Implementing Flutter Clean Architecture for Mobile Tourism Application Development

Ristu Aji Wijayanto
University of Technology Yogyakarta
Yogyakarta, Indonesia

RR. Hajar Puji Sejati
University of Technology Yogyakarta
Yogyakarta, Indonesia

ABSTRACT
The rapid advancement of technology has been a catalyst for several innovative developments across diverse disciplines. The tourism sector is being actively promoted as one of the fields of focus. An increasing number of vendors are currently exploring the development of applications that have the capability to present comprehensive information regarding tourism sites. One issue arises when several vendors engage in application development, as the programs frequently encounter interruptions and upgrades are subject to prolonged durations. Despite the implementation of maintenance practices, the code remains unreadable, exhibiting deficiencies in both its overall pattern and the organization of individual units of code. In the context of existing applications, there exist some limitations, like the absence of enhancements in the authentication functionality and the non-functionality of the maps feature. This study aims to explore the principles of designing applications with a well-organized structure and implementing clean architecture as a structural pattern to enhance the scalability and maintainability of systems. The implementation of a clean architecture facilitates the long-term development and targeted testing of applications. By implementing a segregation of data layers, domains, and clean presentation architecture, the process of categorizing unit code based on its functionality will be facilitated. By employing Test-Driven Development (TDD), a software development approach that prioritizes testing over producing code units, the frequency of errors can be significantly reduced.

General Terms
Clean Architecture, E-Tourism, LCOV

Keywords
Mobile Application, Tourism, Flutter, State Management, Unit Testing

1. INTRODUCTION
The concept of Smart Tourism is closely bound up with the use of digital technology, since it enables organizations to efficiently access an extensive range of information resources. In contemporary businesses and institutions, the integration of technology has become universal. As a result, a location is able to improve its overall visitor experience by using technology and social components in a synergistic manner [1]. According to statistics obtained from the Central Statistics Agency (BPS), it is evident that there has been a consistent upward trend in the number of tourist arrivals in Indonesia. According to available data, the quantity of international visitor arrivals in Indonesia surpassed 7 million visits in 2010 and shown a consistent upward trend until 2019, when the number exceeded 16 million visits [2]. Tourism is a highly popular destination for both domestic and international travelers, owing to its special attraction and appeal. In 2019, Bantul Regency, a constituent of the Special Region of Yogyakarta, documented over 250 tourism spots and attracted a visitor count over 5 million [3]. The proliferation of tourism locations has the potential to significantly contribute to the area economy. The use of the E-Tourism idea is promoted as a method for disseminating information via mobile applications. E-Tourism refers to the use of technology for the purpose of disseminating information and facilitating communication in order to attract tourists. This is achieved by the provision of accessible media services to consumers [4]. However, the advancement of this technology is now limited by challenges related to scalability and application maintenance. The creation of applications is a significant challenge, as it requires continuous adaptation to market demand, technological advancements, competitive business landscapes, and organizational priorities [5].

The undeniable influence of mobile technology applications is seen across several demographics, including millennials, explorers, and experimental travelers. Despite the increasing popularity and significance of mobile applications, a study revealed that over 50% of travelers exhibited a lack of interest in utilizing such programs for their travel purposes [6]. The development and rapid advancement of mobile technology have significantly transformed the human experience. Mobile devices are utilized by individuals for the purposes of engaging in activities such as reading news articles, accessing social media platforms, and playing games. In addition to this, numerous organizations and institutions employ mobile devices as a means of promotion, in response to the rapid expansion of smartphone users. Mobile applications are executed on portable handheld devices that are compact, user-friendly, and capable of being accessed from any location [7]. Thus, mobile applications are regarded as effective platforms for advertising in the sector of tourism [8]. The mobile e-tourism application is designed to offer a range of beneficial functionalities, including access to event details, the ability to locate tourist attractions, curated lists of recommended destinations, interactive maps, and the option to save preferred locations for future reference. These elements aim to enhance the whole holiday experience by providing users with convenient tools and resources.

This research identifies and utilizes a scalable and maintainable mobile tourist application development technique. Flutter clean architecture is used to construct this research application. An architecture design strategy that helps organize, maintain, and test application development across time is needed. In this research, a clean development architecture that leverages dependency inversion to segregate high-level and low-level components is preferable. This architecture also promotes software flexibility and maintainability [9]. Clean architecture guides code structure and layer dependencies. Clean architecture in flutter includes data, domain, and presentation layers. Since each layer has separate duties, structure separation makes the codebase modular and manageable [10]. Development with a clean architecture requires dart and flutter

state management and research support. It generates a simple dart class, using optional arguments, a factory, an interface, and dart to write its functions. Thus, programmers' familiar language and usability make it suitable for complicated system implementation [11]. Application system design requires code knowledge and appropriate architecture. Continued development is planned.

2. CLEAN ARCHITECTURE

In general, develop an application using flutter can be uses a single code base. Otherwise, there are a lot of architecture design patterns that programmers use for Flutter. They are all different ways of managing an app's state. The goal of a design pattern is to provide a clean standard for how our work will be organized, how the components will interact with each other, separate layers so that a change in one is transparent to the others, and most importantly, promote the reuse of blocks of code [12]. This can be done by using a clean architecture that separates components separately and has their own responsibilities, according to predefined logic and flow. Even so, clean architecture is a relative architecture, it does not have a fixed standard, mainly because each state management has its own clean architecture suitability, but still has the same workflow of separating business logic for easy testing and maintainable applications.

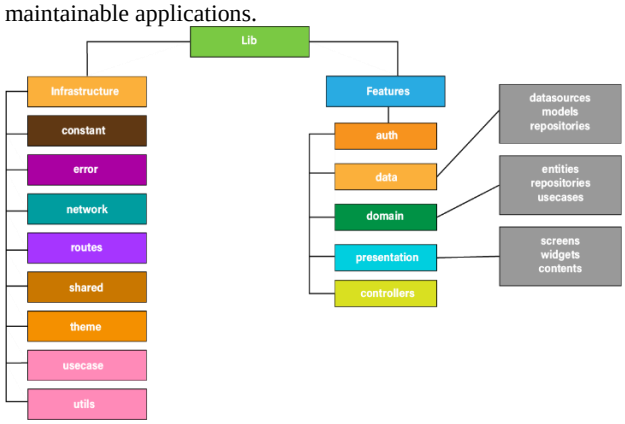

Fig. 1 Flutter Clean Architecture Folder Structure

Clean architecture in the context of Flutter typically consists of three fundamental layers: the data layer, the domain layer, and the display layer, which are organized within a features folder [10]. The clean architecture employed in this study is founded on the author's assumptions and afterwards verified with the relevant stakeholders. This study involves a collaboration with the Head of Marketing at the Bantul Regency Tourism Office, who serves as the primary stakeholder and supplier of data. The primary objective of this architectural design is to achieve a clear separation of concerns and enhance scalability. According to the source cited as [13], the separation of a system into layers that protect business logic from platform-specific implementation can uphold three essential concepts in Software Engineering. These principles include External Modules Framework Agnosticism, Testability, and Isolation.

2.1 Framework Agnosticism

The continual evolution of the mobile application features a design architecture that enables it to operate autonomously without reliance on any particular framework. This is achieved via the implementation of a layered design approach. The application's business logic operates autonomously, without reliance on other libraries or frameworks. Within the realm of Flutter, the business layers of the application are implemented only in Dart, devoid of any cognizance regarding their integration within a Flutter application.

The achievement of increased portability is realized via the development of business logic that is not reliant on any particular framework. This methodology facilitates the utilization of Dart code in web applications and its subsequent transformation into JavaScript [14]. When developers decide to transition from Flutter to an alternate framework such as React Native, they may quickly convert the framework-agnostic business logic to JavaScript, enabling the reuse of both the code and the logical flow of the application.

Furthermore, by employing a framework-agnostic architectural model, frameworks may be utilized as mere instruments, rather than occupying a central role inside the application, as the business logic of the application will not contain any code specific to the framework [13]. However, the application ensures that the framework stays separate from the business logic levels, offering just interfaces and abstract classes. By maintaining a pure Dart layer within the application, it would offer barrier to the frequent changes that occur at the framework level.

2.2 Testability

Testing is flexible using Flutter's Clean Architecture. Business logic is clearly separated from UI and database components, making testing easier [13]. Business rules and the UI can be tested separately, simplifying testing. The inheritance of abstract classes in the core layer simplifies dependency replication during testing. This allows independent, precise evaluation of business rules regardless of implementation. Separate tests for authentication rules can be run independently of the authentication implementation. Software product development requires application testing. Flutter has unit, widget, and integration testing. Unit tests check one function or method under different scenarios. Integration tests complete apps or parts to ensure widgets and services work. Automatic functional testing verifies business logic implementation [15].

2.3 Isolation

The Clean Architecture is a system where the business layer is at the center, isolated from the implementation, allowing for the swapping of outer modules without affecting the business logic. This structure is structured like an onion, with inner layers consisting of more abstract parts and outer layers being more verbose and implementation-specific. Data and Platform layer is the outermost layer, consisting of platform-specific implementations of enterprise and business logic [13]. It includes the GUI, classes that interact with the iOS or Android platform, and classes that interact with databases locally and through the web [15]. Repositories in this layer implement interfaces from the Domain layer, which are injected into the inner layers and used polymorphically to abide by dependency rules.

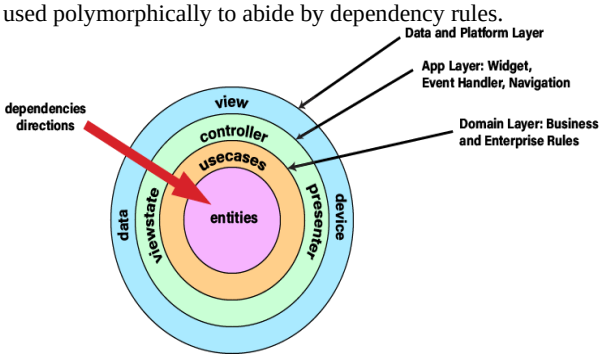

Fig. 2 Flutter Clean Architecture Layer Diagram

Application layer, consisting of Flutter widgets, widget states, event handlers, view controllers, presenters, and navigation controllers, handles all user events issued from the GUI and redirects them to their appropriate use cases in the inner layers. The controllers either update the UI or pass it to the presenter, which executes the appropriate usecases and prepares the result appropriately for the UI [13]. In summary, the Clean Architecture is a system where the business layer is at the center, with the UI and other elements being outer modules that do not impact the business logic.

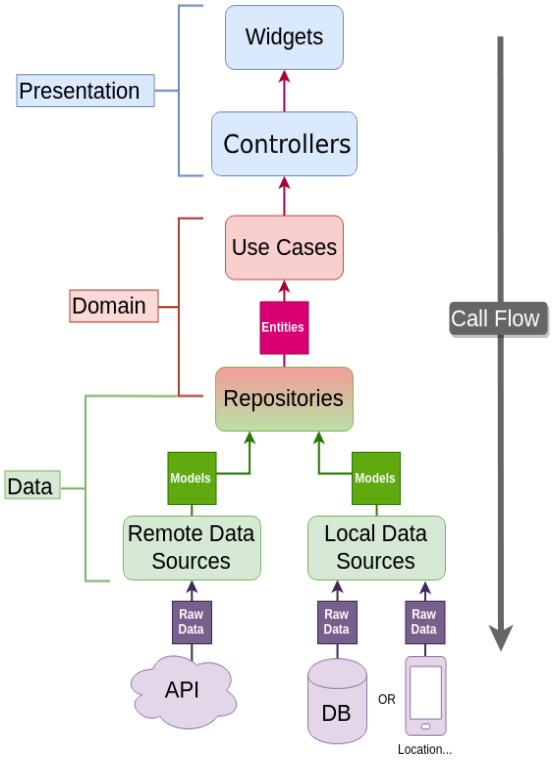

Fig. 3 Flutter Clean Architecture Workflow

The Domain layer is a crucial part of an application architecture, divided into two parts. The outer part is the business rules layer, which contains the main business logic and orchestrates data flow throughout the application [16]. It should not be aware of any outer layers or platform, and can use language-specific libraries like RxDart or the Dart language. The domain layer also contains repositories, interfaces that outline behavior needed by usecases. These repositories allow for dependency injection from the outermost layer into the business rules layer, allowing usecases to call methods through polymorphism. The inner layer, the enterprise rules layer, is comprised of enterprise entities used throughout the application. These entities only interact through usecases that connect them, and are unaware of any other layer. The only reason to modify this layer is if enterprise rules change, as it can be used without being aware of the platform. Inner layers are neither aware of nor dependent on outer layers, but both are aware of and dependent on inner layers. Outer layers represent the concrete mechanisms by which business rules and policies operate, while inner layers are completely independent from implementations. This rule is vital for the success of the architecture, as inner layers are not aware of any classes, functions, names, libraries, etc., present in outer layers [13].

3. RESULT AND DISCUSSION

3.1 Assumption and Hypotheses

The program will provide users with data in the form of informative content regarding various tourist attractions, including comprehensive details about each attraction. Furthermore, the incorporation of clean architecture in the development process of Bantul Tourmate would enhance the ease of application maintenance and testing for developers. The hypothesis of this study proposes that the Bantul Tourmate application can effectively present information about tourist attractions in the Bantul area through the implementation of clean architecture and the utilization of unit testing within the project.

3.2 Implementation

During this phase of implementation, the researchers will explain the process of implementing a clean architecture, as seen in (Fig. 1), by utilizing the Flutter framework and employing GetX state management for data and UI control. The initial phase of implementation involves organizing the folder structure and dividing it according to the levels described in the previous section.

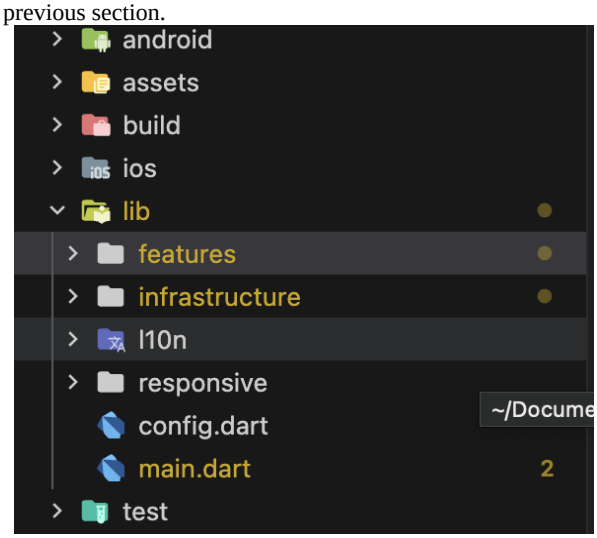

Fig. 4 Main Layer Clean Architecture

The provided visual representation (Fig. 4) illustrates the clean architecture, which comprises two primary components: the feature layer and the infrastructure layer.

3.2.1 Infrastructure Layer

Within the context of Clean Architecture, the Infrastructure Layer serves as a boundary between the internal core, which encompasses the business logic and entities, and the external environment. The major objective is to ensure the integrity of the application's core by keeping it free from irrelevant variables, hence facilitating its maintenance, testing, and adaptability to evolving technologies. This layer additionally facilitates the application's high modularity and pluggability, hence enabling the seamless replacement or update of external dependencies while minimizing any adverse effects on the core functionality.

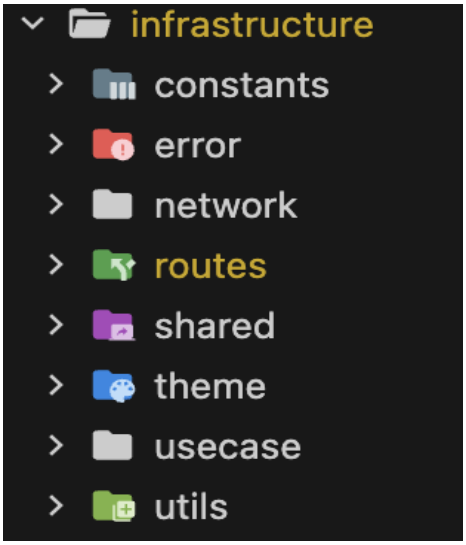

Fig. 5 Layer Infrastructure layer contains core folder or building block for dependencies

3.2.2 Feature Layer

The feature layer is a component within the program that serves to organize and separate distinct features. It specifically separates components such as data, domains, controller, and presentation. The features layer comprises of several architectural layers, including controller, data, domain, and presentation, which are designed based on clean architectural principles. These layers serve the purpose of dividing the code base into separate components.

3.2.2.1 Data Layer

This layer is responsible for the management of datasources, repositories, and models. The system is responsible for managing the retrieval and storage of data.

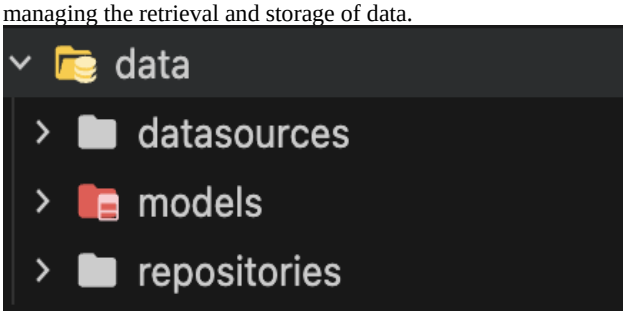

Fig. 6 Three folders in the Data Layer allow network and local data access and administration.

1. Datasources
   Data sources establish a connection between the application and various data repositories, such databases, application programming interfaces (APIs), or local storage. They are responsible for managing data retrieval and storage processes.
2. Models
   This layer is responsible for the management of data sources, repositories, and data models. The system is responsible for the retrieval and storage of data. The model consists of two components, specifically the answer and request. In the response layer, it is necessary to establish the definition of variables that are reliant on data gained from the API. Meanwhile, the request layer encompasses a model that is used as a parameter for transmitting to the API in order to acquire a response.
3. Repositories
   The function of this component is to facilitate the exchange of information between the presentation, domain, and data layers. The system manages user input, directs it to the domain layer's business logic, facilitates data transfers, and governs the flow of information, encompassing navigation and routing. It effectively coordinates the many architectural components of an application.

3.2.2.2 Domain Layer

For maintainable and scalable applications, the domain layer is crucial. Without disrupting other system components, developers can adjust and improve the application's functioning by centralizing the important business logic under this layer. Code reuse, testability, and business adaptability are improved by this architectural approach.

Fig. 7 The domain layer has entity, repositories, and use case folders. The layer encapsulates the data layer model.

1. Entities
   Entities are essential elements or abstract ideas inside the context of business. They contain both data and behavior that connects with a certain concept.
2. Repositories
   Repositories are conceptual structures that establish the formal agreement for accessing and storing data pertaining to the domain entities. The abstraction of data storage and retrieval is employed to ensure the decoupling of the domain layer from the actual data source. Repositories facilitate efficient data retrieval and contribute to the preservation of the principle of separation of concerns.
3. Usecases
   Use cases are a means of representing discrete actions or procedures that an application has the capability to accomplish. Software components are responsible for encapsulating the essential business logic necessary to accomplish a certain job or objective.

3.2.2.3 Presentation Layer

The Presentation Layer functions as the primary interface via which users access the program, and their initial perceptions of the application are frequently shaped by the quality of interaction facilitated by this layer. Thus, it is crucial to provide intuitive and captivating visual elements and interactive features within the Presentation Layer.

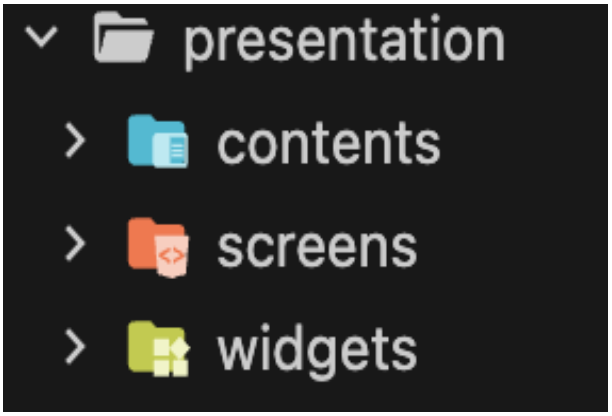

Fig. 8 All folders in the presentation layer display domain and data layer data

1. Contents
   Within this particular layer, the component is responsible for the processing of data received from the controller. Subsequently, it presents this processed data in the form of a user interface (UI).
2. Widgets
   The Widget is charged with the task of presenting the processed display on the levels of information and displays.

3. Screens
   The screen layer primarily receives data that has been processed by the content layer, which functions as a user interface (UI). This data is further refined at the screen layer to provide more particular information.

3.2.2.4 Controller Layer

The controller assumes a crucial position within the broader framework of an application as it assumes responsibility for managing user input, facilitating interactions between the model and view components, managing UI-related logic, and assuring the application's responsiveness and maintainability. The promotion of a clear separation of concerns facilitates the development of a codebase that is characterized by modularity and testability

3.3 Testing Method

The utilization of clean architecture is closely aligned with a development approach that prioritizes the preservation of code maintainability, scalability, and testability. The process of testing serves as a valuable tool for assessing the functionality and performance of our code. Flutter encompasses three commonly utilized types of testing: unit testing, widget testing, and integration testing. Each test possesses unique capabilities. The TDD (Test-Driven Development) approach was employed by the researchers. Test-driven development (TDD) is a component of agile development. However, academics employing this approach do not primarily emphasize agile development itself, but rather its execution. TDD also constitutes a component of the implementation process for application development. Test-driven development (TDD) is a software development approach in which unit tests are written before the corresponding unit code. The primary objective of TDD is to provide a systematic and quantifiable development process that yields well-functioning systems that are straightforward to maintain. LCOV is also utilized in the process of testing. LCOV is a valuable tool that facilitates comprehensive coverage of testing outcomes and presents them in a user interface format, hence enhancing the visibility of the extent to which our unit code is functioning.

3.3.1 User Interface

3.3.1.1 Login Screen

The Login page serves as the primary mechanism for verifying the user's identity, determining if they have registered previously. Additionally, it offers a create account functionality for users who do not possess an account.

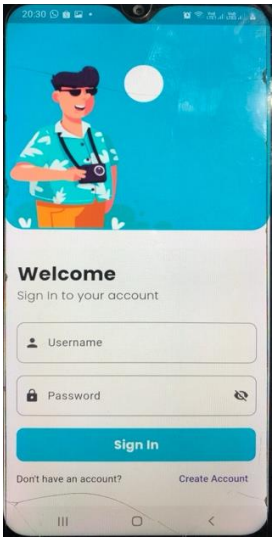

Fig. 9 Login Screen Bantul Tourmate Application

3.3.1.2 Home Screen

The homepage will exhibit the user's name and salutation, along with their location and profile picture. In addition, there are numerous features accessible, including explore, event, weather, and support functionalities. In addition, there are suggested destinations available to assist users who may be uncertain in selecting a tourism location.

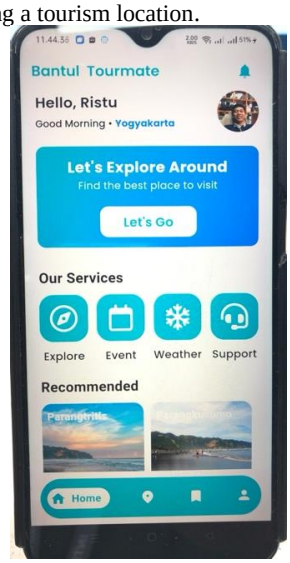

Fig. 10 Home Screen Bantul Tourmate Application

3.3.1.3 Explore Screen

A page that displays various tourist attractions which are the main features in the application. This feature makes it easier for users to find interesting tourist attractions that suit their tastes.

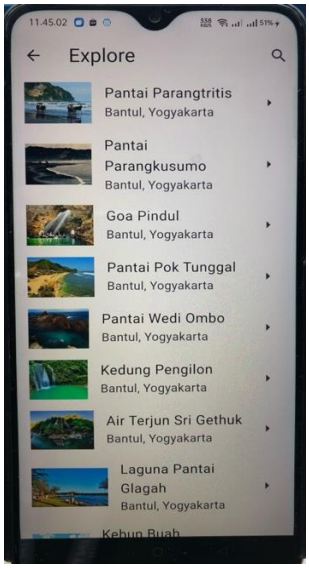

Fig. 11 Explore Screen Bantul Tourmate Application

3.3.1.4 Profile Screen

Users can view their entire name, username, email, and city on their profile. Settings includes an account logout button.

Fig. 12 Profile Screen Bantul Tourmate Application

3.3.1.5 Maps Screen

The page includes geographical markers and provides information about various areas using maps.

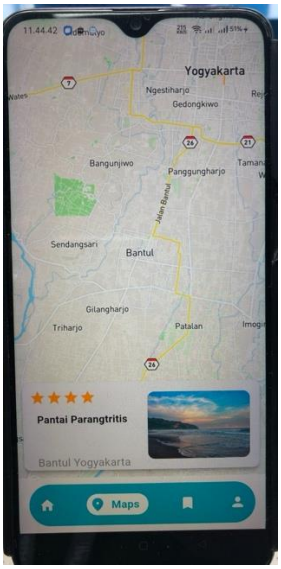

Fig. 13 Maps Screen Bantul Tourmate Application

3.3.1.6 Bookmark Screen

The bookmark page shows data about favorite tourism attractions to help people plan trips.

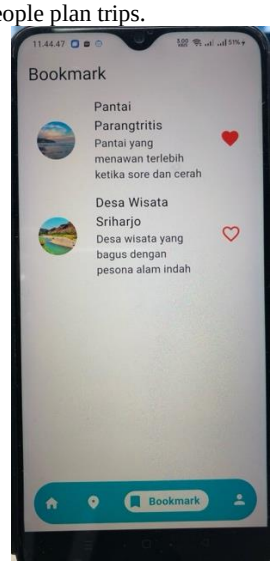

Fig. 14 Bookmark Screen Bantul Tourmate Application

3.3.2 LCOV Experimental Testing

The experiment was conducted utilizing the previously outlined methods. LCOV provides a thorough summary of all tests, encompassing unit tests, widget tests, and integration tests. The author presents information using tabular data derived from tests conducted on the key features, displaying the collected results in the following manner.

Table 1 Compilation of unit testing outcomes categorized by characteristics utilizing LCOV

Featured	Total Case	Result
Auth	164	93,57%
Maps	78	100%
Profile	22	98%

Bookmark	23	100%
Home	119	97,6%
Total	**406**	**97,83%**

According to the data presented in (Table 1), the LCOV report indicates that a total of 406 successful tests were conducted, encompassing integration testing, unit testing, and widget testing. The ultimate outcome achieved was 97.83% when measured as the proportion of tasks executed accurately. Despite the imperfect nature of the acquired results, the implementation of this clean architecture has proven to be effective in terms of functionality, and the anticipated interface is suitable.

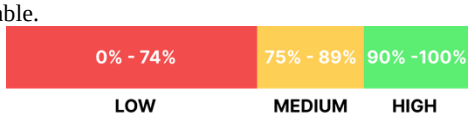

Fig. 15 Classification of Code eligibility indicator on LCOV

The test percentage achieved a value of 97.83% based on the given parameters, placing it in the high category. The findings are highly gratifying since they fall into the green category, signifying that the code is functioning correctly and is prepared for utilization.

3.3.3 Questionnaire Data Sampling

Taking questionnaires to programmers was carried out randomly with programmers who were in accordance with their profession and were related to using Flutter.

Table 2 List of programmer questionnaire questions

No	Questions	Answer
1	Does adopting Clean Architecture aid in isolating business logic from infrastructure code better than not using Clean Architecture?	Yes/No
2	Does the adoption of Clean Architecture allow for easier code maintenance compared to not utilizing it?	Yes/No
3	Does applying Clean Architecture demand more time and effort in the initial development compared to not using it?	Yes/No
4	Can code be tested more effectively with Clean Architecture than without it?	Yes/No
5	Does the adoption of Clean Architecture considerably reduce the complexity of your project compared to not utilizing it?	Yes/No

Table 3 Result of Questionnaire

User	Q1	Q2	Q3	Q4	Q5
Joko	Yes	Yes	No	Yes	No
Herlambang	Yes	Yes	Yes	Yes	Yes
Nurjaman	No	Yes	Yes	Yes	Yes
Difa	No	No	No	Yes	No
Retno	Yes	Yes	Yes	Yes	No

From Based on the acquired results, developers predominantly opt for the affirmative choice from the range of responses between yes and no. Based on these findings, the use of clean architecture can enhance the quality of code development by promoting tidier and more organized code structure, as well as facilitating regular and systematic testing.

3.4 Analysis

The clean architecture pattern is popular. Clean Architecture has been shown to improve code understanding, scalability, and application maintainability. This application uses TDD to arrange its architecture. The architecture has three layers: data, domain, and display.

Data layer testing verifies application-server communication. Data validation and user interface business logic control are handled by the domain layer. Finally, the presentation layer is tested to ensure the displayed information satisfies expectations.

Several auxiliary modules are checked, including the repository module, which verifies the application-server and data layer connections. Module controller testing checks the controller's operation, which is the application's business logic and connects the user interface to the repository's data. Unit test results showed 97.83% passing.

To analyze program authoring and testing efficacy after implementing clean architecture, the author collected questionnaire data from 5 proficient programmers who had expertise with Flutter or mobile apps. The data showed 17 yes and 8 no answers to 5 questions from 5 developers. This shows that it can help developers improve code quality, manage applications, and test them easily.

4. CONCLUSION

The Bantul Tourmate initiative serves as a means to facilitate the transition towards e-governance and to bolster the development of Bantul city as a smart city. The Bantul Tourmate program offers a range of features that assist users, particularly travelers, both local and international, in discovering tourist attractions within Bantul Regency. This task is facilitated by the presence of several features, including but not limited to tourism exploration, event information, maps, and more supportive functionalities. The adoption of Clean Architecture has emerged as a prevalent software development paradigm, which has been effectively applied. Empirical evidence, obtained through the utilization of Test-Driven Development and LCOV methodologies for test outcome visualization, indicates a success rate of 97.83% for unit tests, widget tests, and integration tests. According to the Indicator, the application reaches a state of moderate development, indicating its potential to transition into the production category with continuous maintenance. The successful execution of clean architecture is seen in its ability to effectively present the intended data. In the context of application development, there is an expectation that the implementation of Lean UX methodologies will enable the delivery of a user interface that optimizes the user or client experience. Additionally, it is desirable for the application to be accessible to a wide range of users.

5. REFERENCES

[1] A. S. Dasuki, M. Djamin, and A. Y. Lubis, “The strategy of photovoltaic technology development in Indonesia.” [Online]. Available: www.elsevier.com/locate/renene

[2] A. Mun’im, “PENYEMPURNAAN PENGUKURAN KONTRIBUSI PARIWISATA: ALTERNATIF PERCEPATAN PERTUMBUHAN EKONOMI INDONESIA Improvement on the Measurement of Tourism Contribution: An Alternative to Accelerating Indonesia’s Economic Growth,” 2022.

[3] D. Widiyastuti et al., “Analisis Tingkat Perkembangan Destinasi Wisata Kabupaten Bantul, Daerah Istimewa

Yogyakarta," Spatial Development Journal, vol. 02, no. 01, p. 2023.

[4] S. Saniati, M. A. Assuja, N. Neneng, A. S. Puspaningrum, and D. R. Sari, "Implementasi E-Tourism sebagai Upaya Peningkatan Kegiatan Promosi Pariwisata," International Journal of Community Service Learning, vol. 6, no. 2, pp. 203–212, Jul. 2022, doi: 10.23887/ijcsl.v6i2.45559.

[5] C. Chen, R. Alfayez, K. Srisopha, B. Boehm, and L. Shi, "Why is it important to measure maintainability and what are the best ways to do it?," in Proceedings - 2017 IEEE/ACM 39th International Conference on Software Engineering Companion, ICSE-C 2017, Institute of Electrical and Electronics Engineers Inc., Jun. 2017, pp. 377–378. doi: 10.1109/ICSE-C.2017.75.

[6] N. Alif Amri Nik Hashim, N. Abu Bakar, E. Noreni Mohamad Zain, N. Dalila Mat Yusoff, and N. Hafizah Muhammad, "Travel Mobile Applications Technology: Examining the Reliability and Validity of Instruments," International Journal of Advanced Science and Technology, vol. 29, no. 6s, pp. 3882–3885, 2020.

[7] M. R. Islam and T. A. Mazumder, "Mobile Application and Its Global Impact," 2010. [Online]. Available: https://www.researchgate.net/publication/308022297

[8] X.-S. Yang and Institute of Electrical and Electronics Engineers, Proceedings of the World Conference on Smart Trends in Systems, Security and Sustainability (WS4 2020) : July 27-28, 2020, virtual conference.

[9] D. Esteban, S. Rodriguez, A. E. Rojas, H. Florez, and D. Sanchez, "Towards a Clean Architecture for Android Apps using Model Transformations." [Online]. Available: https://developer.android.com/jetpack

[10] "Flutter — Clean Architecture. Clean Architecture is a software design… | by Samra Khan | Medium." https://medium.com/@samra.sajjad0001/flutter-clean-architecture-5de5e9b8d093 (accessed Sep. 11, 2023).

[11] I. Firman Ashari, M. Fazar Zuhdi, M. Tyaz Gagaman, and S. T. Denira, "Kolepa Mobile Application Development Based on Android Using SCRUM Method (Case Study: Kolepa Minigolf and Coffe Shop)," 2022. [Online]. Available: http://jurnal.polibatam.ac.id/index.php/JAIC

[12] S. Y. Ameen and D. Y. Mohammed, "Developing Cross-Platform Library Using Flutter," European Journal of Engineering and Technology Research, vol. 7, no. 2, pp. 18–21, Mar. 2022, doi: 10.24018/ejeng.2022.7.2.2740.

[13] S. Boukhary and E. Colmenares, "A clean approach to flutter development through the flutter clean architecture package," in Proceedings - 6th Annual Conference on Computational Science and Computational Intelligence, CSCI 2019, Institute of Electrical and Electronics Engineers Inc., Dec. 2019, pp. 1115–1120. doi: 10.1109/CSCI49370.2019.00211.

[14] "Dart overview | Dart." https://dart.dev/overview (accessed Sep. 12, 2023).

[15] N. Kuzmin, K. Ignatiev, and D. Grafov, "Experience of Developing a Mobile Application Using Flutter," 2020, pp. 571–575. doi: 10.1007/978-981-15-1465-4_56.

[16] "Flutter Clean Architecture [1]: An Overview & Project Structure - DEV Community." https://dev.to/marwamejri/flutter-clean-architecture-1-an-overview-project-structure-4bhf (accessed Sep. 13, 2023).