Comparative Analysis of the Use of State Management in E-commerce Marketplace Applications Using the Flutter Framework

Aditya Akhmad Dinatika*1, Muhammad Aminul Akbar2, Aryo Pinandito3

1,2,3Brawijaya University, Malang
1aditya_akhmad@student.ub.ac.id, 2muhammad.aminul@ub.ac.id, 3aryo@ub.ac.id
*Corresponding Author

Received 10 August 2023; accepted 23 August 2023

Abstract. This research aims to analyze the performance of two popular state management tools that have different system approaches, namely Riverpod and GetX, in developing marketplace applications using the Flutter framework. The marketplace type application used is Flutter ecommerce which is available as open source on the GitHub platform. This research aims to provide benefits to Flutter application developers in choosing the right state management tools for marketplace applications. The method used is to compare memory usage, execution time and CPU utilization of the two tools. Testing is done on the feature of displaying the home screen, search and add features. The results of the analysis obtained are that there are differences in memory usage and execution time for the three features and scenarios, GetX provides less memory usage with faster execution time. While in CPU utilization, there tends to be no performance difference between GetX and Riverpod. The conclusion is that GetX state management uses memory more efficiently and is able to provide faster execution time than Riverpod state management with better performance.

Keywords: GetX, Riverpod, State Management, Flutter

1 Introduction

In today's digital era, mobile applications have become a daily necessity for people. One popular type of mobile application is marketplace application, which allows users to buy and sell products online. Marketplace application development requires the right technology to achieve optimal performance and provide a good user experience.

Flutter is one of the frameworks that is now widely used in mobile application development. This framework uses the Dart programming language and has the ability to produce high-quality and high-performance mobile applications. One of the things that affects the performance and quality of code in an application is state management. State management in Flutter is a technique for managing application state and separating state from views. This technique allows developers to create applications that are more flexible and can be changed easily, while improving performance and user experience (Bizzotto, 2021). In Flutter application development, state management tools are used to manage and store data in the application. State Management plays an important role in displaying data changes on widgets. Basically. All widgets contained in the Scaffold body in Flutter code will be rebuilt if there is a change in the data used
even for just one widget. This results in a lot of unnecessary resource allocation. Choosing good state management helps increase the effectiveness and efficiency of any data changes in the Flutter application. The urgency of the state is basically because Flutter uses widgets that are declarative (Bauroziq, 2022), meaning that any changes to the application state will be automatically reflected in the UI display. The existence of the state becomes a trigger to automatically redraw the view from the beginning. State management tools that are popular in Flutter include Riverpod and GetX. Riverpod state management basically uses the state provider system because it is a further development of the Provider state management and offers some more flexible and easy-to-use features than Provider (Riverpod, 2023), while GetX state management uses the concept of observable widgets (Pub.dev, 2023). There are differences in the concept approach in the two state management, so there may be differences in performance or resource usage that need to be analysed.

However, there is no research done to compare the performance of these two state management tools in the development of marketplace applications using the Flutter framework. Therefore, this research will discuss the performance analysis of Riverpod and GetX in marketplace applications using the Flutter framework. The marketplace type application that will be used is an Ecommerce App application that is available as open source on the github.com platform. Ecommerce App which is marketplace-based applications have data complexity, involve a lot of interaction from users, and require fast response and display updates that will affect user satisfaction or user experience. Therefore, Ecommerce applications are suitable as applications used for state management testing which involves a lot of data changes and is directly related to the widget display.

This research is expected to provide benefits for Flutter application developers in choosing the right state management tools for marketplace application development. In addition, this research can also be a reference for researchers and mobile application developers to develop applications with better performance.

2 Previous Research

In this research, researchers use literature in the form of related research that has been done before as a reference and reference. The author studied the research of Muhammad Hafid Nur Azis entitled Comparative Analysis of the Use of State management in Watched Applications Using the Flutter Framework. From this research, the author can take references for performance test techniques and methods in this research. In the reference research, the researcher makes the application watched as the object of research. The Watched application is built with the Flutter framework to display a list of movies and utilize state management to manage data processing. In the study, a comparison was made between two types of state management, Provider and BLoC, on three performance categories: memory usage, execution time, and CPU utilization. The results show no difference in the memory usage category, but there are differences in the execution time and CPU utilization categories. The difference between the research Comparative Analysis of the Use of State Management in the Flutter Commerce Marketplace Application Using the Flutter Framework with this research is in the object of the application being tested and the state management used. This research compares Riverpod state management with GetX state management. In addition, the statistical test of difference uses paired T-Test. In contrast to this study, the difference statistical test will be carried out using an independent t-test.

The second research was taken from Regawa Rama Prayoga, Ghifari Munawar, Rahil Jumiyani, Alifia Syalsabila in 2021 entitled Performance Analysis of BLoC and Provider State Management Library on Flutter. This research discusses the selection of
the right state management library that can improve the performance of Flutter application development for Android. This study focuses on the effect of using the BLoC and Provider state management libraries on the widget build process in Flutter. The experimental results show that using BLoC results in (1) CPU usage of 0.45% (2.14% more efficient), (2) memory usage of 23.27 MB (8.19% more efficient), and (3) execution time of 3.54 seconds (16.36% more efficient). While the use of Provider results in (1) CPU usage of 0.54% (more efficient 2.57%), (2) memory usage of 32.01 MB (more efficient 11.27%), and (3) execution time of 4.10 seconds (more efficient 19.44%). These results were found when BLoC and Provider were implemented on smaller widgets or "leaf widgets". The connection with this research is that researchers take the same calculation method related to test result data both from memory usage, execution time, and CPU utilization. Data that is accurate enough to encourage researchers to perform the same technique in determining the best state management performance in marketplace applications. The difference with this research is the test parameters used by the author, this time without using CPU Utilization testing. In addition, this research also only focuses on functionality, not directly applying the state to the application.

3 Methodology

This research uses quantitative analysis methods. According to Johnson and Christensen (2012), quantitative analysis methods are data analysis techniques used to test hypotheses or answer research questions using numerical data. The reason for choosing this method is because this research has input and output values in the form of numbers, which can make it easier for researchers to find out which state management is better based on the results of the performance test score.

![Figure 1. Methodology](image-url)
3.1. Development

Ian Sommerville (2016), in his book "Software Engineering" describes application development as a process of designing, implementing, and testing software that involves steps such as requirements analysis, architecture design, coding, and system testing and integration. The difference statistical test will be carried out using an independent t-test.

However, according to Pressman, Roger S. (2014), development is not limited to creating new applications, but also involves changes or improvements to existing applications. Feature additions can involve changes to the design, construction, testing, and implementation of existing software to meet new or existing user or organizational needs. The process of developing feature additions often involves needs analysis, design, implementation, and testing to ensure that the new features added work well and meet the desired needs.

In this research, two developments will be carried out on the Flutter E-commerce application. The first application development implements Riverpod as its state management while the second application development implements GetX as its state management. This is done in order to create a means of collecting data to be analyzed and compared in the next stage.

3.2. Data Collection

Data is obtained from Performance Tests conducted on Ecommerce applications that implement State Management Riverpod and GetX.

Performance testing is a testing process carried out to evaluate the performance or performance of a system, product, or process. Performance testing aims to measure and assess the performance of an object by comparing test results with predetermined standards.

According to Prayoga, R. R., et al. (2021), the most relevant test data with state management functionality to measure are as follows:

1) Memory Usage: This is because inefficient state management can affect memory usage in Flutter applications. If state data is not managed properly, there can be excessive memory usage or unnecessary data accumulation, which can increase memory usage in the application. Therefore, choosing an efficient state management method in Flutter can help optimize memory usage in the application.

2) Execution Time: State management can also affect execution time in Flutter applications. The process of updating and managing complex state can take significant execution time, especially if it is not well organized. In some cases, the use of inefficient state management methods can result in increased execution time in Flutter applications, which can affect the responsiveness of the application to user interactions.

3) CPU Utilization: The use of state management in software development can affect CPU performance or CPU utilization. Inefficient or complex state management can cause excessive overhead in managing and updating application state. This can result in higher CPU resource utilisation, as the system has to continuously process the state changes that occur. Therefore, in efficient application design, it is important to consider the optimal use of state management to minimise CPU usage and ensure better overall application performance.

In essence, this research focuses on the urgency of limited user device resources such as memory, CPU speed, and limited user time. This test is oriented towards user satisfaction with the application.
The Memory Usage, Execution Time, and CPU Utilization data will be taken from three test scenarios or certain situations in the application such as:
1) When displaying the Home Page. This scenario represents the performance of state management initiation when the application is first run.
2) When searching on the Search Screen page until the searched product results appear fully on the page. This scenario represents the responsiveness of UI updates when performing a search streamed through a text editing field.
3) When adding product quantity on the Cart Screen page to the total price. This scenario represents real-time data changes involving mathematical operations (summation) of the total price of products in the cart.

For credibility, the data to be collected in this study is 30 for each test parameter per scenario. Except the CPU parameter data will be 30 for a combination of 3 scenarios because there are too many factors that affect CPU utilization such as test pauses, device conditions when features are tested, and the number of repetitions (Aziz, M. H. N., 2022). So, the total data collected will be 420.

3.3. Data analysis
Analysis of test data is the process of interpreting data obtained from tests or experiments to draw conclusions or make judgements based on the data. The analysis of test data in this study involved the following steps:
1) Compiling data: Test data is collected and collated in a format that can be analyzed. Data can be in the form of numbers, text, or other data relevant to the tests performed.
2) Presenting Data: Test data will be presented in the form of graphs and tables to facilitate data understanding and analysis.
3) Analyzing Data: Test data is analysed using statistical methods or other data analysis techniques, depending on the type of data and the purpose of the test being conducted. Data analysis includes normality test and different t-test. Normality test is used to determine the normality of data distribution as a condition for determining parametric or non-parametric different test methods.
4) Interpreting Results: The results of the data analysis are translated into understandable information to make conclusions or judgements on the objectives of the test conducted.

3.4. Conclusion
In the last stage of this research, conclusions will be drawn based on the results of data analysis. Conclusions are drawn to answer the question or purpose of the test conducted. In this research, conclusions will be drawn in the form of statements of fact to answer specific questions regarding "Which state management has better performance between Riverpod and GetX in its application based on memory usage, execution time, and CPU utilization?"

This stage also opens suggestions and evaluations on further research to be more relevant in the future.

4 Result and Discussion
4.1. App Implementation
The e-commerce application project used for Riverpod and GetX state management testing uses clean architecture, which separates Presentation, Application and Data.
In Clean Architecture, each part has clear responsibilities and roles:

1) **Presentation Layer (Display):** This section focuses on the UI and user interface. Widgets are used to organise the display and interaction with the user. The Controller or ViewModel is responsible for managing the state and business logic associated with the view.

2) **Application Layer:** This part contains the business logic or business rules of the application. Services are used to provide specialised functionality, such as integration with external APIs, data processing, application state management, etc. These services are used by the controller or ViewModel to retrieve data and implement business logic.

3) **Data Layer (Data):** This part deals with the management of application data. The Repository is responsible for accessing and managing data from various sources, such as local databases, network APIs, local storage, and others. In this research, the application is limited to accessing only local data connected to the Repository. Clean Architecture is implemented on every feature in the application. While the features that will be applied different state management to be tested include the homepage feature, search feature, and cart feature in which there is an increment function.

4.2. **Application Testing**

Application testing is done using two hardware devices, one hardware device is a computer as a tester to see the performance of the application and the status of the device being tested. the other hardware is a mobile device as a device that is tested to run the E-Commerce application. The hardware specifications used are as follows:

1) **Tester (PC)**
   - Device: Acer Swift 3 SF314-41
   - CPU: AMD Athlon 330U with Radeon Vega Mobile Gfx 2.40 GHz (2 Cores 4 Threads)
   - Memory: 4+16GB DDR4
   - OS: Windows 11

2) **Tested Device (Mobile)**
   - Device: Samsung A03s
   - CPU: Octa-core (4x2.35 GHz Cortex-A53 & 4x1.8 GHz Cortex-A53)
   - Memory: 4 GB/64 GB
The software used includes Flutter DevTools for memory usage and execution time performance tests, Snapdragon Profiler for CPU utilization tests, Macro Recorder for testing automation, and Scrcpy for mobile device screen representation.

4.3. Data Collection

1) Memory Usage

Memory usage data uses megabyte (MB) units obtained from Flutter DevTools, which is a flutter default application for testing purposes. This memory data is contained in the "Memory" section at the top and the data is taken from the dot when the action is performed. In the first scenario (home screen), the action taken is to open the application until the main page appears. In the second scenario (search feature), the action taken is when searching in the search field. While in the third scenario (increment), the action taken is when pressing the button to increase / decrease the quantity of products on the cart page. The test result data can be seen in the figure below:

![Figure 3. Memory usage of home screen in megabyte (MB)](image)

![Figure 4. Memory usage of search feature in megabyte (MB)](image)
2) Execution Time

Execution time data uses units of second (s) obtained from Flutter DevTools which is a flutter default application for testing purposes. In the first scenario (home screen), the data is obtained by calculating the time from when the application is still closed, then opened until the list of all products is displayed. In the third scenario (increment), the data is obtained by calculating the time from when the increment button is clicked which is marked by the comment log, until the application does not display the log anymore or means that the entire increment process has succeeded. The test result data can be seen in the figure below:
3) CPU Utilization

CPU Utilization data uses megabyte (MB) units obtained from Flutter DevTools, which is a Flutter built-in application for testing purposes. CPU utilization data is obtained from the Snapdragon Profiler application in the cumulative CPU Utilization section. Tests were conducted on all three scenarios run at once so that each state management only produces 2 data variables. The series of scenarios starts by clicking on the application icon until the home page appears, followed by using the search feature until the searched product data appears, then clicking on one of the products and adding the product to the cart. Finally, on the cart page, the product quantity is added to the maximum. The test result data can be seen in the figure below:

![Execution Time in 3rd Scenario](image)

Figure 8. Execution time of increment in second (s)

![CPU Utilization](image)

Figure 9. CPU Utilization of all scenarios in second (s)

4.4. Data Processing

The data that has been obtained will pass the normality test first before being analyzed. Normality test is a statistical process used to evaluate whether a sample of data comes from a normally distributed population. This is done to determine the different test method that will be used next. If the data is normally distributed, then the difference test will be carried out by parametric methods. If the data is not normally distributed, then the difference test will be carried out by non-parametric methods. The normality test that will be used in this study uses the Shapiro-Wilk method because the data tested is less than 50 data.
The following are the results of the normality test for memory usage data:

![Table of normality test results for memory usage](image1)

From the table above, it can be seen that memory usage in the first, second, and third scenarios, there is one data variable that has a value of <0.05. This means that all different tests on memory usage parameters will be carried out using non-parametric methods.

The following are the results of the normality test for execution time data:

![Table of normality test results for execution time](image2)

From the table above, it can be seen that the data execution time in the first and second scenario has a value of >0.05. Whereas in the third scenario, there is one data variable that has a value <0.05. Therefore, testing on the first and second scenario execution time parameters will use parametric methods. While testing on the third scenario, execution time parameter will use a non-parametric method.

The following are the results of the normality test for CPU utilization data:

![Table of normality test results for CPU utilization](image3)

From the table above, it can be seen that the data CPU utilization has a value of >0.05. Therefore, testing on the first and second scenario execution time parameters will use parametric methods.

4.5. **Data Analysis**

After the normality test is done. The next step is to perform parametric and non-parametric tests. For all three data memory usage and Execution Time 1st and 2nd
scenarios, the Mann-Whitney test was carried out. For the data execution time 3rd scenario, the first Paired Sample T-test was conducted. The t-test results can be seen in Figures 13, 14, 15 and 16.

**Test Statistics**

<table>
<thead>
<tr>
<th></th>
<th>RAM1</th>
<th>RAM2</th>
<th>RAM3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mann-Whitney U</td>
<td>22.02</td>
<td>28.562</td>
<td>208.00</td>
</tr>
<tr>
<td>Wilcoxon W</td>
<td>547.80</td>
<td>765.562</td>
<td>673.00</td>
</tr>
<tr>
<td>Z</td>
<td>-5.447</td>
<td>-2.211</td>
<td>-3.579</td>
</tr>
<tr>
<td>Asymp. Sig. (2-tailed)</td>
<td>0.000</td>
<td>0.027</td>
<td>0.000</td>
</tr>
</tbody>
</table>

a. Grouping Variable: STATE_MOM

Figure 13. Mann-Whitney U-Test of Memory Usage at scenario 1 and 2

![Image](image1.png)

Figure 14. Mann-Whitney U-Test of Memory Usage at 3rd scenario

**Test Statistics**

<table>
<thead>
<tr>
<th></th>
<th>TIME3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mann-Whitney U</td>
<td>142.50</td>
</tr>
<tr>
<td>Wilcoxon W</td>
<td>607.50</td>
</tr>
<tr>
<td>Z</td>
<td>-5.49</td>
</tr>
<tr>
<td>Asymp. Sig. (2-tailed)</td>
<td>0.000</td>
</tr>
</tbody>
</table>

a. Grouping Variable: STATE_MOM

Figure 15. Mann-Whitney U-Test of Execution Time 3rd Scenario

![Image](image2.png)

Figure 16. Independent T-Test of CPU Utilization

It should be noted that to see determine the hypothesis whether the data has differences or not. This is important as a basis for decision-making. There are 2 hypotheses, namely H0 and Ha. H0 is accepted if the significance value of the t-test is >0.05, meaning there is no difference in the two variables. Ha is accepted if the significance value of the t-test <0.05, meaning there is a difference in the two variables. The details of hypothesis are:

p-ISSN: 2540-9433; e-ISSN: 2540-9824
1) \(H_0 = \text{There is no difference in memory usage/execution time/CPU utilization between Riverpod and GetX state management.}\)

2) \(H_a = \text{There is difference in memory usage when between Riverpod and GetX state management.}\)

From the output of the tests that have been carried out. The following results were obtained:

1) Memory Usage
   - Scenario 1 (Home Screen): \(0.000 < 0.05\), \(H_a\) accepted. Average data when using Riverpod and GetX are 11.002 and 10.157. It means GetX more efficient in memory usage.
   - Scenario 2 (Search Screen): \(0.027 < 0.05\), \(H_a\) accepted. While average data when using Riverpod and GetX are 11.568 and 10.884. It means GetX more efficient in memory usage.
   - Scenario 3 (Increment): \(0.000 < 0.05\), \(H_a\) accepted. Average data when using Riverpod and GetX are 12.454 and 11.286. It means GetX more efficient in memory usage.

2) Execution Time
   - Scenario 1 (Home Screen): \(0.019 > 0.05\), \(H_a\) accepted. Average data when using Riverpod and GetX are 11.002 and 10.022. It means GetX more efficient in memory usage.
   - Scenario 2 (Search Feature): \(0.000 > 0.05\), \(H_a\) accepted. It means there is no different between Riverpod and GetX
   - Scenario 3 (Increment): \(0.000 < 0.05\), \(H_a\) accepted. Average data when using Riverpod and GetX are 0.254 and 0.245. It means GetX more efficient in execution time.

3) CPU Utilization
   CPU utilization T-Test: \(0.370 > 0.05\), \(H_0\) accepted. It means there is no different between Riverpod and GetX in aspect CPU utilization.

4.6. Discussion

The author's assumption of the data that the reason GetX has more efficient memory usage and faster execution time than riverpod is in terms of the reactivity of components in each state management. Based on the program code that has been developed, GetX has an observable widget approach, where the observer will automatically update the component when changes occur in the observed data. So the component does not need to update itself. While Riverpod has a concept where each component must actively subscribe to the state provided by the state provider. Riverpod will notify each subscribed component to update themselves. This causes GetX to require less memory compared to Riverpod and indirectly affects the time in processing data to be shorter.

Further discussion regarding the performance of GetX and Riverpod is taken from Flutter's own community group on the Telegram platform. There is an assumption in the implementation of state management in the industrial world, Riverpod is easier to maintain than GetX. Riverpod architecture looks cleaner and is suitable for developing more complex applications that have long-term functions. It can be further
research related to the use of these two state management to prove whether code complexity and long-term application maintenance affect the performance of state management. So that these aspects can be considered by developers whether to choose GetX with more efficient memory and time resources but difficult to maintain, or choose Riverpod which consumes slightly more memory and time but is easy to maintain.

5 Conclusion

5.1 Conclusion

The results of performance analysis on the homepage scenario show that with the same CPU performance, GetX state management is proven to be more efficient in memory usage and faster in execution time than Riverpod state management. Furthermore, the performance analysis on the search feature scenario also shows that GetX state management is more efficient in memory usage and faster in execution time than Riverpod state management with the same CPU performance. While in the increment scenario, the analysis results also show that GetX state management is more efficient in memory usage and also faster in execution time with the same CPU performance.

So, the conclusion of this research is that GetX state management is considered better than Riverpod in memory usage and in the execution time, but shows no difference in CPU utilization. Apart from the influence of code complexity and easier long-term maintenance of Riverpod which is argued by Flutter developers in the market.

5.2 Suggestion

Suggestions for future research are that researchers can analyse other state management with the same method or with more developed methods if there are new theories in testing state management such as code complexity and long-term maintenance. On the basis of this research, future researchers can also try to find ways to develop Riverpod and GetX state management to be more efficient and produce better performance in the latest version.

References