ISSN 1816-6075 (Print), 1818-0523 (Online)
Journal of System and Management Sciences
Vol. 13 (2023) No. 4, pp. 249-264
DOI:10.33168/JSMS.2023.0415

A Design Study of Microservice Architecture on White Label Travel Platform

I Gede Rahmat Wijaya, Ahmad Nurul Fajar

Information Systems Management Department, BINUS Graduate Program-Master of Information Systems Management, Bina Nusantara University, Jakarta 11480, Indonesia

i.wijaya006@binus.ac.id, afajar@binus.edu

Abstract. The travel industry has experienced a surge in online travel agents (OTAs) offering white-label products to meet the growing demand of travelers. However, the complexity of these systems has increased, leading to challenges in integrating third-party systems and developing new features. To address these challenges, this study explores the use of microservice architecture on a white-label travel platform and provides insights into how Domain-Driven Design can be used as a software design concept to improve the overall system architecture. The research focuses on XYZ company, a company that develops white-label technology for the travel industry. An interview with the company and the result of code analysis tools show that the complexity level of the codes in the system has increased, making it more challenging to integrate third-party systems and develop new features without causing errors in other parts of the system. By using the Domain-Driven Design approach, the author found that several business processes could be divided into several product types. The study's results show that the microservice architecture design can help facilitate the process of transitioning from monolith architecture to microservice architecture on a white-label travel platform in XYZ company. This study also provides a solution to the challenges faced by white-label product service providers when integrating large-scale systems. Additionally, the Domain-Driven Design approach used in this study can be applied to other businesses looking to convert their monolith architecture to microservice architecture. The design of a microservice architecture on a white-label travel platform can also serve as a blueprint for other companies looking to make a similar transition.

Keywords: Microservices, Domain-Driven Design, Online Travel Agent, White Label Product.

1. Introduction

The internet has revolutionized the travel industry, making it more accessible for everyday travelers to book and plan their trips. To meet this growing demand, online travel agents (OTAs) have emerged, providing a range of services such as hotel bookings, airline ticket reservations, and tour packages. To facilitate their operations, many travel businesses have turned to white-label products, which offer pre-built technology solutions that can be easily customized to meet specific business needs. However, as the complexity of these systems has increased, it has become more challenging to integrate third-party systems and develop new features without causing errors in other parts of the system.

The object of this research is XYZ company, which develops white-label technology for the travel industry. Currently, the company has collaborated with several large companies, such as BBC, Nickelodeon, and MTV, to run their online travel business. With the variety of tourism services, integrating large-scale systems, such as with third parties, becomes a challenge for white-label product service providers, and it increases the system's complexity. The development of a system is directly proportional to the complexity of the application's features. As the codebase size increases, problems with monolith architecture also increase. It becomes more challenging to implement new features and modifications to old ones because developers must find the right place to apply these changes.

Code analysis tools (Cyclomatic Complexity) from Microsoft used by the author on the XYZ company system show that the level of complexity of the codes has increased. Start from 2020 with a score of 76.788 to 500.364 in 2022. Higher numbers are bad and lower numbers are good. We can use cyclomatic complexity to get a sense of how hard any given code may be to test, maintain, or troubleshoot as well as an indication of how likely the code will be to produce errors (Code Metrics - Cyclomatic Complexity - Visual Studio (Windows) | Microsoft Learn, 2022).

Typical reasons moving towards microservice architecture are complexity, scalability and code ownership (Kalske et al., 2018). With the complexity that appears in a white label product system in the travel industry, this research exploring the use of microservice architecture on a white-label travel platform. Microservice architecture is an architectural style that breaks down large systems into small functional units, providing better modularity and scalability. This approach is particularly relevant to the travel industry, where businesses must constantly adapt to changing customer needs and market conditions. In addition, this research aims to provide insights into how Domain-Driven Design can be used as a software design concept to improve the overall system architecture. By providing a detailed analysis of the benefits and challenges of this approach, this research will offer practical insights to travel businesses and software developers looking to adopt microservice architecture on their white-label travel platforms.

2. Literature Review

This section will explain the theories related to this research to understand better the context and concepts used in this research. The author will also present several studies as the foundation for this research.

2.1. Online Travel Agent (OTA)

The evolution of OTAs began in the mid-1990s with Microsoft launching Expedia in 1996 in the United States, followed by Priceline in Europe in 1997. The site’s early offerings allowed individuals to book vacation reservations online and avoided the use of travel agents and direct booking reservations. Consumers can now bridge several information gaps with the ability to explore and evaluate options for themselves. From a business perspective, this provides the travel industry an alternative option for selling distressed inventory along with additional exposure (Webb, 2016). OTA gives us several advantages compared to conventional travel agents. If we book through an online travel agent, we see all the options, not just those of hotel chains or airlines (Elliott, 2020). Additionally, a recent study by Piper Jaffray found that consumers find the same or lower rates at online travel agencies 87% of the

time (Elliott, 2020).

2.2. White Label Product

White label products focus on how to rebrand products for sale to other companies. It has been mostly applied in the manufacture of physical products, and seems suitable for software startups, particularly for the opportunity to develop customized applications for different customers in a shorter period of time, when compared to stand-alone applications (Silva et al., 2020). The main principle of white label product service is to create a generic solution. Customers today are not looking for custom content. They are looking for general content and that can be customized or rebranded and easy to use (Sharma, n.d.).

According to (Guzenko, 2020) there are several advantages of using white label products. Quick deployment, Since the solution is built on another provider's server, installation takes minimal time. User-friendly, working with cloud-based services does not require special knowledge. The intuitive interface and detailed manual make it easy for you to get started right from the start. Cost-effective, A brand spends no money on server maintenance, integration, platform customization, testing, design, and licensing. Scalable Unlike ready-to-use solutions, white label platforms are scalable and their capacity continuously adjusted to meet evolving business needs.

2.3. Microservices Architecture

The microservices architecture can be seen as a new paradigm for programming applications through a composition of small services, each running its own process and communicating through lightweight mechanisms. Microservices for their contribution to the application, not for their lines of code (Rademacher et al., 2018). Each service contained in a microservice-based application can be developed independently, so that as an application developer, you are not required to understand the entire existing system to be able to run applications or update applications.

Typically, microservices are organized as a suite of small granular services that can be implemented (developed, tested, and deployed) on different platforms through multiple technological stacks (Larrucea et al., 2018). For service mediation, MSA uses the API layer that acts as the service facade while SOA adopts the concept of messaging middleware for service coordination. Moreover, MSA mostly relies on Representational State Transfer (REST) protocol and simple messaging as remote service access protocols; however, SOA can handle different types of remote access protocols including simple messaging for accessing remote services (Microservices vs. Service-Oriented Architecture – O'Reilly, n.d.).

Splitting application into distinct independent microservices allows managed them by individual teams within software development organization and work them independently. Usually, teams developing microservices are organized around business capabilities, but not technical capabilities. Each new requirement should be addressed by only one microservice to retain independent development (Mayer & Weinreich, 2018).

2.4. Domain-Driven Design

Domain-Driven Design (DDD) is a development philosophy defined by Eric Evans in his seminar work Domain-Driven Design: Overcoming complexity at the Heart of Software (Addison-Wesley Professional, 2003). In Evans' approach to DDD, the main principle is to align the intended application with the domain model. The domain model forms a ubiquitous language that is used among team members and serves as a tool used to achieve this goal (Steinegger et al., 2017).

Domain-Driven Design (DDD) provides a framework that can help get the big picture of a well-designed microservice. DDD has two distinct phases, strategic and tactical. Strategic DDD defines the large-scale structure of the system. Strategic DDD helps ensure that the architecture remains focused on business capabilities. Tactical DDD provides a set of design patterns that you can use to create a domain model. This pattern includes entities, aggregates, and domain services. This tactical pattern will

assist developers in designing microservices that are less inter-service dependent and cohesive (Using Tactical DDD to Design Microservices, n.d.).

Domain-Driven Design (DDD) is one of the proposed methods of extracting microservices from a domain. It evolved after a considerable number of discrepancies arose in the work of domain experts, analysts, designers, and developers. DDD is a good starting point for identifying microservices. However, where the line for the bounded context should be drawn is open to interpretation (Steinegger et al., 2017). It is well known that one of the greatest benefits of microservices is that they are loosely coupled (Vural & Koyuncu, 2021).

2.5. Related Works

Analysis of the microservices architecture has been carried out by several previous researchers. Research with title Challenges When Moving from Monolith to Microservice Architecture (Kalske et al., 2018), they studied the types of transition from monolith architecture to microservice architecture. the goal is to transition the reasons why the company decided to do so, and identify the challenges that may be faced during this transition. Their findings reveal that the typical reasons for switching to a microservices architecture are complexity, scalability, and code ownership. Challenges, on the other hand, can be separated into architectural challenges and organizational challenges.

Another study from (Bjørndal et al., 2020) entitled Migration from Monolith to Microservices: Benchmarking a Case Study presents a methodology that can verify whether migrating to microservices is beneficial or not. They evaluate the proposed solution by conducting comparative experiments on the reference system that has been developed in two versions, monolith and microservice. In addition, the research entitled Deployment and communication patterns in microservice architectures: A systematic literature review by (Karabey Aksakalli et al., 2021) identifies and describes existing deployment approaches, and communication platforms for microservices in the current literature. Next, they aim to describe the barriers identified from this approach as well as the corresponding research directions. Based on their research, they were able to identify three types of dissemination approaches and seven different patterns of communication. In addition, they have identified eight challenges related to implementation and six challenges related to communication issues. In the process of monitoring and testing a microservices system, the research entitled Design, monitoring, and testing of microservices systems: The practitioners' perspective by (Waseem et al., 2021) has the objective of gaining an in-depth understanding of how microservice systems are designed, monitored, and tested in the industry.

3. Research Methodology

This research method will explain the steps that will be taken to design a microservice architecture with a Domain-Driven Design approach. As can be seen in the figure below, several steps will be taken to produce a microservice design and prototype according to the needs in solving the existing problems.

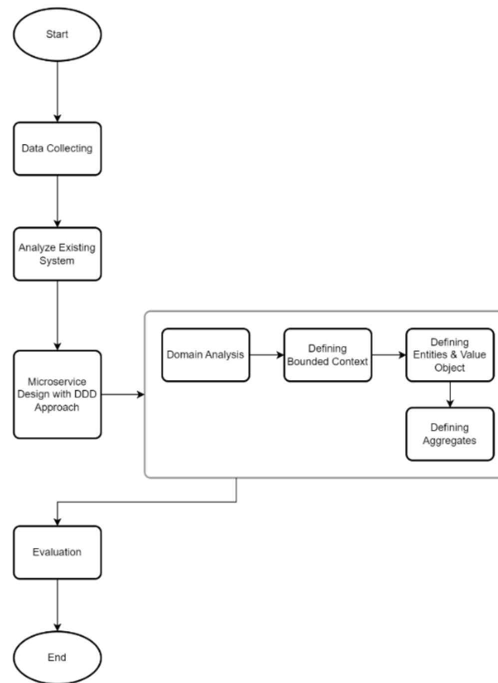

Fig. 1: Research Framework

3.1. Collecting Data & Analyze Existing System

The process of collecting data in this study begins with conducting observations and interviews with parties related to the application. Interviews were conducted with several members of the Product & Engineering Department, namely:

- Technical Delivery Owner, as the party responsible for the strategic planning and system development of the XYZ company.
- Head of Account Manager, as a party that has responsibility for products and cooperation relationships with partners.
- Technical Lead, as the party responsible for transactional features.
- Product Owner, as the party responsible for coordinating the business needs of stakeholders with the engineering team.

After getting the data needed through the interview process and studying the documentation of the XYZ company system, the next step the author will document the results of the current system analysis using the BPMN (Business Process Model and Notation) tools.

3.2. Microservice Design with Domain-Driven Design Approach

The next stage is designing the microservices architecture. In designing this microservices architecture, the author uses the Domain-Driven Design approach to develop applications from XYZ company. The steps are as follows:

- Domain Analysis
- Defining Bounded Context
- Defining Entities & Value Object
- Defining Aggregates
- Using the results of the previous process to identify the services that will be made into microservices.

3.3. Evaluation

The last stage is evaluation. At this stage the evaluation process is carried out by conducting group discussions (focus group discussions) with the Technical Delivery Owner, Technical Lead, Product

Owner, and several developers to ensure that every data and process in the system is appropriate and capable of becoming solutions to problems that exist in XYZ company in the development process.

4. Result and Discussion

Collecting data is the first phase of this research. The results of these interviews can be concluded that the process of developing features and maintainability of the owned system has not been maximized. There are often problems when upgrading or perfecting the libraries owned. With the complexity of the codes that increase yearly, it has the potential to cause the system to be complicated to update in the future.

In addition to the interview process, the author also conducted several literature reviews regarding the implementation of microservices, the application of Domain-Driven Design to studying the business processes of the travel industry. Observation of product documentation owned by XYZ company is also material for the author to better understand their business processes. The BPMN (Business Process Model and Notation) be used by the author to document business processes and will be used as the basis of microservice architecture. Figure 2 below is one of the examples of BPMN for the best-selling types of products in all partners who have installed white label products from XYZ company.

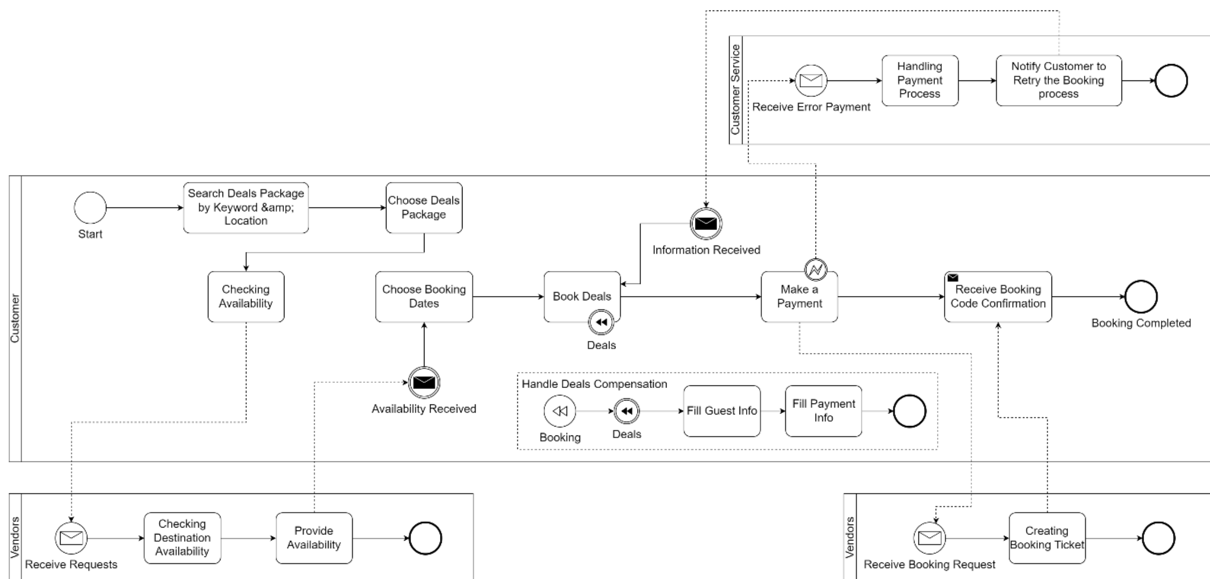

Fig. 2: BPMN of Deals Product

Figure 2 describes the business process from customer searching to ordering product Deals packages available in the application. The process from searching, curating, processing payments, and sending notifications to completing the booking process will involve a third-party vendor issuing a booking code from that vendor. Some of the results of the author's analysis get a business process that is almost the same which will always start with a product search by the customer, followed by the payment process and the ordering process to third parties or processed manually through customer service. Other products available in the XYZ company include flight bookings, hotel bookings, and booking for other activities.

4.1. Domain analysis and Defining Bounded Context

Based on the current business process and system analysis results, an approach with Domain-Driven Design is carried out to solve the current business complexities. Domain-Driven Design is an approach to developing software that enables teams to manage software construction and maintenance effectively. A domain is what the organization does and the world in which that business is shown. From the known

business processes, domain and bounded context analysis can be shown in figure 3.

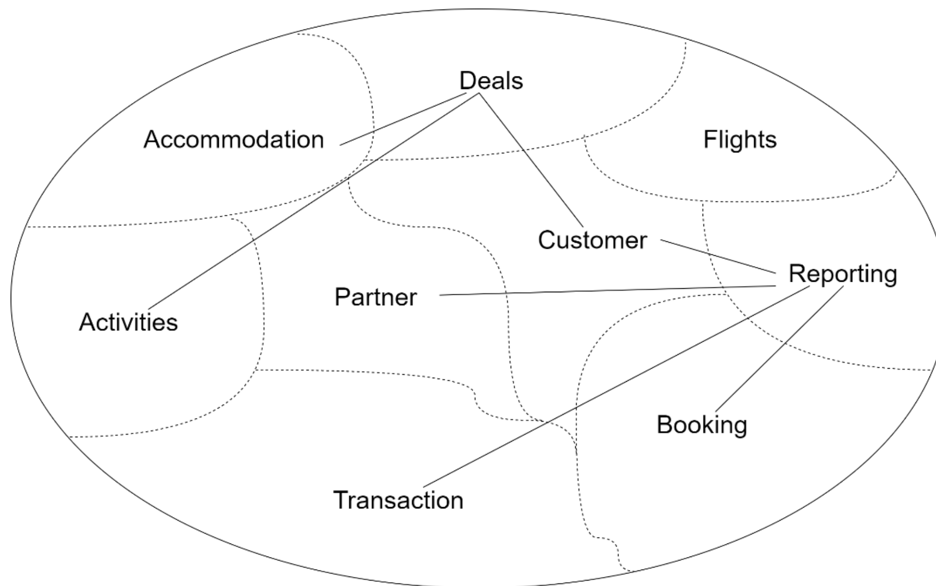

Fig. 3: Domain and Bounded Context

Figure 3 shows that nine sub-domains are modeled based on the results of the BPMN analysis in the previous stage. In the results of this analysis, it is also known that each sub-domain has its own bounded context, in accordance with the concept of the bounded context itself, namely defining responsibilities properly in a particular business domain and in a language that the components of the context can understand. As an example, the context of "customer" can vary in each existing sub-domain. For example, in the Accommodation sub-domain the language used is "guest" but in the Flight sub-domain the language used is "passenger". However, if we look at it from a business process, this can be referred to as a "customer" where they will make transactions according to the product they want.

4.2. Defining Entities and Value Object

In the Domain-Driven Design approach, there are stages for analyzing entities. Entities are unique and capable of continuous change over long periods. The changes may be so extensive that objects may appear much different. However, it is an object with the same identity. Apart from entities, there is also a concept called value object. A value object is an object that represents a defining aspect of a domain without having its own identity. This stage focuses on organizing and identifying sub-domain information from the results of the previous analysis. The following is an entity design in each sub-domain where the future goal is to facilitate implementation between interrelated entities.

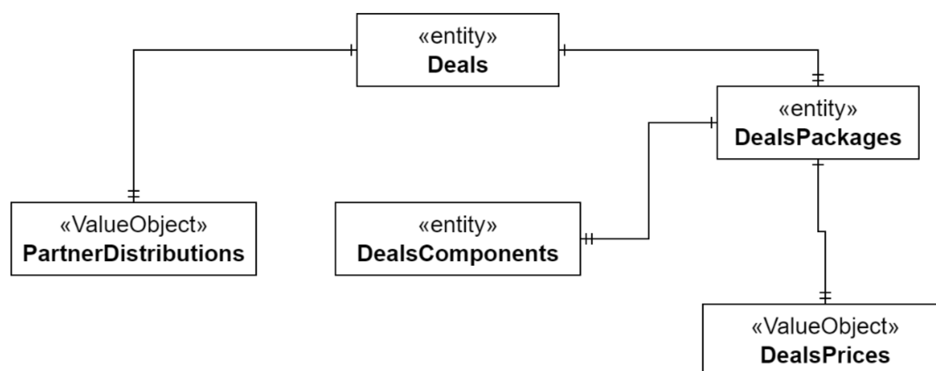

Fig. 4: Deals Entities and Value Objects

Figure 4 shows the entity design of sub-domains and bounded context Deals, divided into three entities and two value objects. These entities are interconnected with each other related to information from the deals service package. Each entity has its own attribute in accordance with the duties of each entity. The Deals Components entity is an entity from the Deals sub-domain that forms and manages the Deals package components. Attributes such as Type will be used to mark a component, while examples of Type are Accommodation, Activities, and so on, where the Type will determine the Product Id and component behavior when used by entities or other bounded contexts.

Analysis of Value Objects in the Deals sub-domain obtained value objects such as Deals Price. Deals Price stores information about Price and Currency of each Price attribute contained in entities that are members of the Deals sub-domain. With a business process that allows customers to see products with various currency options, this value object will make it easier to map prices according to the currency choices desired by customers.

Fig. 5: Flight Entities

The entities in figure 5 are entities that support any data stored in the Flights sub-domain. Flight Searches, Flight Passengers and Flight Ticket are the entities found from the results of the analysis. Each entity has its own attribute in accordance with the responsibilities of each entity. Based on the results of the analysis, it is known that flight data such as availability of schedules, airlines and issuance of airplane tickets are fully carried out in a collaborative process with third parties. So that the XYZ company does not store the data and only needs information about the searches of each customer. Flight Searches are used to store these things, which will be used as data for better decision-making.

The Flight Passengers entity is an entity used to store passenger data. These attributes have a role in storing data related to airplane tickets that customers have purchased. This entity will not store customer personal data because customer data will be concentrated in one Customer sub-domain. The next entity is Flight Ticket, which is an entity that is used to store ticket data in connection with airplane ticket orders that customers have made. The PNR attribute stores flight number data followed by other attributes such as departure, arrival, and airline used. Each passenger can have more than one flight ticket in each order. For example, a passenger who wants to travel to a location and return to the initial location (return) will have at least two data entities in Flight Ticket.

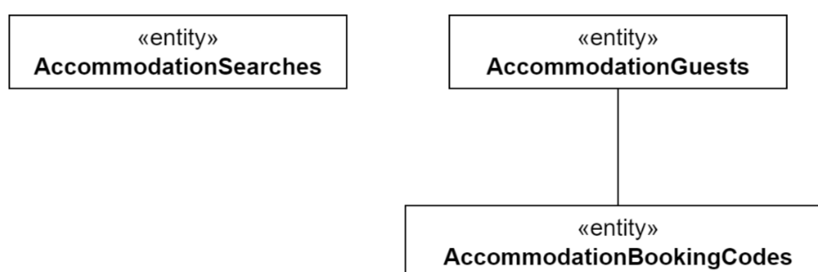

Fig. 6: Accommodation Entities

Figure 6 shows the entities from Accommodation. The Accommodation Searches entity, just like Flight Searches, has the same purpose to be used in storing customer search data. In this case, it becomes part of the accommodation context, which stores data such as destinations until the date of the planned stay. Attribute Query has the role of storing this data so that it can be used by XYZ company to determine business steps based on customer search interests. The Accommodation Guests entity with its attributes is used when the customer has made a hotel booking process. The existing attributes support information about guest data that will stay overnight. This guest data will be sent to third-party services as part of the hotel booking process.

The Accommodation Booking Codes entity stores the booking code, check-in date, and check-out date of customers who have made reservations. The Attribute Code stores a code in the form of a combination of letters and numbers, which functions as a unique code for each booking that customers can use to confirm hotel reservations to the XYZ company or to the intended hotel.

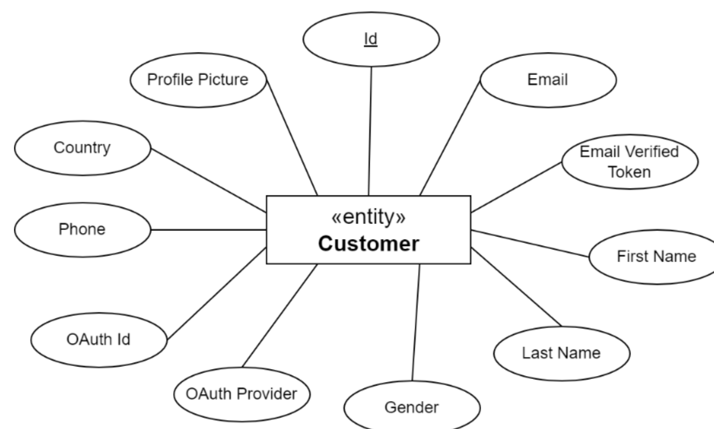

Fig. 7: Customer Entity

Figure 7 shows the entities from the Customer sub-domain. There is only one entity based on the analysis results. The information surrounding the customer entity such as Id, Email, and others is an attribute of the customer entity. The Customer entity has attributes that are used to store customer data. Attribute Id is used as the primary identity of the customer with a combination of Email and Email Verified Token as a customer activity validator. There are also OAuth Id, and OAuth Provider attributes to store information on third-party services that customers use to log in using the OAuth2 authentication standard.

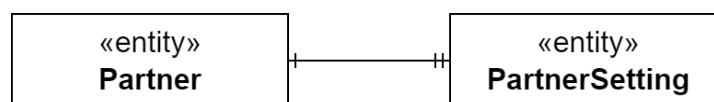

Fig. 8: Partner Entity

Figure 8 shows the Partner entity which has several attributes to store the data needed for the brand configuration that will be used as a display for the partner's white-label product. The Partner Setting entity has attributes that store each data such as color standards, website meta title configurations, third-party analytic code, menu structure configurations and so on. When the customer accesses the website page of the Partner, the system from the XYZ company will read the configuration and will adjust the appearance according to the configuration managed by the Partner.

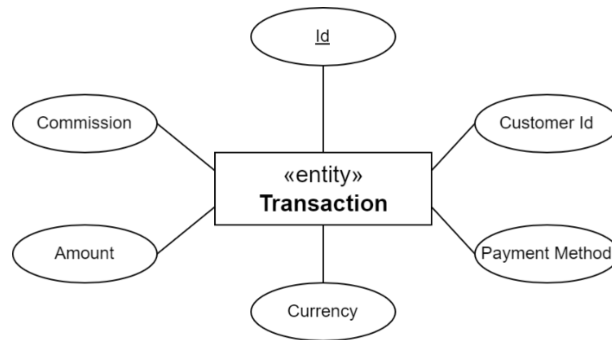

Fig. 9: Transaction Entity

Figure 9 shows the Transaction entity, which stores customer transaction data. In making transactions, customers can choose the payment method the system provides. The payment method is stored in the Payment Method attribute, and the currency choice is stored in the Currency attribute. Attribute Commission is an attribute that stores commission data for XYZ company from every transaction made by a customer.

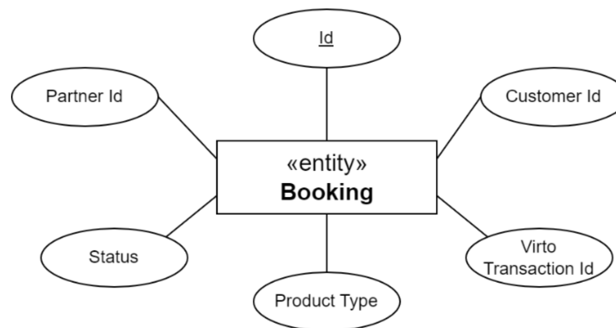

Fig. 10: Booking Entity

Figure 10 shows the Booking entity used when the transaction process has been completed. Using third-party Virto Commerce services, each product ordered will be stored in the third-party system and connected to the Booking entity with the Virto Transaction Id attribute. The Product Type attribute is used to identify the type of product ordered, such as Accommodation or Deals or Flights or Activities.

4.3. Defining Aggregates

Generally, aggregates are modeled from business objects where the object is subject to a boundary that can be treated as a single unit. In Domain-Driven Design aggregate must comply with several rules. This rule ensures that a unit can be independently. One of the rules is a reference to the Aggregate Root, this rule determines that every transaction process such as create, read, update, and delete can only be done through the aggregate root. This is to maintain data consistency and ensure that each entity contained in the aggregate gets the same data changes.

Another rule is that references between Aggregates must use the Primary Key. This rule requires that communication between aggregates must use an aggregate's primary key or identity. The aggregate must use the primary key to be processed by other aggregates if there is a transaction process such as create, read, update, or delete. Based on these explanations and rules, the following is an aggregates design by the results of the analysis in the previous stages:

Table 1: Aggregate Mapping

Aggregate Root	Entities or Value Objects
Deals Aggregate	Deals
	Deals Packages
	Deals Prices
	Partner Distributions
Deals Component Aggregate	Deals Component
Flight Search Aggregate	Flight Searches
Flight Passenger Aggregate	Flight Passengers
	Flight Tickets
Accommodation Search Aggregate	Accommodation Searches
Accommodation Guest Aggregate	Accommodation Guests
	Accommodation booking Codes
Customer Aggregate	Customers
	Oauth Providers
Partner Aggregate	Partners
	Partner Settings
Transaction Aggregate	Transactions
	Currency
Booking Aggregate	Bookings

4.4. Microservice Design

With the results of domain and aggregate analysis in the previous phase, identifying service candidates will be easier. Analysis of the currently running architecture can also be a foundation for determining the required microservice candidates. The following are service candidates based on the results of the author's analysis:

Fig. 11: Microservices Candidates

Based on figure 11, there are 13 candidate services according to the results of the previous analysis. Each service has its own business process, which has been adapted to the BPMN analysis carried out previously. Of the service candidates, Vendors Service is a generic service that can be duplicated according to the vendors working with XYZ company. As an example of the current system, the XYZ company cooperates with Agoda as one of the vendors to search and process hotel room reservations, so with this architectural design, Vendors Service can be duplicated into Agoda Vendor Service which focuses on integration and communication with APIs from Vendors.

Microservices run on multiple processes, services, and servers. Therefore, each service must have a communication strategy by the needs of the service. In general, the communication pattern can be divided into two parts: synchronous and asynchronous. The synchronous pattern is a sending process to waiting for a response from the intended service. This pattern requires that the request sender wait for a response from the intended service before continuing the desired process. The protocol that supports this pattern as it is generally used is HTTP. While the asynchronous pattern is a sending process without waiting for a response from the intended service. This pattern can be used to send messages to several services simultaneously to process data. The protocol that supports this pattern is AMPQ.

Based on the concept of communication patterns, the service candidates that have been obtained previously are subjected to further analysis processes to determine communication patterns that suit the needs and business processes being carried out. Following are the results of the service candidate analysis with the required communication patterns:

Table 2: Service Communication Pattern

Service	Communication Pattern
Deals Service	Synchronous (HTTP, REST API)
Deals Vendor Service	Asynchronous (Message Bus, AMPQ)
Booking Service	Synchronous (HTTP, REST API)
Reporting Service	Asynchronous (Message Bus, AMPQ)
Partner Service	Synchronous (HTTP, REST API)
Flight Service	Synchronous (HTTP, REST API)
Flight Vendor Service	Asynchronous (Message Bus, AMPQ)
Customer Service	Synchronous (HTTP, REST API)
Transaction Service	Asynchronous (Message Bus, AMPQ)
Activities Service	Synchronous (HTTP, REST API)
Activities Vendor Service	Asynchronous (Message Bus, AMPQ)
Accommodation Service	Synchronous (HTTP, REST API)
Accommodation Vendor Service	Asynchronous (Message Bus, AMPQ)

The client must communicate with several services in the microservices architecture to carry out the desired business processes. To simplify communication from client to service, an API Gateway is needed to simplify communication. API Gateway acts as a bridge between clients and services in microservices-based services. API Gateway will map every request from the client to the intended services based on specific patterns. Based on this concept, the following is an overview of the microservice architecture with communication patterns and API Gateway:

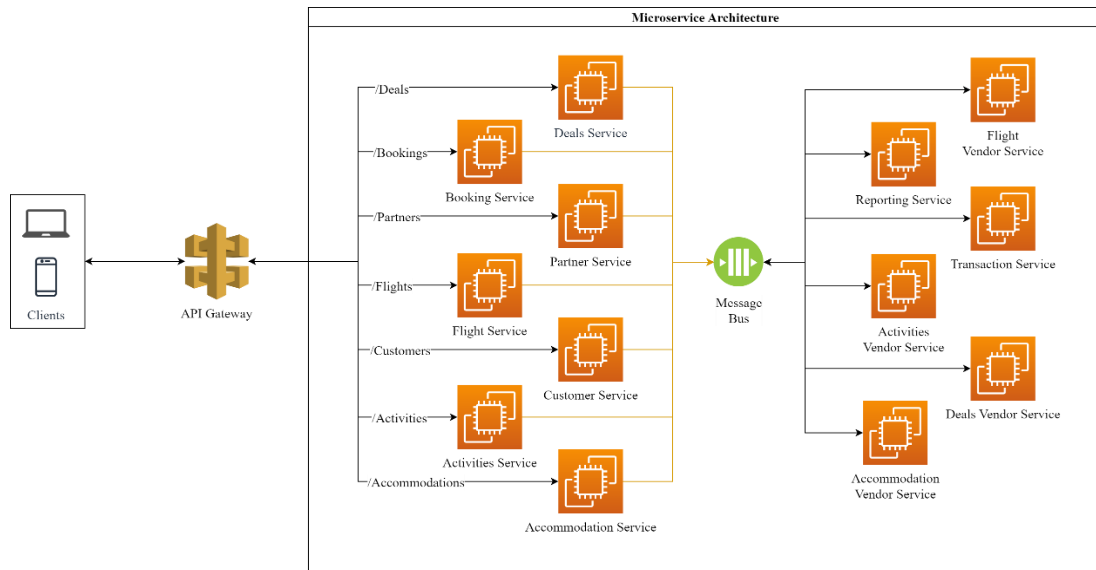

Fig. 12: Microservices Architecture with API Gateway

In supporting the independence of each service, data access will be separated according to the needs of each service. Each service will have its database and can only be managed by the service concerned. If other services require information from specific services, these services must communicate between services to obtain the required data. Based on the architectural design above, the following is the database communication design for each service:

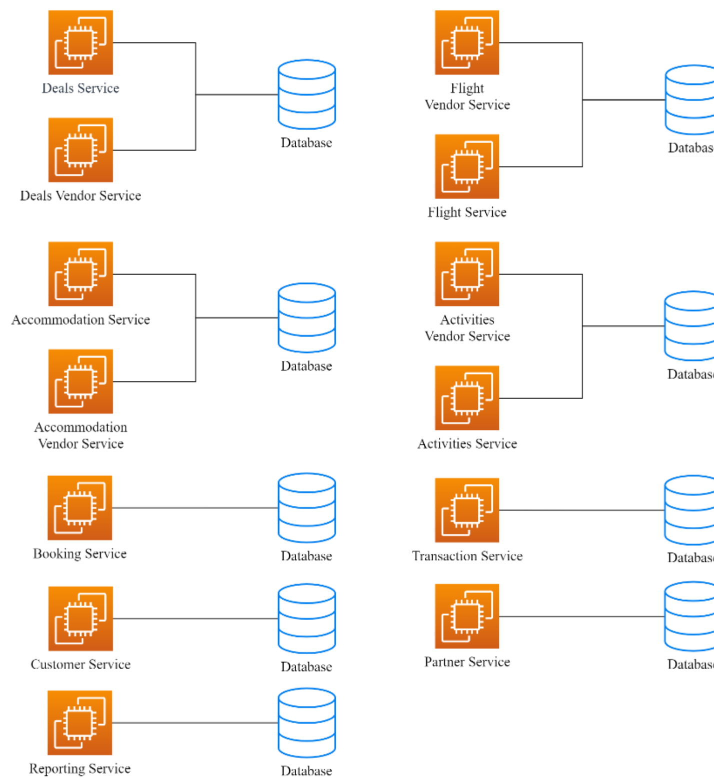

Fig. 13: Database design for each services

From figure 13, it can be seen that several services manage the same database. It is possible because the context of the service is still within one aggregate boundary and business process. With the

separation of the database for each service, it is hoped that each service can process each data more independently and is more complimentary in determining the required data storage technology.

4.5. Evaluation

To measure the analysis's suitability in this study with the expected needs, a discussion forum agenda (Focus Group Discussion) was created, which focused on discussing the design of microservice architecture at the XYZ company. The author scheduled this forum with selected participants from the Engineering division at XYZ company. This activity was attended by 8 participants consisting of Technical Delivery Owners, Technical Leads, DevOps, Product Owners, Fullstack Developers, and Frontend Developers.

In this forum, the idea of designing a microservice architecture with a Domain-Driven Design approach is explained on a system owned by XYZ company. The author explains from business processes to microservices design according to the analysis results. There are several notes from the results of the focus group discussion activities, such as:

- Basically, mapping every service on the system from XYZ company makes understanding every feature it has easier.
- Implementation of microservice will make the maintainability process easier.
- The complexity of the codes can be reduced by dividing each feature into the respective services according to the business processes they have.
- It is necessary to pay attention to the migration strategy from monolith to microservices, both in terms of feature migration and data migration.
- A dedicated development team is needed to focus on implementing this microservice design.

5. Conclusion

This research resulted in a microservice architecture design on a white-label travel platform at XYZ company. Using the Domain-Driven Design approach, the author found that several business processes are divided into several product types. By using BPMN, the author can better understand each business process so that it can be easier to design a microservice architecture according to business needs. With the complexity of the business and existing systems that are already owned, Domain-Driven Design can solve these complexities with each stage of the analysis of this approach. With the design of this microservice architecture, it is hoped that it will provide benefits to facilitate the process of changing applications from monolith architecture to microservice architecture on a white label travel platform in XYZ company. The existence of this architectural design will make it easier for XYZ company to develop strategies for implementing microservice architectures.

With the design of this microservice architecture and the results of the evaluation that has been carried out, the author suggest designing a data migration strategy that is currently owned, where with the system currently running, a mature strategy is needed so that fatal errors do not occur. In addition, a good monitoring implementation strategy is needed so that each process between services can be known in detail so that if there is an error in the system, the source of the problem can be quickly identified.

By using the SCRUM framework, the microservice architecture design can be implemented efficiently. The implementation of scrum framework in software development is expected to be responsive to changes and deliver quality products to market quickly (Panjaitan et al. 2022). With a dedicated development team and an appropriate framework, the implementation and migration of the architecture are likely to proceed smoothly.

Acknowledgements

This research would not have been possible without the help from my lecturer Mr. Ahmad Nurul Fajar, the XYZ Company Development Team Mr. Ped, Mr. Agni, and Mrs. Yulanda for their cooperation

through the whole process. At the end, author really grateful for the guidance from all mentor in designing this.

References

Bjørndal, N., Bucchiarone, A., Mazzara, M., Dragoni, N., & Dustdar, S. (2020). *Migration from Monolith to Microservices : Benchmarking a Case Study*. https://doi.org/10.13140/RG.2.2.27715.14883

Code metrics - Cyclomatic complexity - Visual Studio (Windows) | Microsoft Learn. (2022, October 26). https://learn.microsoft.com/en-us/visualstudio/code-quality/code-metrics-cyclomatic-complexity?view=vs-2022

Elliott, C. (2020, February 16). *Should You Use An Online Travel Agency For Your Next Trip?* Forbes. https://www.forbes.com/sites/christopherelliott/2020/02/16/should-you-use-an-online-travel-agency-for-your-next-trip/?sh=554566b75f22

Guzenko, I. (2020, May 4). *Do You Need White-Label Technology Or A Do-It-Yourself Dream Team?* https://www.forbes.com/sites/forbestechcouncil/2020/03/04/do-you-need-white-label-technology-or-a-do-it-yourself-dream-team/?sh=71db75812d02

Kalske, M., Mäkitalo, N., & Mikkonen, T. (2018). Challenges When Moving from Monolith to Microservice Architecture. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *10544 LNCS*, 32–47. https://doi.org/10.1007/978-3-319-74433-9_3

Karabey Aksakalli, I., Çelik, T., Can, A. B., & Teki̇nerdoğan, B. (2021). Deployment and communication patterns in microservice architectures: A systematic literature review. *Journal of Systems and Software*, *180*, 111014. https://doi.org/10.1016/J.JSS.2021.111014

Larrucea, X., Santamaria, I., Colomo-Palacios, R., & Ebert, C. (2018). Microservices. *IEEE Software*, *35*(3), 96–100. https://doi.org/10.1109/MS.2018.2141030

Mayer, B., & Weinreich, R. (2018). An Approach to Extract the Architecture of Microservice-Based Software Systems. *Proceedings - 12th IEEE International Symposium on Service-Oriented System Engineering, SOSE 2018 and 9th International Workshop on Joint Cloud Computing, JCC 2018*, 21–30. https://doi.org/10.1109/SOSE.2018.00012

Microservices vs. service-oriented architecture – O'Reilly. (n.d.). Retrieved October 9, 2022, from https://www.oreilly.com/radar/microservices-vs-service-oriented-architecture/

Panjaitan, I., & Legowo, N. (2022). Measuring Maturity Level of Scrum Practices in Software Development Using Scrum Maturity Model. *Journal of System and Management Sciences*, *12*(6), 561–582.

Rademacher, F., Sorgalla, J., & Sachweh, S. (2018). Challenges of domain-driven microservice design: A model-driven perspective. *IEEE Software*, *35*(3), 36–43. https://doi.org/10.1109/MS.2018.2141028

Sharma, C. (n.d.). *Applicability of Design System to White-Label Service Development*. Retrieved April 27, 2022, from https://aaltodoc.aalto.fi/handle/123456789/42763

Silva, F., Souza, R., & Machado, I. (2020). Taming and Unveiling Software Reuse opportunities through White Label Software in Startups. *Proceedings - 46th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2020*, 302–305. https://doi.org/10.1109/SEAA51224.2020.00057

Steinegger, R., Giessler, P., Hippchen, B., & Abeck, S. (2017). *Overview of a Domain-Driven Design*

Approach to Build Microservice-Based Applications.

Using tactical DDD to design microservices. (n.d.). Retrieved May 11, 2022, from https://docs.microsoft.com/en-us/azure/architecture/microservices/model/tactical-ddd

Vural, H., & Koyuncu, M. (2021). Does Domain-Driven Design Lead to Finding the Optimal Modularity of a Microservice? *IEEE Access*, *9*, 32721–32733. https://doi.org/10.1109/ACCESS.2021.3060895

Waseem, M., Liang, P., Shahin, M., Di Salle, A., & Márquez, G. (2021). Design, monitoring, and testing of microservices systems: The practitioners' perspective. *Journal of Systems and Software*, *182*, 111061. https://doi.org/10.1016/J.JSS.2021.111061

Webb, T. (2016). From travel agents to OTAs: How the evolution of consumer booking behavior has affected revenue management. *Journal of Revenue and Pricing Management*, *15*(3), 276–282. https://doi.org/10.1057/rpm.2016.16