UNIVERSITY of RWANDA

Research and Postgraduate Studies (RPGS) Unit

Design Microservices and Event driven Eco-System for eTax

By

NIYIGENA Pierre Fourier

Ref: 221027385

A dissertation submitted in partial fulfilment of the requirements for the degree of Masters in Software Engineering

Supervisor: Dr. Alexander NGENZI

Co-Supervisor: Dr. Frederic NZANYWAYINGOMA

DECLARATION

I declare that, this project work entitled, "**Design Microservices and Event driven Eco-System for eTax**" is original and has never been submitted to any University of other Institution of Higher Learning. It is my own research whereby other scholar's writings were cited and references provided. I thus declare this work is mine and was completed successfully under the supervisor of Dr. Alexander NGENZI and Dr. Frederic NZANYWAYINGOMA.

NIYIGENA Pierre Fourier

ACKNOWLEDGEMENTS

On the benefits of this work, I would like to thank all the contributions of the individuals, Lecturers, Students and families who helped to accomplish this works. This work couldn't have accomplished without your willing moral and technical support.

I would like Special to thank Dr. Alexander NGENZI and Dr. Frederic NZANYWAYINGOMA for your tireless guidance and supervision in achievement of this work.

I would like to thank My Wife Jowe KABIBI KACYIRA, for your moral support and encouragements.

Thanks to my colleague students for your encouragements.

May the Almighty God bless all of these people!

CERTIFICATION

The undersigned certify that I have read and hereby recommend for acceptance by the University of Rwanda, College of Science and Technology (UR/CST); a dissertation: "**Design Micro services and Event driven Eco-System for eTax"** submitted in partial fulfilment of the requirements for the degree of Master of Science in software engineering, at the University of Rwanda, College of Science and Technology, School of ICT, department of Computer Science, option of Software Engineering.

Signature: …….

Date: ……./……../2024

NIYIGENA Pierre Fourier

Signature: **Signature:** 

Date: …../…../2024 **Date:**18 / September/2024

Dr. Alexander NGENZI **Dr. Frederic NZANYWAYINGOMA**

Supervisor **Co-Supervisor**

Signature:

Date: …../…../2024

Names:

Head of Department

Abstract

Today's technology sector is growing quickly, government institutions are running various software applications. Among those institutions there are revenue collection agencies. The government revenue collection agencies are providing online services (eTax) to the taxpayers in order to facilitate them to have access to the services and make easy for them to accomplish their duties and monolithic architecture is common for most of their system. This brings problem when it comes to add new features in the system like adding payment module, notifications services and etc. Adding one functionality to the monolithic system will require to change the whole system. Another problem is related to the deployment where the system is deployed on one server and if the server is down for any reason, it makes the whole service unavailable. It is also hard to work on parallel task in this architecture since one person should wait another one to accomplish one task. This study is aiming to develop the microservices and even driven system to for government revenue collection (eTax) as a solution to the resilience issue, scalability problem, system deployment problem and increase the productivity of development team. To achieve our objectives, the agile model were used as software development process model. Observations, interviews and documentations techniques were used in requirements gathering, system analysis, designing. For modeling we used Unified Modeling Language diagram. To implement our solution Spring boot framework with its Kafka messaging were used as Asynchronous communication. To deploy our solution, we packaged the eco- system in docker containers which can be deployed in any operating system.

By combining all those concepts and tools our developed system is very scalable, flexible, decoupled, and resilient and simple to deploy.

Keywords: *Monolithic, Micro services, Gateway, Kafka, Keycloak, Docker.*

List of Acronyms

API: Application Programming Interface

CPU: Central processing unit

DB: Database

DMS: Declaration Microservice System

EDA: Event-Driven Architecture

ERD: Entity Relation Diagram

GDPR: The General Data Protection Regulation

HIPAA: The Health Insurance Portability and Accountability Act

HTML: Hyper Text Markup Language

IAM: Identity and Access Management

IP: Internet Protocol

OS: Operating system

SSO: Single sign-on

PAYE: Pay As You Earn

SDLC: Software Development Life Cycle

SQL: Structured Query Language

TIN: Taxpayer Identification Number

TMS: Taxpayer Microservice Menu

UI: User Interface

UML: Unified Modeling Language

XML: Extensible Markup Language

XHTML: Extensible Hyper Text Markup Language

VAT: Value-added tax

WHT: Withholding Tax

Table of Content

List of Figures

List of Tables

CHAPTER 1: INTRODUCTION

1.1 Introduction

In today's rapidly evolving technological landscape, the design and implementation of software systems have become more intricate than ever before. Today we need to provide a solution for this downtime issue, the service is needed 24/7. Traditional monolithic architectures[1] struggle to keep pace with the demands for scalability, flexibility, and responsiveness. We need the high availability system. The system should be resilient (Ability to resist to different type of failure). Use Cases grow every day and time to go to the market is considered due to the need of service new or upgrade the existing one. We need to adapt a technology which is flexible and adaptive to the changes and adjustment (scalability). In recent years, adapting event driven architecture is a solution to the listed issues. It provides a data sharing at scale and a distributed[2] data feature. The event stream data sharing is the heart of the event driven. With microservice, we are trying to divide a huge monolithic application into small piece of application called microservice which are loosely decoupled, cohesive and well defined.

Today application systems are collecting a massive amount of data. In distributed architecture, the microservices are stateless. We need to integrate data exchange between the services. Decoupling system into small piece of interconnected service is an answer to the system resilient. It is also the response to the anatomy of the teams working parallel and can lead to the product completion at time. Adopting event driven is good choice to have a suitable solution to the current issues system faced, but it requires to learn new tools, new concepts and this can affect the productivity of institutions. It also affects the way system are deployed and this sometimes require to change hardware. The containers technologies [3] are very useful to handle deployment environment by packaging codes and tools into one container.

Considering government revenue collection, providing a microservice and event-driven architecture to the existing system can be a solution to the resilience issues, scalability problem [4], increase team productivity and deployment problems current faced by the old fashion. This will also provide the interoperability with other system such as payment collection system and other government systems. This will help also the taxpayers to accomplished the duties on time and avoid the charges they always face when the systems are not available on the last minute of the declaration and payment deadlines.

1.2 Background and Motivation

1.2.1 Background of this research project

In recent years, there has been a significant shift towards distributed computing models to accommodate the growing complexity and scalability requirements of modern applications. Traditional monolithic architectures often struggle to meet these demands due to their tightly coupled nature, where changes or updates in one part of the system can impact the entire application. Event-Driven Architecture addresses these challenges by promoting loose coupling, resilience, and scalability through asynchronous communication and event propagation. Microservices architecture advocates breaking down large, monolithic applications into smaller, autonomous services that are independently deployable, scalable, and maintainable. Each microservice focuses on a specific business capability, communicating with other services through lightweight APIs. This approach promotes agility, allowing teams to develop, deploy, and update services independently, thereby accelerating innovation and reducing time-to-market. Event-Driven Architecture (EDA) represents a paradigm shift in software design where systems respond to events rather than relying on direct, synchronous communication between components. Events can signify various occurrences within a system, such as user actions, sensor inputs, or system alerts, and can trigger corresponding actions or workflows across distributed services. This approach promotes scalability, flexibility, and responsiveness in modern applications, making it a compelling area for research and implementation.

1.2.2 Motivation of this research project

The motivation behind researching Microservices and Event-Driven Architecture stems from its potential to revolutionize how applications are designed, developed, and deployed. By leveraging events as first-class citizens in system design, EDA enables systems to react in real-time to changes and inputs, facilitating agility and responsiveness crucial for today's dynamic business environments. Moreover, EDA aligns well with modern development practices such as microservices, serverless computing, and cloud-native architectures, further driving its adoption and relevance in industry and academia.

This research help taxpayer to have accessibility to the service provided by tax authority in easy way via different channels. This will help the taxpayers pay to do their duties on time and avoid the penalties of later payment. On the other hand, this research project will provide best practice

to the development team of revenue authority where they will be able to deliver products quickly and also provide efficient software solutions. Lastly the 3rd parties including payment providers will be able to integrate with revenue authority system easy and in secure manner.

1.3 Problem Statement

The traditional approaches to designing and deploying software applications, characterized by monolithic architectures and synchronous communication patterns, are increasingly inadequate to meet the demands of today's dynamic and scalable digital environments. As organizations strive for agility, scalability, and resilience, there is a growing recognition of the potential benefits offered by integrating Microservices and Event-Driven Architecture (EDA). Today, user requirements are becoming complex in exponentially manner and it is very difficult to meet user expectation using traditional approach. There is a need of integrated effort to conduct many researches in order to choose the best approach to respond to this challenge. It is very difficult to manage and coordinate work in traditional way where number of team is huge [5], components tightly coupled and codebase is entangled. We need to deliver a software product faster. With traditional way (Monolithic) is very hard to increase productivity due to dependency of all modules. When we want to change small thing, we need to call all team member and also run test for the whole system. When there is a bug in one component, the whole system is down until the bug is fixed. The more we mix component together the more it becomes hard to fix the bug and also it takes long to have a release.

Another challenges web application systems are facing is availability. Today application systems are receiving many requests at the time due to the huge number of users requesting services over internet. There is a need of maintaining availability, stability, durability of those systems and also provide auto recovery process in case of failure. There is a need of implementing tools to monitor those systems.

1.4 Study Objectives

1.4.1 General Objective

The general objective of this research is to develop the microservices and even driven system to for government revenue collection (eTax) as a solution to the resilience issue, scalability problem, system deployment problem and increase the productivity of development team.

1.4.2 Specific Objectives

- ✓ To Analyze and Understand Microservices and Event-Driven Architecture.
- ✓ To Integrate of Microservices and Event-Driven Architecture with real world application.
- ✓ To Design, Develop and implement Event-Driven with in Microservices.
- ✓ To Evaluate the Performance and Scalability with in Microservices.

1.5 Hypotheses

In Chad, it is very difficult to declare and pay taxes easily. Taxpayers have to move from their office to the tax collection office and make declaration from there. Also, the developed online solution is still hard to integrate it with external entities like payment providers and taxpayer registration office. With this research project we will develop a secure, scalable and resilient eTax system and will be packaged into image containers for easy deployment.

1.6 Study Scope

This research project will focus on the two tax services such as declaration and payment. It will also focus on developing microservices and even-driven architecture to provide those services. We will implement an API gateway [6] for all external request and secure this API gateway with open authentication server. And lastly, we will package those microservices into image containers for the deployment.

1.7 Significance of the Study

Technology is growing every day; researchers are spending days and nights finding new way of doing things and increase the productivity in software engineering. On the other hand, learning new technologies is also a key factory of success. In this research we will use the knowledge gained from master's programs and use them in practice for the real word application. API Gateway Pattern as one of Microservices Design Patterns will be used in this research. Use of Agile model in this research is also another aspect to be considered.

1.8 Organization of the Study

The next section of this research project is grouped into 6 chapters as follow: The chapter 1 focus on introduction of this research project; the chapter 2 focus on the literature review and previous research of the microservices and even-driven architecture together with the deployment method of them. The chapter 3 describe the methodology and the data collection methods used in this

research project. The chapter 4 consists of the analysis we made within this research project in order to be able to do the system design. It also contains the list of the tools used in this research project. The chapter 5 contains the result of this research and analysis together with presentation of some tools in action. The chapter 6 contains the conclusion and the recommendations for further research in this area of taxes collection and also in software development field.

1.9 Conclusion

This chapter is an introduction of the research project to be conducted in this report. We started by illustrating the problem this research is trying to address. Then we presented the way we will address those problem at the end of this research. We also discussed on the scope of this research and the structure of this report. At the end of this research taxpayer will be able to declare and payment taxes, on the other hand payment collector's system and other external system will be to interact with eTax system in a reliable and secure manner. Lastly the development team in revenue collection office will increase the productivity.

CHAPTER 2: LITERATURE REVIEW

This literature review explores the concepts of microservices and event-driven architecture in the context of eTax, focusing on their benefits, challenges, and real-world applications. By analyzing a range of scholarly articles, conference papers, and industry reports, this review aims to provide a comprehensive understanding of the integration of microservices and event-driven architecture in eTax systems, highlighting their impact on scalability, flexibility, maintainability, and fault tolerance. The review also examines the key considerations and best practices associated with implementing these architectural patterns, along with the potential future directions for research and development.

Related work

The following section illustrates the few research papers related to this work. Florian Auer, et al [7] wrote the paper on microservices. The goal of this paper was to propose an evidence-based decision support framework for companies that need to migrate to microservices, based on the analysis of a set of characteristics and metrics they should collect before re-architecting their monolithic system. Jawad Sadeka, et al[8] wrote a paper on design and implementation of medical searching system based on microservices and serverless architectures. In this paper, the authors proposed system architecture to access clinical trials and medical devices in one place, providing corresponding visualization and analysis features.

In 2018, Kai Jander, et al [9] wrote paper on Defense-in-depth and role authentication for microservices systems. The authors made a study on authenticity and confidentiality of microservices and demonstrate that the calls have to be secured even for internal calls. In this paper Authors present how standard cryptographic primitives can be combined to provide a flexible communication system providing a high level of security even when easy to manage low-entropy authentication secrets are used. Svetoslav Zhelev, et al[10] made research on how to use microservices and event driven architecture for big data stream processing. The researchers outline microservices and EDA challenges, advantages and potential problems concerning big data stream processing. The research on Microservices for Data Analytics in IoT Applications Current Solutions Open Challenges and Future Research Directions by Safa Ben Atitallah [11] aims to address a survey about the adoption of the microservices paradigm for supporting data analytics in IoT applications. This work presents the first review paper that discusses how

microservices technology is being used in IoT applications gives a brief overview of IoT, data analytics, and microservices.

The research on Event-Driven Architecture Overview by Brenda M. Michelson[12] demonstrates the power of event-driven architecture in conjunction with service-oriented architecture, and as a broader architectural strategy. Lastly Ali ABOU KHALIL write a report on Event-Driven Perception for Robotics. In this report Author implemented circuit characterization utilizing simulation and experiments and detection of the direction and orientation of a sliding object on iCub skin patch using machine learning technique.

Smith et al., in 2018 explored the role of API Gateways in microservices architectures, demonstrating how they facilitate service communication, request routing, and load balancing. Johnson and Lee, in 2019 studied the impact of API Gateways on microservices security, highlighting their role in implementing security policies such as rate limiting, IP whitelisting, and authentication. Li et al., in 2019 examined the performance overhead of API Gateways in high-throughput scenarios, concluding that while they introduce some latency, their benefits in terms of traffic management and security outweigh the performance costs. Garcia et al., in 2020 conducted a study on the scalability of API Gateways, showing how they can handle increased loads through horizontal scaling and efficient resource utilization. Kim et al., in 2020 investigated the security features of API Gateways, such as token validation, encryption, and intrusion detection, and their effectiveness in protecting microservices. Alvarez et al., in 2021 analyzed the role of API Gateways in ensuring compliance with regulatory standards like GDPR and HIPAA by managing data flow and access control.

Jones et al., in 2018 explored Kafka's role in event-driven architectures[13], demonstrating its capabilities in decoupling services and enabling real-time data processing. Smith and Lee, 2019 studied the integration of Kafka with microservices, showing how it facilitates scalable and resilient communication between services. Li et al., 2019 examined the performance of Kafka in real-time analytics, highlighting its high throughput and low latency characteristics[14]. Garcia et al., 2020 conducted a comparative analysis of Kafka[15] and traditional batch processing systems, concluding that Kafka significantly improves the timeliness and accuracy of data insights. Kim et al., 2020 investigated Kafka Streams, a library for building stream processing applications, and its effectiveness in handling complex event processing tasks[16].

Garcia et al., in 2018 explored the role of Keycloak in enhancing IAM in cloud environments, emphasizing its support for single sign-on (SSO), identity brokering, and social login. Kim and Lee, 2019 conducted a study on the integration of Keycloak with microservices, demonstrating how it simplifies authentication and authorization processes in distributed systems. Singh et al., 2020 evaluated the security features of Keycloak, such as two-factor authentication (2FA), and its ability to comply with various regulatory standards like GDPR and HIPAA. Alvarez et al., 2021 analyzed Keycloak's role in implementing fine-grained access control policies and its impact on data security and compliance[17].

Jones and White, in 2018[18] conducted a comparative analysis of containerization tools and concluded that Docker provides significant advantages in terms of ease of use and ecosystem support. Li et al., in 2019 examined the performance overhead of Docker containers compared to traditional virtual machines, highlighting that Docker[19] offers near-native performance due to its lightweight nature. Kim et al., in 2020 proposed optimization techniques for Docker image management, reducing build times and image sizes through layer caching and efficient image layering. Smith et al., 2017 explored the use of Docker in microservices architecture, demonstrating how it improves deployment efficiency and scalability in cloud environments[20].

Summary

The above researches demonstrate the need and the advantages of using microservice over traditional architecture. Adoption of best practices for the development of microservices-based applications and Data management in microservices-based IoT applications. Logging and metrics are essential for distributed applications and the use of such platform is strongly recommended. Kafka is a good choice for event-driven broker.

Conclusion

This literature review highlights the foundational concepts, benefits, and challenges of microservices and event-driven architecture, as well as their application in tax systems. The integration of these architectures offers a robust framework for developing scalable, flexible, and responsive eTax systems. The insights gained from this review will inform the design and implementation of the proposed architecture for the eTax case study.

CHAPTER 3: RESEARCH METHODOLOGY

3.1. Introduction

This chapter focus on the software development methodology used to implement this research project. This chapter delves into the research approach employed to design a microservice architecture with an event-driven approach for the eTax. It outlines the specific methods used to gather data, analyze the existing system, and evaluate the effectiveness of the proposed architecture.

3.2 Data Collection Methods

3.2.1 Literature Review

A comprehensive review of existing literature on microservices, event-driven architecture, and agile methodologies was conducted. This provided the theoretical foundation and informed the design decisions.

3.2.2 Case Studies

Case studies from existing implementations of microservices and event-driven architecture were analyzed to identify best practices, common challenges, and solutions.

3.2.3 Interviews and Surveys

Interviews with domain experts and surveys of practitioners were conducted to gather qualitative insights on the practical aspects of designing and implementing microservices and event-driven architecture.

3.2.3 Document Analysis

Project documentation, including architecture diagrams, design documents, and sprint artifacts, was analyzed to assess the implementation and evolution of the architecture.

3.3 Research Design

The research design follows a qualitative approach combined with practical implementation and iterative evaluation. The methodology aligns with agile principles, focusing on adaptability, incremental development, and continuous feedback.

3.3.1 Research Questions

To understand better this project, we have selected different research questions based on type of practitioners of eTax. The list of those question can be found in the Annexes.

3.4 Software Development Model

3.4.1 Agile Software Development

This section provides some insights into our approach for implementing agile methodology in the development of microservices for our eTax project. As you know, Agile is a flexible and iterative methodology [14] that promotes collaboration, adaptive planning, continuous improvement, and rapid delivery of software solutions. It is particularly well-suited for projects involving microservices, which are small, independent, and loosely coupled components that work together to provide a larger system. By combining agile principles with the microservices architecture, we aim to enhance our development process and achieve better outcomes.

Agile Principles

Agile emphasizes frequent communication, customer collaboration, and the ability to respond to change. We will adhere to the following agile principles in our microservice development:

- ***Individuals and interactions over processes and tools***: We prioritize effective communication and teamwork to achieve project success.
- ***Working software over comprehensive documentation***: Our focus is on delivering functional microservices that provide value to end-users.
- ***Customer collaboration over contract negotiation:*** We value continuous feedback from stakeholders to ensure the developed microservices align with their requirements.
- ***Responding to change over following a plan***: We embrace change and adapt our approach as needed to maximize the value of the microservices.
- ***Continuous Improvement:*** Agile methodology promotes a culture of continuous improvement. Team members regularly reflect on their work, identify areas for improvement, and make necessary adjustments.

3.4.2 The Agile Software Development Life Cycle (SDLC)

The Agile Software Development Life Cycle (SDLC)[15] is an iterative and incremental approach to software development that emphasizes flexibility, collaboration, and continuous improvement. It is a popular methodology used in the software industry to manage and deliver projects more effectively.

The Agile SDLC typically consists of the following phases:

- ***Requirements gathering:*** In this phase, the project team works closely with stakeholders to identify and prioritize the requirements for the software product. User stories and product backlog are often used to capture and manage requirements.
- ***Planning:*** The team plans the project iterations or sprints based on the prioritized requirements. The duration of each iteration is typically short, usually ranging from one to four weeks. The team also estimates the effort required for each user story or task.
- ***Design:*** During this phase, the team designs the architecture, system components, and user interfaces. The focus is on creating a high-level design that supports the requirements and enables flexibility for future changes.
- ***Development:*** The actual coding of the software takes place in this phase. The team works on developing small, incremental features or user stories. Continuous integration and automated testing are often employed to ensure the quality of the code.
- ***Testing:*** Testing is an integral part of Agile SDLC. Each iteration is thoroughly tested to identify defects and ensure that the software meets the specified requirements. The team performs various types of testing, including unit testing, integration testing, and user acceptance testing.
- ***Delivery:*** At the end of each iteration, a potentially shippable increment of the software is delivered. This allows stakeholders to see and provide feedback on the working software. It also enables early value delivery and helps in managing risk.
- ***Review and retrospective:*** After each iteration, the team conducts a review to assess the progress, gather feedback, and identify areas of improvement. A retrospective meeting is also held to discuss what went well, what could be improved, and any necessary adjustments to the process.

These phases are repeated in a cyclical manner throughout the project, with each iteration building upon the previous ones. The Agile SDLC promotes close collaboration between the development team and stakeholders, encourages adaptability to changing requirements, and prioritizes the delivery of working software at regular intervals.

3.4.3 Reason for using Agile Software Model

The Agile model was selected for this research project for the following reasons:

- This research project consists of developing taxes functionalities which depends each other's. Agile will help us to divide these functions into small functions which can be developed and released in short time.
- This project involves many stakeholders such as taxpayers, tax agents and the 3rd parties. It is very important to regular interact with them in order to present them the work done at time and if there is a need of adjustment it can be done at early stage.
- This project consists of developing microservices with the tax office team. Agile model will help the development and also in software management.
- Agile guide on the creation of team shat should be foster productivity and collaborative.

By adopting Agile methodology in our research project, we aim to achieve greater flexibility, faster delivery, and increased customer satisfaction. Collaboration and effective communication within the team and with stakeholders will be critical for success. Let's embrace the Agile mindset and work together to deliver high-quality microservices that meet the needs of our users.

3.5 Analysis and Validation

3.5.1 Iterative Analysis

The architecture and implementation were continuously analyzed and refined through Agile iterations. Each sprint produced a potentially shippable product increment, allowing for regular **validation against requirements.**

3.5.2 Performance Testing

Performance testing was conducted to ensure the microservices architecture met the required scalability and responsiveness standards. This included load testing and stress testing of individual services and the overall system.

3.5.3 Usability Testing

User feedback was solicited through usability testing sessions to validate the effectiveness and efficiency of the system from the end-users' perspective.

3.6 Ethical Considerations

Ethical considerations included ensuring the confidentiality and privacy of interview and survey participants. Data was anonymized where necessary, and informed consent was obtained before conducting interviews.

3.7 Limitations

The research methodology is primarily qualitative and iterative, which may not cover all potential scenarios and challenges in a production environment. Additionally, the reliance on agile practices may lead to differing levels of detail and completeness in various iterations.

3.8 Summary

This chapter outlined the research methodology for designing microservices and event-driven architecture using agile software development. The approach integrates theoretical insights with practical implementation, emphasizing iterative development, continuous feedback, and adaptation to evolving requirements.

CHAPTER 4: SYSTEM DESIGN AND ANALYSIS

4.1 Introduction

The design process encompasses a series of structured steps to create an effective, scalable, and user-centric solution. This process involves translating system requirements into detailed system specifications and architectural models.

4.2 Requirement Analysis

After analysis the eTax ecosystem, the following functions was considered as the main functions of the system:

- Taxpayers' signup
- Tax account creation
- Taxes registration
- Tax Declarations
- Declaration payment

4.3 Conceptual Design

The expected services are integrated in the following architecture. At the start a taxpayer sign up into a portal application to get authentication. Then the taxpayer login into portal application and make tax declarations. After submitting the declaration, taxpayer will submit the declaration to the declaration microservice and receive the declaration number in the return. With the declaration number, taxpayer can pay this declaration into portal application or any other external system by sending the payment details to the payment microservice. The payment microservice will then handle the payment process and up to the successful it will return the payment reference. At the same time the payment microservice will update taxpayer's account into taxpayer microservice and also send the notification to the taxpayer via a notification service. The following figures illustrate the high-level architecture of developed eTax ecosystem and use case diagram shows how the actors(users) interact with eTax system. The picture shows the taxpayers tasks, the agents' tasks, the external system tasks and the payment collectors' tasks.

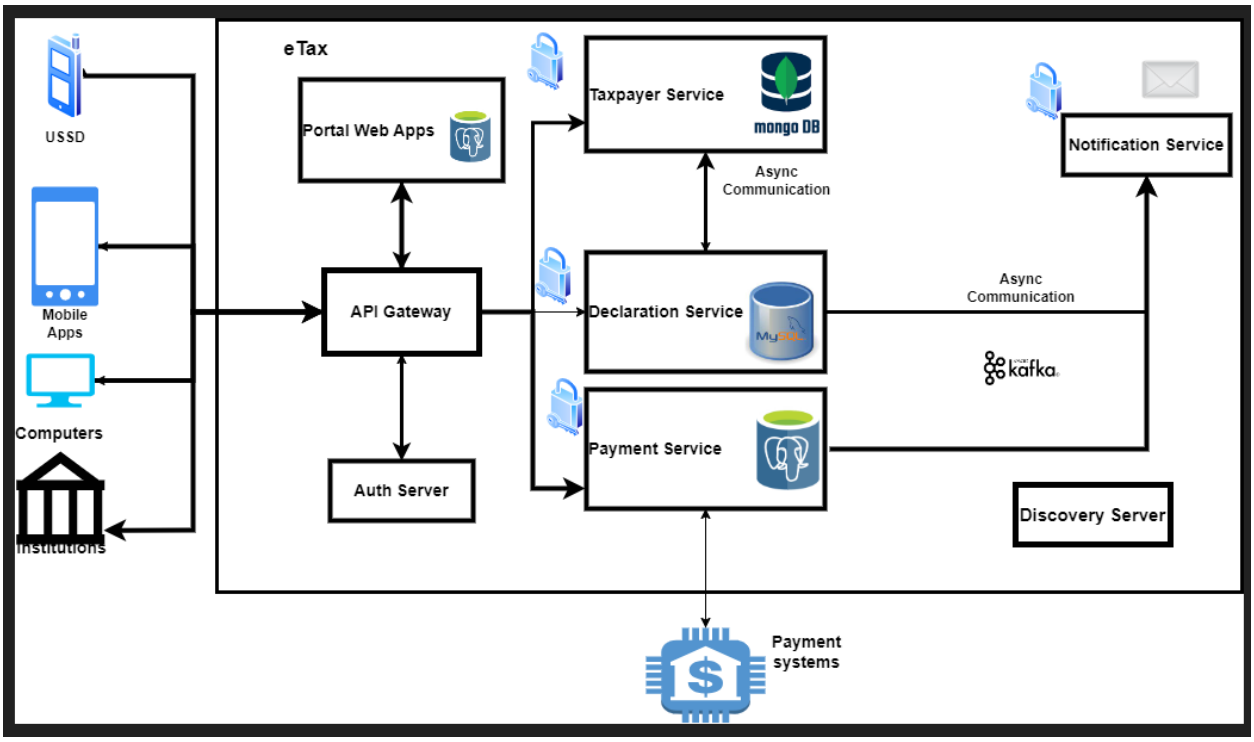

Figure 1: System Architecture of eTax

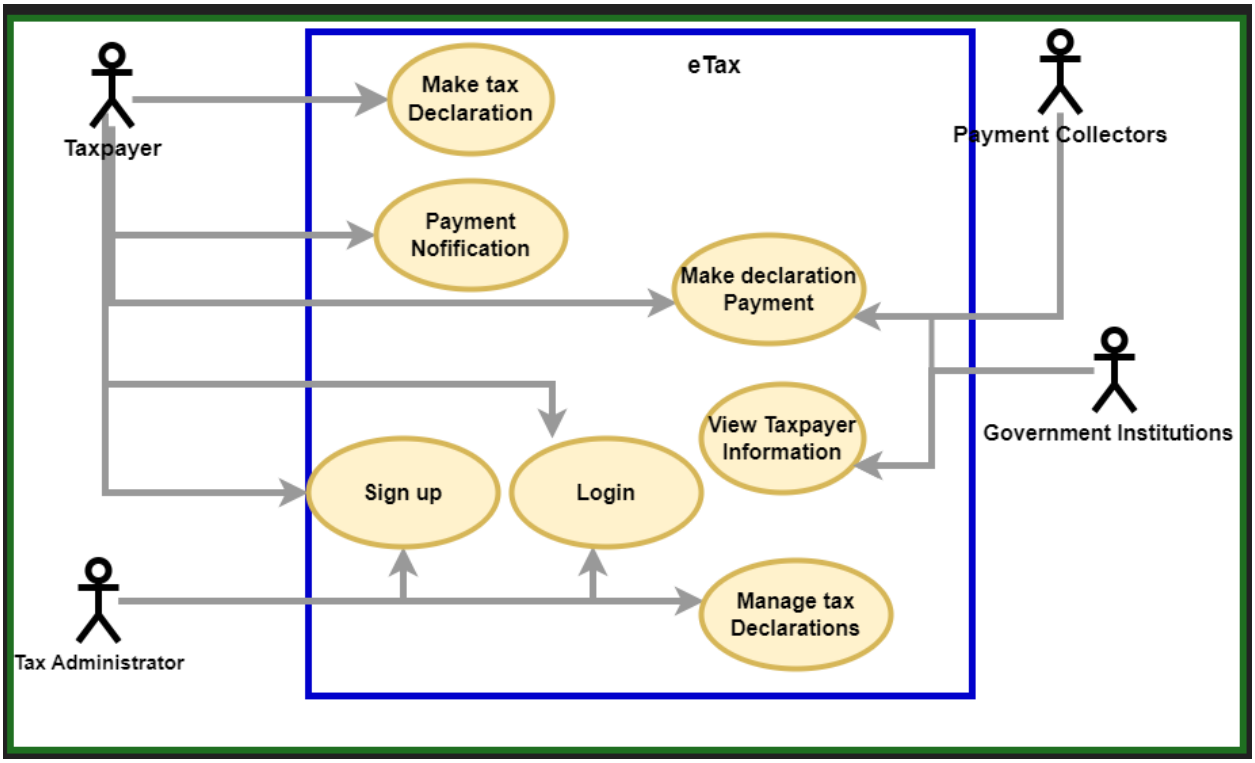

Figure 2: Use case diagram of eTax.

4.4 Detailed Design

In this project we developed seven microservices for eTax platform named as api-gateway service, discovery-server service, notification-service, payment-service, portal-service, taxpayer-service, declaration-service using spring boot framework. Among these microservices, some are stateless like api-gateway, discovery-server, and notification-service. The payment-service and declaration-service use MySQL database, the portal-service used PostgreSQL database and has a graphic user interface developed using Thymeleaf framework and finally we have taxpayer-server with use the Mongo database. The eTax platform functionalities start by a taxpayer or an agent accessing portal-service application and make a sign. After a user fill sign-up form and submit it. An authorized Tax collection agent will login into eTax portal and approves the sign-up information. If the request belongs to the taxpayer, the approval process will start from portal service send a request to the taxpayer service via a synchronous communication to check if the submitted TIN number exist into the taxpayer database via Taxpayer service. To send this request we used Spring WebClient interface from Spring Web Flux module. Note that the data in portal service are stored in Postgres DB server while the data from taxpayer service are stored in Mongo database. To manage inter process communication, we have developed a discovery service using Spring Cloud Netflix library from Spring Cloud Project. For accessibility and security concern we have implemented an API gateway service using Spring Cloud Gateway Library into our eTax to handle all requests coming from users. The developed API gateway service is protected by Keycloak security server. All request should be authenticated by this server. After user login, the eTax shows the list of menus depending on the type of user who logged in.

We have grouped them into two components as Taxpayer Portal and Tax collection Agent portal.

4.4.1 Tax Collection Portal

The tax collection agent is a person who work for tax collection institution. Once logged into our portal service, he or she has access to the following microservices as per the following figure shows.

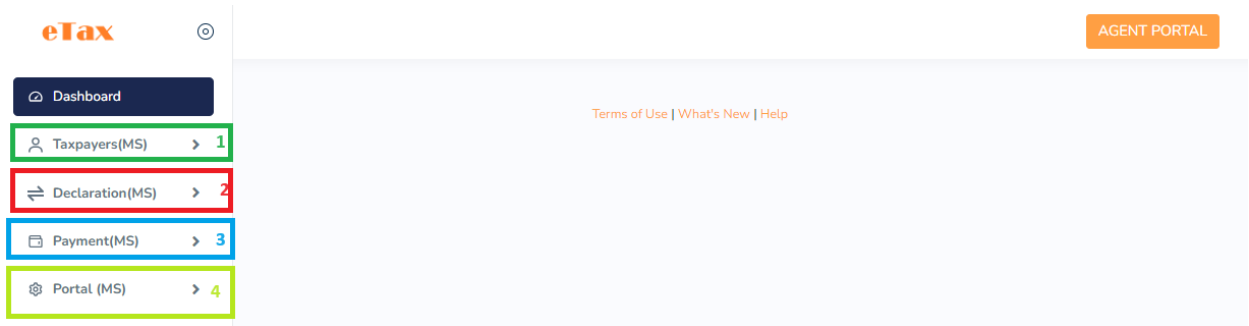

Figure 3: Tax Collection Portal Menus

4.4.1.1 Taxpayer Microservice Menu (TMS)

The Taxpayer Microservice Menu (TMS) gives access to the taxpayer service. The taxpayer service is an independent service running on its own container named as **taxpayer-service** and is using Mongo database also with its own container named **mongo-taxpaye**r with port **27018** as storage. This service is responsible to manage taxpayers' information like Tax Identification Number (TIN), names, address and etc. The tax collection agent can add, modify taxpayer's information from the Portal service. The following figure show the list of taxpayers

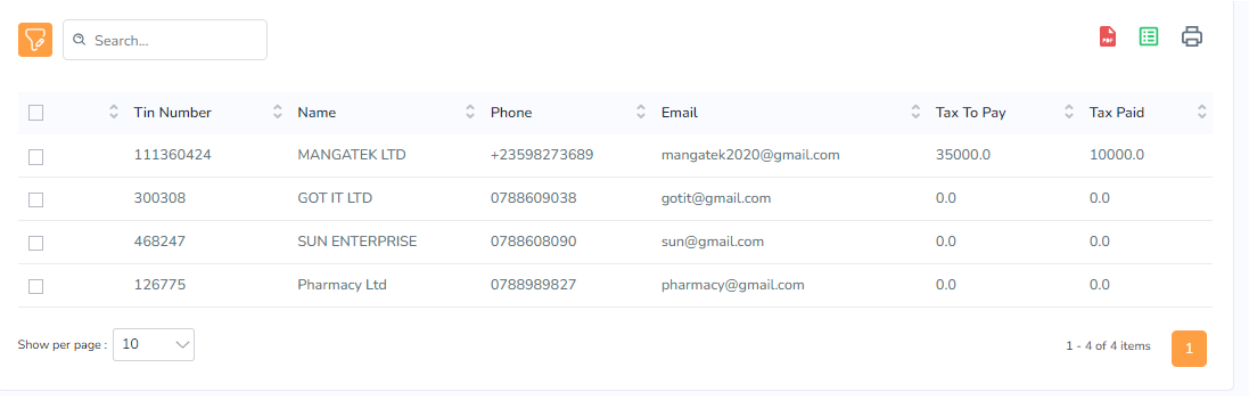

Figure 4: List of taxpayers on web portal

As mentioned, the declaration service use Mongo database via a self-container called mongo-taxpayer. The following picture show the list of taxpayers in the database using MongoDB compass tool.

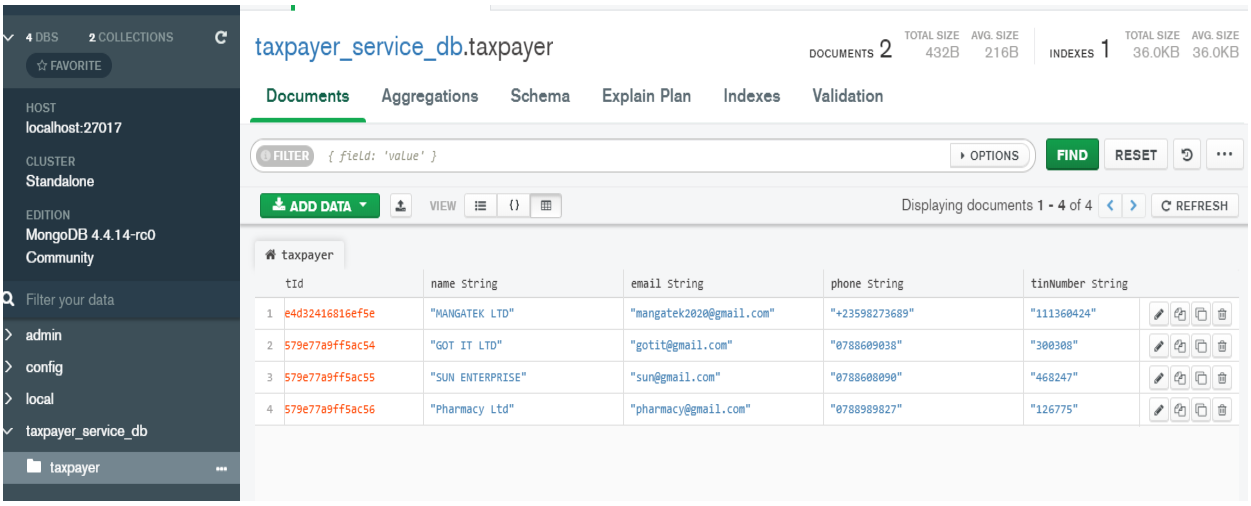

Figure 5: List of taxpayers on MongoDB Compass

The taxpayer service is also update by Kafka event every time taxpayer made declaration or payment. We will explore this feature more in coming section. The following figure shows the view of one taxpayer account details called “MANGATEK LTD” with TIN “111360424”. This tax account will be affected when taxpayer makes declaration and payment. Those 2 transactions will be discussed in the Taxpayer Portal section.

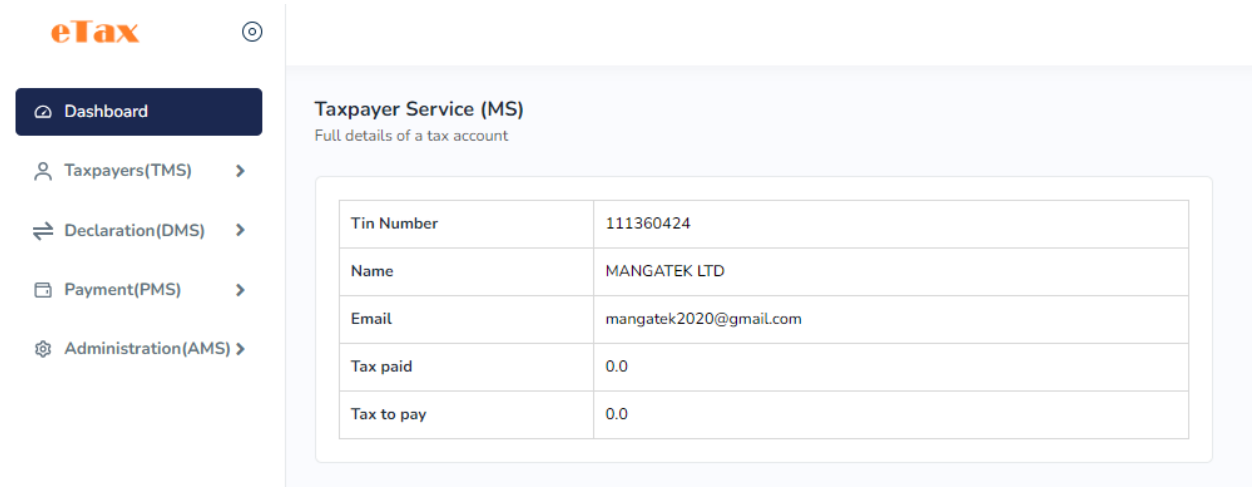

Figure 6: Tax Account Details view in Taxpayer Microservice

4.4.1.2 Declaration Microservice Menu

The Declaration Microservice Menu gives access to the declaration microservice. The declaration microservice is an independent container runs on the port **8484** and has MYSQL database as storage on the port **3307.** This service is being called every time a taxpayer wants to

make a payment and updated on the payment successful. The tax collection agent can view the details of each declaration.

4.4.1.3 Payment Microservice Menu (PMS)

The Payment Microservice Menu gives access to the payment microservice. The payment microservice is an independent container and has Postgres database as storage**.** This service is being called every time a taxpayer makes a payment.

4.4.1.4 Administration Menu (AM)

The Administration Menu gives tax collector agent access to the Portal Microservice. This portal has a graphical user interface (UI) which helps to manage the following activities:

- View List of sign-up requests to be approved.
- Approve/Reject sign-up requests
- View List of users.
- Manage Declaration periods
- Manage Taxes
- View tax declarations from taxpayer portal
- View payments from taxpayer portal

4.4.2 Taxpayer Portal

As mentioned in the previous section, this project has two types of users, tax collection agent and taxpayer user. In this section we will focus on the taxpayer user functionalities. When a taxpayer is logged in into his portal it has access to the following menus:

- Declaration Menu
- Payment Menu

The following figure shows the taxpayer menus as described previously.

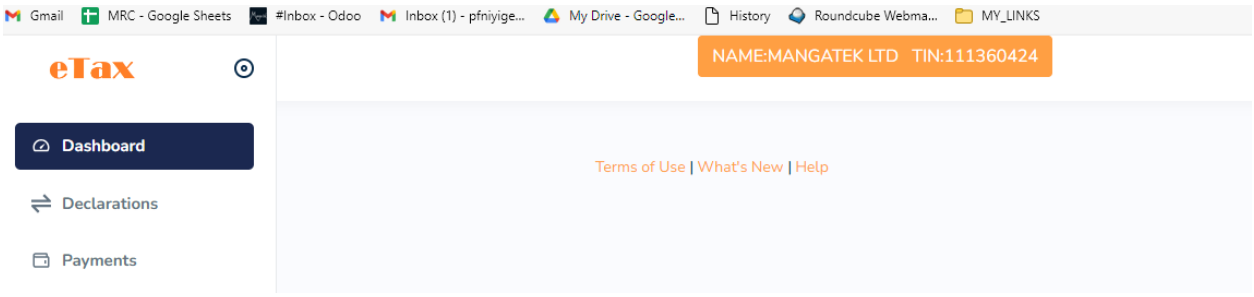

Figure 7: Taxpayer menus

4.4.2.1 Declarations Menu

Tax declaration, also known as tax filing or tax return, is a process where individuals or entities report their income, expenses, and other financial information to the government for the purpose of calculating and paying taxes. In this project we handle three monthly taxes such as Value-added tax (VAT 18%), Withholding Tax (WHT 15%) and Pay As You Earn (PAYE 3%). By clicking to this menu, logged taxpayer has access to the list of its declarations. Taxpayer can add new declaration, submit declaration and approval the saved declaration. The following figure show tax declaration form of our project. Once the declaration form submitted, the declaration is saved into taxpayer portal.

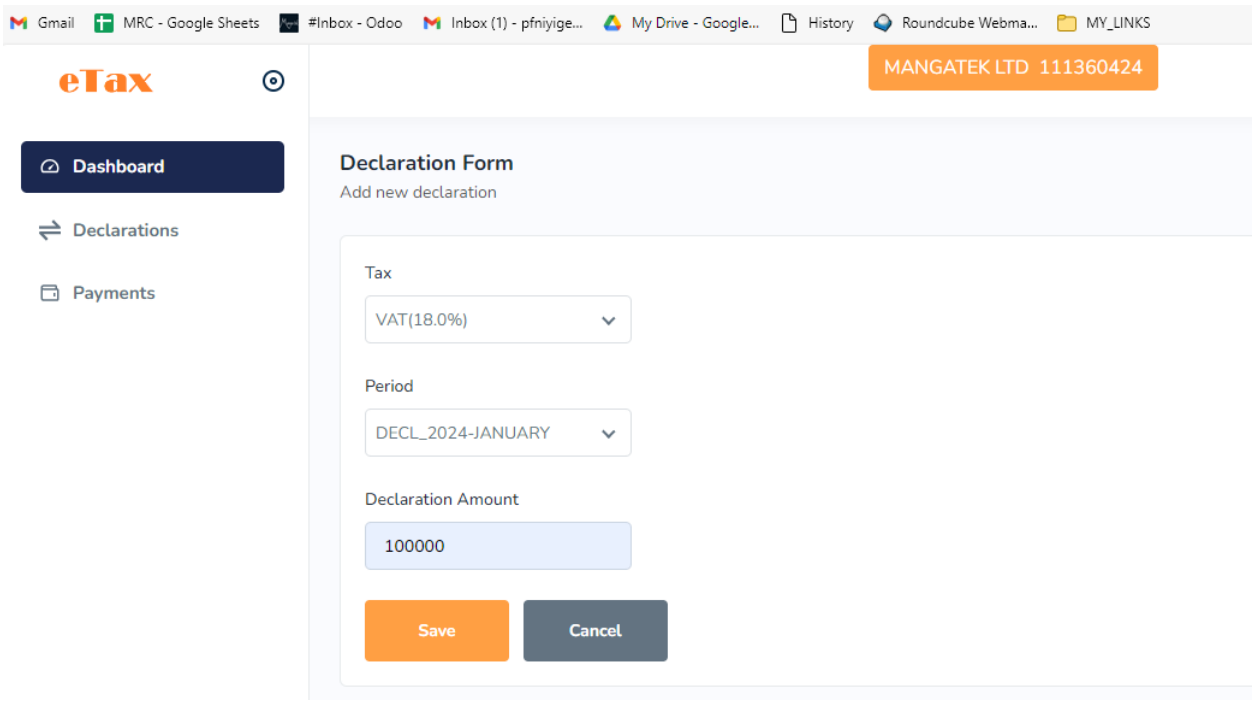

Figure 8: eTax declaration form

At this time the created declaration has pending status, and taxpayer can modify it before validation. The following figure shows declaration details and the action buttons for approval or reject.

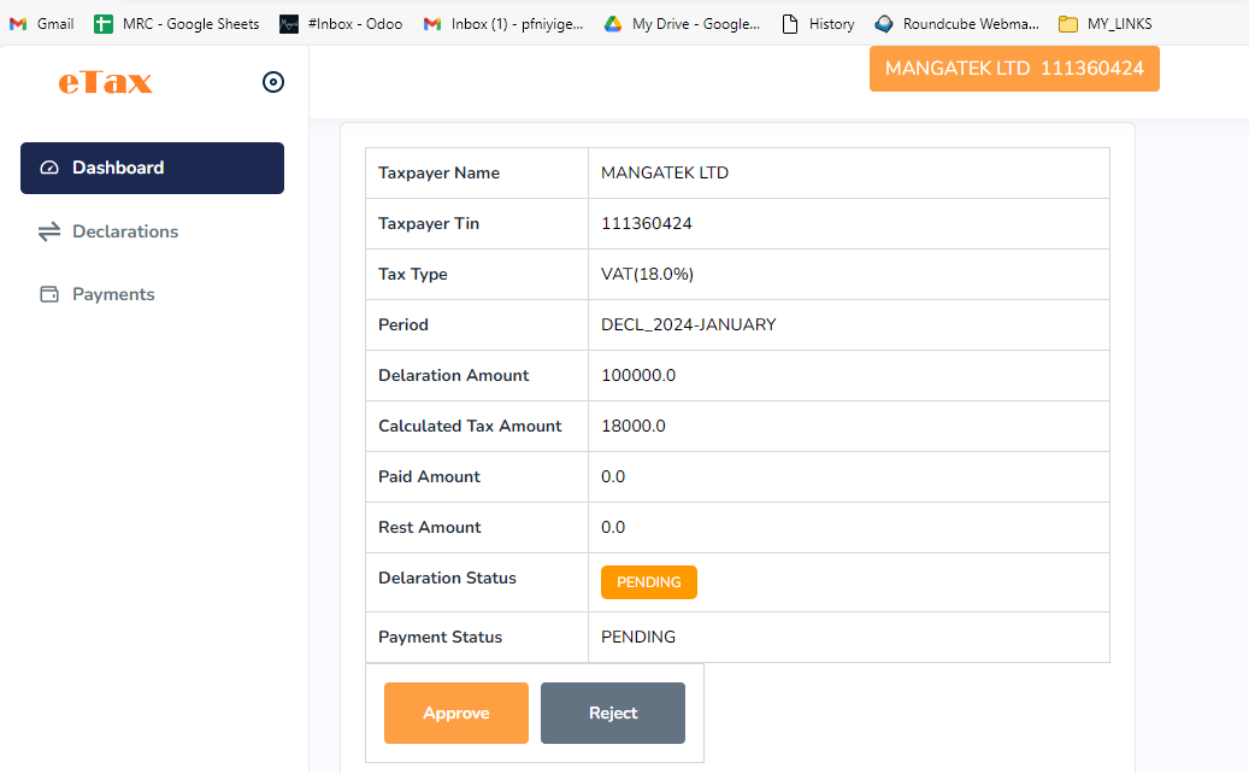

Figure 9: eTax declaration details

Declaration Approval

At this step when taxpayer approves the declaration, the portal service sends this declaration to the declaration microservice via a post method thought rest api. And this declaration is available for payment. The following figure shows declaration details inside declaration microservice.

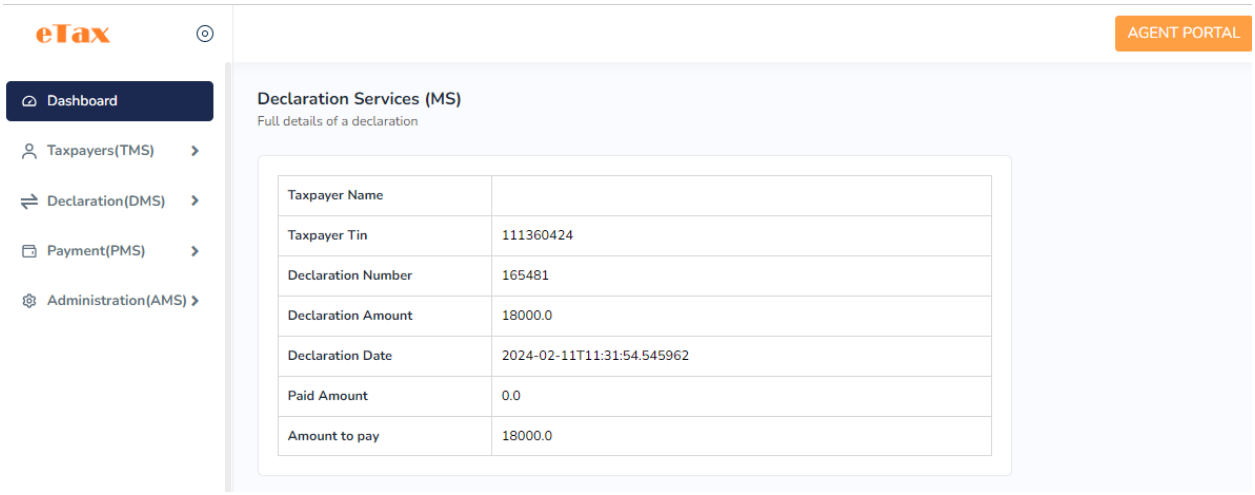

Figure 10: eTax declaration view in DMS

At the same time, once declaration microservice receives declaration transaction from portal, it updates taxpayer's tax account via an Kafa event. Even if the taxpayer microservice is down, this event keeps the data until the service is available to consume this event. The following figure show taxpayer's tax account after approval process.

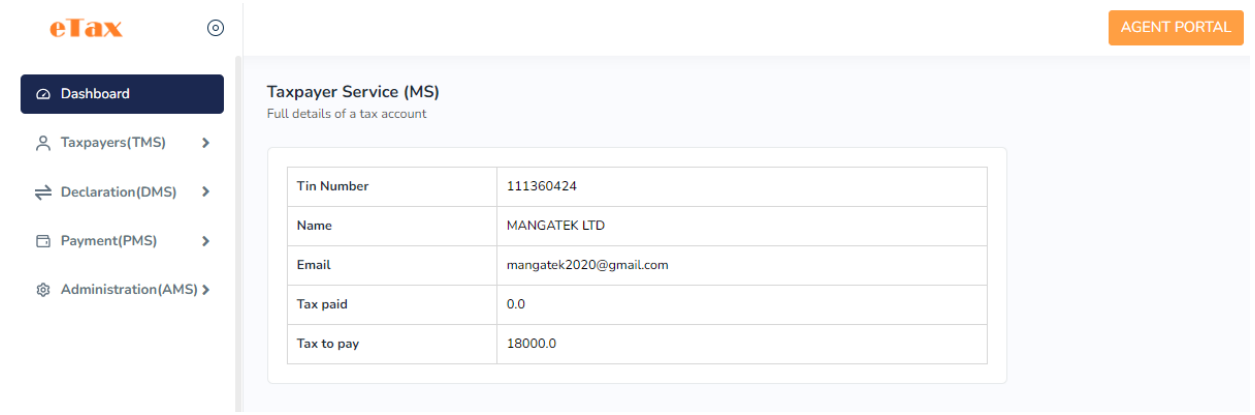

Figure 11: eTax tax account details after approving declaration.

Declaration Payment

After the process of declaration approval, the next step is to make payment. With this project we have integrated our eTax platform with one of telecoms payment gateway called MOOV Mobile Money. Once the taxpayer selects this payment, the request is sent to the payment microservice to handle payment, then the payment will select the gateway to be used. The following figure shows the payment form.

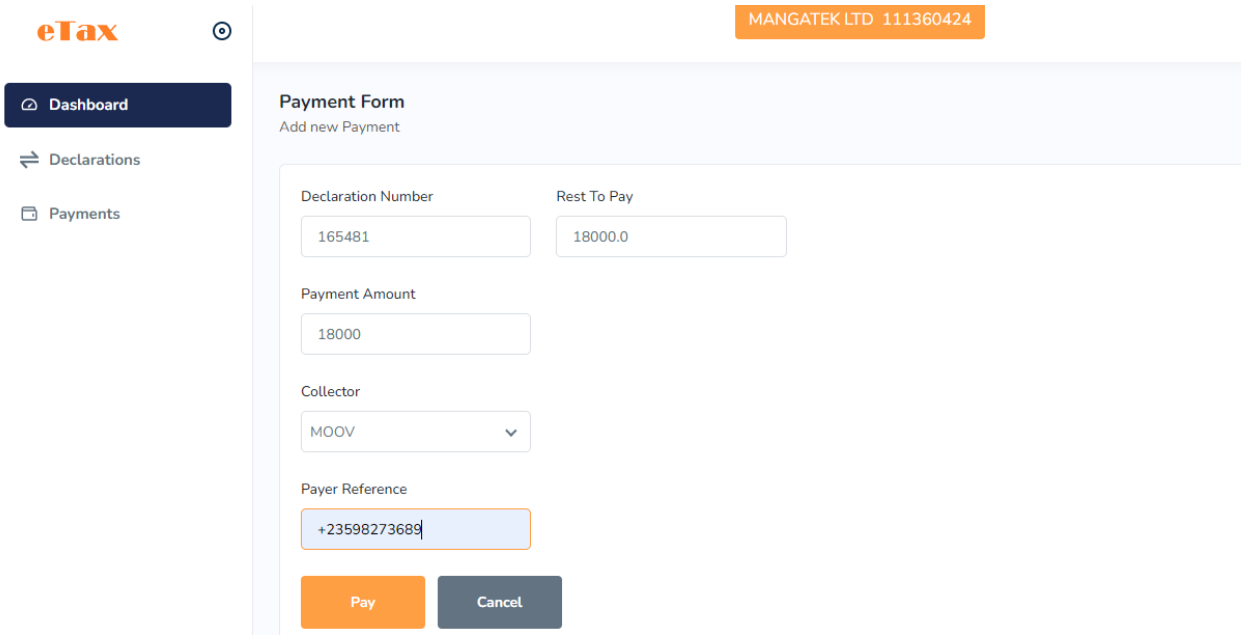

Figure 12: The payment form

On the payment successful, the payment service will update taxpayer's account and declaration service via Kafka event and also will send another Kafka event to the notification microservice for notification message.

The following figure shows tax account details after payment.

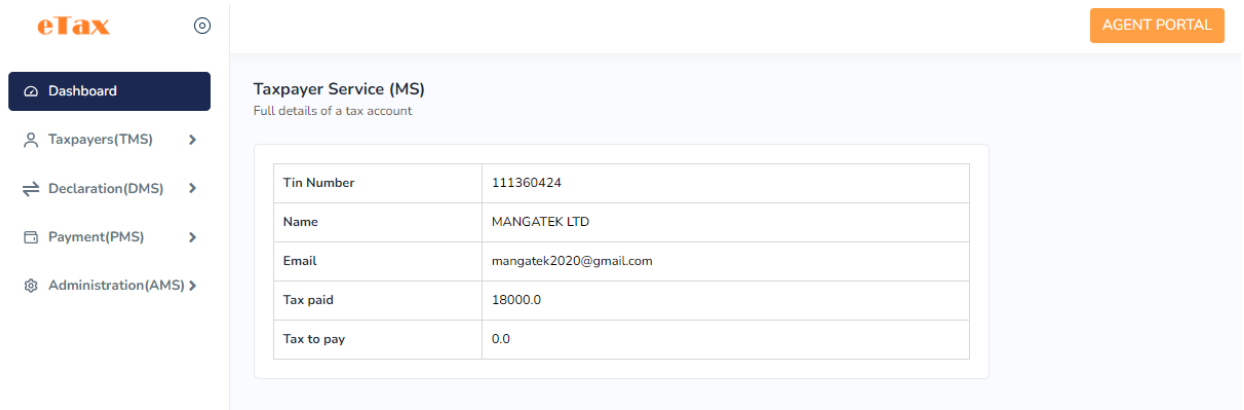

Figure 13: Updated tax account details after payment

The following figure shows Kafka topic for notification.

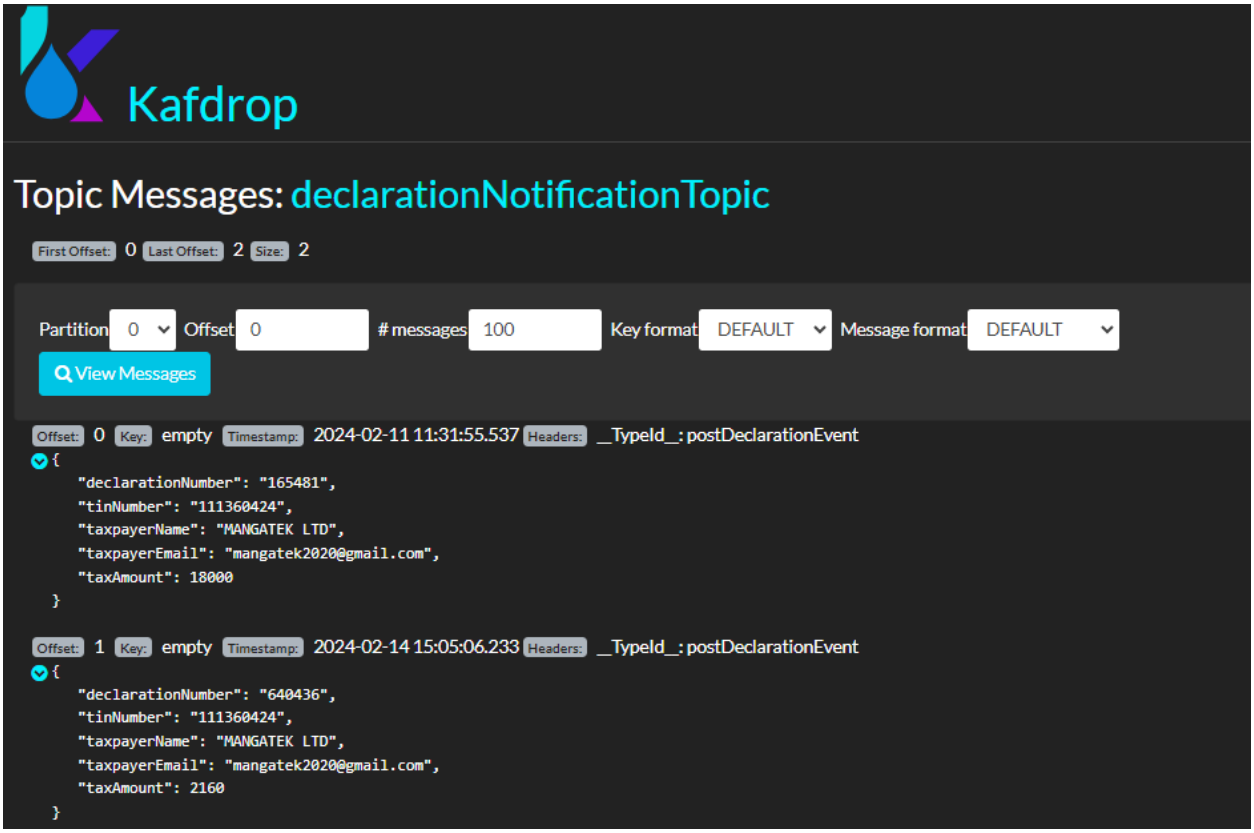

Figure 14: Topic Messages" declarationNotificationTopic"

4.4.2.2 Payments Menu

Payment's menu gives access to the list of payments. With this list a taxpayer can view the details of each payment made.

4.5 Metadata Repository

The developed ecosystem has 4 repositories as it is microservices architectures. Those repositories are independent each other but share some references.

4.5.1 Portal Repository

Portal service is a web application which store data into a Postgres Database. The following figure show the entity relationship diagram of portal repository.

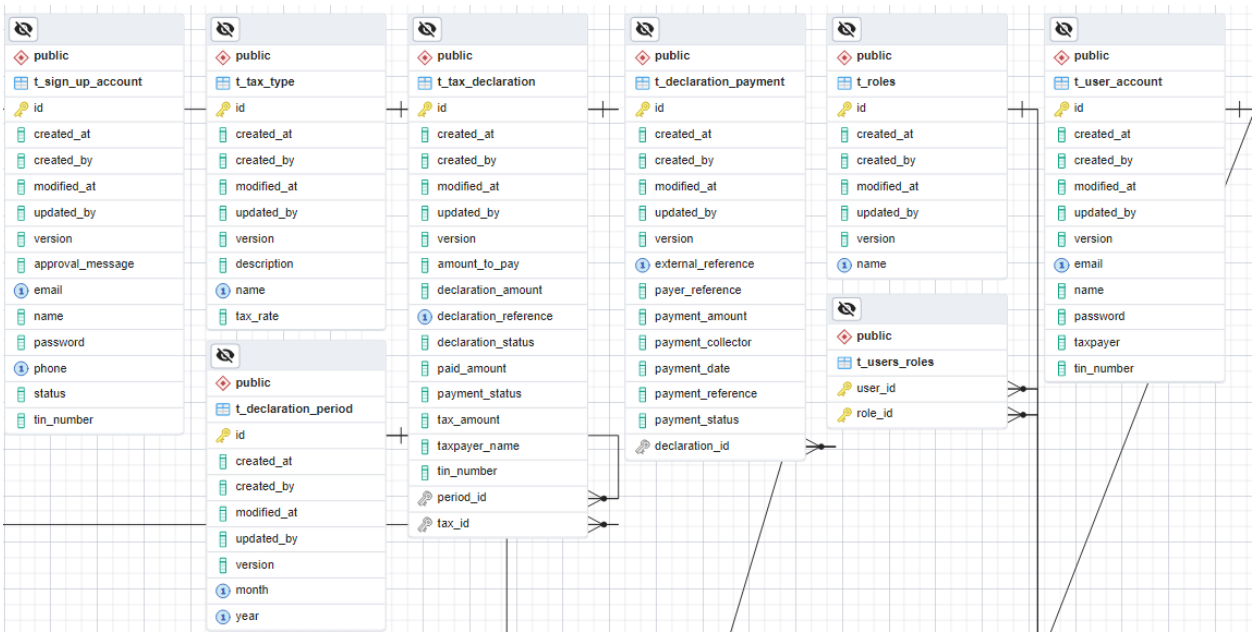

Figure 15: Portal ERD Diagram

4.5.2 UML class diagrams of eTax

ETax developed is a multi-database. Some of the databases are non-relational database. With the following figure we illustrate the class diagrams for domain models.

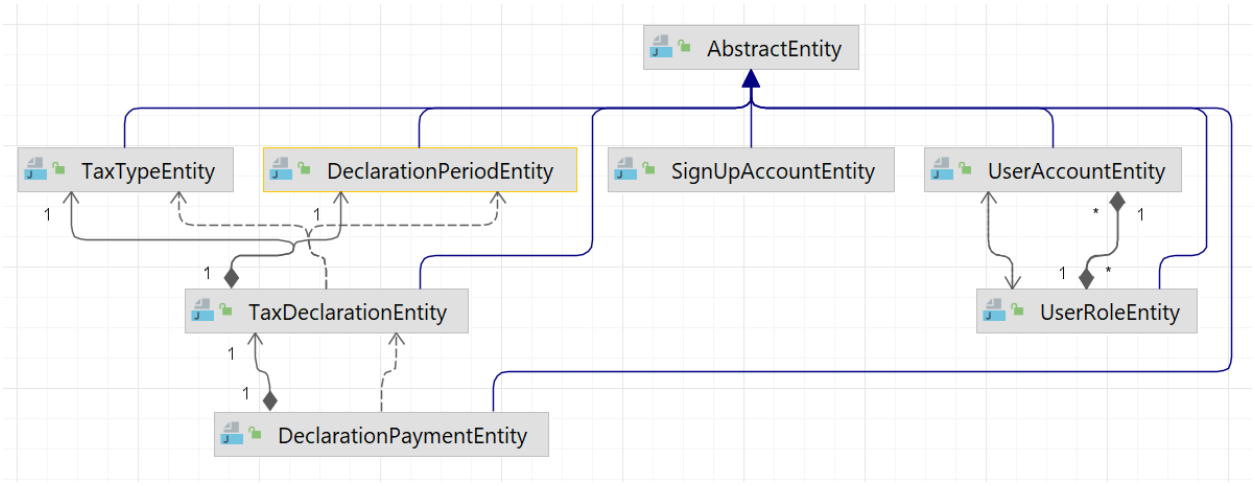

Figure 16: Class diagram for portal domain model.

4.6 Technology Selection

4.6.1 Docker

Docker is a platform designed to make it easier to create, deploy, and run applications by using containers[21]. Containers allow a developer to package up an application with all parts it needs, such as libraries and other dependencies, and ship it all out as one package. This ensures that the application runs seamlessly on any computing environment.

4.6.2 Docker Compose

Docker Compose is a tool that allows you to define and manage multi-container Docker applications. It enables you to define an entire application stack, including services, networks, and volumes, in a single YAML file called "**docker-compose.yml**"[22]. This file acts as a configuration blueprint for your application.

4.6.3 Docker hub

In this research project, the developed microservices was converted into docker images and pushed to the docker hub for accessibility and easy deployed to any OS container. The following figure shows our developed images into docker hub.

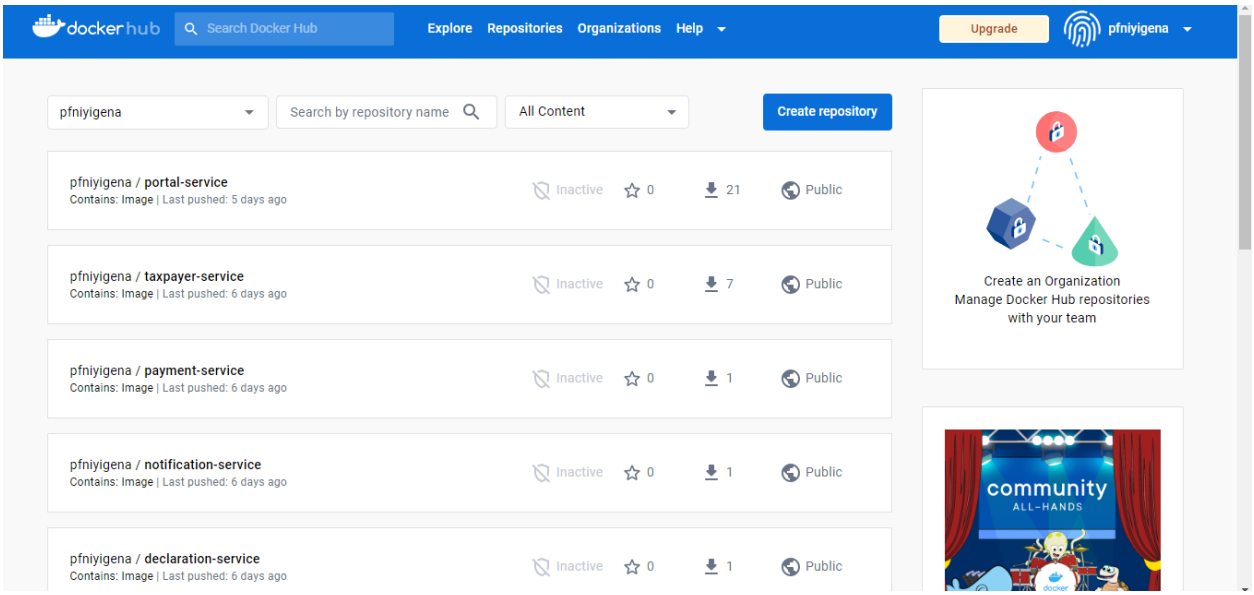

Figure 17: Docker hub of eTax images

Source: https://hub.docker.com/

4.6.4 Docker Desktop

To administrate the developed images into our OS, we have a docker desktop installed into our windows machine. Form this application, we can start, restart, stop our containers. The following figure show eTax containers into a windows machine.

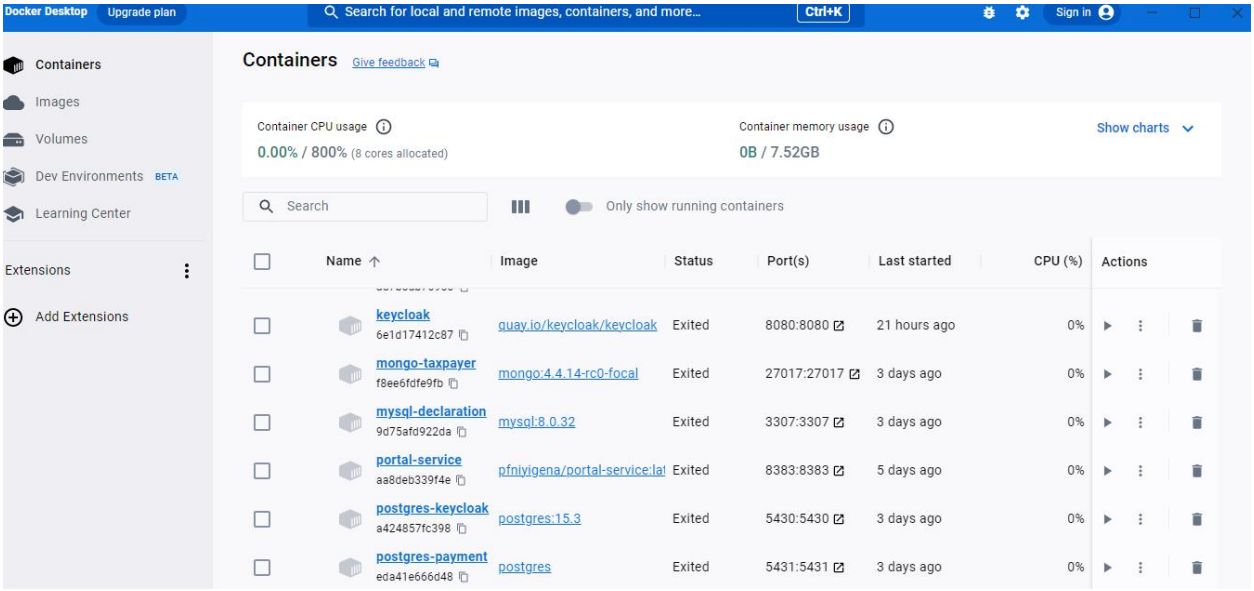

Figure 18: Docker desktop of eTax in action

4.6.5 Zipkin

Our developed eTax eco-system uses a distributed monitoring and tracing tool called Zipkin. With Zipkin interface, we are able to monitor and trace the requests exchanges among our eTax eco-system.

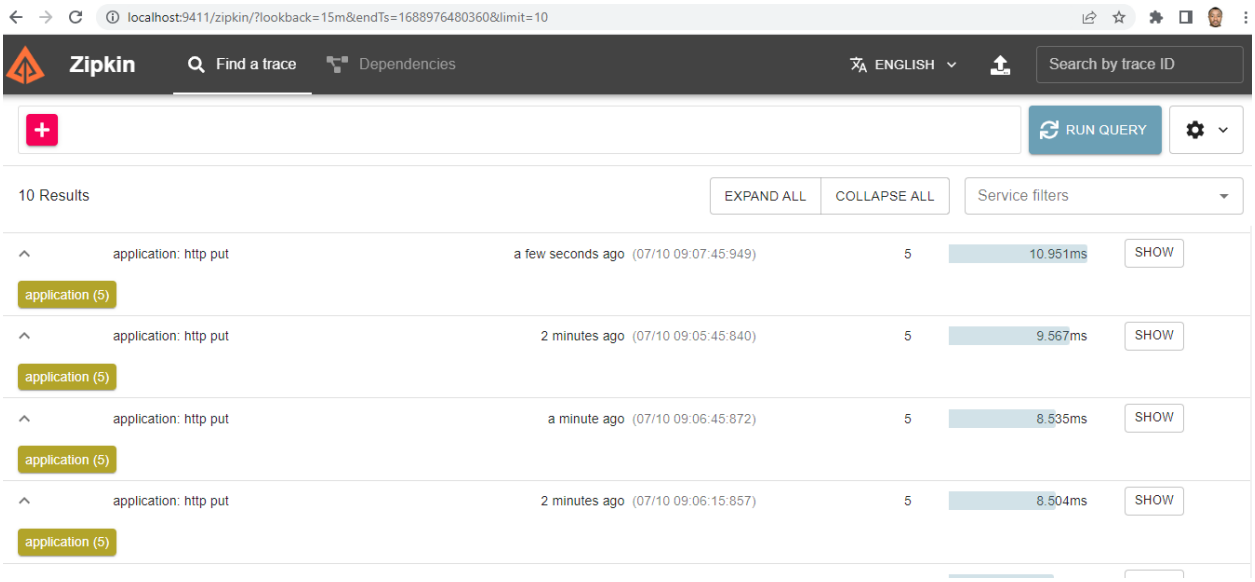

Figure 19: Zipkin interface of eTax in action

4.6.6 Keycloak Server

For authentication of our eTax, we have installed a Keycloak server which manage authentication of our gateway by providing OAuth 2.0. All requests coming from outside of our eTax need to be authenticated by this server. The following figure shows our configurations details in Keycloak server.

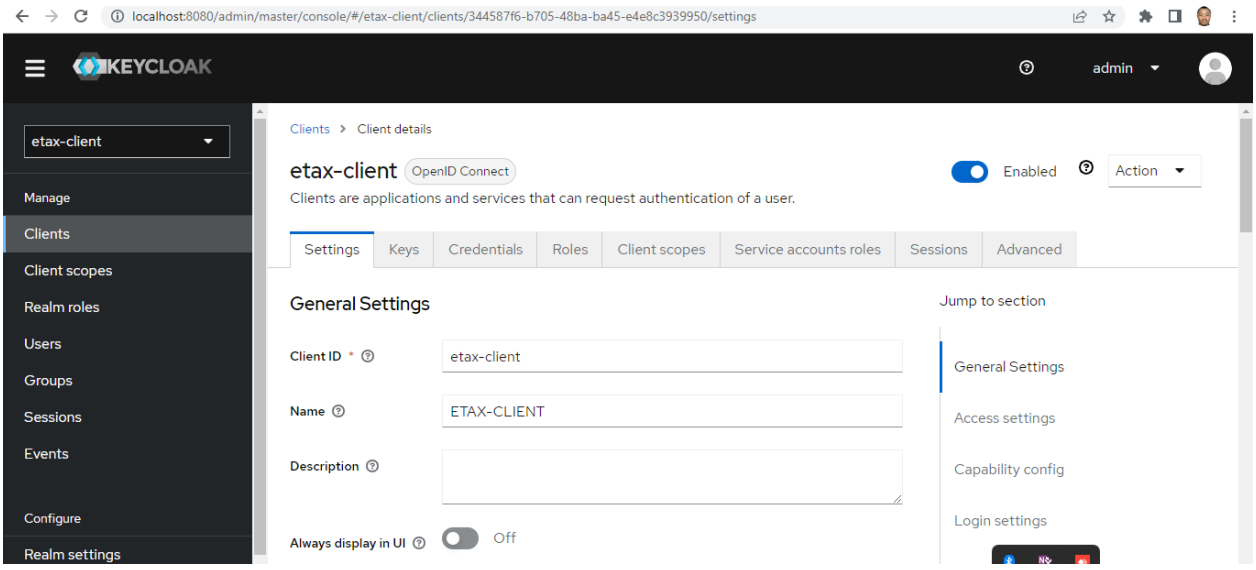

Figure 20: Keycloak server configuration of eTax

4.6.7 List of tools used to develop eTax

The development of eTax eco-system involves of multi complex of tools. The following table illustrate the list of tools used in this project.

No	Name	Description	Version
1	Spring boot framework	Java-based enterprise framework used to create different types of enterprise applications	3.0.4
2	IntelliJ IDEA	Integrated Development environment for developing computer software.	2021.3(Ultimate Edition)
3	Maven	Open-source tool to build automation while developing java application	3.9.2
4	Thymeleaf	Java XML/XHTML/HTML5 template engine that can work both in web and non-web environment.	3.1.1
5	Spring Cloud	An event-driven framework to build microservice applications.	2022.0.3
6	MongoDB	A NoSQL relation database tool documented oriented.	6.0.6
7	MongoDB Compass	A Graphic User interface tool for MongoDB.	1.38.2
8	PostgreSQL	A relational database tool based on Objects	12.0
9	PgAdmin	A Graphic User interface tool for PostgreSQL	4.7.0
10	MySQL	A relational database tool	8.32.0
11	MySQL Workbench	A Graphic User interface tool for MySQL	8.32.0
12	Git	An open-source tool used as code control version. It helps to track changes of codes	2.35.0

		and in collaboration among software developers	
13	GitLab	An open-source tool used to visualize a git repository	14.0
14	Postman	An open-source tool used to build and test API	10.15
15	SonarQube	An open-source platform for inspections of codes quality and bugs detection	3.4.0
16	Docker	A platform developed to help developers to make images and manages containers in OS virtualization	24.0.4
17	Jib	Java library from Google used to create docker image from source codes without using docker file	3.2.1
18	Docker hub	It is an online repository where developer can push their container images.	N/A
18	Docker Desktop	A tool that contains local docker images	24.0.4
18	Zipkin	An opensource distributed monitoring and tracing tool	3.0.4
19	Keycloak	An opensource software product to allow single sign-on with identity and access management aimed at modern applications and services	2.3.0
20	Kafka	An open-source project use to design and deploy messaging queues	7.0.1

Table 1: List of tools used to develop eTax

4.7 Usability Testing

To complete the eTax eco-system, we have selected some users to interact with our developed project. Five taxpayers, two tax professionals and software developers were chosen and they have registered into the platform and performed some transactions and tests. The taxpayers were happy to see one platform which manages the declarations and the payment with a single access. The feedback from tax professionals was also positive due to access of taxpayer registries with all details like tax accounts, real time payments transactions. As this platform gives availability to communicate with any external platform via an API, developer was posting some transaction from different tools like Postman, SoapUI.

4.8 Conclusion

In this section, we demonstrated the proof of concept of our eco-system. All developed microservices are interacting each other via synchronous and asynchronous communication. Tax declaration was submitted and paid using our service and notification was sent to the taxpayer email.

CHAPTER 5: RESULTS AND ANALYSIS

5.1 Introduction

5.2 API Gateway

This section presents the results of the implemented API Gateway of our project, focusing on its implementation, performance evaluation, and analysis of the collected data. The analysis provides insights into the effectiveness, efficiency, and overall impact of the API Gateway implementation.

System Architecture: The API Gateway system was successfully implemented following the architectural design described in Chapter 4. The components, including web portal, taxpayer service, declaration service, payment service, notification service, were integrated and configured as planned.

Response Time Analysis: To measure the response time of the API Gateway, requests were sent to various endpoints, and the time taken for processing and response was recorded. Statistical analysis, including mean, median, and standard deviation, was performed to assess the consistency and efficiency of the response times.

Security Analysis: The effectiveness of the implemented authentication and authorization mechanisms was assessed through penetration testing and analysis of access logs. Security vulnerabilities and potential improvements were identified.

5.3 Event Driven Architecture

This section presents the results obtained from the implementation and evaluation of the Event-Driven Architecture (EDA) using Kafka. The analysis focuses on key performance metrics, system behavior, and the overall effectiveness of Kafka in supporting event-driven systems.

Performance Metrics: One of the primary metrics evaluated was the system throughput, measured in messages processed per unit of time. Results indicate a substantial improvement in throughput compared to traditional monolithic architectures. Kafka's ability to handle a high volume of events concurrently was evident, showcasing its suitability for real-time event processing.

Latency: Latency was another critical metric analyzed to understand the responsiveness of the system. The average latency observed during peak loads demonstrated a negligible impact on overall system performance. Kafka's design principles, such as partitioning and parallel processing, contributed to maintaining low latency even under heavy workloads.

Scalability: Scalability tests were conducted to assess Kafka's ability to handle an increasing number of events and adapt to growing demands. Results revealed a linear scalability, highlighting Kafka's effectiveness in scaling horizontally. This is particularly advantageous for applications requiring elasticity in handling varying workloads.

Fault Tolerance and Reliability: The analysis of fault tolerance and reliability demonstrated Kafka's robustness in maintaining data integrity and system availability. Failover scenarios were simulated, and Kafka ensured seamless recovery with minimal data loss. This resilience is crucial for mission-critical applications where downtime is not acceptable.

5.4 Dockerization

The journey towards modernizing software deployment and infrastructure management has led to the adoption of containerization technologies, with Docker[18] emerging as a prominent player in this paradigm shift. This section presents the results and analysis of the Dockerization process applied to our application, underscoring the transformative impact on various facets of development, deployment, and operations.

Containerization Success: The Dockerization process of the application was successful, with the successful creation and deployment of Docker containers encapsulating the entire application stack. This includes the application code, dependencies, and required configurations.

Image Size Optimization: The Docker images were optimized to reduce their size, resulting in more efficient storage and faster container deployment times. This optimization was achieved by minimizing unnecessary components and leveraging multi-stage builds.

The following figure show docker images size of our developed service

```
mysql                              8.0.32            412b8cc72e4a   9 months ago    531MB
mongo                              4.4.14-rc0-focal  d903df4e44df   21 months ago   438MB
confluentinc/cp-kafka              7.0.1             5069d65bcc55   2 years ago     780MB
confluentinc/cp-zookeeper          7.0.1             3a7ea656f1af   2 years ago     780MB
pfniyigena/notification-service    latest            54abb0cac1d3   54 years ago    333MB
pfniyigena/declaration-service     latest            e98f6418976e   54 years ago    362MB
pfniyigena/payment-service         latest            8ffdfe70f4f7   54 years ago    355MB
pfniyigena/taxpayer-service        latest            569a39b565f7   54 years ago    338MB
pfniyigena/api-gateway             latest            a40cc2490db9   54 years ago    322MB
pfniyigena/discovery-server        latest            a611ca55aaa6   54 years ago    325MB
pfniyigena/portal-service          latest            cf51540d18dc   54 years ago    376MB
PS F:\courses\MSE\modules\thesis\final\2024\codes\etax-main>
```

Figure 21: eTax docker images size.

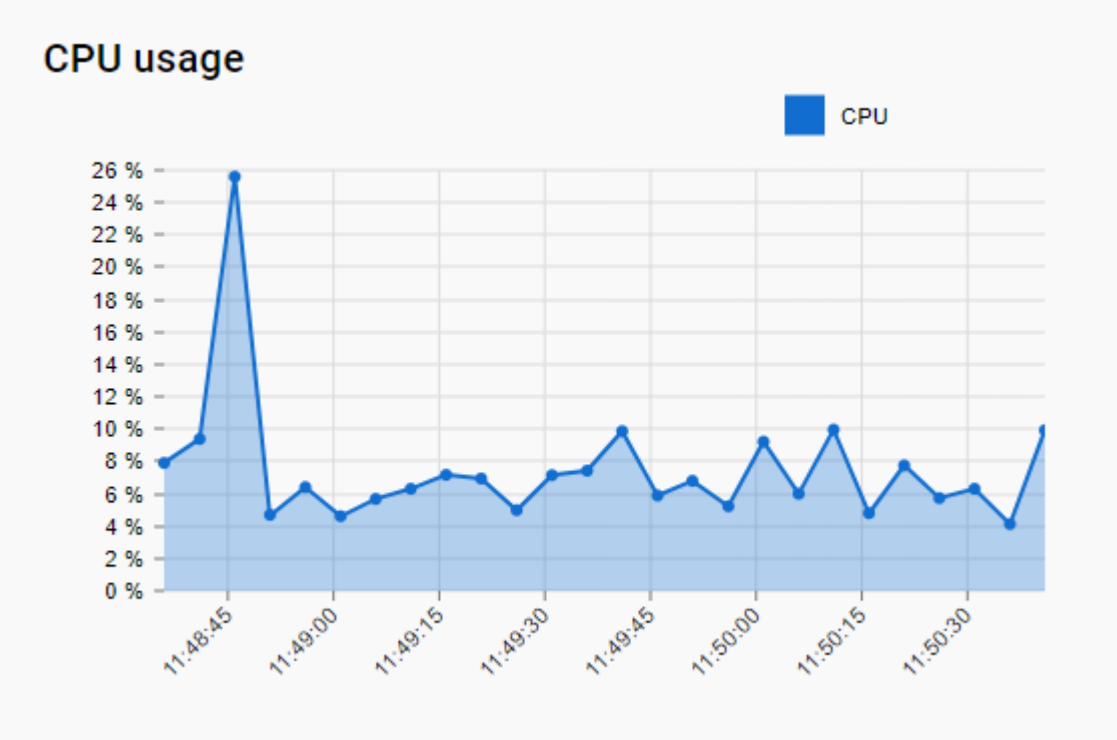

Figure 22: eTax CPU usage (All 16 docker images)

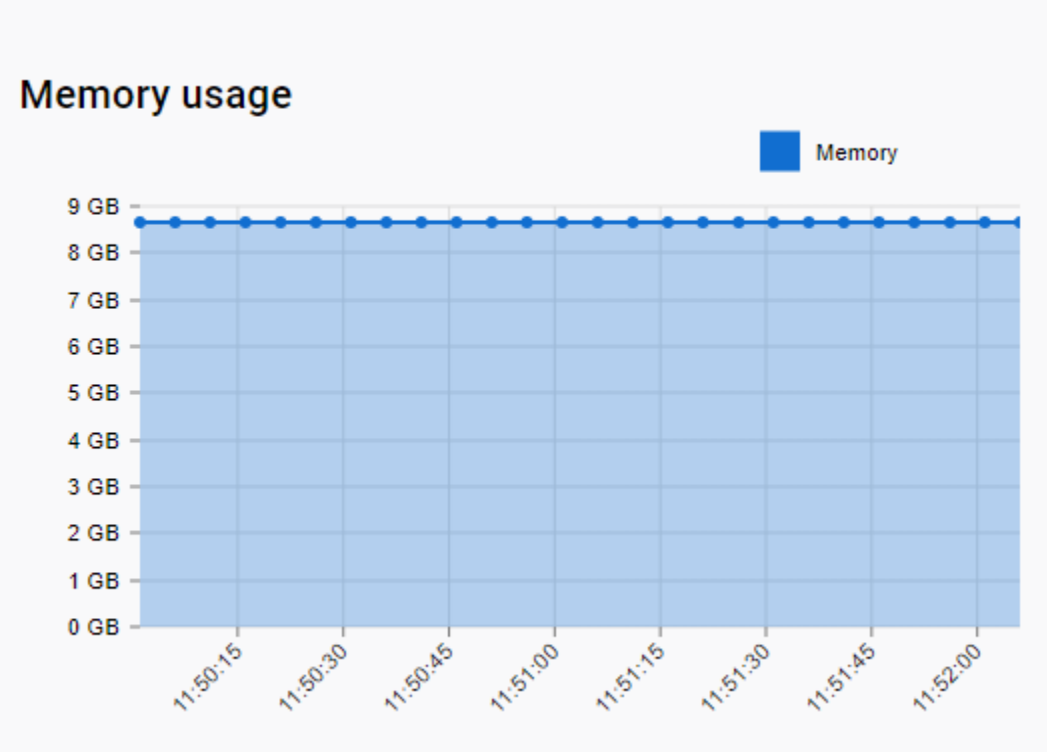

Figure 23: eTax Memory usage (All 16 docker images)

Versioning and Rollback: Docker's versioning capabilities facilitated easier management of application versions. Additionally, the ability to roll back to previous versions in case of issues contributed to increased system reliability and maintainability.

User Experience: Feedback from end-users indicated a positive experience with the Dockerized application. The simplified deployment process and improved performance were well-received, contributing to user satisfaction. The adoption rate of Dockerized deployments within the development and operations teams surpassed expectations. Team members reported increased efficiency, collaboration, and ease of managing application dependencies.

The following table highlights some of the key differences between traditional deployment and Docker deployment. Docker's containerization provides advantages such as improved isolation, consistent environments, portability, and easier scaling, making it a popular choice for modern application deployment. The following table outlines key differences between the traditional deployments with docker deployment.

Feature	Traditional Deployment	Docker Deployment
Isolation		
Dependencies	Manual installation of dependencies.	Dependencies are packaged within containers
Environment Consistency	May vary across different environments.	Consistent environment due to containerization.
Portability	Less portable across different systems.	Highly portable due to containerization.
Scaling	Scaling may involve more manual effort.	Easy scaling with container orchestration tools.
Resource Efficiency	Resource utilization may be suboptimal.	Efficient resource utilization with containers.
Deployment Speed	Slower due to manual setup and config.	Faster deployment with pre-packaged containers.
Rollback	Complex rollback procedures.	Easier rollback by switching to previous image.
Versioning	Manual version control.	Version control through container images.
Collaboration	May require extensive documentation	Simplified collaboration with container images.
Dependency Conflicts	Common due to shared dependencies	Reduced conflicts due to encapsulated dependencies.
Maintenance	More time-consuming maintenance task.	Simplified maintenance with containerization.

Table 2: Table outlines key differences between the traditional deployment with docker deployment

5.5 Conclusion

In conclusion, the Dockerization of the application has proven to be a valuable enhancement to the development, deployment, and operational aspects of the system. The improvements in scalability, performance, security, and development efficiency highlight the benefits of adopting containerization technologies.

CHAPTER 6: CONCLUSION AND RECOMMENDATION

6.1 Conclusion

This research demonstrates the importance of using service discovery to create multiple instances of microservices in eTax. The use of API gateway demonstrates how easy and efficient is to implement a single secured one entry point of all other microservices. The use of even driven architecture in eTax demonstrates the implementation of asynchronous communication between to components of the microservices. Finally, the introduction of dockers images and containers show how easy and efficient it is to deployment together a microservices with all package dependencies together

6.2 Recommendation

Future researches should investigate the issues and problems the use of microservices and event architecture might cause on both development teams and the customers satisfaction. They should also implement the other modules of eTax such as accounting, tax recovery etc. In addition, the researches should look into another concept of Kubernetes which is an open-source tool for managing docker containers.

LIST OF REFERENCES

[1] Antonio Bucchiarone et al,"From Monolithic to Microservices" ,*THE IEEE COMPUTER SOCIETY*,p50, 2018.

[2] Brendan Burns,"Designing Distributed Systems",*O'Reilly Media, Inc*.,p45-57 2018.

[3] Sandeep Rawat,"CI/CD Pipeline with Docker and Jenkins Learn How to Build and Manage Your CI/CD Pipelines Effectively",*BPB Online*,p45-57,2023.

[4] H. F. Oliveira Rocha,"Practical Event-Driven Microservices Architecture". *Apress*, p 56,2022.

[5] Alex Campbell,"Agile All You Need to Know about Agile Software Development. Team and Project Management using Scrum",*Alex Campbell*,p50-137,2020.

[6] M. Sharma,"Full Stack Development with MongoDB",*BPB Publications*,p50-300, 2018.

[7] F. Auer, V. Lenarduzzi, M. Felderer, and D. Taibi,"From monolithic systems to Microservices: An assessment framework", *Inf. Softw. Technol.*, vol. 137, 2021,.

[8] J. Sadek, D. Craig, and M. Trenell, "Design and Implementation of Medical Searching System Based on Microservices and Serverless Architectures Microservices",*Procedia Comput. Sci.*, vol. 196, pp. 615–622, 2022.

[9] K. Jander, L. Braubach, and A. Pokahr, "Defense-in-depth and Role Authentication for Microservice Systems", *Procedia Comput. Sci.*, vol. 130, pp. 456–463, 2018.

[10] S. Zhelev and A. Rozeva, "Using microservices and event driven architecture for big data stream processing," *AIP Conf. Proc.*, vol. 2172, no. December 2017.

[11] S. Ben Atitallah, M. Driss, and H. Ben Ghzela, "Microservices for Data Analytics in IoT Applications: Current Solutions, Open Challenges, and Future Research Directions," *Procedia Comput. Sci.*, vol. 207, pp. 3938–3947, 2022.

[12] B. B. M. Michelson and E. Links, "Event-Driven Architecture Overview 2011", *Architecture*, 2011.

[13] E. Eldor, "Kafka Troubleshooting in Production". *Elad Eldor*, p 50-350,2023.

[14] Bill Bejeck,"Kafka Streams in Action",*Manning Publications Co*,p50-300, 2024.

[15] D. D. Scott, "Kafka in Action",*Manning Publications Co,p50-300,*2019.

[16] M. Needham, "Building Real-Time Analytics Systems". *O'Reilly Media, Inc* 2023.

[17] S. Thorgersen and P. Silva, "Keycloak - Identity and Access Management for Modern Applications." *Packt Publishing*, 2021.

[18] D. G. N. Schenker, "The Ultimate Docker Container Book". *Packt Publishing,* 2023.

[19] L. Rice and M. Hausenblas, *Kubernetes Security*. 2018.

[20] N. (Telecommunications engineer) Poulton, "Docker deep dive : zero to Docker in a single book". 2020.

[21] B. L. Iverson and P. B. Dervan, "Docker and Kubernetes for Java Developers Scale".

[22] N. Poulton, "Weapons-grade container learning Docker Deep Dive Zero to Docker in a single book 2023 Edition," 2023.

APPENDIX

Survey Questions

Instructions:

Please answer the following questions to the best of your ability. Your responses will remain anonymous and will be used for research purposes only. Thank you for your time and valuable input.

1. **How frequently do you file your taxes?**
   - Once a year
   - Quarterly
   - Other (please specify)
2. **Have you used electronic tax filing services (eTax) in the past?**
   - Yes
   - No
3. **If you have used eTax services, please rate your overall experience on a scale of 1 to 5, with 1 being very poor and 5 being excellent.**
4. **What specific aspects of eTax services do you find most useful? (Select all that apply)**
   - Ease of use
   - Time-saving
   - Accessible from anywhere
   - Ability to review and correct errors
   - Automatic calculation of tax owed/refund
   - Other (please specify)
5. **What challenges have you faced while using eTax services, if any? (Select all that apply)**
   - Technical difficulties
   - Unclear instructions or terminology
   - Difficulties in entering or uploading data
   - Security concerns
   - Lack of customer support
   - Other (please specify)
6. **Would you prefer a more simplified eTax interface, or do you appreciate a comprehensive interface with advanced features?**
   - More simplified interface
   - Comprehensive interface with advanced features

- No preference

7. **How satisfied are you with the current level of guidance provided within the eTax system?**
   - Very satisfied
   - Satisfied
   - Neutral
   - Dissatisfied
   - Very dissatisfied
8. **How likely are you to recommend eTax services to others?**
   - Very likely
   - Likely
   - Neutral
   - Unlikely
   - Very unlikely
9. **If you have any suggestions or improvements to the existing eTax system, please provide details below.**
10. **Are there any additional comments or feedback you would like to share regarding your experience with eTax services?**

Thank you for your participation in this survey! Your feedback is highly valuable and will contribute to the improvement of eTax services.

Interviews Questions

- ✓ How would you design the communication between systems? Can you explain the key components and patterns involved?
- ✓ When designing eTax system, how would you ensure data consistency and integrity across different systems?
- ✓ How would you handle service failures and ensure fault tolerance in an eTax?
- ✓ Can you discuss the role of asynchronous messaging and how it applies to eTax systems?
- ✓ How would you handle versioning and backward compatibility when evolving an ETax system? Can you provide some strategies or best practices?
- ✓ In an eTax system, how would you handle long-running processes or tasks that require coordination between multiple system?
- ✓ Can you discuss the scalability and performance considerations when designing eTax systems for handling tax-related transactions?
- ✓ What are some common security measures for an eTax system?
- ✓ How would you ensure security and data privacy for an eTax system?
- ✓ Can you describe the process of deploying eTax in a production environment?
- ✓ What tools or technologies do you use for monitoring eTax?

API Gateway properties file of eTax project

```
eureka.client.serviceUrl.defaultZone=http://eureka:password@localhost:8761/eureka
spring.application.name=api-gateway
logging.level.root= INFO
logging.level.org.springframework.cloud.gateway.route.RouteDefinitionRouteLocator= INFO
logging.level.org.springframework.cloud.gateway= TRACE
## Search Taxpayer Service Route
spring.cloud.gateway.routes[0].id=taxpayer-service-search
spring.cloud.gateway.routes[0].uri=lb://taxpayer-service
spring.cloud.gateway.routes[0].predicates[0]=Path=/api/taxpayer/search
## All Taxpayer Service Route
spring.cloud.gateway.routes[1].id=taxpayer-service-all
spring.cloud.gateway.routes[1].uri=lb://taxpayer-service
spring.cloud.gateway.routes[1].predicates[0]=Path=/api/taxpayer/all
## New Taxpayer Service Route
spring.cloud.gateway.routes[2].id=taxpayer-service-create
spring.cloud.gateway.routes[2].uri=lb://taxpayer-service
spring.cloud.gateway.routes[2].predicates[0]=Path=/api/taxpayer/new
## Declaration Service Route
spring.cloud.gateway.routes[3].id=declaration-service-place-declaration
spring.cloud.gateway.routes[3].uri=lb://declaration-service
spring.cloud.gateway.routes[3].predicates[0]=Path=/api/declaration/placeDeclaration
## Declaration Service Route
spring.cloud.gateway.routes[4].id=declaration-service-all
spring.cloud.gateway.routes[4].uri=lb://declaration-service
spring.cloud.gateway.routes[4].predicates[0]=Path=/api/declaration/all
```

```
## Payment Service Route
spring.cloud.gateway.routes[5].id=payment-service-place-payment
spring.cloud.gateway.routes[5].uri=lb://payment-service
spring.cloud.gateway.routes[5].predicates[0]=Path=/api/payment/placePayment
## Payment Service Route
spring.cloud.gateway.routes[6].id=payment-service-all
spring.cloud.gateway.routes[6].uri=lb://payment-service
spring.cloud.gateway.routes[6].predicates[0]=Path=/api/payment/all
## Discover Server Route
spring.cloud.gateway.routes[7].id=discovery-server-web
spring.cloud.gateway.routes[7].uri=http://localhost:8761
spring.cloud.gateway.routes[7].predicates[0]=Path=/eureka/web
spring.cloud.gateway.routes[7].filters[0]=SetPath=/
## Discover Server Static Resources Route
spring.cloud.gateway.routes[8].id=discovery-server-static
spring.cloud.gateway.routes[8].uri=http://localhost:8761
spring.cloud.gateway.routes[8].predicates[0]=Path=/eureka/**
spring.security.oauth2.resourceserver.jwt.issuer-uri= http://localhost:8181/realms/etax-client
#Distributed tracing
management.tracing.enabled=true
management.tracing.sampling.probability=1.0
management.zipkin.tracing.endpoint=http://localhost:9411/api/v2/spans
logging.pattern.level="trace_id=%mdc{traceId} span_id=%mdc{spanId} trace_flags=%mdc{traceFlags} %p"
```

Docker compose file of eTax project

```yaml
---
version: '3.1'
services:
  ## Mysql Declaration Service Config with Mysql database
  mysql-declaration:
    container_name: mysql-declaration
    image: mysql:8.0.32
    volumes:
      - ./mysql_declaration_data:/var/lib/mysql
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: declaration_service_db
      MYSQL_USER: etax
      MYSQL_PASSWORD: MYSQLnpf20208.
    expose:
      - "3307"
    ports:
      - "3307:3307"
    command: -P 3307
    restart: unless-stopped
  ## Postgres Keycloak Config with Mysql database
  postgres-keycloak:
    image: postgres:15.3
    container_name: postgres-keycloak
    environment:
      POSTGRES_DB: keycloak
      POSTGRES_USER: keycloak
      POSTGRES_PASSWORD: password
      PGDATA: /data/postgres
    volumes:
      - ./postgres_keycloak_data:/data/postgres
    expose:
      - "5430"
    ports:
        - "5430:5430"
    command: -p 5430
    restart: unless-stopped
    healthcheck:
      test: "pg_isready -U postgres"
  ## Postgres Payment Service Compose Config
  postgres-payment:
    container_name: postgres-payment
    image: postgres
    environment:
      POSTGRES_DB: payment_service_db
      POSTGRES_USER: etax
      POSTGRES_PASSWORD: npf2020.
      PGDATA: /data/postgres
    volumes:
      - ./postgres_payment_data:/data/postgres
    expose:
```

```
    - "5431"
  ports:
    - "5431:5431"
  command: -p 5431
  restart: unless-stopped
## Postgres Portal Service Compose Config
postgres-portal:
  container_name: postgres-portal
  image: postgres
  environment:
    POSTGRES_DB: portal_db
    POSTGRES_USER: etax
    POSTGRES_PASSWORD: npf2020.
    PGDATA: /data/postgres
  volumes:
    - ./postgres_portal_data:/data/postgres
  ports:
    - "5432:5432"
  expose:
    - "5432"
  restart: unless-stopped
## Mongo Taxpayer ServiceMongo Docker Compose Config
mongo-taxpayer:
  container_name: mongo-taxpayer
  image: mongo:4.4.14-rc0-focal
  restart: unless-stopped
```

```
  ports:
    - "27017:27017"
  expose:
    - "27017"
  volumes:
    - ./mongo_taxpayer_data:/data/db
##  Zookeeper
zookeeper:
  image: confluentinc/cp-zookeeper:7.0.1
  container_name: zookeeper
  environment:
    ZOOKEEPER_CLIENT_PORT: 2181
    ZOOKEEPER_TICK_TIME: 2000
##Broker
broker:
  image: confluentinc/cp-kafka:7.0.1
  container_name: broker
  ports:
    - "9092:9092"
  depends_on:
    - zookeeper
  environment:
    KAFKA_BROKER_ID: 1
    KAFKA_ZOOKEEPER_CONNECT: 'zookeeper:2181'
    KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: PLAINTEXT:PLAINTEXT,PLAINTEXT_INTERNAL:PLAINTEXT
    KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://localhost:9092,PLAINTEXT_INTERNAL://broker:29092
```

```
    KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
    KAFKA_TRANSACTION_STATE_LOG_MIN_ISR: 1
    KAFKA_TRANSACTION_STATE_LOG_REPLICATION_FACTOR: 1
## Zipkin
zipkin:
  image: openzipkin/zipkin
  container_name: zipkin
  ports:
    - "9411:9411"
##Keycloak
keycloak:
  container_name: keycloak
  image: quay.io/keycloak/keycloak
  ports:
    - "8080:8080"
  environment:
    KEYCLOAK_ADMIN: admin
    KEYCLOAK_ADMIN_PASSWORD: admin
    ##ENV KC_DB_URL: 'jdbc:postgresql://postgres:5430/keycloak'
    ##ENV KC_DB_USERNAME: keycloak
    ##ENV KC_DB_PASSWORD: keycloak
    ##ENV KC_DB: postgres
  command:
    - start-dev
    - --import-realm
  volumes:
```

```
    - ./realms/:/opt/keycloak/data/import/
  ##depends_on:
  ##  - postgres-keycloak
## Eureka Server
discovery-server:
  image: pfniyigena/discovery-server:latest
  container_name: discovery-server
  ports:
    - "8761:8761"
  environment:
    - SPRING_PROFILES_ACTIVE=docker
  depends_on:
    - zipkin
## Api gateway
api-gateway:
  image: pfniyigena/api-gateway:latest
  container_name: api-gateway
  ports:
    - "8181:8181"
  expose:
    - "8181"
  environment:
    - SPRING_PROFILES_ACTIVE=docker
    - LOGGING_LEVEL_ORG_SPRINGFRAMEWORK_SECURITY= TRACE
  depends_on:
    - zipkin
```

```
156       - discovery-server
157       - keycloak
158   ## Taxpayer service
159   taxpayer-service:
160     image: pfniyigena/taxpayer-service:latest
161     container_name: taxpayer-service
162     environment:
163       - SPRING_PROFILES_ACTIVE=docker
164       - LOGGING_LEVEL_ORG_SPRINGFRAMEWORK_SECURITY= TRACE
165     depends_on:
166       - mongo-taxpayer
167       - broker
168       - zipkin
169       - discovery-server
170       - api-gateway
171   ## Portal-Service Docker Compose Config
172   portal-service:
173     container_name: portal-service
174     image: pfniyigena/portal-service:latest
175     environment:
176       - SPRING_PROFILES_ACTIVE=docker
177       - SPRING_DATASOURCE_URL=jdbc:postgresql://postgres-portal:5432/portal_db
178     ports:
179       - "8383:8383"
180     depends_on:
181       - postgres-portal
```