CHAPTER 4
ANALYSIS AND DESIGN

4.1. Analysis

This chapter provides a detailed explanation and discusses the method of solving problems that have been specified in chapter I. The main objective of this project is to seek an alternative architecture from the previously used monolith architecture and to implement the microservice architecture.

The main difference of both architectures lies on how the architectures designed as shown in figure 4.1, the monolith architecture is an architectural style that consists of different components, combined into a single program, while the microservice architecture is a loosely coupled, small and independent architectural style that can be deployed independently and runs in its own unique process.

Figure 4.1 Monolith and Microservice Architecture

Based on the problem formulation that has been identified in chapter I, the first problem formulation of this project is discussing how microservice architecture average response time when there are a lot of users accessing the application compared to monolithic architecture. Performance is one of the most important parts of architecture, therefore performance testing is an important step in the application development process, it determines how these architectures perform in different scenarios utilizing different testing setups. Response time were utilized to

compare the performance of the tested application, response time is the time it takes for a client to receive a response from the server for a request of a specific service.

The second problem formulation discussing can the microservice architecture have a lower average response time compared to monolith architecture, that in this project the applications will undergo a load test that will test user and list features services at the same time, after completing the tests both application response time will be compared to find out which architecture has the lowest response time.

The last problem formulation discussed about does the amount of data affect both architecture performance. In this project there are 2 test plan configurations that will be tested in load test for both applications, the first test plan configuration will consist of 100 total requests and the second test plan configuration will consist of 200 total requests. After the test has been conducted, the result of the test will show whether the amount of data affects the applications performance or not.

4.2. Design

4.2.1. Service Decomposition

The monolith architecture in the List Feature Application backend uses REST API as framework style for a web API and uses HTTP request method to access and process the data, the main HTTP method that the application uses is GET, POST, PUT, and DELETE. HTTP method GET usually is used for reading or viewing from the data request, the POST method usually is used to create data from the data request, the PUT method is used to update the data from data request and DELETE method is used for deleting the data from data request. Both of the request and the response for the API uses JSON to serialize or deserialize the data, as seen in figure 4.2

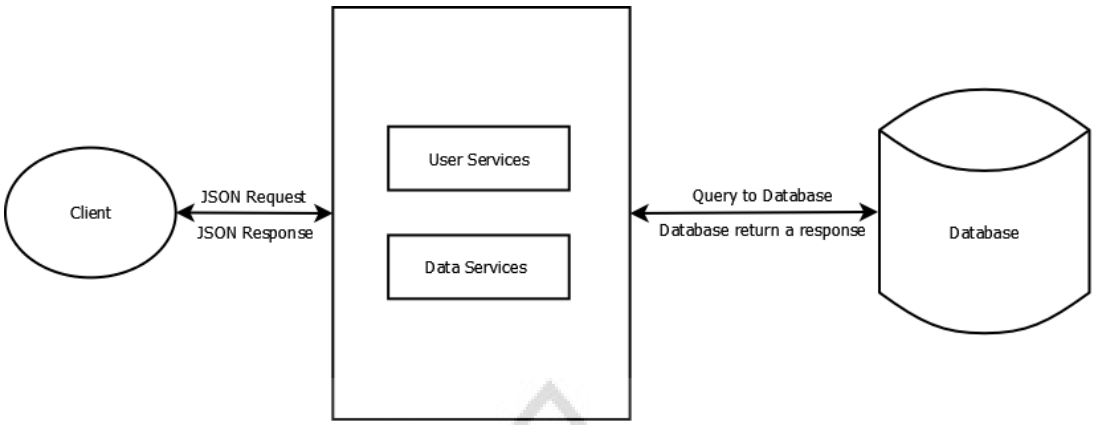

Figure 4.2 Monolith Architecture

Figure 4.2 shown above describes the services that are in the monolith architecture will be decomposed into small components, that run independently and will use Go Kit as a framework to build the API services. Go kit is a go framework that has a collection of go packages and libraries for implementing microservices. Go kit has 3 major components that will be implemented in this project, the 3 components consist of transport layer, endpoint layer and service layer. Transport layer provide concrete transports to distribute the endpoints to the client that will consume the service. Endpoint layer will convert every service into RPC style endpoints, each of the endpoints exposing the services using a transport layer and a single endpoint can be exposed by multiple transport. Services layer is the core of the application that will implement the business logic that will be converted into an endpoint and later will be exposed using the transport layer.

4.2.2. Rebuild API Function

The List Feature Application backend that previously used monolith architecture is created using Go programming language. The application uses REST API as architecture for web API that takes JSON data type as request data type and response data type. The frontend of the application will send the request as JSON data type that consists of a key value pair and HTTP method that matches the backend request. The backend will process the request and create a query to the database and return the response to the frontend as JSON data type.

The monolith application has 2 main services namely User Services and List Services. User services as shown in figure 4.3 below, handle all of the user related services, for now it has 2 main

services that handle the login and register of the user, it does not rule out the possibility that in the future, another user services will be added, such as adding roles to the user and service to change user password. Register service only takes 3 requests that consist of username, email and password. The application also uses Json Web Token or JWT for user authentication, JWT token can be obtained by the user if the user already registered the account and uses the account to login to the application.

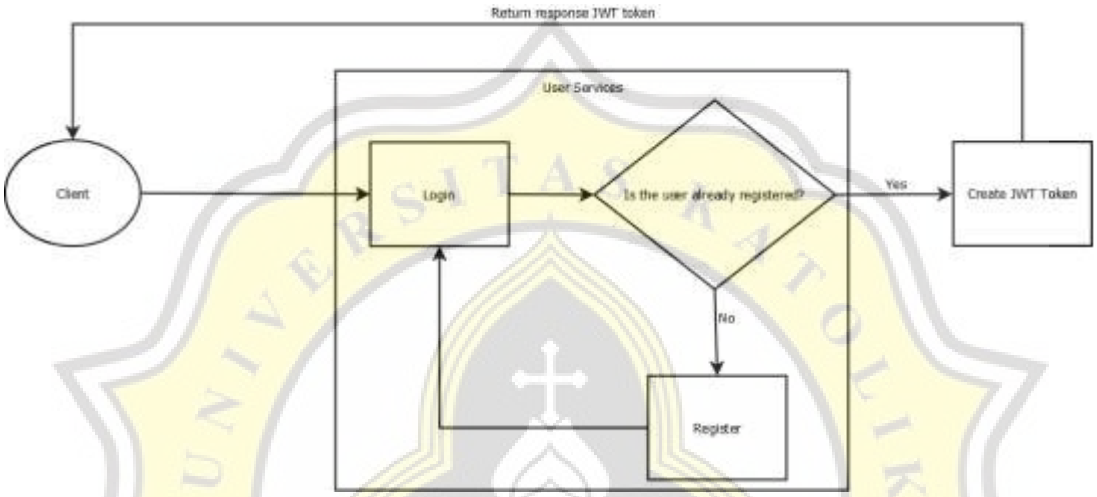

Figure 4.3 User Service Workflow

List Services as shown in figure 4.4 below, handle all of the list services that consist of list application that uses HTTP method GET to view the list either view all list of the application or view the list by id, the list that taken from database wil consists of id, feature name, description, live date, segmentation, channel, limit transaction, limit daily and limit monthly. Insert data function use HTTP method POST to insert the data to the database, the data consists of id, feature name, description, live date, segmentation, channel, limit transaction, limit daily, limit monthly and user name. Update data function use HTTP method PUT to update the data that consists of feature name, description, live date, segmentation, channel, limit transaction, limit daily, limit monthly, update data, update by and use id to get the data in the database. Delete data function use HTTP method DELETE to delete the data using the id. To access the list service JWT token is needed, the JWT token must be sent in headers as authentication, the authentication of the token takes place in middleware that is implemented in all list service routes. List services consists of list of application which has been made by one of the divisions in PT Bank Negara Indonesia Persero (TBK), it expected this list features application will be used by all of the employee in the

divisions as a pocket application to view, register, edit and delete the list of the application that has been made.

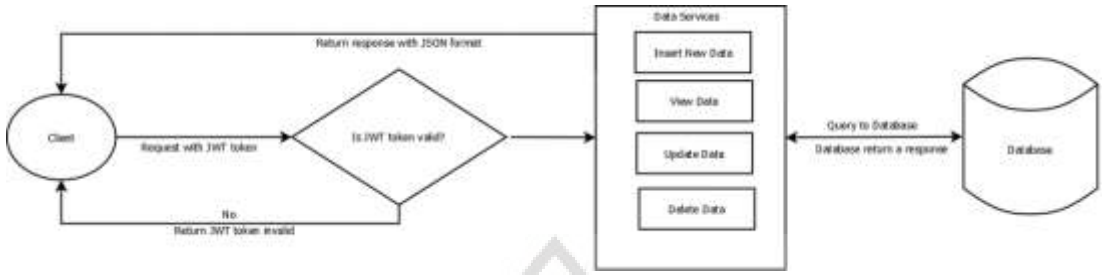

Figure 4.4 List Services Workflow

The 2 main services will be separated into 2 applications that run independently that are called microservice. The microservices application will use Go Kit as the framework that builds upon Go programming language.

4.2.3. Testing

The main objective of this project is to seek an alternative architecture from monolith architecture to microservice architecture, after implementing the architecture a comparison needs to be done to prove that microservice architecture can perform better when there are a lot of users accessing the application compared to monolith architecture. Response time were utilized to compare the performance of both architectures. Response time is the time it takes for a client to receive a response from the server for a request of a specific service, the lower the response time the better the performance is. To compare both of the architectures, testing software is needed. JMeter is an application to test a functional and performance of the services, JMeter has a number of features to perform a load test to both architectures. JMeter also produces the test report dashboard that can be seen in the form of a web page, the report dashboard contains several metrics including complete statistics of response time, the comparison will be based on these reports. Both application services will undergo 3 performance test scenarios, and after both architecture services complete the test, the average response time of both architectures will then be compared.

Table 4.1. Test Scenarios

Test Scenario	Test Plan Configuration				
	Threads(user)			***Ramp-up period(s)***	***Loop Count***
1	100	200	500	60	1
2	100	200	500	60	1
3	100	200	500	60	1

The first test scenario is to test both architectures in the same environment, the second test scenario is to test both architectures in virtual machine, the microservice architecture will be tested in 2 virtual machines for both services, that will simulate the core of the microservice that is distributed computing, the monolith architecture will also tested in 1 virtual machine because monolith architecture are supposed to be a single unit application. The third test scenario will test both architectures in different environments, the monolith architecture in the base operating system while the microservice architecture in the virtual machine.

From the table 4.1 above there are 3 test plans that have various configurations that will be tested in both scenarios, the first test plan configuration will cover 100 total requests, 60 seconds ramp-up period and 1 loop count. that will make it total 100 users in a span of 60 seconds for each service, while both services will be tested at the same time the total thread that the application will be served is 200. The second test plan configuration will cover 200 total requests, 60 seconds ramp-up period and 1 loop count, that will make it total 200 requests in a span of 60 seconds for each service, while both services will be tested at the same time the total thread that the application will be served is 400. The third test plan configuration will cover 500 total requests, 60 seconds ramp-up period and 1 loop count, that will make it total 500 requests in a span of 60 seconds for each service, while both services will be tested at the same time the total thread that the application will be served is 1000. The 3 performance tests were conducted to see how significant the architectures were from 200 to 400 to 1000 total requests.

The comparison of the average response time for both architectures will be carried out in all scenarios, the comparison will use weighted arithmetic means to calculate the average of the test results and the comparison will be based on the calculation. The weighted arithmetic mean is

used to calculate the sum of the number multiplied by the weight and divided by the sum of the weight, it will use the formula like the following function.

$$W = \frac{\sum_{i=1}^{n} w_i X_i}{\sum_{i=1}^{n} w_i}$$

Figure 4.5 Weighted Arithmetic Mean Formula

The W in figure 4.5 above is the weighted average or the values that will be the result, the in formula above is the weights applied to X values, while the X is the data values that will averaged that in this projects X will be the test results. There are 2 services that will be tested simultaneously in all testing scenarios, that is user services and list features services. The weights that will be used in the formula will be based on the HTTP method GET, POST, PUT and DELETE that is used in list features services, the weights will be ranged from 1 to 4 with 1 is the lightest and 4 is the heaviest weight. Based on the paper conducted by S. Alam et al, the PUT method will have 4 weights as the heaviest methods, because in the operations the PUT method will retrieve and overwrite existing resources[2]. The POST method will have 3 weights because in the operations the method will create a new resource. The DELETE method will have 2 weights because in the operations the method will only delete existing resources. The GET method will have 1 weight because in the operation the method will only retrieve the resource.