JURNAL ECOTIPE
Electronic, Control, Telecommunication, Information, and Power Engineering

Volume 10, Issue 2, Oktober 2023, pp. 191-199
ISSN 2355-5068 ; e-ISSN 2622-4852
DOI: 10.33019/jurnalecotipe.v10i2.4327

MyNote: Multiplatform Reactive Integration System with The Clean Architecture Development Concept

Arrijal Amar Ma'ruf [1], Dedi Gunawan[2*]
[1,2] Department of Informatics Engineering, Faculty of Communications and Informatics, University of Muhammadiyah Surakarta, Indonesia

ARTICLE INFO

Article historys:

Received : 08/08/2023
Revised : 31/08/2023
Accepted : 11/10/2023

Keywords:

Clean Architecture; Multiplatform; React; Reactive Programming; WebFlux; Webview

ABSTRACT

The application needs in the current digital era are very broad and varied. In its development, these applications transform, thus demanding application developers to create applications that suit their devices. This study applies the concept of the reactive paradigm which is implemented into a multiplatform application system and the concept of clean architecture in the process of creating a simple note management application "MyNote". The concept of reactive programming uses Spring WebFlux on the server and React.js with RxJs on the client. The multiplatform concept on Android is implemented in Webview. Performance analysis is conducted using the load test, and stress test. This research aims to provide information on how to implement the concept of multiplatform applications and reactive programming with the clean architecture development method for the MyNote application. In addition, it also can be used as a consideration in implementing this concept stack in production-ready products. This study succeeded in making the MyNote application using a combination of clean architecture, multiplatform, and reactive programming concepts with a percentage of 93.75% application of the concept, and 80% application development.

Corresponding Author:

Dedi Gunawan
Department of Informatics Engineering, University of Muhammadiyah Surakarta, Surakarta, 57102, Indonesia
Email: dedi.gunawan@ums.ac.id

1. INTRODUCTION

In today's digital era, there is a wide range of applications needed. These applications have evolved in terms of their appearance, usability, and operation in line with the advancements in hardware development. As a result, developers are required to create applications that are suitable for these devices. However, during the development process, developers often face technical difficulties such as continuous data access and less flexible and scalable application structures. To overcome these challenges, developers can implement a multiplatform system combined with a reactive paradigm and a clean architecture. Multiplatform applications can provide convenience for users in choosing the platform to be used [1]. There are various types of application development architectural styles. While all of these architectures differ somewhat in detail, they are very similar. All of them have the same goal, namely the separation of interests. They all achieve this separation by dividing the software into layers. Each has at least one layer for business rules, and another for interfaces [2]. An important feature of Clean Architecture in the client base is that the UI and data source can be changed without any problem. This characteristic allows testing business rules without UI, database, services, or any other external dependency [3]. Reactive programming has recently gained popularity as a paradigm that is well-suited for developing event-driven and interactive applications [4]. Reactive programming is a programming paradigm that has asynchronous data streams at its core [5]. A reactive system is a system

JURNAL ECOTIPE
Electronic, Control, Telecommunication, Information, and Power Engineering
Volume 10, Issue 2, Oktober 2023, pp. 191-199
ISSN 2355-5068 ; e-ISSN 2622-4852
DOI: 10.33019/jurnalecotipe.v10i2.4327

that continuously communicates with the environment outside the system and responds to it at a time determined by the environment, not the program itself [6]. Reactive streams on the official documentation are initiatives to provide asynchronous stream processing standards with non-blocking backpressure [7]. Non-blocking is another term for asynchronous or overlapped processes [8].

MMyNote is a useful app for taking notes, whether for personal use or for collaborating with others. You can personalize the default themes to make your notes stand out and look appealing. The app is compatible with various devices. The goal of this study is to create a preliminary version of MyNote that has not yet been optimized for performance. This will allow us to evaluate its performance and gather raw data on its capabilities. We will conduct load, stress, and spike tests on the server to assess its performance. This study will only focus on discussing the development and analysis concept. MyNote is available on multiple platforms and is created using Progressive Web Apps (PWA) technology, which combines various web technologies to provide users with a superior experience [9]. On mobile, it uses Webview, which is an Android system component that enables Android applications to render web pages and communicate with web servers [10]. However, it is important to note that there is a risk of security breaches, particularly Cross-site scripting (XSS) attacks, which can occur through HTTP Requests by stealing cookie data, hijacking sessions, and impersonating user data [11]. The application of the reactive paradigm in MyNote will use the Functional Reactive Programming (FRP) method in Java, with a framework Spring WebFlux on the server and RxJs on the client. Using a framework allows us to build web applications faster [12]. FRP itself is denotative (based on precise, simple, implementation-independent compositional semantics, which precisely determines the meaning of each type and building block (program)) and temporally continuous (has a certain tempo) [13]. This study provides comprehensive guidance on how to seamlessly integrate multi-platform applications and reactive programming using the clean architecture development approach in the MyNote app. It can also serve as an excellent reference for implementing this technology stack in production-ready products, based on thorough analysis and research.

2. RESEARCH METHOD

2.1. Build Application Architecture

In the early stages of developing the MyNote application, an application structure was created with a clean architecture for both the server and client which is more directed towards the application of the dependency rule concept, so the structure will be like one-way traffic. The implementation of clean architecture on the client is combined with the Model View ViewModel (MVVM) structure. MVVM is an architecture adapted for modern User Interface (UI) development platforms where appearance is the responsibility of a designer rather than a classic developer [14], in MVVM, UI is located inside the View layer, and the data layer is inside the Model and ViewModel layer.

2.2. Build Application

We use TypeScript programming language with the React.js and RxJs libraries for the client and Spring WebFlux framework for the server. The data connection between the server and the client uses the Short Polling technique, in Short Polling the client sends regular requests to the server and each request tries to "pull" events or available data. If no events or data are available, the server returns an empty response and the client waits for an interval before sending another polling request [15]. Server code testing is carried out by functional Application Program Interface Representational State Transfer (API REST) testing, namely system testing and interface testing on the Postman application. Client code tests are carried out using the system testing method and UI testing. After the development of the server and client is complete, reconfiguration and synchronization between the server and client will be carried out. After reconfiguring, the application will be built using Gradle, Gradle is a build tool that is used as a means of organizational development and project-specific development standards [16]. So Gradle is like a framework but it works in the build application process.

2.3. Analyze Application

After the system is built locally, the next stage involves performing load tests, stress tests, and spike tests. The load test is conducted using Postman while JMeter is used for stress and spike tests. If the system meets the analysis target which is greater than equals to 10 test cases, the application is then

JURNAL ECTIPE
Electronic, Control, Telecommunication, Information, and Power Engineering
Volume 10, Issue 2, Oktober 2023, pp. 191-199
ISSN 2355-5068 ; e-ISSN 2622-4852
DOI: 10.33019/jurnalecotipe.v10i2.4327

hosted on Google Cloud servers. The server uses a Virtual Private Server (VPS) with e2-small type engine specifications on the Compute Engine. After hosting the application, the server undergoes load, stress, and spike tests using different test cases. The subsequent stage involves creating a Webview on the Android system using Kotlin.

3. RESULTS AND DISCUSSION

Previous studies have implemented Clean Architecture [17], Reactive Programming [18], and Multiplatform concepts separately, using independent experimental programs and methods, systems, libraries, and frameworks. However, in this study, we have implemented all three concepts in one complex experimental application called MyNote. The application has been built using specific programming languages, Java and Typescript, and specific libraries and frameworks, WebFlux and RxJs. The purpose of this application is to analyze the performance of Reactive Programming on the server side. Although the application lacks performance on the server side, it was built to gain a better understanding of performance in a reactive system that is built with the WebFlux framework. The knowledge gained from this study is combined to serve as a reference for developing production systems.

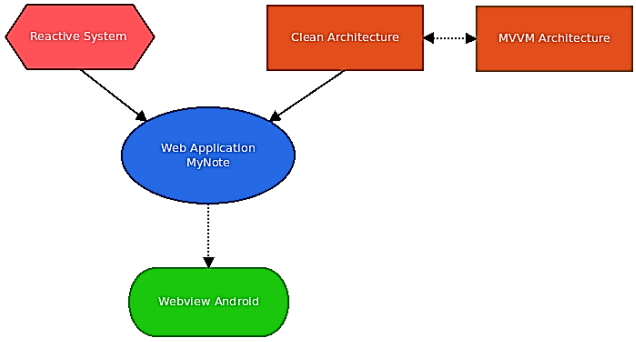

Figure 1. MyNote Implementation System Scheme Concept

3.1. Clean Architecture

The client structure has been updated to follow the clean architecture principles of MVVM and it was a success. The changes made were minor but made sure that the dependency rules remained intact. The client structure includes five components - model, use case, adapter, configuration, and view. The model component, also known as the Entities layer, holds objects that define business rules. The use case component, known as the Use Cases layer, applies these business rules through application logic functions or methods on the client. The adapter component, also known as the Adapter layer, transforms data and implements gateway functions from the use case layer to the outer layer. The configuration component, located outside the View layer, along with the adapter component, acts as the Model (M) in the MVVM architecture and includes client configuration libraries like the Redux library. Lastly, the view component is responsible for displaying application graphics and acts as the View View Model (VVM) layer in the MVVM architecture. The container sub-component represents the View (V) layer, while the view controller represents the View ViewModel (VVM) layer.

The infrastructure of the application server has been carefully crafted following the principles of clean architecture. Object streams, namely Mono for data 0..1 and Flux for data 0..N, have been utilized from the Reactor Core library of the Spring Webflux framework to achieve this remarkable feat. The Java module, consisting of four modules - Entities, Use Case, Adapter, and Application - has been implemented to enforce the dependency rule. These four modules have been built with the help of Gradle build tools. The Entities module defines the business rules object, including the main, repository, and storage model. The Use Case module implements business rules by employing application business logic, inner converter between the main model, application storage method, application security method for authentication object, and entity logics. The Adapter module acts as a gateway and boundaries between applications and the external layer, interacting with the database and incorporating the Spring Data library (version 3.0.3 (Spring Data Commons) and version 3.0.5 (Spring Data R2DBC)). Lastly,

JURNAL ECOTIPE
Electronic, Control, Telecommunication, Information, and Power Engineering
Volume 10, Issue 2, Oktober 2023, pp. 191-199
ISSN 2355-5068 ; e-ISSN 2622-4852
DOI: 10.33019/jurnalecotipe.v10i2.4327

the Application module houses all libraries, drivers, and configurations, including security, database, error handling, and router configurations. The Reactor Core library on each layer has a different version to ensure flexibility and scalability, guaranteeing that all necessary resources are available and the software is built and deployed on the server based on this module.

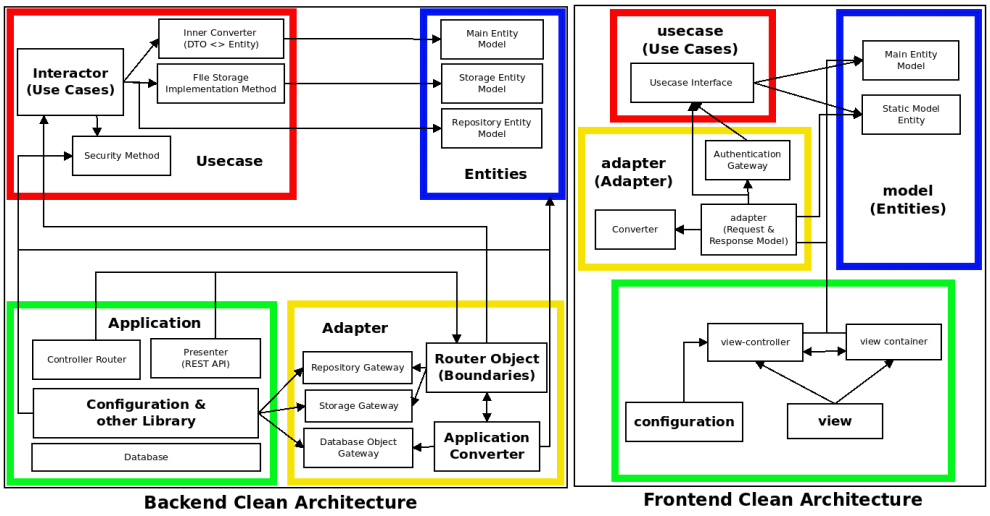

Figure 2. Clean Architecture MyNote

After a thorough examination of the business policies governing the structure of this endeavor, it can be confidently asserted that the implementation of clean architecture has been a resounding success. The framework is designed in such a way that the outgoing layers are responsible for enforcing specific business regulations, while the general layers handle the implementation of broader business policies. The layers are interconnected in a one-way manner, with the upper layer having no direct connection with the lower layer, but the lower layer is connected to the upper layer. Moreover, the library's implementation ascertains that any changes made in the lower layer do not impact the upper layer, while the lower layer accommodates any changes effectively. The architecture is meticulously organized, making it simple to read. Additionally, it is scalable, as the addition or removal of multiple features does not require any modifications to the existing structure. The code is uncomplicated to maintain, and the structure is flexible, allowing it to adjust swiftly to any modifications. However, the build size is substantial because of the independent libraries in each layer and the necessity to separate functions.

3.2. Multiplatform

The MyNote application is a multi-platform tool that can be accessed via an internet connection and a web browser. Android users can access the application through a web browser or an Android application (.apk) using Webview technology. To create this application, We utilized the Android Studio Dolphin (version: 2021.3.1 Patch 1) and Kotlin (version: 213-1.7.20-release-for-android- studio-AS6777.52), along with 12 GB RAM, 2 core CPU, 4 vCPUs, 2.30GHz, and Intel(R) Core(TM) i5-6198DU CPU. They also used Gradle plugins and dependencies to facilitate the development build process. To ensure that MyNote Webview smoothly on Android, several steps must be taken. These include granting internet permission, using the 'NoActionBar' theme for the activity, enabled DOM storage and Javascript feature in Webview settings to enable storage and Javascript access. The Javascript feature is optional depending on the use of Javascript. To run MyNote, Android version 5.0 (Lolipop) or newer is required, with Android version S_V2 (Android 12) being the optimal target system.

Our analysis of the MyNote Webview application revealed several advantages and disadvantages. The advantages include fast development time, small application size, and ease of implementation. However, the application requires an internet connection, which impairs the user experience in offline mode. Additionally, the application load time is slow and demands substantial resources due to the use of the default browser in Webview.

JURNAL ECOTIPE
Electronic, Control, Telecommunication, Information, and Power Engineering
Volume 10, Issue 2, Oktober 2023, pp. 191-199
ISSN 2355-5068 ; e-ISSN 2622-4852
DOI: 10.33019/jurnalecotipe.v10i2.4327

3.3. Reactive Programming

The client has successfully implemented reactive programming to process request and response data from adapters at the ViewModel layer. The implementation of reactive programming is using asynchronous and synchronize. The pooling technique is utilized for request profiles (requests that retrieve user profile data) using the GET method. The request profile is the main object pooling and triggers other requests (chains effect), which affect the display and performance. To handle continuous changes in appearance, data validation and comparison are conducted and in this state reactive programming has been implemented with some transformation data. Security filters have 2 conditions, if the path request is public access will be granted to all requests, and if the path request is private access will be restricted to only a few people, It can be done with an access token, if the access token has been outdated, it will be refresed automatically to new ones. So the view will be automatically updated without a refresh page. On the server side, the reactive concept is applied in the system and programming itself. The system uses threads on each request automatically scalable with the help of Netty Server and the database side uses a connection pool with a value of 50. That means up to 50 connections to the database can be opened concurrently. and the programming side uses functional programming that is almost the same as client-reactive programming.

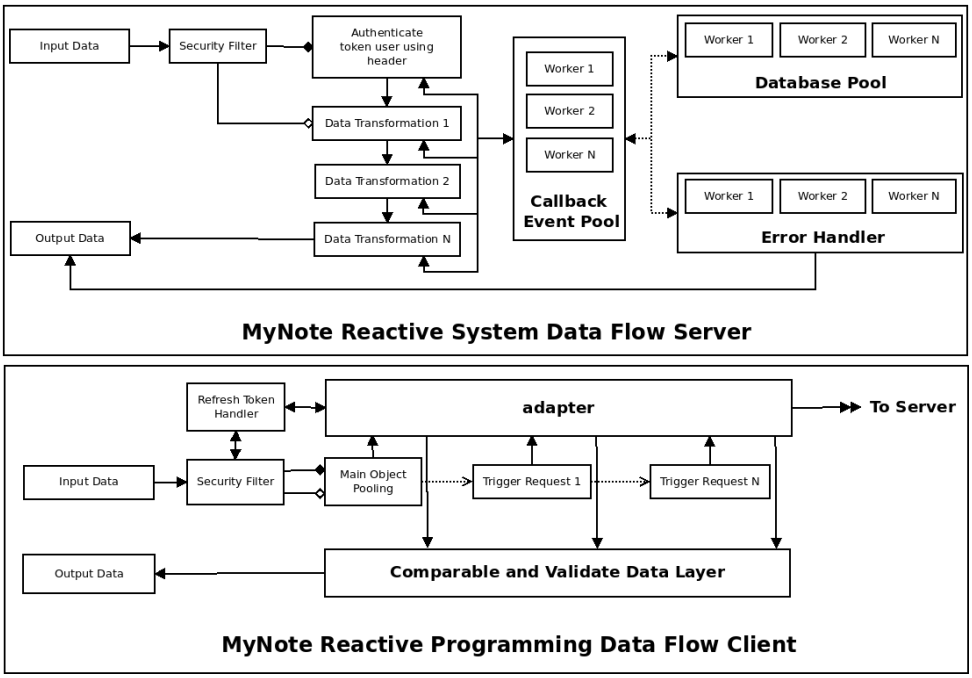

Figure 3. Reactive Data Flow in MyNote

Regarding the MyNote client system, there have been various analyses conducted on the use of reactive structures. One of these analyses suggests that the mapping function can be combined with Promise and async functions when retrieving data within the function. The decision to use Redux and similar libraries depends on whether you want to distribute data efficiently, not on the endpoint (next value), and calling the function (subscribe) without using input does not use pure functions. As a result, changes to the function's outcome depend on external variables. By using reactive structures, the code can be tidied up instead of using general functions, leading to a successful implementation of the reactive system. However, there are some potential drawbacks to consider. Too many processes display data, which must be validated and compared with previous data, making it less efficient. Additionally, too many data requests are sent to the server at one time due to the chain effect, making processing heavier. Despite these challenges, there are several advantages to using reactive programming on the client of the MyNote application, including neater and easier-to-read code arrangement, efficient use of code, and a clearer data processing flow.

The process of analyzing load tests, stress tests, and spike tests is divided into two significant phases: pre-hosting and hosting. The evaluation of the test results is based on two crucial factors: response time and success rate. Server behavior is also taken into account while presenting the analysis

JURNAL ECOTIPE
Electronic, Control, Telecommunication, Information, and Power Engineering
Volume 10, Issue 2, Oktober 2023, pp. 191-199
ISSN 2355-5068 ; e-ISSN 2622-4852
DOI: 10.33019/jurnalecotipe.v10i2.4327

findings. To conduct the load test, we utilized the Postman application version v9.31.28, while for the stress test and spike test, we used the JMeter application version 5.6. These tests were carried out on two different devices. The first device, a local device, had 12 GB RAM and 2 core CPUs, 4 vCPUs, 2.30GHz, Intel(R) Core(TM) i5-6198DU CPU. The second device was a VPS hosting device e2-small, which had 2GB RAM and 1 core CPU, 0.5 GHZ, with the default machine configuration. The analysis of the test results was conducted rigorously, comparing the results obtained from both devices. The comparison was based on the response time and success rate, along with the server behavior observed during the tests. The results obtained from the analysis were highly informative and helped us to identify the areas that required improvement.

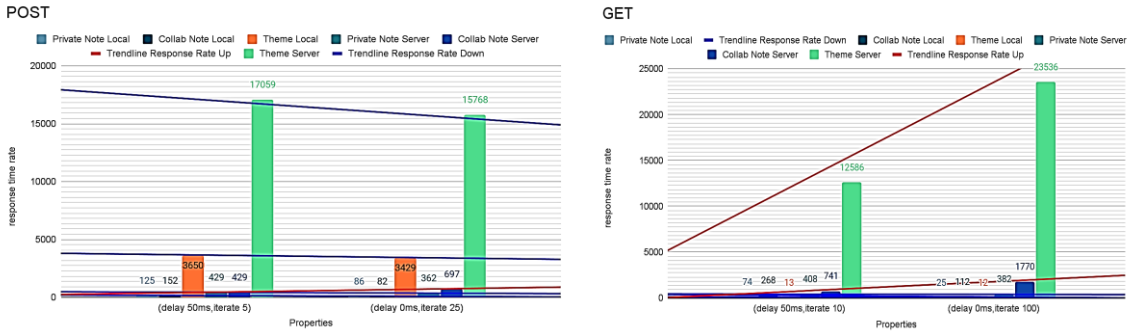

Figure 4. POST and GET Load Test

In the process of making POST requests, there are two possibilities: the response rates may either increase or decrease with a ratio of 1:5 or 20%:80%. Upon conducting the tests on both the local and server, it was observed that all response rates on the local side had decreased by 100%. On the other hand, the Collab Note Server response rate increased, leading to an overall response rate decrease of 77% on the server side. These findings indicate that external factors such as the internet and server environment have a significant impact on the response rate, as concluded through the analysis of the local response rate. Similarly, when making GET requests, response rates may either increase or decrease with a ratio of 2:3 or 40%:60%. Again, all response rates on the local side decreased by 100%, while the Collab Note Server and Theme response rates increased on the server side, resulting in a 33% decrease in overall response rate. However, it should be noted that GET collab note requests on the server experienced a decrease in response rate with a final average result of 936ms for requests 1-53, but an increase in response rate with a final average result of 1,770ms for requests > 53. In contrast, GET theme requests on the server showed a continuously decreasing response rate, with a starting response rate of 127,750ms. From these findings, it can again be concluded that external factors play a major role in affecting response rate. However, the results are considered insufficient for drawing conclusive implications.

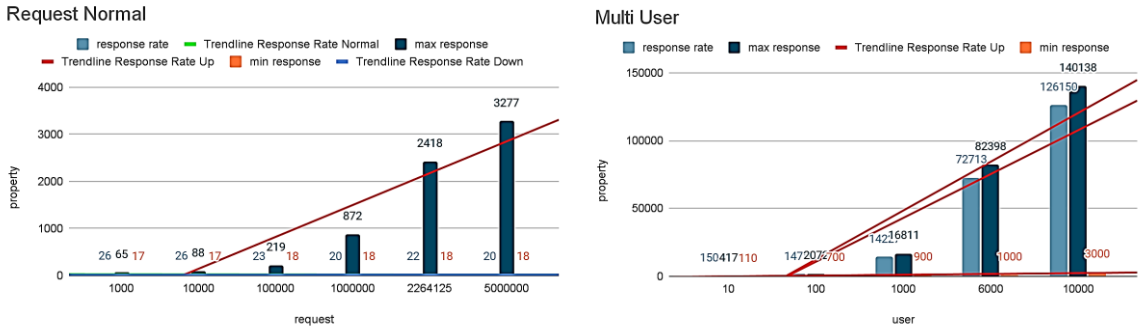

Figure 5. The Difference Between Normal Request and Multi-User Request in Local

The graph on the left shows normal request data with 1,000 to 5,000,000 iterations, indicating that all requests were successful without any errors. The response rate is normal, the maximum response rate always stays up, and the minimum response rate is also normal. However, for requests exceeding 1,000,000 iterations, the minimum response rate is not as accurate since it uses an estimated lower limit,

JURNAL ECOTIPE
Electronic, Control, Telecommunication, Information, and Power Engineering
Volume 10, Issue 2, Oktober 2023, pp. 191-199
ISSN 2355-5068 ; e-ISSN 2622-4852
DOI: 10.33019/jurnalecotipe.v10i2.4327

resulting in spike requests occurring. This graph has not yet reached the limits of the machine's capabilities. On the right side graph, multi-user requests were used with 10 to 10,000 users and a varying number of iterations. For example, 1,000 iterations were used for 10 users, 100 iterations for 100 users, and 10 iterations for the rest. All requests were successful without any errors. The iteration was implemented to evaluate the server's behavior when a collapse request (a request sent simultaneously) occurs. Both the response rate and maximum response rate always experience a significant increase. However, the minimum response rate for users exceeding 6,000 uses an estimated lower limit, resulting in less accuracy. This graph has reached the machine's limit, as when the number of users reaches 10,000, the machine eats up 10.9 GB of RAM and 92% CPU.

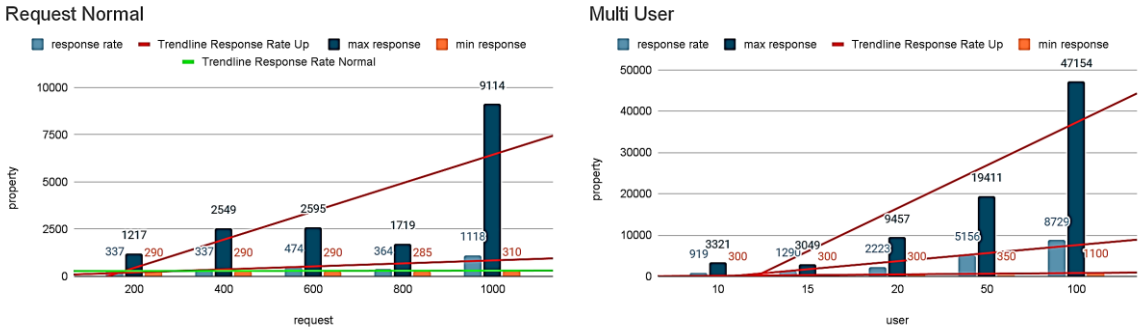

Figure 6. The Difference Between Normal Request and Multi-User Request in Server

The graph on the left shows the results of a normal request with a request iteration range of 200 to 1,000. All requests were successfully sent without any errors. Under normal circumstances, both the response rate and the maximum response have increased, while the minimum response has decreased. However, for requests exceeding 500, there were some spikes in the response rate. On the other hand, the graph on the right shows the results of multi-user requests with 10 to 100 users, with a constant number of iterations set at 50. All requests were also successfully sent without any errors. In this case, both the response rate and the maximum response consistently experienced a significant increase. However, the minimum response for users greater than or equal to 20 was estimated based on the lower limit of the graph, making it less accurate. Additionally, the response rate for users greater than or equal to 15 also showed some spikes in requests. To carry out the spike test, we used the results of the stress test and divided it into two parts: local and server. Both parts involved testing with minimum to maximum limit conditions, with no pauses in between. Each round was set to follow the sequence of minimum, maximum, and then minimum again. In the maximum condition, we used 10 iterations per user, while in the minimum condition, we used unlimited iterations with a timeout of 5, 1, and 3 seconds per round. If the timeout was exceeded, we moved to the maximum condition.

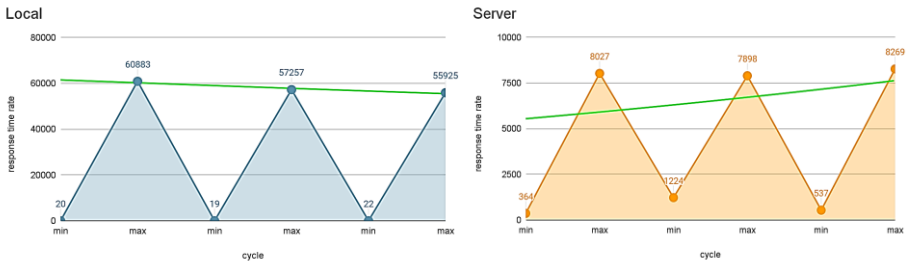

Figure 7. Spike Test Local and Server

Local spike tests showed a decrease in response rate per rotation under maximum conditions, while the response rate remained stable under minimum conditions. Meanwhile, the server experienced an increase in response rate per round under maximum conditions, with stability under minimum conditions. The local system remains stable with no errors, exhibiting 100% resilience. However, the server experienced a request error of 0.2% out of 5,000 (10 requests) in the first round under maximum conditions, resulting in a resistance of 99.96%. This calculation is derived from the total success rate

JURNAL ECOTIPE
Electronic, Control, Telecommunication, Information, and Power Engineering
Volume 10, Issue 2, Oktober 2023, pp. 191-199
ISSN 2355-5068 ; e-ISSN 2622-4852
DOI: 10.33019/jurnalecotipe.v10i2.4327

per round, with 2 data points (minimum and maximum) per round. With 3 rounds, the result would be (((100 x 5) + 99.8) / 6)% = (599.8/6)% => 99.96%.

4. CONCLUSION

When developing multiplatform applications with Webviews, using clean architecture and reactive programming together can bring many benefits. These concepts make the development process more organized and the structure easier for the team to understand. Updating applications becomes more flexible and scalable, and code maintenance is simpler. Using these concepts also leads to faster Android application development and smaller application sizes, reducing memory and overflow errors. User requests can be handled efficiently, even on low-end devices, and multiple requests can be accommodated at once. Additionally, the application can respond well to changes in system load and appearance.

However, certain factors must be considered to achieve the best results when implementing these concepts. These include requiring significant time and resources, lightweight UI display data on the server, user internet connection, server feedback events to improve user experience (such as loading animations), server location and speed, and server security (such as synchronizing and validating unique token data). It's also important to implement back pressure on data servers, pool connections in the database, and use optimized queries (such as using ID and date for paging and sorting instead of limits and offsets). Mapping data access steps in functions is also crucial, as is selecting the connection method between client and server, minimizing the effects of request chains, and minimizing data validation and comparison processes. For data comparison, the dispatch until change method can be used.

Based on our assessment, the MyNote application, which uses clean architecture, reactive programming, and multiplatform concepts, has a value of 93.75%. This is calculated from 100% clean architecture client and server, 100% multiplatform, and 75% reactive programming (for a less elastic application). On the MyNote application side, 80% production has been achieved.

REFERENCES

[1] R. Choirudin and A. Adil, "Implementasi Rest Api Web Service dalam Membangun Aplikasi Multiplatform untuk Usaha Jasa," *MATRIK : Jurnal Manajemen, Teknik Informatika dan Rekayasa Komputer*, vol. 18, no. 2, pp. 284–293, May 2019, doi: 10.30812/matrik.v18i2.407.

[2] "Clean Coder Blog." https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html (accessed Aug. 04, 2023).

[3] D. Sanchez, A. E. Rojas, and H. Florez, 'Towards a clean architecture for android apps using model transformations', *IAENG International Journal of Computer Science*, vol. 49, no. 1, pp. 270–278, 2022.

[4] E. Bainomugisha, A. L. Carreton, T. van Cutsem, S. Mostinckx, and W. de Meuter, 'A survey on reactive programming', *ACM Computing Surveys (CSUR)*, vol. 45, no. 4, pp. 1–34, 2013.

[5] T. John, *Hands-On Spring Security 5 for Reactive Applications: Learn effective ways to secure your applications with Spring and Spring WebFlux*. Packt Publishing Ltd, 2018.

[6] M. Bernhardt, *Reactive Web Applications: Covers Play, Akka, and Reactive Streams*. Simon and Schuster, 2016.

[7] https://www.reactive-streams.org (accessed Aug. 04, 2023).

[8] D. Syme, T. Petricek, and D. Lomov, "The F# Asynchronous Programming Model," in *Practical Aspects of Declarative Languages*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 175–189. Accessed: Aug. 04, 2023. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-18378-2_15

[9] D. A. Hume, *Progressive Web Apps*. Manning, 2017.

JURNAL ECTIPE
Electronic, Control, Telecommunication, Information, and Power Engineering
Volume 10, Issue 2, Oktober 2023, pp. 191-199
ISSN 2355-5068 ; e-ISSN 2622-4852
DOI: 10.33019/jurnalecotipe.v10i2.4327

[10] M. A. El-Zawawy, E. Losiouk, and M. Conti, "Vulnerabilities in Android webview objects: Still not the end!," *Computers & Security*, vol. 109, p. 102395, Oct. 2021, doi: 10.1016/j.cose.2021.102395.

[11] A. B. Bhavani, "Cross-site Scripting Attacks on Android WebView," *arXiv.org*, Apr. 28, 2013. https://arxiv.org/abs/1304.7451

[12] A. Rakhmadi and E. Listiyanto, "PERANCANGAN LIBRARYUMS-CMS MENGGUNAKAN CODEIGNITER," Jan. 01, 2010. http://hdl.handle.net/11617/2104

[13] S. Blackheath, *Functional Reactive Programming*. Simon and Schuster, 2016.

[14] Kexugit, "Introduction to Model/View/ViewModel pattern for building WPF apps," *Microsoft Learn*. https://blogs.msdn.microsoft.com/johngossman/2005/10/08/introduction-to-modelviewviewmodel-pattern-for-building-wpf-apps/ (accessed Aug. 04, 2023).

[15] P. Saint-Andre, "RFC 6202: Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP," *IETF Datatracker*, Apr. 05, 2011. https://datatracker.ietf.org/doc/html/rfc6202 (accessed Aug. 04, 2023).

[16] T. Berglund and M. McCullough, *Building and Testing with Gradle*. "O'Reilly Media, Inc.," 2011.

[17] Y. N. Nugroho, D. S. Kusumo, and M. J. Alibasa, "Clean Architecture Implementation Impacts on Maintainability Aspect for Backend System Code Base," in *2022 10th International Conference on Information and Communication Technology (ICoICT)*, Aug. 2022. Accessed: Aug. 22, 2023. [Online]. Available: http://dx.doi.org/10.1109/icoict55009.2022.9914890

[18] J. Setiadi and A. H. Rangkuti, "Design and development of assets management system using spring webflux and command pattern," in *2022 International Conference on Information Management and Technology (ICIMTech)*, Aug. 2022. Accessed: Aug. 22, 2023. [Online]. Available: http://dx.doi.org/10.1109/icimtech55957.2022.9915197