Journal of Theoretical and Applied Information Technology
15th December 2022. Vol.100. No 23

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

ANALYSIS AND DEVELOPMENT OF MICROSERVICES ARCHITECTURE IN LOAN APPLICATION SYSTEM OF COOPERATIVE ENTERPRISE IN INDONESIA

REYNALDI LIE[1], AHMAD NURUL FAJAR[2]

[1,] Information System Management Department,

BINUS Graduate Program - Master of Information Systems Management,

Bina Nusantara University,

Jakarta, Indonesia 11480

E-mail: [1]reynaldi004@binus.ac.id, [2]afajar@binus.edu

ABSTRACT

PT XYZ is a cooperation enterprise currently using monolithic IT (Information Technology) system architecture. Load test and stress test results of the system showed that development of capabilities of the loan application system of the enteprise is still possible to be done. This research is purposed to analyze the loan application system design based on microservices as possible alternative to replace the current IT monolithic system architecture in PT XYZ by enhancing the performance of the loan application system in scaling up business with the ability to establish low dependency among applications. The research is exclusively aimed to solve the system architectural problem in cooperative enterprises with PT XYZ as the example. The method used in this research is microservices-based system design with DDD (Domain Driven Design) approach by determining bounded contexts, followed by classification of entities, aggregates, and services that are going to be materialized in the design. Analysis results confirmed that the services can be seen from three different contexts, namely information context, loan application context, and loan review context according to the functionalities of each service component. The researcher suggested software stack emphasizing on processes automation at PT XYZ to support the microservices architecture design.

Keywords: *Service Oriented Architecture, Microservices Design, Domain Driven Design, System Architecture, Loan Application*

1. INTRODUCTION

Savings and loan cooperatives are the pillars of Indonesia's economy and have always been an economic agent that synergizes with other economic agents, namely BUMN (*Badan Usaha Milik Negara*), the Indonesia's equivalent for SOEs (State-Owned Enterprises) and BUMS (*Badan Usaha Milik Swasta*), the Indonesia's equal of private enterprise in the joint effort of bringing equally prospering society. To realize and develop the democratization of economy, these economy agents, particularly enterprises, must be built in accordance with the Article 33 Section (1) of UUD 1945 (The 1945 Constitution of the Republic of Indonesia). The success of the cooperatives will be the determining factor of Indonesian economic stability, especially in the uncertainty of Covid-19 which end has yet to be seen, where many parties might need fundings from cooperatives.

According to the records of OJK (*Otoritas Jasa Keuangan*), the Indonesian Financial Services Authority, every year has saw the increase of credit disbursement significantly [1]. The data retrieved from the period of December 2018 where the credit disbursement of creditors in Java Island reached about 19.62 trillion rupiahs whereas outside of Java, the disbursement reached approximately 3.05 trillion rupiahs in amount. The following year, in December 2019 showed drastic increase in the credit disbursement where creditors in Java Island disbursed 69.82 trillion rupiahs and creditors outside of Java disbursed 11.67 trillion rupiahs. The very next year also saw increase, exactly in December 2020, where Java Island has accumulated disburse amount of 132.38 trillion rupiahs while outside of Java counted as 23.52 trillion rupiahs.

The accumulation of the disbursement amount were calculated from various sources, namely from borrowers' accounts, including both consumer and working capital loans. More than 140 creditors are offering loan services across Indonesia annually. May 2020 has the greatest number of creditors providing loan services to customers, at 161 companies.

The rise of Covid-19 outbreak at the beginning of 2020 has not reached its conclusion. Indonesia's government, like many countries around the world, has enforced physical distancing as a countermeasure against the viral infection, which the Indonesian government called PSBB (*Pembatasan Sosial Berskala Besar*) or literally translated as Large-Scale Social Distancing. This limited people's activities greatly and ended up reducing purchasing power immensely. This occurrence affected Indonesia's economy negatively as in the same year, there was a decrease with the percentage of 2.07% [2]. Business owners responded to PSBB by implementing WFH (Work from Home) and WFO (Work from Office) rules. WFH and WFO are done alternately to maintain the continuity of business activities without ignoring the government's advice to physically distancing with each other to slowdown the spread of Covid-19. Another challenge for the business owners is the necessity of adapting and innovating amongst the Covid-19 situation. With or without the pandemic itself, continuous adaptation and innovation are necessary to adjust with the present conditions and challenges. People have insurmountable number of needs, along with the necessity of adapting and innovating, are the reason why a modern system must be developed to answer those needs. One of the possible ways to provide the ability to keep doing activities while communicating with the society among a condition which greatly restricts face to face interaction is to adopt a well-established information system and hasten the development of the digitalization of a company so that the company can move in a dynamic way.

Besides, the globalization era gives more opportunities for cooperative enterprises to increase in number. The cooperatives are competing with each other as the result [3]. This circumstance forces every cooperative to be in constant improvement and perfecting the business to be able to withstand the threat of competitors and to maintain the cooperatives overall existence for long term. The advancement of business environment nowadays forces business owners to use many strategies and techniques to support the improvement of business, cooperatives are not an exception to this.

The strategy for improving or perfecting cooperative business lines can be realized by conducting digital transformation which demands an organization to be nimble and to adopt suitable innovation method with good acceleration to ensure that the newly developed digital services can be brought to customers, partners, and staffs alike. To achieve this, many organizations took the approach of building a seamless application based on cloud with good flexibility, where it is easier to add and renew digital services to suit the everchanging business requirements and technological capabilities [4]. Old monolithic applications may be acceptable for daily operations, yet these applications are not suited to construct digital services. The traditional monolithic system architecture and software development method will always be hinderances in pushing forward to the desired digital transformation within an organization [5].

Information system is commonly built with monolithic architecture, where each component of the system is intended to be built as one unity. A single monolithic architecture usually comprises of tens or hundreds of business functions used simultaneously in one software release. Microservices, on the other hand, usually summarizes a single business function so that it can be scaled and used separately. Developing application for a large enterprise by spreading cloud facilities by creating and managing a number of microservices give the possibility of another alternative to improve monolithic architecture. The challenges that a developer might encounter within monolithic architecture are the questionable adaptation capability of the system toward system requirement changes, mainly in managing codes complexity and its maintainability, where this will cause bottleneck to the services distribution process due to tightly coupling of the codes. Therefore, a consideration of devising another architectural alternative to create an adaptive system management to suit requirement changes whenever needed, is needed [6]. In an architectural design focused on services development, every company should at least update the system and services weekly. PT XYZ, a cooperative enterprise in Indonesia situated at Jakarta that the researcher studied on, is no exception. The case study serves to compare the updating process in monolithic

architectural design versus microservices architectural design.

Another challenge in using monolithic architecture is the system development dependence on stack of the main technologies which limit the possibility to develop new functions with newer application framework or a different programming language [7]. With the increasing functions amount, monolithic architecture gets more complex and need larger team of developers to support the applications within [8]. Scalability in monolithic architecture is also limited, as individual functions cannot be run repeatedly separately after usage. The repetition must be done to the entire application which can be very time consuming.

Based on these explanations, if the previous system is maintained, it is going to bring disadvantages for the enterprises using the system, particularly in competing with other enterprises in fulfilling customers' needs. The consequence of using the traditional system is that the developers have to face the complexity of system development and must spare a lot of time to deal with it, which can result in slowdown of the advancement of the system compared to the system used by competitors. High operating costs are inevitable in handling complex system structure as the management needs large team to run the process.

For instance, loan system in PT XYZ is a system using monolithic architecture where every components of the application inside are built, designed, implemented, and maintained within a single union of codes [9]. The biggest drawback of this architecture is the developer might face adversity in increasing capacity needed for the system. Constant fixing and improvement of the system is difficult as all components should be increased in capacity, even if only one component needs capacity increase while the rest do not need changes in capacity. For example, the number of SMEs (Small Medium Enterprises) in Indonesia is at 59.2 million. This number indicates the chance for request of application services coming from the customers to be relatively high [10]. Monolithic system renders the application to be slow responding and cannot keep up with the ever-increasing number of users.

Apache Benchmark is used as a tool for testing the services in PT XYZ with its currently used monolithic architecture which showed that the capability of the system can process request of to 3 RPS (Request Per Second). A series of test is also conducted afterwards by changing the testing duration variable and the number of services that the system can provide within a preset time, using Execution Time as the parameter for measuring duration and Concurrency as the parameter in measuring how many services can the system provide within given duration. This measuring method is called load testing, which purpose is to measure how a website can be accessed within certain parameters like execution time and concurrency. RPS is used to gauge the amount of request that can be accepted by the web server within certain duration. Based on the test results as shown in Table 1, the RPS in monolithic system architecture can be considered as lacking.

Table 1: Load Testing of Monolithic Architecture Using Apache Benchmark

No	Execution Time (s)	Concurrency (#)	Monolithic RPS (#/s)
1	10	10	2.98
2	50	10	3.02
3	100	10	2.92
4	150	10	2.99
5	200	10	2.94
6	250	10	2.92
7	300	10	2.87

Apart from doing load testing, monolithic architecture capabilities can also be tested with stress testing. Stress testing is also done using Apache Benchmark with the preset parameters being the number of services requested and the number of services requested at the same time. Stress testing is used to measure how a web can be resilient and responding requests from users even though the number of requests are high, regardless of the requests are made simultaneously or not. The parameters for testing are Number of Request and Concurrency, where Number of Request represents the number of requests acceptable by the web server and Concurrency represents the number of requests acceptable by the web server at the same time. The precise illustration for the stress testing result can be seen in Table 2 below.

Table 2: Stress Testing of Monolithic Architecture Using Apache Benchmark

No	Number of Requests (#)	Concurrency (#)	Monolithic RPS (#/s)
1	10,000	200	10.24
2	5,000	200	9.31
3	2,000	200	8.66

Test results indicated that monolithic architecture is capable of supporting the current services layout, but by combining the results of

both load testing and stress testing, it is safe to assume that the performance of the services present at the current business type can be improved even further.

As implied by reference [11], with a total time required for testing of three minutes, only a minute is necessary to discover that the average response time of applications built on monolithic architecture is significantly lower than the response time needed for applications built on microservices architecture. The accumulated usage of RAM (Random Access Memory) by clients of applications built as microservices are higher than the RAM usage by clients on monolithic applications. Although the frequency of testing done was relatively low as most of the enterprises' server are strong, response time should not be a problem. Similar result is also discovered in the usage of CPU (Central Processing Unit).

Microservices architecture originated from Service Oriented Architecture or abbreviated commonly as SOA, one of the possible system architectural style to develop business and technological solution in the form of software as composition of split logics loosely coupled for ease of operation and distribution into separate parts without dependency on each other in order to shape services. Service orientation or SO for short, is the new paradigm to shape services with a set of design principle like self-description, storing personal information, autonomous, abstraction, standard contract, division of services contents, and ease of use in locating the services. All of these are set to fulfill the principle of SOA. This paradigm is the evolution from the traditionally object-oriented business into components-based manipulation to enforce services with great flexibility and interoperability, without neglecting the principle of information hiding for the sake of privacy. SO is intended to devise software solutions as single and/or combined services, in regard to SOA architectural style running in distributed computerization or can be referred to as SOC (Service Oriented Computing).

SOA approach has several advantages as opposed to traditional architectural approaches such as the opportunity for repeated use, better independency as a platform, lower maintenance and development cost, observable with results determined by network variable and factor, and various services to choose from with dynamic composition and dynamic information stream [12].

By understanding the basics of SOA, one can recognize microservices architecture which is an evolved form of SOA as one alternative of a more measurable architecture with higher flexibility than SOA. To encourage digital transformation efficiently, many organizations are exploring methodologies and architectural style to develop new software. In microservices architecture, information system is designed to be distributed into separate parts to provide more focused and more specific services. Large issues will be categorized into small pieces of solutions grouped into one service, where each service has its own responsibility. This approach creates an information system comprising of several services which can be managed and distributed independently. Thus, the system can adapt better to changing needs [6].

Another reason for the suggestion of microservice architecture in this research is to reduce difficulty in the process of scaling up the current business, especially in the business segment of loan service in PT XYZ. Business scale up is the effort to prepare and support the growth and the development of a business, be it from the system aspect, resources, processes, technology, or partners [13]. An enterprise is ready to perform business scale up if the customer retention rate of that company is high to the point of receiving large quantities of demand from customers, but with limited services capacity, facing difficulty in fulfilling the requests. High dependency among applications in monolithic system can be a barrier in scaling up business of a company. PT XYZ is also vulnerable to the threat if system architecture replacement is not done sooner. The need of migration from monolithic to microservices architectural style is to give PT XYZ the capacity of balancing the readiness of the cooperative enterprise in scale up processes and helping in its realization.

A data from [14] indicated that 68% of for-profit company using IT system in running their business had implemented microservices to products and its development, while 26% of them have learnt microservices but haven't implemented it. The remaining 6% didn't learn and implement microservices. This reference also pointed out from a case study in UppLabs, a Fintech company from United States that the company couldnot scale up the business without replacing the monolithic architectural style.

The final desired outcome of using microservices architecture is lessened difficulty in doing business by improving services dedicated to

stakeholders of enterprises, like bridging between producers and buyers through an integrated information system. Digitalization in a company support guidance and accompaniment across space and time as the processes of delivering services can be done in real-time and online environments.

Microservices infrastructure can be developed using the approach of DDD (Domain Driven Design). DDD's main area of focus is domains, including its conception, relationship with each other, and existing business processes. DDD relies on bounded contexts to identify services within the microservices architecture. Business processes must be grouped according to the functions. Every group will be treated as bounded contexts of the system. These contexts will later form microservices in the microservices architecture [15]. However, in real life practice, there aren't many companies taking total advantage of information system. Some creditor companies, like cooperatives, are still using manual system which slowdown the process of decision-making regarding loan disbursement and several other things related with credit. Slow decision-making can hinder the process of retrieving daily customer reports like loan applicant report and credit disbursement report. Data also suffer to the risk of inaccuracy. These problems can be tackled by building a computerized system with better integration [16]. DDD approach is expected to be the beginning of formulating solutions to dissolve or solve these problems. The application of DDD approach in designing microservices architecture can be beneficial in achieving the maximum advantage of loose coupling with reduced implementation complexity by separating domains and subdomains using DDD approach.

The researcher is piqued to design microservices architecture for PT XYZ's loan application system with DDD approach. The design idea for the loan application system is carried out by interviewing the CTO (Chief Technology Officer) from the cooperative to learn more about what the current monolithic system in loan application system is capable of doing. The design is hoped to offer an alternative of systemic solution for PT XYZ in enhancing the performance of the loan application system in bringing convenience in scaling up business with low dependency among applications. The research is exclusively aimed to solve the system architectural problem in cooperative enterprises, as opposed to the referred past studies that mainly involved IT companies. Research questions are formulated as follows:

1. How using microservices architecture to design the loan application system for PT XYZ can reduce the difficulty of the system design processes?
2. How the DDD (Domain Driven Design) approach can help in designing microservices for the loan application system of PT XYZ cooperative enterprise?

2. LITERATURE REVIEW

In this section, the researcher discussed related terminologies to provide better understanding of the theories surrounding the research focus for the readers. The theories consisted of definition and contribution of past researches, mainly from journals and literatures.

2.1 General Theory

General theories defined some terms with direct connection to the field of computer studies. These theories were taken from journals, books, or previously documented researches that contained insights and suggestions to improve the area of IT (Information Technology).

2.1.1 Service oriented architecture

SOA is an architecture solution with loose coupling mechanism and one way to implement it is by using web services. To answer the challenge of competition from economic pressure, the IT industry in general has always tried to improve system performance and customer services with efficient cost management. This tendency in behavior among IT industry agents somehow led to the changing of IT system infrastructure into SOA for to have a nimbler system for the best interest of business owners. This initiative, however, has brought to light the security issues that SOA possess [12].

By implementing SOA, enterprises will have effective, flexible, scalable, and repeatedly useable IT infrastructure. This concept supports digital transformation in a company with the ability to be adaptive in either internal business structure or external environment [17].

SOA plays the role of illustrated architectural paradigm that is grouped into six separate dimensions which consists of business value, strategic goals, intrinsic interoperability, combined services, flexibility, and evolutionary improvement. The ultimate purpose of this architectural paradigm is to divide several business procedures into smaller parts which can be connected and reused in another business scenarios whenever needed [18].

SOA is formed by four core components, they are application frontend, services, services repository, and services bus of an enterprise. In application frontend, the users can initiate, control, and cease the services, where functional units are separated so they are accessible from long distance and taken care of and updated independently. Different user groups can access the services from built communication networks and build infrastructure that is divided into equal parts to facilitate integration and collaboration, both internally and/or externally. Services repository has a structure to store and access services data to enable the users to determine the services characteristics and functions as registry or information contained in the database, which governs the SOA projects in access services catalogue, all of which are improving continuously in an interactive and controlled environment. After a series of processes, the enterprise's service bus brings to conclusion by implementing communication system across services where all of them are connected by the already established SOA environment.

Service consumer and service provider communication flows on the internet in SOA environment starts with the service provider component publishing information about the services in service registry. Service consumer will find the service information and service provider component information from the service registry. Lastly, these two components will undergo the process of integration where both will bind with each other through internet connection. This processes are illustrated in Figure 1 below.

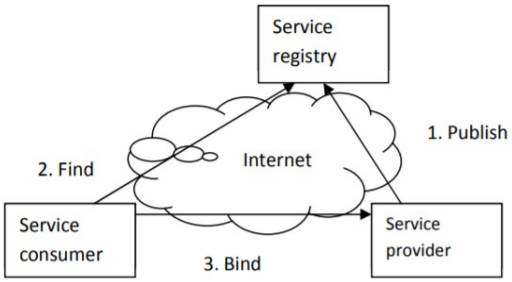

Figure 1: SOA Main Structure [19]

SOA is very strict when it comes to privacy protection because information flow or program codes cannot be seen by anyone, closing the possibility of knowing how the services will be grouped to complete certain processes in the architectural environment. So, to measure the degree of potential information leak in SOA-based web services proves to be a challenging feat.

2.1.2 Microservices architecture

Adoption of microservices architecture to develop software is getting more and more popular as a mean to replace traditional system architecture since cohesive services group and loose coupling are the benefits of adopting microservices architecture. Microservices are built as solutions which take form as properly defined domains and generating many components with similar ability to communicate and operate together. Usage of microservices support the implementation and maintenance process with good resilience and scalability. However, these perks can unintentionally create a more complex ecosystem that requires additional effort of communication and collaboration in tackling the problems that might surface from the inside [20].

Microservices is reliable in building tough, measurable, independent, and fast improving applications. But since the services are autonomous and are built from groups of small services, a careful approach is needed to build the applications. Reference [21] provided a comprehensive explanation about microservices such as the components, benefits, and challenges of implementing the architectural style.

The components shaping microservices comprises of these characteristics:

1. Microservices are small in size, moves independently, and can be grouped flexibly. Thus, only a small team of developers are needed to describe and maintain the services.
2. Every service has separate code base.
3. Service can be updated separately without having to update the entire services.
4. Each service is responsible for their respective data.
5. Every service can communicate with each other through API. Implementation detail of one service is hidden from another service.
6. It is not necessary for services to share the same technological basis, user manual, and workflow.

Other component beside the services itself that is also vital for the materialization of microservices is API gateway. API stands for Application Programming Interface and acts as an entry point for clients. API gateway will forward the request to suitable service in the backend part of the system.

Converting to microservices provide various benefits such as:

1. Nimbleness. Fixing bugs and releasing new features in applications are made easier by using microservices, as opposed to traditional architecture where new features release can be put on halt whenever bugs are being fixed.
2. Small team and better focus, making creation, testing, and implementation of new features to be less time consuming.
3. Small code base, making addition of new features to be less challenging as each service are not reliant to each other.
4. Freedom to combine many different technologies to suit development needs.
5. Isolation of error, like circuit breaking to prevent errors in one service to affect another area of microservices architecture.
6. The ability to adequately allocate resources into certain areas of needs accordingly.
7. Data isolation gives the ability to update at will, minimizing the potential unwanted updates to undesired parts of the services.

Microservices' benefits also come with new challenges that the developer team must tackle, like:

1. Increase in complexity of the entire system architecture.
2. Development and testing of the services can be difficult as available tools can lack in compatibility to gauge the dependency of a service, especially if the services are growing fast.
3. Lack of proper governance, since microservices emphasized on decentralized environment.
4. Possibility of congestion and latency in network. API must be designed carefully to avoid getting into this issue.
5. Data integration becomes a problem as each service being in charge of respective data, generating data consistency issues.
6. Logging management issues.
7. Versioning of services may lead to potential backward or forward compatibility issues.
8. Proper expertise are absolute requirement in handling microservices. The team of developers that are going to be tasked with everything related with microservices cannot be just anyone, it must be experts or people with experience in handling the complexity of microservices.

2.1.3 Domain driven design

DDD (Domain Driven Design) is a methodology based on models that is popular in use for capturing knowledge of certain domains with relevance to a software design. To encourage understanding about domains and the authenticity of generated design, DDD emphasizes on an agile collaborative modeling from domain experts and software engineers. Presently, microservices architecture is developing as an architectural style for distributed software system with high requirements for scalability and adaptive ability. DDD can also dissect domain and define contexts among the domain, where each are grouped into a coherent domain concept. These contexts are suitable for microservices functionality in providing different business capabilities. Putting DDD into practice in microservices design, however, create some challenges regarding the exclusion of some services from domain model, infrastructure components modeling, and domain modeling in an autonomous group [22].

During strategic phase of DDD, business domain are being mapped and defined as bounded contexts for domain modeling. DDD is a tactical approach in classifying domain models with high precision. This tactical pattern is applied into bounded contexts. Applying entities and aggregates pattern in microservices will help identifying natural boundaries for internal services. Using common principle, microservices must be greater than aggregates, and must be smaller than bounded contexts. The initial step would be to supervise this tactical pattern and apply it into the bounded contexts. These are the summary for DDD's tactical pattern:

1. Entities, which can be defined as an object with unique identity stuck and persist during the course of time. For instance, in banking application, customers and bank accounts can be classified as entities. An entity can be distinguished from another entity by looking at GUID (Global Unique Identification Number) or the main lock in a database. Entities identities can include some bounded contexts and can last longer than an application usage period, e.g.: bank account or customer ID issued by the bank are not tied to application usage period. Entities attributes can change over time, like one's legal name can be changed but not the main identity of that person. An entity can be linked to other entity.
2. Value objects. These objects do not have an identity. The only way to distinguish one value object from another is to examine its attributes values. Value objects are immutable. To renew value objects, new examples are always required to replace the outdated value objects. Value objects can have a method to

summarize domain logic, but it shouldn't have a side effect to the objects itself. The example of value objects are color, date and time, and currency value.

3. Aggregates, which define the consistency boundary around an entity or more. Entities in aggregates are called root. Searching for an aggregate can be done by knowing the root entity identifier. Other entities that are derived from root can be referred to by following the clues from the main root. Aggregates' main purpose is to model a transactional invariance. In real life, the relationship network between each aggregate is more complex. For instance, customers make a request, the request contains product specification that the customers desire, products are delivered by suppliers, and so on. If the application modifies some related objects, it is necessary to discover the way to guarantee its consistency, tracking the invariance, and enforce it. Traditional application mostly uses database transaction to ensure consistency. However, in a distributed application, this is almost impossible to do. Single business transaction can only reach some of the data storage, or taking too much time to do, or maybe require assistance from third-party services. In the end, it depends on the application, not the data layers, to apply the invariance needed in the domain. This is how modeling using aggregates are done.
4. Domain services and applications. In DDD terminologies, services are objects used to implement some logics without any status. DDD differentiates between domain services and application services, providing technical functionality, like user authentication or SMS (Short Message Service). Domain services are often used to model behavior surrounding many entities.
5. Domain events, to communicate with other parts of the system when something occurs. As the name suggests, domain events area of concern is only the occurrences in a domain. Example of a thing unrelated with a domain is "notes are put into the table" and the example of a domain event is "Delivery is cancelled". Domain events are relevant in realization of microservices architecture, since microservices are distributed and not sharing data storage. And so, domain events serve as a way for microservices to coordinate with each other. Some DDD patterns like factory, repository, and module are useful in implementation of microservices, but irrelevant in designing boundaries between microservices.

2.2 Specialized Theory

According to reference [23], Fintech, a portmanteau of the word financial and technology, is a usage of technology in delivering a new and a better form of financial services. One of the reason to adopt Fintech is despite the convenience of information technology has tackled costs and functionalities related issues, the seemingly inevitable intermediary costs of using financial services never change for more than a century.

Reference [24] stated that Fintech is a part of the growth of innovation process in financial industry that is risky theoretically but valuable. This finding is supported by the latest proof that Fintech gives substantial value to the investors. Reference [25] also stated that Fintech is the point where financial services and technology collides. This is not a new phenomenon for banking companies and financial services providers in the era of Fintech the last few years. The evolution of Fintech history are as follows:

1. Fintech 1.0 (1866 to 1967). The first Fintech era began from the installation of the first transatlantic undersea telecommunication cable and the invention of ATM (Automatic Teller Machine), where finance and technology are combined to mark the first period of financial globalization. A technology like telegraph also supported the relationship of finance across boundaries, enabling fast transmission in financial information exchange, transaction, and payment.
2. Fintech 2.0 (1967 to 2008). The second evolution in Fintech history is when it remained in exclusive sectors for decades and was mainly dominated by conventional financial services providers regulated by the authorities that used the technology to provide financial products and services. This Fintech age is the witness of the beginning of the introduction of electronic payment and clearing system. Also, usage of ATM was commercialized with many units of the machine being produced, which is also supported by the deployment of online banking system worldwide. In mid '90s, financial services industry became the largest purchaser of IT assets, a record which hasn't been surpassed until now.
3. Fintech 3.0 (2008 to the present): Since the global financial crisis in 2008, startup companies with disruptive nature and

powerful technology enterprises began to send financial products and services directly to business or known as B2B (Business-to-Business) and the society where newcomers had tendency to focus at providing single solution designed to offer better experience in only a single product or service. The key connection of this behavior with Fintech 3.0 is that the competition amongst banks and other financial services providers were no longer from the same businesses. New competitors have surfaced in the form of Fintech companies, which reshaped the way financial services are delivered by using advance technologies. Fintech companies in this era are capable of mirroring the disruption, as seen by wider economic scope with the emergence of online platforms like Amazon or Uber that managed to overcome the already established conventional financial system. Instead of replacing the current system, Fintech companies have invented an important infrastructure instead. Fintech is also capable of developing on its own and have been acknowledged worldwide. McKinsey, a global consulting firm, has calculated that more than 2,000 startup companies offered financial services, and predicted that there might be 12,000 more that will become Fintech companies.

Nowadays, Fintech companies have vast improvement, so the concept of Fintech itself must be implemented to firms or organizations that support the improvement of economics, like cooperatives, whose system currently are very traditional. Cooperatives need to adopt a system that is used by Fintech companies, or else, the cooperatives may be less favored by costumers of the financial industry if they view Fintech companies as a more attractive alternative to use financial products and services.

Cooperative is derived from the Latin word *coopere* which is translated literally as cooperation or cooperative in English. Cooperation means striving together to achieve mutual goals. In accordance with the 1945 Constitution of the Republic of Indonesia Number 25 Article 1 Section (1), cooperatives are business entities that aims to bring prosperity to its members by cooperating with each other through active participation in the services provided by the cooperatives [26].

One of the most common types of cooperatives easily found across the world, is savings and loan cooperatives, or often called CU (Credit Union) [27]. The purpose of CU is to encourage its members to save their funds, allocate the accumulated funds from the members to provide loan for members in need according to the principle of democracy. There are approximately 88,000 CUs and other cooperatives around the world. In 1986, the membership of CUs almost reached 114 million people with total accumulated savings of 540 billion US dollars and total loan outstanding of 380 billion US dollars.

2.3 Relevant Past Researches

Research by reference [28] entitled "*Modeling Microservices with DDD*" explained that DDD can help in providing functional comprehension on microservices architecture. Although, the method to apply this idea is unclear to domain practitioners in common. DDD is a domain modeling technique developed in early 2000 while microservices are developed and popularized years after in 2015. This past research discussed the basic conception of DD and why and how it can help developing microservices with better availability, scalability, reliability, and capability of modification. The researcher navigated domain model built with DDD into microservices design based on synchronous REST and asynchronous or reactive communication protocols. The researcher dwelled further into five scenarios of microservices surrounding the DDD's aggregates, bounded contexts, domain events, and other strategies for bounded contexts to interact with each other.

A research by reference [29] focused on an IT company developing e-procurement application as the basic product system to fulfill the needs of automation in internal procurement processes discovered that the system was built on monolithic architecture where the application is enveloped in a large package and had high dependency among modules. The modules were defined as the entire process of integration, where if changes occurred to certain module to adapt with business processes, this will affect the other module functions. An alternative for another system architecture style should be put into consideration to have a more adaptive e-procurement application system and to be able to overcome issues generated from usage of the old architectural style. This past research suggested the design of transformational process using microservices architecture with DDD approach. The results indicated that microservices architecture is capable of being more adaptive towards change where the modules are designed to be standalone modules.

Research entitled "Implementation of Microservices Architecture to Comrades Application Backend" by reference [30] was conducted with Comrades, a smart application that acts as an information and educational medium about HIV/AIDS disease, as the research object. Comrades help every HIV/AIDS sufferer to exchange information and console each other while able to interact with general society. In its development, this application partnered with the community of HIV/AIDS patients within close proximity. It was predicted that the users of the application would increase due to the partnership. As a preventive measure to handle the surge of application usage, performance upgrade is necessary for Comrades. This research sought to improve the performance and availability of Comrades for its users. This could be achieved by implementing microservices architecture into the web application that was previously built on monolithic architecture. DDD was used as the approach to split the services. From architectural perspective, the physical aspect would change after implementation of the currently advanced technologies like Docker, Kubernetes, and API gateway.

Reference [15] implied that Docker can help developers to devise a system to develop, implement, and spread services, especially to achieve the desired level of system performance in e-commerce web services. Docker will contain all information center and dependence of the system used within a single package. This can simplify the process of implementation and distribution of the system. In designing system using microservices architecture, the entire system is broken down into several smaller services. Afterwards, some factors of service separation should be paid attention to, such as service dependency and service communication. Microservices should be developed with care unless the difficulties of system development will only increase.

Lessons learned from the conclusion of each relevant past research suggested that the advantages of using microservices architecture in developing a system are flexibility and maintenance of the system. System performance can also be maximized with the freedom to build the system using different programming languages and databases. Changes and improvement in one service won't affect the other services in work, as long as the service does not depend on each other. This is very important as business processes can continue to improve, so long as the system has good adaptability to changes. Using the example of these past works, the design of microservices architecture using DDD approach for PT XYZ loan application system is expected to yield satisfactory results. However, these past researches have limited amount of information about the technological requirements needed to carry out the design processes of microservices, such as the software stack. This research is done by circumventing the limitation with borrowing data from PT XYZ's existing system to learn more about the software stack and which part of them can be complemented or replaced with a better alternative to better suit the newly designed microservices architecture.

From a critical analysis of the literatures, research questions related with computing contribution in the current research can be stated as follows:

1. What software requirements are ideal to support the design of microservices with DDD approach?
2. How improving the IT system can contribute to improve the competitiveness of a cooperative enterprise toward Fintech companies?

3. DESIGN METHODOLOGY

3.1 Reference Framework

This research is motivated by the needs of converting PT XYZ's system to a more advanced system architecture using microservices from the currently used monolithic architecture. Microservices are chosen because it is tougher, more measurable, can be implemented independently, and quicker in improvement than monolithic architecture. The steps that the research is going to take begin with collecting data and analyzing the existing architecture system. Adoption of DDD approach is going to be done to design microservices, followed by analysis of the existing application functionalities to understand its workflow before designing microservices.

3.2 Research Flow

The research flow to realize the design of microservices architecture using DDD approach is depicted in Figure 2 below.

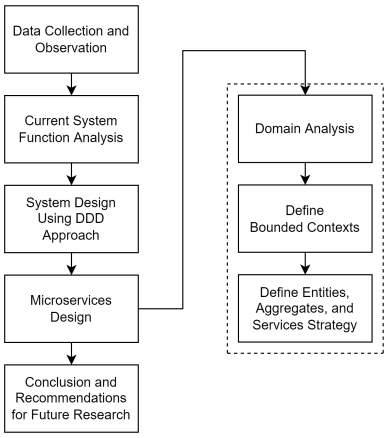

Figure 2: Research Flow

3.2.1 Data collection and observation

Data is collected by interviewing the PIC (Person in Charge) of the system management in PT XYZ and by observing the existing system. These PICs are the CTO and a representative from the backend developer team. The purpose of interviewing the parties responsible for the system is to get a comprehensive details about the currently running loan application system including the system architecture to discover which areas of the system can be improved by migrating the entire system into microservices architecture with every aspects related with loan application as the top priority of concern. From the interview, the flaws of the current system is learnt. Another piece of valuable information showed that the loan application system consists of 5 different modules, namely Customer Services, Loan Application, Loan Calculation, Loan Information, and Register modules.

3.2.2 Function analysis

Function analysis identifies the business processes in PT XYZ, including documenting the features in the existing system. Each feature has its function and actor or user that can access certain feature. This analysis is focused on the existing components in the current system model, such as user manual, user journey from the loan application system. The function analysis results serve as the reference and guidance to design system specifications for the microservices architecture.

3.2.3 System design using DDD approach

DDD is intended to optimize the microservices design for PT XYZ and is expected to portray what the technology and system configuration should be like for the migration to take place. DDD processes identify and classify domain model with these following approaches:

1. Analyzing business domains in PT XYZ to better understand the workflow of application functionalities. This analysis will yield informal domain descriptions which later will be perfected to be a pack of formal domain model.
2. Defining bounded contexts, where the contexts contain domain model representing related subdomains through a larger application.
3. Applying tactical DDD patterns to define entities, aggregates, and bounded contexts of domain services from the loan application system.
4. Using the results from previous steps to begin designing microservices for PT XYZ.

3.2.4 Microservices design

Mapping of loan application concept needed by PT XYZ into features that will be developed into a new loan system is executed with designing microservices with the previously done DDD approach. Reimagining the system design with different architectural style, if done correctly will reduce the difficulty of development. Definition of bounded contexts, entities, aggregates, and services design strategy using DDD approach will make this effort into less laborious for the developers. The design processes are ended with writing conclusion of the research and giving suggestions for improvement possibilities.

3.3 Research Object

PT XYZ, the cooperative enterprise established since December 2019 in Indonesia is the research object in this study. The company offers products like savings, collateral-based loan, and payroll loan. The current monolithic architecture used by PT XYZ has four layers, they are: presentation layer, logic layer, data layer, and database as illustrated in Figure 3.

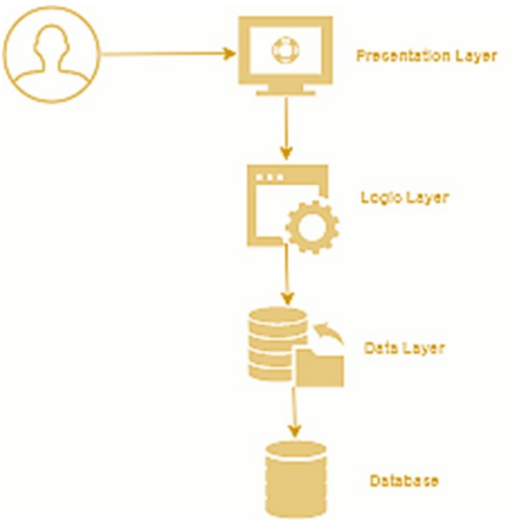

Figure 3: Monolithic Architecture at PT XYZ

4. RESULTS AND DISCUSSION

4.1 Collection of Secondary Data

The main source of data for this research is retrieved from the internal documents of PT XYZ, such as BRD (Business Requirement Document), FSD (Functional Specification Document), and TSD (Technical Specification Document) for written information and interview with the CTO and backend developer representative to have clearer ideas of the actual business processes of the loan application system. Relevant literatures also play prominent role in shaping framework of thinking for the system design planned in this paper. The entirety of the information obtained from the secondary data sources are discussed further in subsections in this chapter.

4.2 Current System Function Analysis

Analysis on system functions reflect the state of system that the research object currently uses to deliver loan services to the cooperative's members.

4.2.1 Existing infrastructure

The existing infrastructure is monolithic in nature with a relatively simple workflow. The application in use is web-based that can only be accessed through the cooperative official website link given to users and visitors alike. Parties attempting access to the application will be directed to Cloudflare interface. Cloudflare's main function is to act as the security layer applied by PT XYZ to protect the main system from the threat of DDoS (Distributed Denial-of-Service). The overall infrastructure used by the cooperative is using third-party services. Aside from using Cloudflare, the enterprise also uses service from a well-known e-commerce tycoon Alibaba such as Alibaba Cloud which includes ECS (Elastic Computer Service), OSS (Object Storage Service), and ApsaraDB (Apsara Database). ECS is a virtual server with stable performance and better efficiency than physical server which can cost more due to heavy requirement to buy capable hardware. Unstructured data or objects like images in large numbers can be stored in OSS while for grouping of data and information to make it easier in identification of data is handled with ApsaraDB. The concise illustration of the existing infrastructure is as shown in Figure 4.

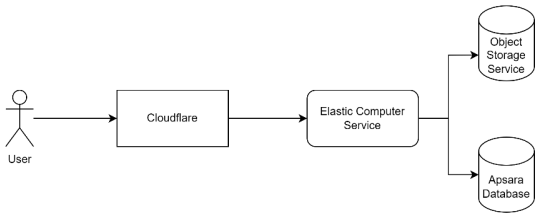

Figure 4: Existing Infrastructure at PT XYZ

4.2.2 Information workflow

Information workflow is divided into several parts according to the purpose of each information. Information workflow in PT XYZ includes Registration Workflow, Loan Application Workflow, and Loan Calculation Workflow.

Registration Workflow depicts the process of every individual that attempts to register for the cooperative membership in PT XYZ. This is the basic procedure to be eligible for using the services of the company, like savings and loan to abide by the mandatory requirement of a cooperative where only members are allowed to use the services of the enterprise. The process begins by filling the registration form. One of the components required in the registration form is the applicant's e-mail (electronic mail). System will give different response to the submission of the e-mail, depending on whether the e-mail has been registered previously or not. If the e-mail has been registered before, the applicant is requested to submit another unused e-mail address. Once new e-mail address is submitted, the system will check for its validity. If it is not used yet, the system will deem it valid and send OTP (One Time Password) to the inbox of the registered e-mail address. Applicant must open the e-mail containing OTP information from the applicant's inbox and enter the correct combination to the provided column in the application interface. There is a possibility that the OTP combination will enter the spam folder instead of the intended inbox folder as a safety precaution measure applied by the e-mail service provider to prevent malicious e-mail

to enter the inbox folder of the e-mail address' owner. Inputting the correct OTP combination will complete the registration processes, making the applicant a formal member of the cooperative. However, if the OTP combination is wrong, the applicant should request for another OTP and recheck their e-mail folders and reinput the new OTP to complete the registration processes.

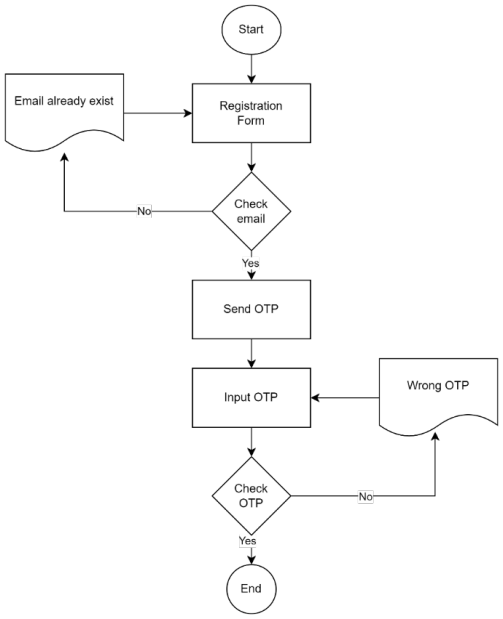

Figure 5: Registration Workflow at PT XYZ

Loan Application Workflow depicts the process of applying for loan or credit involving two parties, namely the cooperative member as the applicant, and PT XYZ as the issuing party. Loan application starts with the applying member filling the application form and submit personal identity documents like photocopies of ID Card and Tax ID. Assigned staff will check the validity of the documents by using e-KYC (electronic Know-Your-Customer). If one of the submitted documents do not comply with the terms, like unclear or poor visibility of the information contained in the documents and/or the information in the documents do not match the records in e-KYC, then the applicant is required to submit clearer documents. If the applicant cannot comply or refuse to comply, the application will be rejected immediately by the staff. When the documents submitted met the requirements of the cooperative, a survey and an interview will be scheduled for the applicant to learn the reason for their application, to recognize their personal background, and to know what collateral the applicant will offer to PT XYZ should the loan application receives approval from the committee. Following the survey and interview, the assessment to determine if the applicant is creditworthy or not will be conducted. The assessment consists of credit collectability rating in PEFINDO credit bureau, an Indonesian credit bureau in charge of credit rating information nationwide, and appraisal of the offered collateral by internal appraisal officer from PT XYZ. If both assessment are passed, the application will be approved. If one or both of the assessment are not passed, application will be rejected. The applicant is obliged to handover the asset that is offered as a collateral for the loan after request for loan has been granted. The collateral will be bound by law via credit pact with a notary as the authorized personnel to legalize the credit pact. Pact must be signed by the applicant and a representative from PT XYZ. Signed pact is required for credit disbursement. After receiving loan, the applicant is officially a borrower to the cooperative.

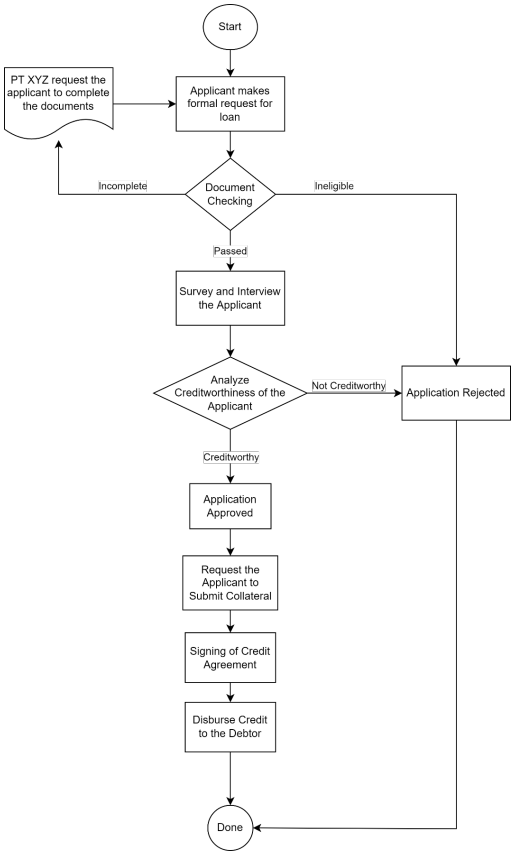

Figure 6: Loan Application Workflow at PT XYZ

Loan Calculation Workflow is a simple workflow where the visitor to the web application of PT XYZ can access a calculator feature built on it to estimate the costs to spend for paying loans issued to the applying member should the loan request be granted. The process is very linear where

it begin with the visitor accessing the application and clicking on the calculator feature without having to login, as loan account is given only to borrower. Visitor that is not a borrower doesn't need to have a borrower account to access the calculator feature. Then, on the loan calculation interface, the visitor must input the desired loan amount from the range between IDR 5 million to IDR 1 billion and choose credit tenor with the range between 1 month to 12 months. The next step requires the visitor to click on Calculation button. The system will respond by displaying estimation of the costs with the composition such as provision fee, amount of loan disbursed, and monthly interest based on tenor chosen. Calculation result cannot be displayed if the information is not properly given by the visitor.

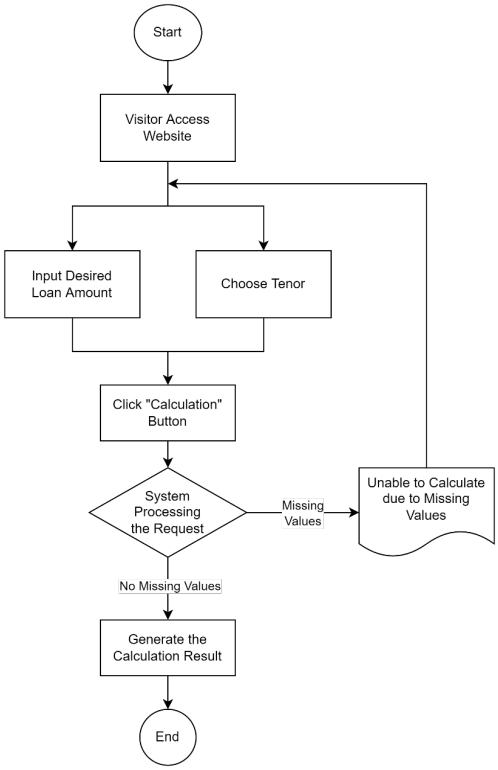

Figure 7: Loan Calculation Workflow at PT XYZ

4.2.3 User workflow

User workflow enables users to engage in activities that may vary depending on users' access rights. PT XYZ divided user workflow into two parts, they are Information Workflow and Customer Services Workflow.

Loan Information Workflow portrays the process of retrieving borrowers' personal information kept in the database. Every borrowing member will be given access to a personal account with a unique password to retrieve the respective borrower's loan information through the application. These information are exclusive and private to every holder of each respective account. Thus, only the owner of the account is authorized to access it or PT XYZ's staff with access authorization. The flow is quite simple. Borrower only needs to access application website, input the given account name and password, then the system will direct to a number of information like borrower's name, borrower's unique identifier number, loan amount, tenor, next payment amount, payment due date, phone number of the PIC from PT XYZ assigned to the borrower's account, and another information related with the identity of the borrower. To exit from the information window, the borrower as the user only needs to logout from the account.

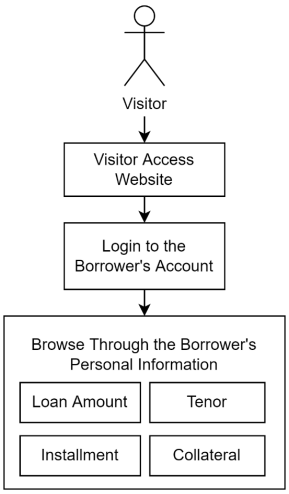

Figure 8: Loan Information Workflow at PT XYZ

Customer Services Workflow explains how the website application visitors can discover solutions to problems or issues the visitors faced or may encounter in the scope of the cooperative's services independently, especially those related with loan processes. Upon visiting the website, a menu dedicated to answering questions from the visitors can be located on the website. The menu, when clicked, will direct the visitors to compilation of FAQs (Frequently Asked Questions). The FAQs are accompanied by relevant answers to help visitors in solving problems. If the FAQs cannot solve the problems or if the visitors are unable to find the expected answers, then the visitors can send request to be assisted by a customer service using live chat feature.

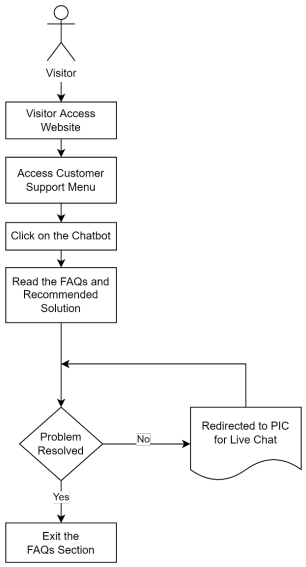

Figure 9: Customer Services Workflow at PT XYZ

4.2.4 Endpoints identification

Endpoints in the existing infrastructure is identified to make mapping of the problems easier in the process of converting from monolithic architecture to microservices architecture. Identified endpoints are later grouped based on the domains the endpoints interact with and categorized based on its functions. Endpoints came in the form of API bridging applications with each other in interacting with the backend system. API versioning is also added to anticipate the need of migrating application to new API while maintaining REST API currently in use. Table 3 in appendices section contains the full list of endpoints being part of the existing API. The list can help in tracking the trails of the domains interacting with the endpoints.

4.2.4.1 Endpoints ownership

Endpoints ownership correlates with parties involved in the business processes of PT XYZ, particularly as users of respective endpoints. Ownership identification of endpoints are necessary to tackle future problems or when APIs are changed. Owners that can create, maintain, and modify the endpoints according to each owner authorization in PT XYZ are grouped into Administrator & User, Credit Committee, dan Customer Care, as listed in Table 4 in appendices section.

4.2.4.2 Endpoints functionalities

In entirety, the loan services for the cooperative members are divided into 3 major groups representing each respective function. The existence of the services are complimenting with each other. The services comprised of Common Services, Loan Services, dan User Services. Common Services have services assisting the process of an individual attempting to join the cooperative membership of PT XYZ. Loan Services are a group of massive service helping in processing loan applications from the members from the initial step of applying through the approval process by credit committee. The self-service tool to count the estimation of credit costs are also a part of the Loan Services. User Services include services facilitating the members of the cooperative in obtaining information to dissolve or solve loan related problems, like the assistance from appointed staff to answer inquiries that is not contained in the list of FAQs. The complete list of endpoints functionalities can be seen in Table 5 in the appendices section.

4.3 Designing System with DDD Approach

One of the foundation of a proper software development is to instill awareness about domain. This is the basis of the approach of DDD in designing microservices architecture as the replacement of the monolithic architecture PT XYZ uses in this study. This cooperative enterprise in its role as the deliverer of loan or credit to the members ought to consider the stakeholders interests in loan-related activities, be it the cooperative members, or staffs in charge of delivering the loan products to the members, or regulatory firms as the watchful eye toward the loan delivery processes.

The current system architecture is entirely monolithic in handling every aspect of the cooperative services, including loan services. The consequences of using this traditional system is the heavily tied large number of codes required in running the system functions properly. Services delivery and features addition are slowed down as the result. Microservices with DDD approach are expected to reduce dependency among the loan services.

Domains are crucial in deliverance of services in real life practice. Domains work by informing the requirements and acceptance criteria of the to be developed system. Thus, domains can display high-level of segregation to various business areas. Software developers using DDD approach work in a team that must abide by the compliance to general patterns in software development to have every program capable in working together in building software. Precise focus division is needed by putting contexts boundaries in designing the microservices for PT XYZ.

Domain model captures the concept and process of certain business domain that strongly require understanding of the related business domain. The best way to comply with the need is to do event storming to better understand processes carried out within a domain. Process begins from an event context happening inside the domain which is viewed as a fundamental element within a model.

4.4 Designing Microservices

The perks of using microservices architecture highlighted in this design plan is the capability of reducing dependency among services, as explored before in this paper to achieve the desired system outcomes. System design will use 3 service candidates derived from the existing endpoints categories, such as Common Services, Loan Services, and User Services. Services in the microservices architecture design for PT XYZ will be contained separately as independently functional applications. In the architectural model, API gateway will play a vital role as the entry point of clients to connect clients with the requested services. System management will connect the services with nodes, identify failure should it occur, and balance the services across nodes. Figure 10 illustrates the difference between the existing monolithic architecture in PT XYZ and the design plan for microservices architecture.

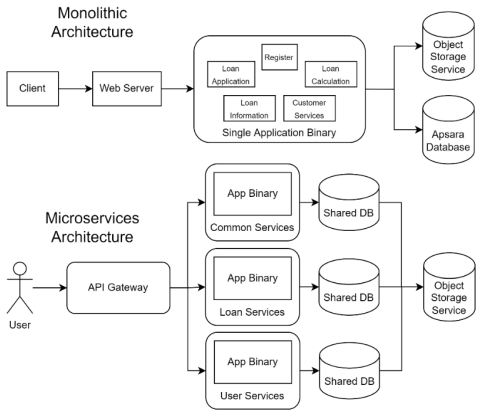

Figure 10: Monolithic Versus Microservices in PT XYZ

4.4.1 Domain analysis

The interaction of domain with each other, including subdomains based on the 5 modules of PT XYZ are as illustrated below in Figure 11.

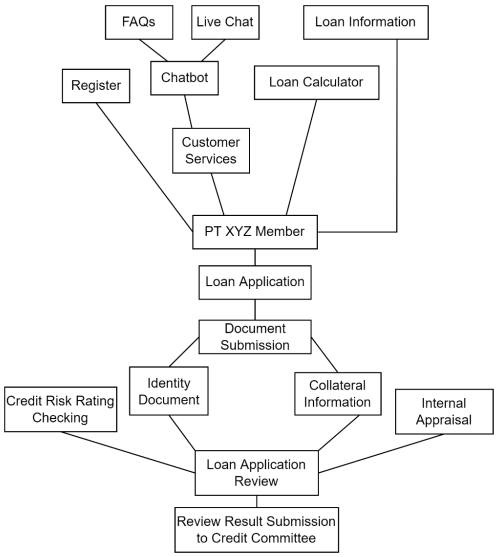

Figure 11: Domain Analysis in PT XYZ

Every domain and subdomain have these roles assigned to help the execution of these tasks:

1. Loan Application is centered in the diagram as it is the business core, to provide loan for the cooperative's members. Loan Application consists of every process related to loan applications submitted by members to the loan department of PT XYZ. Subdomains correlated with this domain are PT XYZ Member and Document Submission.
2. PT XYZ Member is acting as users' management to connect cooperative members as the candidate to apply loan with various services, such as: Register, Customer Services, Loan Calculator, and Loan Information.
3. Register helps an individual to register to become member of the cooperative, in order to be eligible for using the products and services in the cooperative.
4. Customer Services acts as a self-service feature to connect web application visitors with FAQs about anything related with the cooperative, including the loan product information. Chatbot subdomain is included into Customer Services domain to deliver the FAQs and answers to the visitors. Live Chat subdomain connects visitors with the staff responsible for answering questions excluded from the FAQs.
5. Loan Calculator helps visitors in estimating the costs of loan services. Detailed discussion of Loan Calculator can be found at the information workflow subsection previously discussed.

6. Loan Information is exclusive to the borrowers as it has private information related with borrowers and will differ for every borrower.
7. Document Submission focuses on handling documents submitted by applicants of loan. Identity Document and Collateral Information are the extension of Document Submission, where Identity Document handles applicants' personal identity information and Collateral Information handles information of the collateral that the applicants offered to PT XYZ as the loan issuing party.
8. Loan Application Review is a domain in charge of the review processes of submitted loan applications. There are 4 components of review, which also included Identity Document and Collateral Information and two other components from other subdomains namely, Credit Risk Rating Checking and Internal Appraisal with the former is in charge of obtaining credit collectability history of related applicant and the later in charge of the valuation of offered collateral. Loan Application Review domain is also correlated with Review Result Submission to Credit Committee domain that is tasked with collective review to determine the approval status of loan applications—to reject it, or to approve it, based on information from the aforementioned 4 components.

4.4.2 Defining bounded contexts

Bounded contexts can be regarded as a contextual semantic border in a domain where every component of software has its own meaning and unique set of tasks. Bounded contexts is a central pattern and the deciding factor of assigning domain roles in DDD approach. Bounded contexts are used by software developers to know when to cease from pouring more concepts into the domain model in design process. Bounded contexts set in this research are classified into Information Context, Loan Application Context, and Loan Review Context. The merging of these domain contexts are executed by using context mapping integration with RESTful HTTP (Representational State Transfer Hypertext Transfer Protocol). The illustration for bounded contexts set for the DDD approach can be seen in Figure 12.

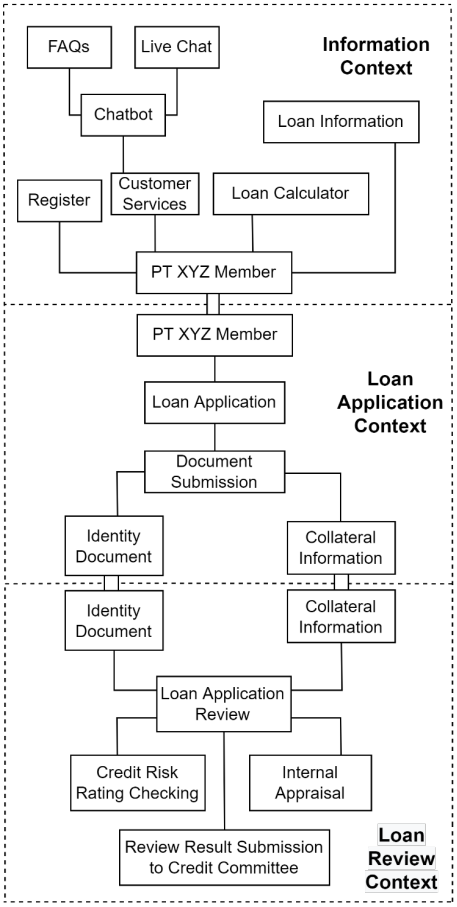

Figure 12: Bounded Contexts in Domain Model of PT XYZ

4.4.3 Defining entities

Following the completion of business domain mapping and bounded contexts setting, entities modeling is necessary to govern domain information. Entities modeling in DDD approach basically involves data and behavior and application of additional DDD patterns (aggregates, value objects, etc.) that can support long term success of the plan for designing microservices in this study.

Entities function by representing domain objects, defined by identities, continuity, and persistence from time to time, and is not limited to the attributes consisting of the entities itself. Entities' identity can span across microservices or bounded contexts. Although entities need identity, there are objects in the system that doesn't need identity, like value objects. The following are model design for entities that are going to be used in domain design in this research, each accompanied by respective explanations.

User Entity is a part of the domain storing and managing access data and information of users involved in accessing the loan system, such as cooperative members, reviewers, committees, and admin staffs and customer service staffs.

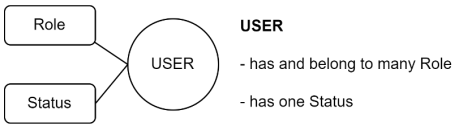

Figure 13: User Entity

Member Entity is a part of the domain storing and manage data of processes in member registration, where the record of the data in the form of e-mail, name, national ID number, Tax ID, and other forms of personal information lie inside. Member Entity is connected to User Entity as the User Entity is aggregated to OTP verification code information required in the process of membership registration verification.

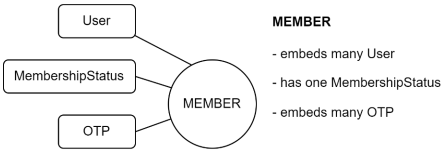

Figure 14: Member Entity

Borrower Entity is a part of the Loan Application domain. This entity is aggregated to Member Entity as an enforcement of a cooperative's basic rule where only a member is allowed to apply for loan services. Information recorded in this entity are IdentityData and GuarantorData. IdentityData stores record of borrower's personal information, similar to the member's personal information stored in Member Entity, whereas GuarantorData stores information related with collateral ownership, like collateralized asset certificate of ownership.

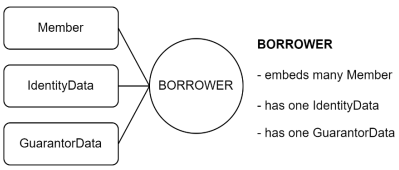

Figure 15: Borrower Entity

Like Borrower Entity, Loan Entity is also a part of the Loan Application domain, which is responsible for storing and managing loan application data. Loan Entity is aggregated to Borrower Entity that stores the important information needed by Loan Entity. Both entities are correlated with each other to realize the Loan Application domain processes, especially in Loan Application Review and Loan Application Approval involving Reviewer and Committee teams. In reviewing and decision-making of loan approval, Loan Entity will record ReviewerResult information containing specific details of examination, application status change, rejection of application, or forwarding to the Committee team for final approval process. CommitteeResult records the final decision of approving or rejecting the loan application, if the final decision is to approve the application, it will also have details of the loan amount, payment terms, covenant to fulfill, and another piece of information related with loan disbursement. All the approval results will also later be recorded as Loan Result Scheme. In every loan application information, system will also record every changes in the process of application in ApplicationStatus.

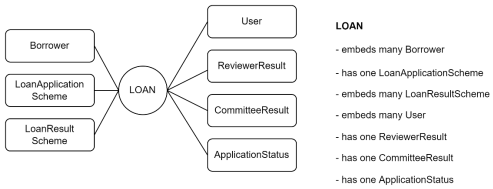

Figure 16: Loan Entity

Loan Result Scheme Entity is a part of the Loan Information domain but also has relationship with Loan Application domain. This entity plays the role of keeping and managing the approval information and the loan specification exclusively associated with the respective applicant. This entity is aggregated to Borrower Entity that keeps essential information like borrower's personal data. For easier information storing, Loan Result Scheme Entity is further divided into smaller information groups, such as: LoanAmount, LoanTenor, Installment, Collateral, and LoanPaymentStatus. LoanAmount has the accepted amount of loan disbursement. The amount of loan could be equal or less than the amount requested, but not greater. LoanTenor is the credit term ranging from 1 month to 12 months, depending on the approving decision of the committee. Installment is monthly expenditure that the borrower must pay as obligatory act of using the loan in accordance with LoanTenor. Collateral has the collateralized asset information, consisting of the collateral type, name of the owner stated in the collateral legal document, and another information related with the collateral. LoanPaymentStatus is useful in monitoring the

payment status of the borrower to ensure that when payment is due, the installment and other credit costs are paid accordingly to prevent the collectability rating of the borrower to fall into the category of NPL (Non-Performing Loan).

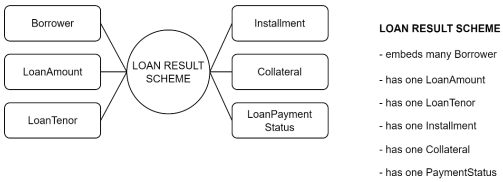

Figure 17: Loan Result Scheme Entity

FAQ Entity is the central source of information that acts as the knowledge base of services and general information surrounding PT XYZ. FAQ Entity is broken down into smaller groups, they are Question and Answer. Question is a compilation of the question mostly asked by visitors while Answer provides the answers to the questions. Another small component called Topic is applied to categorize each question and answer.

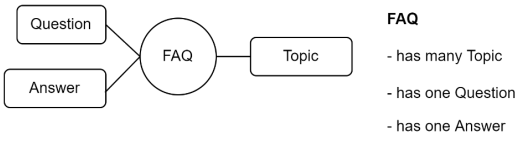

Figure 18: FAQ Entity

4.4.4 Defining aggregates

Aggregates definition is done to model transactional invariance. Business process in real life has a complex relationship network, where applications are often not properly distributed. A business transaction can span across several data storage, runs longer than expected, or require third-party services to run. The execution in the end depends on the application to enforce invariance needed by domain, not the data layer. Aggregate modeling can help in bounding contexts around one or more entities to ensure consistency, tracking invariance, and enforce them. For instance, the defined aggregates depicted in Figure 19 in the appendices section has Loan Application running aggregate function to Borrower, particularly in determining borrower and guarantor information. These information are needed to be located and maintained for consistency in business transaction. However, from aggregate's perspective, Borrower aggregate is not responsible to Loan Application aggregate to convince that BorrowerID is unique. It is up to Loan Application aggregate itself to entirely maintain its invariance in the context of Loan Application as a domain in the system.

4.4.5 Defining services as the core component of microservices design

Services to create the microservices design needs to be formed in regard to the DDD approach. The components for materialization of services into the microservices architecture for PT XYZ loan application system are discussed in the subsubsections below.

4.4.5.1 Designing single services

Single services are designed to be able to operate independently yet capable of communicating with each other with a light connection mechanism, like HTTP. The services are designed to suit every business line capabilities for it to be able to run independently with automatic deployment by deployment server. Each service has center management, enabling different programming languages and database to be used for each service. The single services consist of a web server, focused service binary for certain domains, and Redis cache system.

4.4.5.2 Database for every service

Database are designed for every service with respect to consistency of aggregate forms with bounded contexts. Creating database this way will reduce coupling of the services.

4.4.5.3 Container strategy

Planning for microservices architecture design on loan application system comes with consideration for containing the service applications. This containing is executed to distribute services in infrastructure level prioritizing convenient practice and service capabilities to operate simultaneously yet independent. The researcher uses Docker for services containment in this design, which uses vary, i.e.:

1. Provide binary images that prepare installation, configuration, and testing for every software needed to form the services for the designed microservices architecture. Data distribution is made convenient with the inclusion of every necessary data needed for the process.
2. Interactively create images for software developers to learn the software installation process transparently. This ability ease the documentation process of system design with expected precision.
3. Usage of Docker can reduce the significance of fixing and improvement of the system with versioning, changes in system dependency, security level, addition of features, or deletion

of outdated software that may potentially harm the system.
4. Docker tackles constraints in adoption and reuse of existing system with its reproductive capabilities for more convenient adaptation process to the existing system workflow by management using script.

Container has logic containing mechanism where applications can be separated by decoupling that will enable easier use with maintained consistency. Different containers can exchange information with each other with this mechanism.

4.4.5.4 API gateway strategy

API gateway connects clients and services with its capabilities of running tasks across sectors with aggregation, SSL (Secure Sockets Layer) checking, and load balancing. In this design plan, the researcher recommends KrakenD, an API gateway services with benefits including:

1. Information aggregation from various information sources to the central endpoints.
2. Manipulating responses along with grouping and wrapping.
3. Filtering and shrinking of response to enable abstraction.
4. Limiting unnecessary connection with rate limiting function.
5. Enabling circuit breaker and/or another system security measures.
6. Providing various password protocol encoding formats.
7. Compatible with various middleware and plugins.
8. Configuration of DSL (Digital Subscriber Line).
9. Support for usage of scripting language across sites.
10. Limiting host-based connection.
11. User quota management.
12. Support to SSL.
13. Support to OAuth CCG (Open Authorization Client Credential Grant).
14. Protection against clickjack.

KrakenD offers benefits that normal API gateway may not be capable of as it uses backend for frontend approach, enabling a more efficient process of adopting existing APIs into newer system's API, making it suitable for designing microservices.

4.4.5.5 Communication strategy

Services relying on microservices-based system runs simultaneously across more than one server or host. This simultaneous interaction becomes a necessity to maintain the complex system to be able to run without disruption. By combining two communication protocols across services, namely sync communication protocol and async communication protocol to suit various services requirements, the need for simultaneous interaction can be carried out with less hassle.

Sync communication protocol works when a request is sent, and system waits for response from service receiving the request. The limitation of this protocol lies on the inability of clients continuing tasks before server respond to the request. In contrast with sync communication protocol, async communication protocol doesn't rely on server's response to continue doing tasks. The recommended sync communication protocol and async communication protocol for the microservices architecture design are HTTP and Kafka respectively.

4.4.5.6 Proxy and firewall strategy

Firewall and proxy act as the first line of defense for system security to protect the system from external threats like hackers attempting to disable the system. Both protect the system from harm yet located at different parts of the system. Firewall is located at network later whilst proxy is placed at application layer. Firewall halts all unauthorized access to the system while proxy acts as an intermediary between the clients and the network. Cloudflare, a software useful in mitigating DDoS attacks is capable of doing both tasks of securing the system. Cloudflare can mitigate the risk of private information theft in the loan application system like the cooperative members' data and borrowers' data.

4.4.5.7 Resilience strategy

Resilience in the context of IT system means capability of a system to recover partially or fully from the aftereffect of system failure. Circuit breaker can be used in microservices architecture as a preventive mean to circumvent system failure. System defense can be upgraded if the circuit behavior is suited for partial or full network interaction with the services. Circuit breaker isolates services with problems so cascading won't spread to other services to ensure stability of the clients and the overall services. The researcher suggests for a custom build circuit breaker for the microservices design to give convenience in customization whenever needed.

4.4.5.8 Monitoring strategy

Services monitoring are going to be done with logging as a supporting tool in analyzing issues in the system and to better understand the needs of borrowers as the service users to improve loan products and services performance.

Graylog is a log management software suggested by the researcher to monitor the services in PT XYZ microservices architecture design. It is capable of reducing complexity and time required to explore large amounts of data so clients of the system can interpret data to make decision faster.

4.4.5.9 Testing strategy

The tiniest unit of separated business logic is the basis of constructing microservices. To ensure the business logic can interact with each other through network, testing on this unit is important. Testing is conducted by validating each business logic or service separately.

Katalon Studio is a software suggested for testing of service applications qualities and functions. The software is capable of testing web-based applications and APIs without requiring mastery of a specific programming language. This removes one hurdle for testers in doing automated testing. Both expert testers or beginners can utilize Katalon Studio easily because of its simple yet informative UI (User Interface). Testing can also be done across different web browsers, which means testing on several browsers can be done at the same time to save time and effort from the testers.

4.4.5.10 IAM (Identify and access management) strategy

IAM (Identify and Access Management) can be utilized to manage access and user identities surrounding system, application, and network so each user can have access rights accordingly, or to identify if the user is the authorized person, in regard to access time and access purpose.

Keycloak can be used to support IAM processes with its varying features, they are:

1. Users don't have to authenticate the applications one by one. The authentication process can be replaced by Single-Sign On, making one attempt is enough to login to all applications. The same principle also applies for logout, where Single-Sign Out enables logging out from all applications with only one attempt.
2. Login is made convenient with options to login from social network.
3. Keycloak can be opted to connect to LDAP (Lightweight Directory Access Protocol) or to active directory.
4. Admin console makes it possible to activate or deactivate application features, managing application authorization and services, and managing users.
5. Using standard protocol that supports OpenID Connect, OAuth 2.0, and SAML (Security Assertion Markup Language).

In entirety, Keycloak is helpful to manage user authorization to suit the needs of developer team or as desired.

4.4.5.11 CI/CD (Continuous integration/ continuous development) strategy

As explained before, containing using Docker can be a solution to lessen the gap between applications and system infrastructure. However, if applications grow in number in the microservices architecture, the management and deployment of services will be harder as services containment won't suffice. And so, usage of container on services can be merged with CI/CD approach for automatic deployment processes of the service whenever update is needed to the applications.

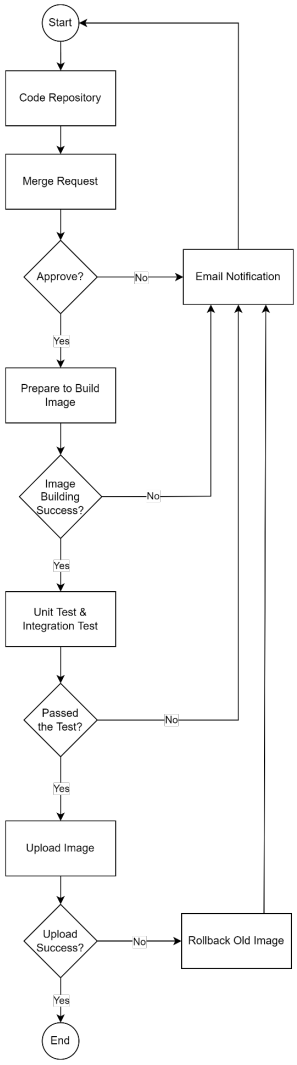

Figure 20: CI/CD Workflow in Microservices Design for PT XYZ

CI/CD is a combination of CI (Continuous Integration) and CD (Continuous Deployment). CI is common in software development. It works by storing application codes in the repository to be tested automatically and quickly with high testing frequency. A completed CI process will be continued to CD process immediately. CD works by using script that will take the codes from the codes center in the repository to prepare for construction and installation of applications into the services environment. CD doesn't manual interference nor confirmation.

CI/CD approach for the design in this research will use GitLab, a multifunctional version control software to operates several software simultaneously. GitLab helps different developers to collaborate in system management and reduce difficulty in program updating processes automatically without rendering the system to experience downtime whenever update is needed. For every update to be performed, the system must be able to continue working so that delivery of loan services to the cooperative members will be timely. Figure 20 shows the workflow of CI/CD that is applied to microservices design for PT XYZ.

4.5 Findings and Discussion

A series of discussions surrounding the services development aspect in microservices design for PT XYZ correlates with recommended software to shape the design into a solid architecture design. Software stack in this respect, is a unity of system pillar to support the system. Table 6 is listing down the software components proposed for the design plan of microservices using DDD approach in this research paper.

Table 6: Software Stack for Microservices Architecture Design of PT XYZ

No.	Category	Candidate Name
1	Programming Language	Golang
2	Database	Postgre Database
3	Cache	Redis
4	Web Server	Google Cloud Platform
5	API Gateway	KrakenD
6	Logging	Graylog
7	WEB UI	Next JS
8	Container	Docker
9	Firewall	Cloudflare
10	Proxy	Cloudflare
11	Sync Communication	HTTP
12	Async Communication	Kafka
13	Access Management	Keycloak
14	Circuit Breaker	Custom Build
15	Test Framework	Katalon Studio
16	CI/CD	Gitlab
17	Versioning Control	Gitlab

Modernization of loan services in a cooperative is semi-automatic. PT XYZ is no exception as the cooperative enterprise needs a dynamic system architecture design that can move across sectors. Microservices as the evolution of traditional SOA, can answer that need. Use of microservices is suitable for loan application system of PT XYZ whose services are divided into domains and subdomains.

Microservices architecture benefits that PT XYZ can experience if this cooperative decides to convert from monolithic architecture into microservices architecture are as follows:

1. Distribution of software composition capable of communicating to the entire network. System failure related to loan business line working across domains can be minimized or mitigated quickly.
2. Services dependency with each other can be minimized by building a dedicated database for each service category.
3. Services in loan business line can be contained and excluded from each other. Every service can be programmed using a different programming language.
4. Microservices design for PT XYZ is capable of producing a more effective API result and existing APIs can be reused in the future. All of these processes can run in a system environment with high flexibility.
5. System users can choose which communication protocol to use between sync or async protocols, depending on the needs.
6. Conversion into microservice architecture doesn't lower protection against threat of system attack by external parties.
7. If system failure occurs in one of the services, the system can disable that service to prevent the error from affecting other services.
8. Can effectively accommodate the effort to monitor the services available because of the system capability of managing large amount of data in a short time. Decision-making regarding issues surrounding the loan application system can be made faster.
9. Testing on the system is easy and fast and can be performed by developers lacking the ability of running difficult programming languages.
10. Better control to user access, especially in assigning authorization for each system user.

11. Continuous integration and services deployment are possible to do because every developers or system administrators can work together for maintenance or deployment of the services.

In comparison with the previous researches highlighted during the analysis of literatures to construct the theoretical framework for this research, the microservices architecture design has addressed the software requirements in a more specific way compared to the referred past researches. Moreover, these past researches only mentioned the necessity of setting bounded contexts among domains and its subdomains but lacking any concrete suggestions of what software are needed to spawn a more realistic microservices architecture design. Peering into the business orientation of a cooperative enterprise, the company emphasizes on delivering financial services to customers, which is similar to Fintech companies. Fintech companies, particularly the Fintech 3.0 companies outshine cooperatives when it comes to utilization of information technologies in business. The revolutionary form of financial service providers have been using technology to deliver services for customers since the day of the Fintech 3.0's inception. A cooperative enterprise can improve significantly by merely enhancing the IT system of the cooperation. By adapting the use of a more computerized operating mechanisms that are properly regulated, the companies are more likely to catch the attention of potential customers and existing customers who favor the use of IT-based financial services. Converting to microservices will bring greater flexibility in modifying the system should the need arise in the future. With so little to no problems to face, the cooperative enterprises can dedicate more focus into delivering services to customers. The use of IT itself is not just to increase competitiveness among financial service providers but is a complete necessity. Keeping up with the technological trends in the financial industry can also prevent the Fintech companies to dominate the entire industry.

Microservices design for PT XYZ doesn't offer solutions without making room for possible issues, mainly from security perspective. Even though with the mitigation measures to defend from cyber-attacks. That is why DDD approach with proper setting of bounded contexts on domains and subdomains are proposed to lessen the risk by shrinking the code base of services. Large code base can create a security hole where system is more likely to be vulnerable to attacks. The proposed design is going to be discussed in FGD (Focus Group Discussion) where system practitioners in PT XYZ are taking part in the discussion, including CTO and the IT team comprises of UI/UX, system developer, frontend developer, web developer, QA, and system analyst. The purpose of conducting FGD is conducted to reach consensus on which part of the proposed design can be implemented in the future.

5. CONCLUSION

5.1 Research Conclusions

The design of microservices using DDD approach yielded a detailed analysis that answers these formulated research questions:

1. How using microservices architecture to design the loan application system for PT XYZ can reduce the difficulty of the system design processes?
   Reduction of dependency among service groups in microservices architecture gives convenience toward IT system design processes at PT XYZ. Each service can be contained without having to be dependent on each other.
2. How the DDD (Domain Driven Design) approach can help in designing microservices for the loan application system of PT XYZ cooperative enterprise?
   SOA is implemented by focusing business into services, where the main service in this case is loan service provided by PT XYZ for its members. Microservices have pivotal role in breaking apart the loan services into smaller subcomponents. These subcomponents are visualized in 5 different modules, which is also classified similarly in the current monolithic architecture, namely: Customer Services, Loan Application, Loan Calculation, Loan Information, and Register modules. The complexity of design using microservices are apparent due to it using large amounts of code, creating a potential for security breach. To circumvent it, DDD (Domain Driven Design) is used to define bounded contexts to reduce the code base in every domain and subdomain in the scope of designed system. The reduction of code base will reduce the chance of hacker attacks.

In addition, the realization of microservices depends on the final decision of the authorized personnel in PT XYZ. The design components take care of the monotony in the current monolithic system design but may create

the problem of complexity. Some components require the handling by experts, while other components might be friendlier to inexperienced programmers. Dissection of the domains and subdomains can be time consuming as it is necessary to execute the task carefully. Reducing code base with careful setting of bounded contexts among domains and subdomains is essential to prevent future errors to occur. Microservices, like monolithic architecture also possess its own flaws that should not be taken lightly. Despite these difficulties, the potential benefits of implementing microservices can overshadow the design difficulties, among these benefits is the guaranteed capability in handling large number of system users, which in the end can boost PT XYZ's business performance in delivering loan for members of the cooperation.

Thus, by enhancing the performance of the loan application system in scaling up business with the ability to establish low dependency among applications, the research is exclusively aimed to solve the system architectural problem in cooperative enterprises like PT XYZ.

5.2 Research Limitations

Determining the correct approach in designing microservices can proved to be a challenge. Since there are a lot of mention about DDD in various literatures, the approach is believed to be the most reliable in designing microservices architecture. However, whether it truly is the most reliable approach in designing microservices is still a matter of question. There is other approach called Clean Architecture proposed by a software practitioner named Robert C. Martin or more well-known by the nickname Uncle Bob [31]. The author claimed that it is a method suitable for designing microservices, but the narratives are built in a book instead of a research paper, making it not suitable to be cited for this research. Not to mention it is also viewed as being impractical and not suitable for rapid application development or product prototyping [32]. It can be concluded that the limitation of this research lies in the difficulty in discovering the possibility of other approaches better than DDD in designing microservices architecture.

5.3 Future Research Recommendations

Microservices might be the solution to reduce dependence among services. However, usage of microservices are more complicated compared to traditional system architecture, which needs intensive care and special countermeasure to tackle potential issues. The researcher recommends which area needs to be improved furthermore, in order for the microservices to be able to run smoother with less issues. These recommendations are not limited to PT XYZ as the research object of this case study, but also for other researchers interested in similar field of study to inspire future researches to be conducted, with the hope that an even better suggestions can be created to perfect the already designed IT system architecture. These are those recommendations:

1. Freedom to use various programming languages between services creates service integration issue. Usage of technological enablers that support the development of microservices without relying on a specific programming language is advised to prevent integration trouble.
2. System practitioners in an organization or firm must have knowledge of domain modeling in an IT system because it is indirectly useful in reducing codes used to prevent hacking risk from unauthorized parties.
3. DDD implementation plan can unnecessarily waste effort and time. DDD is actually more effective to be used on a system with complex domain structure. An approach of system development that is compatible with both simple and complex systems are strongly needed.
4. Microservices existence depend on automation process too much. Cloud technology used to store data needs to be put into consideration carefully so in the case of force majeure, the services can continue to work fine as is.

REFERENCES:

[1] Otoritas Jasa Keuangan, "Perkembangan Fintech Lending Desember 2020," *Otoritas Jasa Keuang.*, pp. 1–11, 2020, [Online]. Available: https://www.ojk.go.id/id/kanal/iknb/data-dan-statistik/fintech/Documents/Statistik Fintech Lending Desember 2020.pdf.

[2] Badan Pusat Statistik, "Ekonomi Indonesia 2020 Turun sebesar 2,07 Persen (c-to-c)," 2021. https://www.bps.go.id/pressrelease/2021/02/05/1811/ekonomi-indonesia-202 (accessed Jul. 15, 2022).

[3] Sudarsono and Edilius, *Koperasi dalam Teori & Praktik*, 5th ed. Rineka Cipta, 2010.

[4] D. J. Teece, "Business models and dynamic capabilities," *Long Range Plann.*, vol. 51, no. 1, pp. 40–49, Feb. 2017, doi:

10.1016/J.LRP.2017.06.007.

[5] F. Dahri, A. M. El Hanafi, D. Handoko, and N. Wulan, "Implementation of Microservices Architecture in Learning Management System E-Course Using Web Service Method," *Sink. J. dan Penelit. Tek. Inform.*, vol. 7, no. 1, pp. 76–82, Jan. 2022, doi: 10.33395/SINKRON.V7I1.11229.

[6] G. Munawar and A. Hodijah, "Analisis Model Arsitektur Microservice Pada Sistem Informasi DPLK," *Sink. J. dan Penelit. Tek. Inform.*, vol. 3, no. 1, pp. 232–239, 2018, [Online]. Available: https://jurnal.polgan.ac.id/index.php/sinkron/article/view/197/125.

[7] Y. Sun, S. Nanda, and T. Jaeger, "Security-as-a-service for microservices-based cloud applications," *IEEE 7th*, pp. 50–57, 2015, doi: 10.1109/CloudCom.2015.93.

[8] C. Richardson and F. Smith, "Microservices: From Design to Deployment," *NGINX, Inc.*, p. 80, 2016, [Online]. Available: https://www.nginx.com/resources/library/designing-deploying-microservices/.

[9] M. Villamizar *et al.*, "Evaluating the Monolithic and the Microservice Architecture Pattern to Deploy Web Applications in the Cloud," *2015 10th Colomb. Comput. Conf. 10CCC 2015*, pp. 583–590, Nov. 2015, doi: 10.1109/COLUMBIANCC.2015.7333476.

[10] K. Ayuwuragil, "Kemenkop UKM: 3,79 Juta UMKM Sudah Go Online," *CNN Indonesia*, 2017. https://www.cnnindonesia.com/ekonomi/20171115161037-78-255819/kemenkop-ukm-379-juta-umkm-sudah-go-online (accessed Jul. 15, 2022).

[11] A. Barczak and M. Barczak, "Performance comparison of monolith and microservices based applications," 2021, [Online]. Available: https://www.iiis.org/CDs2021/CD2021Summer/papers/SA354XK.pdf.

[12] K. Anjaria and A. Mishra, "Quantitative analysis of information leakage in service oriented architecture based web services," *Kybernetes*, vol. 46, no. 3, pp. 479–500, 2017, doi: 10.1108/K-07-2016-0178.

[13] Sasana Digital, "Scale Up Bisnis: Pengertian, Fungsi, Strategi, dan 7 Cirinya," 2022. https://sasanadigital.com/pengertian-dan-strategi-scale-up-bisnis/#:~:text=Scale Up Bisnis adalah suatu,proses%2C teknologi%2C hingga mitra. (accessed Sep. 05, 2022).

[14] UppLabs and T. Smirnova, "From legacy monolith app to microservices infrastructure. Case study," 2020. https://upplabs.medium.com/from-legacy-monolith-app-to-microservices-infrastructure-case-study-90b57821b7ea (accessed Sep. 05, 2022).

[15] J. A. Suthendra and M. A. I. Pakereng, "Implementation of Microservices Architecture on E-Commerce Web Service," *ComTech*, vol. 11, no. December, pp. 89–95, 2020, doi: 10.21512/comtech.v11i2.6453.

[16] S. Santoso, N. Azizah, and A. Astari, "Aplikasi Sistem Informasi Pengajuan Kredit Berbasis Web Pada PD. BPR Kerta Raharja Cabang Balaraja," *Konf. Nas. Sist. Inf. 2018*, pp. 849–855, 2018, [Online]. Available: http://digilib.mercubuana.ac.id/manager/t!@file_artikel_abstrak/Isi_Artikel_822617291516.pdf.

[17] C. Dremel, M. M. Herterich, J. Wulf, J.-C. Waizmann, and W. Brenner, "How AUDI AG Established Big Data Analytics in Its Digital Transformation," *MIS Q. Exec.*, vol. 16, no. 2, pp. 81–100, 2017, [Online]. Available: https://www.researchgate.net/publication/317232875_How_AUDI_AG_Established_Big_Data_Analytics_in_its_Digital_Transformation.

[18] M. Fischer, F. Imgrund, C. Janiesch, and A. Winkelmann, "Directions for future research on the integration of SOA, BPM, and BRM," *Bus. Process Manag. J.*, vol. 25, no. 7, pp. 1491–1519, 2019, doi: 10.1108/BPMJ-05-2018-0130.

[19] H. F. E. Yamany, M. A. M. Capretz, and D. S. Allison, "Quality of Security Service for Web Services within SOA," *Serv. 2009 - 5th 2009 World Congr. Serv.*, no. PART 1, pp. 653–660, 2009, doi: 10.1109/SERVICES-I.2009.95.

[20] I. L. Salvadori, A. Huf, B. C. N. Oliveira, R. dos S. Mello, and F. Siqueira, "Improving entity linking with ontology alignment for semantic microservices composition," *Int. J. Web Inf. Syst.*, vol. 13, no. 3, pp. 302–323, 2017, doi: 10.1108/IJWIS-04-2017-0029.

[21] Microsoft, "Microservices architecture design," 2019. https://docs.microsoft.com/en-us/azure/architecture/microservices/ (accessed Jul. 17, 2022).

[22] F. Rademacher, J. Sorgalla, and S. Sachweh, "Challenges of Domain-Driven Microservice Design," *IEEE Softw.*, pp. 36–43, 2018, doi:

10.1109/MS.2018.2141028.

[23] A. V. Thakor, "Fintech and banking: What do we know?," *J. Financ. Intermediation*, vol. 41, 2019, doi: 10.1016/j.jfi.2019.100833.

[24] M. A. Chen, Q. Wu, and B. Yang, "How Valuable Is FinTech Innovation?," *Rev. Financ. Stud.*, vol. 32, no. 5, pp. 2062–2106, 2019, doi: 10.1093/rfs/hhy130.

[25] Consumers International, "Banking on the future: An exploration of fintech and the consumer interest," *Financ. Dev.*, vol. 56, no. 1, pp. 24–25, 2017, [Online]. Available: https://www.consumersinternational.org/media/154710/banking-on-the-future-full-report.pdf.

[26] T. S. Partomo, L. Krisnawati, and R. Soejoedono, *Ekonomi skala kecil/menengah dan koperasi*. Ghalia Indonesia, 2002.

[27] World Council of Credit Unions, "World Council of Credit Unions Official Website," 2021. https://www.woccu.org/ (accessed Jul. 18, 2022).

[28] P. Merson and J. Yoder, "Modeling Microservices with DDD," *Proc. - 2020 IEEE Int. Conf. Softw. Archit. Companion, ICSA-C 2020*, pp. 7–8, 2020, doi: 10.1109/ICSA-C50368.2020.00010.

[29] A. N. Fajar, E. Novianti, and Firmansyah, "Design and Implementation of Microservices System Based on Domain-Driven Design," *Int. J. Emerg. Trends Eng. Res.*, vol. 8, no. 7, pp. 3058–3062, 2020, doi: 10.30534/ijeter/2020/30872020.

[30] C. S. Budi and A. M. Bachtiar, "Implementasi Arsitektur Microservices pada Backend Comrades," p. 6, 2019, [Online]. Available: https://elib.unikom.ac.id/files/disk1/801/jbptunikompp-gdl-cahyantose-40046-8-20.10114-a.pdf.

[31] R. C. Martin, "Clean Architecture: A Craftsman's Guide to Software Structure and Design," *Prentice Hall*. p. 432, 2017, [Online]. Available: https://www.amazon.com/Clean-Architecture-Craftsmans-Software-Structure/dp/0134494164%0Ahttps://www.safaribooksonline.com/library/view/clean-architecture-a/9780134494272/.

[32] D. Lezhnev, "When clean architecture is not worth it," 2017. https://lessthan12ms.com/when-clean-architecture-is-not-worth-it.html (accessed Sep. 09, 2022).

Table 3: Endpoints List in Existing Infrastructure of PT XYZ

No	Function Name	URL
1	Register Membership	{domain}/api/v1/user/register
2	Verify Member Registration Form	{domain}/api/v1/user/register/?email={email}
3	Get OTP	{domain}/api/v1/user/getOTP
4	Verify OTP	{domain}/api/v1/user/verifyOTP/?otp={otp_code}
5	Create Membership ID Number	{domain}/api/v1/user/create
6	Apply Loan	{domain}/api/v1/loan_application/
7	Check Applicant's Personal Identity Information	{domain}/api/v1/loan_application/checkBiodata/?npwp={npwp}&?ktp={ktp}
8	Check Applicant's Identity Documents	{domain}/api/v1/loan_application/checkId/?data={data}
9	Check Collateral Information	{domain}/api/v1/loan_application/checkCollateral/?data={data}
10	Check Guarantor Information	{domain}/api/v1/loan_application/checkGuarantor/?data={data}
11	Created Unique Identifier for Loan Application	{domain}/api/v1/loan_application/create
12	Input Collateral Appraisal Information	{domain}/api/v1/loan_application/editCollateral/?data={data}
13	Input Loan Collectability Information	{domain}/api/v1/loan_application/editPaymentCollectibility/?data={data}
14	Forward Loan Application to Committee	{domain}/api/v1/loan_application/submitForApproval
15	Change Loan Application Data	{domain}/api/v1/loan_application/edit/?applicant={applicant_id}
16	Change Loan Application Status	{domain}/api/v1/loan_application/editStatus/?applicant={applicant_id}
17	Publish Loan Application Result	{domain}/api/v1/loan_application/notify/?applicant={applicant_id}
18	Display Loan Application Approval Result	{domain}/api/v1/loan_application/detail/?applicant={applicant_id}
19	Calculate Loan	{domain}/api/v1/loan_calculator/
20	Check Loan Calculation Form	{domain}/api/v1/loan_calculator/calculate/?tenor={tenor}&amount={amount}
21	Display Loan Calculation Result	{domain}/api/v1/loan_calculator/result
22	Login	{domain}/api/v1/login
23	Verify Login Form	{domain}/api/v1/loan_information/verifyLogin/?id={memberId}&pass={password}
24	Display Borrower's Information Category	{domain}/api/v1/loan_information/dashboard
25	Display Loan Amount Information	{domain}/api/v1/loan_information/loanAmount
26	Display Tenor Information	{domain}/api/v1/loan_information/tenor
27	Display Installment Information	{domain}/api/v1/loan_information/installment
28	Display Collateral Information	{domain}/api/v1/loan_information/collateral
29	Display Chatbot	{domain}/api/v1/customer_service/chatbot
30	Display FAQ	{domain}/api/v1/customer_service/chatbot/faq
31	Redirect to Live Chat	{domain}/api/v1/customer_service/livechat

Table 4: Endpoints Ownership in Existing Infrastructure of PT XYZ

No	Function Name	Endpoints Ownership
1	Register Membership	Administrator & User
2	Verify Member Registration Form	Administrator & User
3	Get OTP	Administrator & User
4	Verify OTP	Administrator & User
5	Create Membership ID Number	Administrator & User
6	Apply Loan	Credit Committee
7	Check Applicant's Personal Identity Information	Credit Committee
8	Check Applicant's Identity Documents	Credit Committee
9	Check Collateral Information	Credit Committee
10	Check Guarantor Information	Credit Committee
11	Created Unique Identifier for Loan Application	Credit Committee
12	Input Collateral Appraisal Information	Credit Committee
13	Input Loan Collectability Information	Credit Committee
14	Forward Loan Application to Committee	Credit Committee
15	Change Loan Application Data	Credit Committee
16	Change Loan Application Status	Credit Committee
17	Publish Loan Application Result	Credit Committee
18	Display Loan Application Approval Result	Credit Committee

19	Calculate Loan	Administrator & User
20	Check Loan Calculation Form	Administrator & User
21	Display Loan Calculation Result	Administrator & User
22	Login	Administrator & User
23	Verify Login Form	Administrator & User
24	Display Borrower's Information Category	Administrator & User
25	Display Loan Amount Information	Administrator & User
26	Display Tenor Information	Administrator & User
27	Display Installment Information	Administrator & User
28	Display Collateral Information	Administrator & User
29	Display Chatbot	Customer Care
30	Display FAQ	Customer Care
31	Redirect to Live Chat	Customer Care

Table 5: Endpoints Functionalities in Existing Infrastructure of PT XYZ

No	Function Name	Endpoints Ownership
1	Register Membership	Common Services
2	Verify Member Registration Form	Common Services
3	Get OTP	Common Services
4	Verify OTP	Common Services
5	Create Membership ID Number	Common Services
6	Apply Loan	Loan Services
7	Check Applicant's Personal Identity Information	Loan Services
8	Check Applicant's Identity Documents	Loan Services
9	Check Collateral Information	Loan Services
10	Check Guarantor Information	Loan Services
11	Created Unique Identifier for Loan Application	Loan Services
12	Input Collateral Appraisal Information	Loan Services
13	Input Loan Collectability Information	Loan Services
14	Forward Loan Application to Committee	Loan Services
15	Change Loan Application Data	Loan Services
16	Change Loan Application Status	Loan Services
17	Publish Loan Application Result	Loan Services
18	Display Loan Application Approval Result	Loan Services
19	Calculate Loan	Loan Services
20	Check Loan Calculation Form	Loan Services
21	Display Loan Calculation Result	Loan Services
22	Login	Loan Services
23	Verify Login Form	Loan Services
24	Display Borrower's Information Category	Loan Services
25	Display Loan Amount Information	Loan Services
26	Display Tenor Information	Loan Services
27	Display Installment Information	Loan Services
28	Display Collateral Information	Loan Services
29	Display Chatbot	User Services
30	Display FAQ	User Services
31	Redirect to Live Chat	User Services

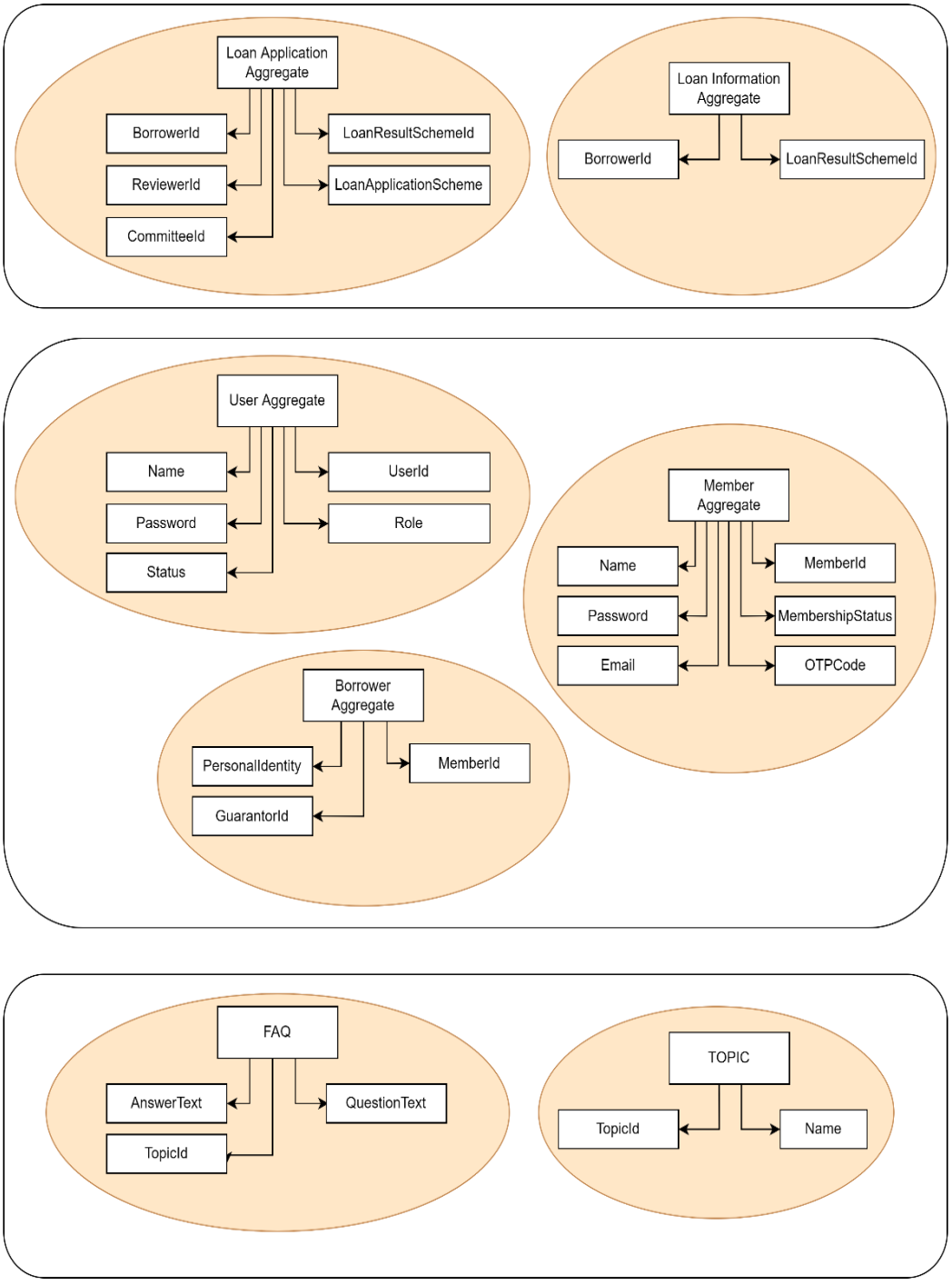

Figure 19: Domain Aggregate