatsApp/Instagram			
Any Other Technology			

3. Kindly fill how the final attachment grade is calculated

Item	Percentage	Remarks
Up to date well organized Log book		
Industry Supervisors Score		
Academic Institutions Assessor Score		
Final Attachment Report Score		
Any Other		

4. What are some of the measures you propose to improve the functions of industrial Attachment docket?

Item	Yes	No
Introduction of an interactive, high available web based system		
Any other		

5. What is the minimum number of hours a student is supposed to be exposed to workplace training? \_\_\_\_\_

Thanks in advance.

### iii) Project Schedule

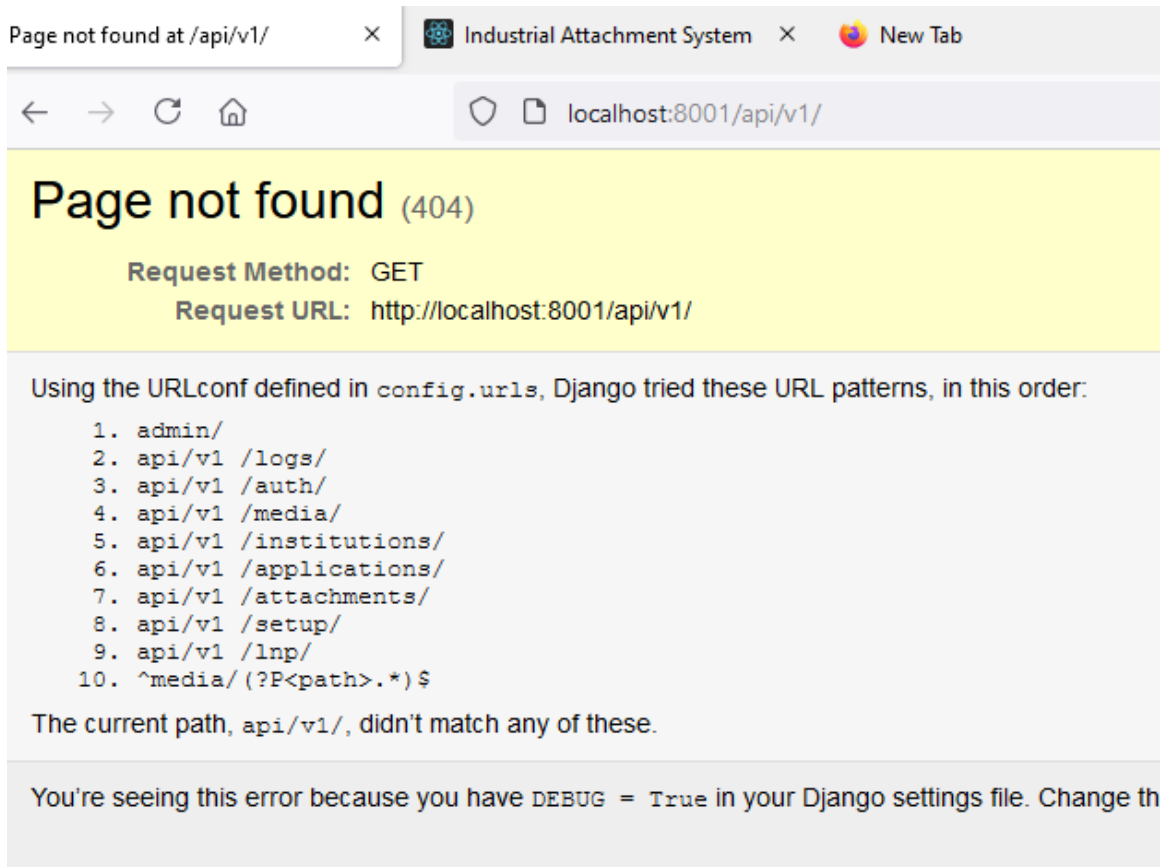
Activity	Months				
	March	April	May	June	July
Proposal Drafting	1 Month				
Data Collection and Analysis		1 Month			
Prototype Design			1 Month		
Prototype Implementation				1 month	
System Testing					1 Month
Documentation	5 months				

### iv) Project Budget

Description	Unit Price	Cost (Ksh)
Computer System	80,000	80,000
Accessories	7,000	7,000
Subscription fee to journals and e-books	15,000	15,000
Data collection	10,000	10,000
Journal Paper Publications	10,000	10,000
Overheads	8,000	8,000
<b>Total</b>		<b>130,000</b>

## v) Sample User Interface

### a) Attachment Microservice Endpoints (Error Page)



The screenshot shows a web browser window with the following details:

- Page not found at /api/v1/
- Industrial Attachment System
- New Tab
- localhost:8001/api/v1/
- Page not found (404)**
- Request Method: GET
- Request URL: http://localhost:8001/api/v1/

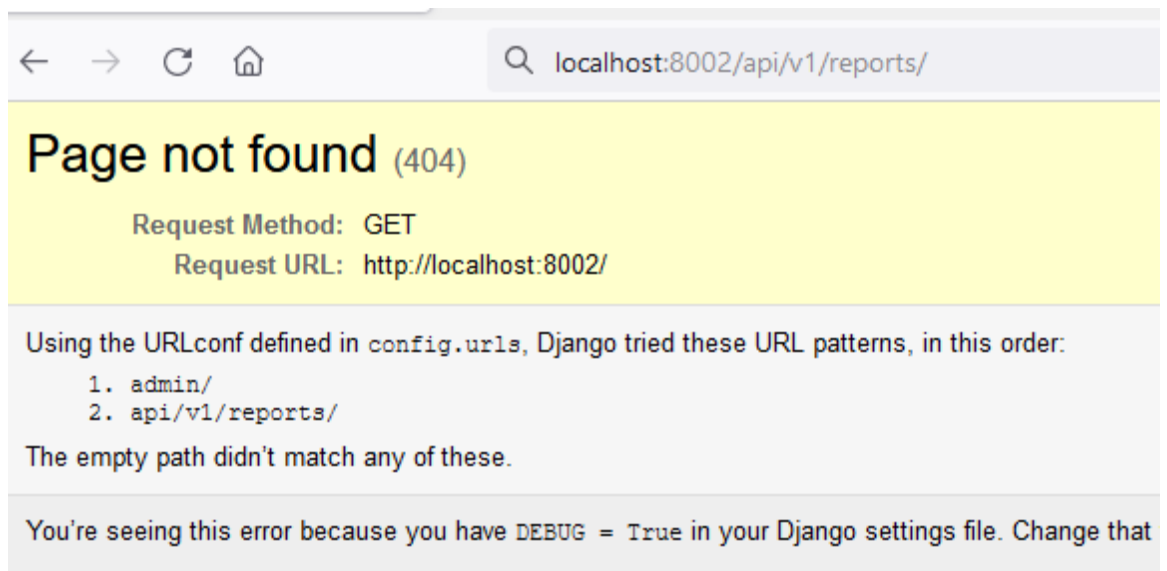
Using the URLconf defined in `config.urls`, Django tried these URL patterns, in this order:

1. admin/
2. api/v1 /logs/
3. api/v1 /auth/
4. api/v1 /media/
5. api/v1 /institutions/
6. api/v1 /applications/
7. api/v1 /attachments/
8. api/v1 /setup/
9. api/v1 /lmp/
10. ^media/(?P<path>.\*)\$

The current path, `api/v1/`, didn't match any of these.

You're seeing this error because you have `DEBUG = True` in your Django settings file. Change th

### b) Report Microservice Endpoint (Error Page)



The screenshot shows a web browser window with the following details:

- localhost:8002/api/v1/reports/
- Page not found (404)**
- Request Method: GET
- Request URL: http://localhost:8002/

Using the URLconf defined in `config.urls`, Django tried these URL patterns, in this order:

1. admin/
2. api/v1/reports/

The empty path didn't match any of these.

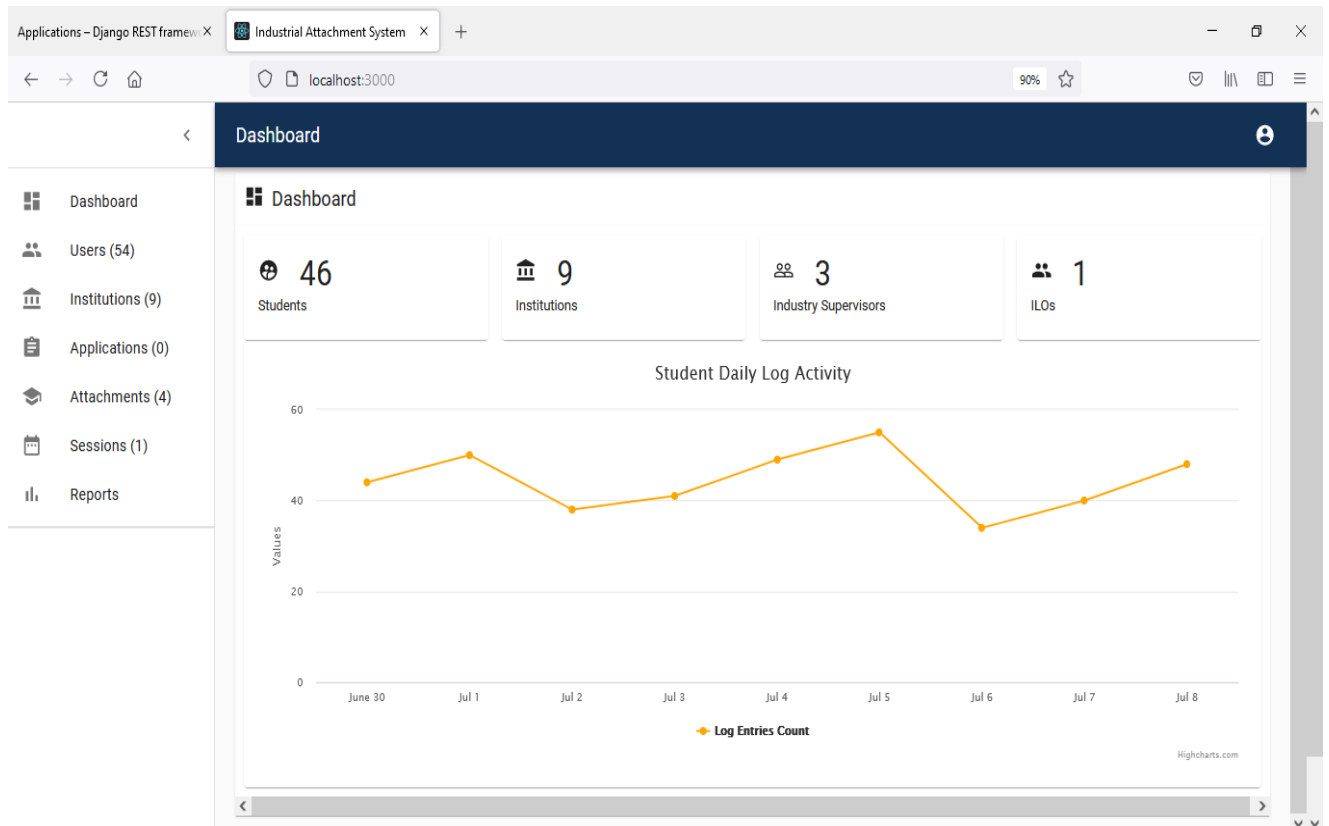
You're seeing this error because you have `DEBUG = True` in your Django settings file. Change that



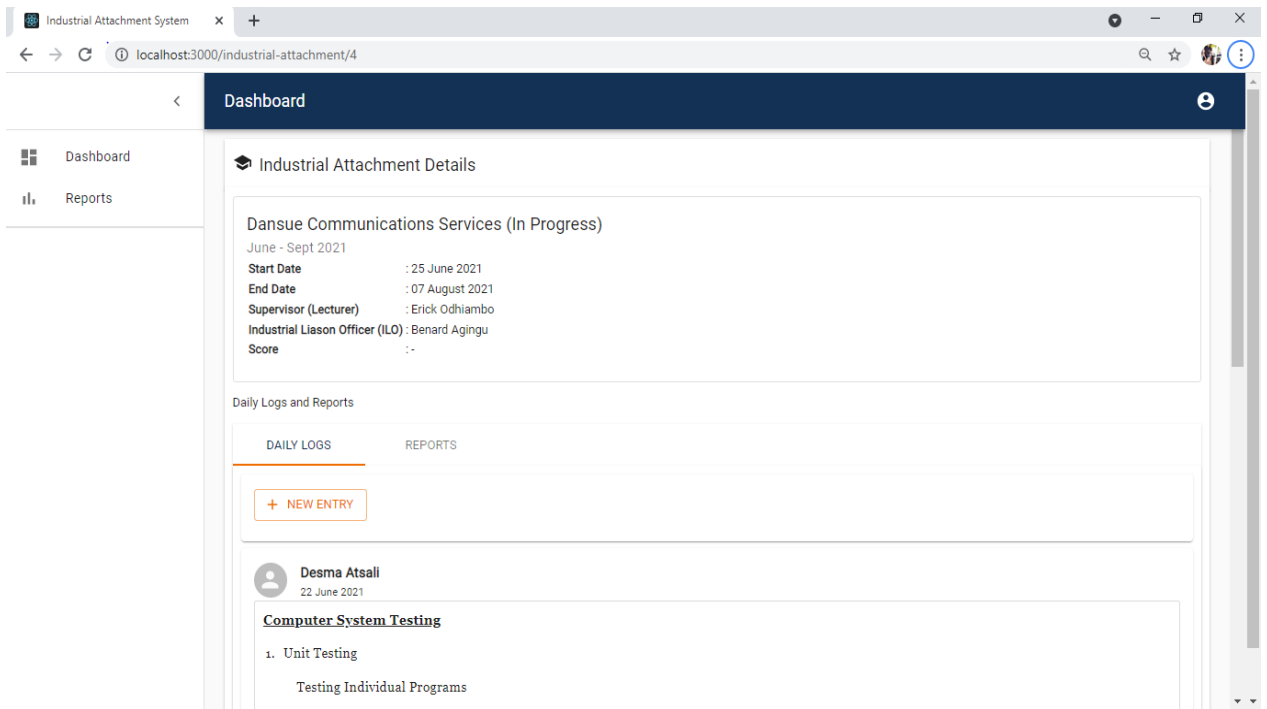
### c) Authentication Service User interface



### d) Admin User Interface Dashboard



## e) Student User Interface



## vi) Sample Code

### a) User interface docker file

```
FROM node:14

RUN mkdir -p /usr/src/app

WORKDIR /usr/src/app

COPY package.json package-lock.json ./

COPY . ./

RUN echo n | npm install

RUN echo n | npm install --save react-rte-image
RUN echo n | npm install --save react-rte-material
RUN echo n | npm install --save react-rte

EXPOSE 3000
```

## b) Dockerfile at the backend

```
FROM python:3

RUN mkdir -p /usr/src/app \
    && apt-get -y update \
    && pip install -U pip

WORKDIR /usr/src/app

COPY requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt

EXPOSE 8001
```

## c) Sample ReactJS Useraction.tsx file

```
import { ReducerAction } from "../types";
import {
  FETCH_USERS_FAILURE,
  FETCH_USERS_REQUEST,
  FETCH_USERS_SUCCESS,
} from "./userTypes";
import http from "../../components/common/api-axios";
import { USERS_URL } from "../../components/common/constants";
import store from "../store";

export const fetchUsersRequest = (): ReducerAction => {
  return {
    type: FETCH_USERS_REQUEST,
  };
};

export const fetchUsersSuccess = (payload: object): ReducerAction => {
  return {
    type: FETCH_USERS_SUCCESS,
    payload: payload,
  };
};

export const fetchUsersFailure = (error: string): ReducerAction => {
  return {
    type: FETCH_USERS_FAILURE,
    payload: error,
  };
};
```

```

export const fetchUsers = (query: string = "") => {
  return (dispatch: Function) => {
    dispatch(fetchUsersRequest());
    http
      .get(`${USERS_URL}?${query}`, {
        headers: {
          "Content-Type": "application/json",
          Authorization: store.getState().auth["accessToken"],
        },
      })
      .then((response) => dispatch(fetchUsersSuccess(response.data)))
      .catch((error) => dispatch(fetchUsersFailure(error.message)));
  };
};

export const deleteUser = (id:string) => {
  return http
    .delete(`${USERS_URL}${id}`, {
      headers: {
        "Content-Type": "application/json",
        Authorization: store.getState().auth["accessToken"],
      },
    })
};

```

#### d) Sample Serializer.py file

```

from rest_framework import serializers
from rest_framework.exceptions import ValidationError

from applications.models import Application

class ApplicationSerializer(serializers.ModelSerializer):
    student_name = serializers.ReadOnlyField()
    institution_name = serializers.ReadOnlyField()
    supervisor_name = serializers.ReadOnlyField()
    ilo_name = serializers.ReadOnlyField()

    class Meta:
        model = Application
        fields = (
            "id",
            "student",
            "session",
            "session_name",
            "supervisor",
            "ilo",
            "institution",
            "start_date",

```

```

        "end_date",
        "status",
        "student_name",
        "institution_name",
        "supervisor_name",
        "ilo_name",
        "display_name",
    )

```

### e) Authentication.py file

```

import requests
from okta_jwt.jwt import validate_token
from rest_framework.authentication import BaseAuthentication
from rest_framework.exceptions import AuthenticationFailed

from authentication.models import User

ISSUER = "https://dev-43093150.okta.com/oauth2/default"
USER_INFO_URL = "https://dev-43093150.okta.com/oauth2/default/v1/userinfo"
CLIENT_ID = "0oatrrh375vmuEMJ5915d6"
AUDIENCE = "api://default"

class OktaAuthentication(BaseAuthentication):
    def authenticate(self, request):
        access_token = request.META.get("HTTP_AUTHORIZATION")
        if not access_token:
            return None
        try:
            payload = validate_token(access_token, ISSUER, AUDIENCE, [CLIENT_ID])
            email = payload["sub"]
            user, created = User.objects.get_or_create(email=email,
username=email)

            # If user has just been created or they don't have a profile setup,
            # call the userinfo endpoint to retrieve their profile
            if created or not user.name:
                self.fetch_user_profile(access_token, user)
            return user, None
        except Exception as e:
            raise AuthenticationFailed(str(e))

    def fetch_user_profile(self, access_token, user):
        try:
            headers = {
                "Content-Type": "application/json",
                "Authorization": "Bearer {}".format(access_token),
            }
            response = requests.post(USER_INFO_URL, headers=headers)
            user_dict = response.json()

            user.name = user_dict.get("name")
            user.save()
        except Exception as e:
            print(str(e))

```