J-COSINE (Journal of Computer Science and Informatics Engineering)
Vol. 10, No. 1, June 2026 E-ISSN:2541-0806
Accredited Sinta-4 by RISTEKDIKTI Decree No. 79/E/KPT/2023 P-ISSN:2540-8895

Clean Architecture Implementation for Android Application Rinjani Visitor

I Dewa Agung Adhinata Budhiastawa[1]*, Dwi Ratnasari[1], Noor Alamsyah[1], Andy Hidayat Jatmika[1], Raphael Bianco Huwae[1], M. Taqwa Mahajaya[2]
[1]Dept Informatics Engineering, University of Mataram
Jl. Majapahit 62, Mataram, Lombok NTB, INDONESIA

[2] CV Softcopy
Mataram, Indonesia
Email: agungnata2003@gmail.com, [dwi.ratnasari, nooralamsyah, andy, Raphael.bianco.huwae]@unram.ac.id, maha_jaya@rocketmail.com

Corresponding Author

Abstract **The growing complexity of mobile applications necessitates scalable architectural solutions to streamline maintenance and feature expansion. This study presents the implementation of Clean Architecture in the *Rinjani Visitor* Android application, an ecotourism platform for Mount Rinjani National Park, Indonesia. The application integrates local tourism services while addressing challenges in code maintainability. Leveraging Flutter and Dart, the architecture is structured into three decoupled layers (presentation, domain, and data) to isolate business logic from technical implementation. The methodology encompasses requirement analysis, feature-based code structuring, and user acceptance testing via black-box methods. Results demonstrate successful layer integration, with all 21 tested functionalities adhering to specifications. The modular design simplifies collaborative development, reduces code interdependency, and supports agile enhancements. The study advocates periodic architectural reviews to sustain these benefits amid evolving technological demands.**

Key words**: Clean Architecture, Flutter, Android application, Ecotourism.**

I. INTRODUCTION

The rapid advancement of information technology that opens access to many people greatly affects the field of information in everyday life. This phenomenon is caused by various factors such as the influence of globalization and the phenomenon of Industry 4.0. This is reinforced by the increasing number of people who use smartphones in every aspect of life, based on the results of studies showing that the adoption of smartphone use is increasing from year to year throughout the world, especially in Indonesia [1].

The increasing adoption of smartphones over time in the world opens up opportunities for application developers to create their own applications [2], competing with other competitors in the digital world. Therefore, many companies and entrepreneurs are involved in creating their services in the field of smartphones, especially applications on the Android operating system [3].

Mount Rinjani National Park in West Nusa Tenggara is the most popular ecotourism destination for domestic and foreign tourists [4]. With very diverse flora and fauna in the mountain environment, the Wallace border, the meeting point between the typical flora and fauna of Asia and Australia [5]. Although this ecotourism has experienced economic growth from year to year, the amount of community involvement in local ecotourism, especially in the Senaru Village area, is still low. Local community participation is still very limited even though Senaru Village is part of Gunung Rinjani ecotourism. Although there are 18 travel entrepreneurs and 11 tourist facilities in Senaru Village, all of them are held by entrepreneurs outside the region, while local people only work as porters and guides [6].

Therefore, CV (Commanditaire Vennootschap) Softcopy formed a development team that aims to develop an information system that helps local ecotourism business practices in the Mount Rinjani area, called Rinjani Visitor. Rinjani Visitor is a collaborative project between CV Softcopy and MBKM (Merdeka Belajar Kampus Merdeka) Interns through the government's Kedaireka Matching Fund program [7], focusing on the design of an integrated digital information system for ecotourism business practices in the Mount Rinjani area.

Fig. 1. Rinjani Visitor App Logo

The Rinjani Visitor information system is a local tourist information system that provides tourist services, lodging, and climbing Mount Rinjani from local partners in Senaru Village to tourists visiting the Mount Rinjani region. This information system consists of 3 parts, namely a server that provides information about tourism on Mount Rinjani, with an API that can be accessed to get

information about packages from cooperating partners, then the Rinjani Visitor website which is an official web page that provides information about services and where to purchase packages, and the Rinjani Visitor mobile application made with Flutter, as the main package purchase application for visitors.

Flutter is a Software Development Kit from Google that can be used in developing cross-platform smartphone applications, Android, iOS, Web, and Desktop [8]. Flutter uses Dart as the main programming language of this framework [9]. Flutter uses the concept of widgets as a basic base for composing interface blocks on the application. Any view that exists on a Flutter app is all based on widgets, from buttons to components to pages [8]. Flutter has an advantage over other frameworks, which is that the code used will be compiled into native code without using an interpreter in the process, which increases the performance of the program development process [10].

In the development process, the complexity of the application increases along with the addition of application features. This can make the code maintenance process more difficult, affecting the development team's ability to improve and maintain the existing application. Therefore, the use of software architecture will greatly help the team in reducing the burden in the development process, while reducing the risk of accidental damage during the application development process [11].

Therefore, the process of developing an application requires the right software architecture method. For this reason, the author will use the Clean Architecture method as the main architecture in developing the Rinjani Visitor mobile application.

Clean Architecture, as proposed by Robert C. Martin, emphasizes the separation of things by organizing source code into distinct, self-contained layers [12]. This architectural style is designed to be independent of frameworks, user interfaces, databases, and external agencies, thereby increasing testability and separating core business logic from technical implementation details.

Clean Architecture focuses on grouping tasks in the source code based on layers that serve to separate the tasks in the code. [13]. This architecture has the advantage of being independent of the framework or language used, independent of the database used, and isolated from external influences. These advantages make Clean Architecture can be applied in various source codes that require Clean Architecture implementation without depending on the framework and language used [14].

In the implementation process, Clean Architecture has a standard rule called the Dependency Rule. The rule states that the source code can only depend on the source code in the inner layer, and the source code in the inner layer must not know or depend on the source code in the outer layer [11]. In implementing Clean Architecture into Flutter Android application projects, it is generally divided into three layers, namely the presentation layer, the domain layer, and the data layer [14].

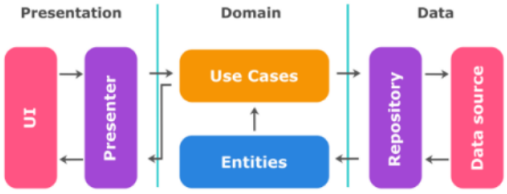

Fig. 2. Clean Architecture for Android app development [14]

Therefore, the main issue can be formulated as follows: how to apply Clean Architecture to the Rinjani Visitor application, and how to effectively implement Clean Architecture in code form, with the limitation that the architecture implementation only focuses on the source code in the Android Flutter application.

The purpose of implementing Clean Architecture is to facilitate maintenance and development for the team, separate responsibilities between layers, and enable feature expansion without overhauling the entire system. This research will demonstrate how these purposes are met by detailing the structuring of source code into distinct features and layers (presentation, domain, and data) to separate responsibilities by using a practical implementation example within the authentication feature, showing how layers interact while remaining independent. After that, the application will be tested through user acceptance testing, confirming that the architecture supports a positive user experience. By demonstrating these points, this study shows how the implementation is useful for maintaining source code, simplifying feature expansion, and providing a better application experience for users.

II. Literature Review

In the process of implementing clean architecture, a literature study with existing research is needed to guarantee that the implementation process is carried out correctly. The elements to be examined in such a study include research related to clean architecture and its implementation, extreme programming, Flutter, and the use of user acceptance testing.

For the reference of clean architecture implementation, the research of MVVM Design Pattern Implementation and Clean Architecture on Android Application Development (Case Study: Agree Application) will be utilized as the primary reference [14]. This research explains the stages of the process of clean architecture implementation, which is meticulously detailed and easy to understand. This includes how to implement Clean Architecture with the development methodology that is already running, as well as the stages of explaining its application starting from requirements analysis, source code structuring, to implementation (Domain, Data, Presentation). These implementation stages do not prioritize the Scrum development methodology employed by this research. Consequently, the Clean Architecture implementation stages in this research can be utilized as the primary

reference point, along with methods such as extreme programming.

Another research for implementing the clean architecture was also found in the research Implementation of Clean Architecture in Food Ordering Application using the Slope One Algorithm Method [15]. This research focused on creating a food delivery app by implementing the clean architecture and comparing it with another architecture. The result showed that a food delivery app that uses the Clean Architecture principle has a higher level of maintenance than the application of the MVC architecture, namely 78.2% compared to 54.8%, where 78.2% can be said to be maintainable because it gets a value higher than 67%. Even though there is no software development methodology like XP or similar in this research. In terms of methodology, this study uses a fairly simple method, starting from design, clean architecture implementation, and application of the nutrient content algorithm for its application.

For reference regarding the usage of extreme programming in the field of general Android applications, study about Developing a Class Scheduling Mobile Application for Private Campus in Tangerang with the Extreme Programming (XP) Model is used as a reference [16]. this research is the development of mobile applications for private campus purposes. this research uses the Flutter framework in the process of making it & uses Black Box Testing for testing purposes, so it is suitable to be used as a reference in explaining the implementation of clean architecture in applications developed using Flutter.

Then, for information about the main Extreme Programming references, references from the research Agile software development: Methodologies and trends are used as a reference [17]. this is because this research has more detailed information about the agile method itself, especially the XP method. in this research, there are also more complete XP life cycle stages, so this research can be used as a reference for the preparation of the XP methodology to be used.

Based on the results of the research, the research conducted is the implementation of clean architecture on the Rinjani Visitor mobile application. the development stages will use extreme programming, with the implementation process starting from analysis, structuring the source code, implementing clean architecture, and applying black box testing, in this case, User Acceptance Testing.

III. METHODOLOGY

The development methodology used is the Extreme Programming (XP) method [18]. Compared to other agile methods, XP is suitable because it emphasizes team collaboration, is more flexible with uncertain changes, and is appropriate for small teams. In the Agile Software Development trend research conducted by Al-Saqqa, the full version of the Extreme Programming development life cycle consists of 6 parts, namely the exploration phase, planning phase, iterations to release phase, production phase, maintenance phase, and death phase [17].

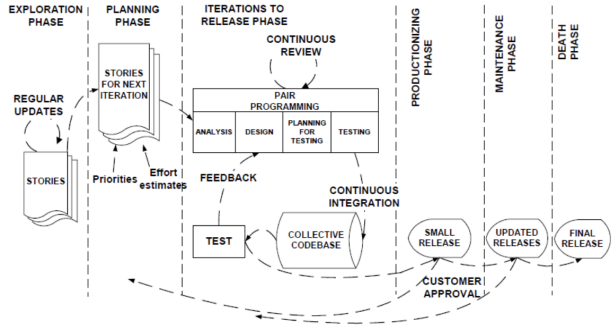

Fig. 3. Extreme Programming Life Cycle [17]

In the process of implementing clean architecture in the Rinjani Visitor application, the process is only carried out when the application is in the development stage, so the phases used are limited to the exploration phase, planning phase, and iteration to the release phase. this method also carries out several modifications to suit the needs of the team, as in the following figure:

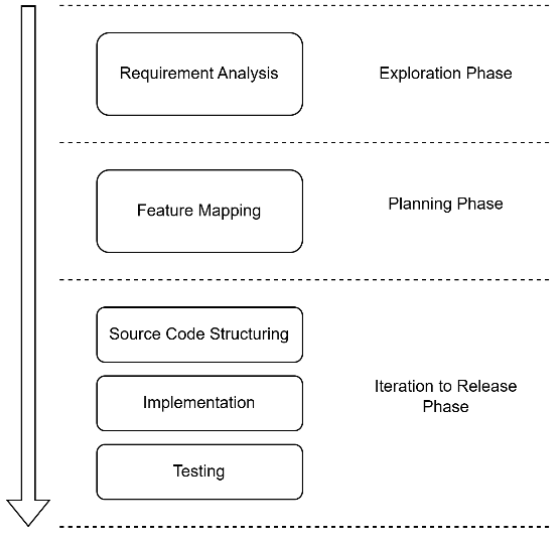

Fig. 4. Detailed Extreme Programming Phases

The following is a detailed explanation of each stage that will be carried out:

1) Exploration Phase

In the exploration phase, the team will undergo requirement analysis. the team and client will collaborate to define user requirements and user stories [19]. Use cases in this phase are needed to get a basic overview of the interrelationships between features in the application, so that they can be grouped into appropriate features. at this stage, the application use cases have been created by the main team so that we only adjust them according to the needs of the mobile application.

2) Planning Phase

In the planning phase, the process of grouping features based on the given use cases is carried out. by grouping them into features, this allows the team to focus on writing

code based on features without being related to one another in the next stage. the result released in this phase is a feature table that can be used as a reference for implementing clean architecture in the next phase.

3) Iteration to Release Phase

The iteration to release phase is the implementation phase in the form of source code based on the results of the planning phase. However, in this project, formal development iterations were not strictly enforced due to dynamic team availability and overall progress. Therefore, this section will not detail discrete iteration cycles. Instead, it will focus directly on the practical, source-level implementation of Clean Architecture within the Rinjani Visitor application, beginning with code structuring and followed by an in-depth look at the authentication feature.

In this phase, the mobile application will undergo source code structuring based on feature lists. Subsequently, the implementation of clean architecture will be executed at each feature. Due to the substantial number of features that have been grouped, it is impractical to explain each of them in this research. Consequently, only one feature will be discussed. For this case, the feature that will be the focus of discussion is the authentication feature.

The application will also undergo Black Box Testing, specifically User Acceptance Testing. This will ensure that the implementation of clean architecture will not alter the functionality of the app. Black Box Testing is a testing method that focuses on software functionality specifications without having to know the design and program code to determine whether the system is running according to the specified specifications [20]. This testing has the advantage that the implementation process does not involve specific programming skills [21]. The User Acceptance Testing method works by testing each application feature based on the application feature specifications without knowing how the application works. If the feature runs well, it will be declared a success, and vice versa if the feature has problems, it will be declared a failure. The results of this test will be used to evaluate the application being developed [22].

IV. Result & Discussion

This section details the process and outcomes of implementing Clean Architecture in the Rinjani Visitor application. The discussion is organized according to the modified Extreme Programming (XP) phases outlined in the methodology. The discussion is organized into three main parts, namely Project Development, Clean Architecture Implementation and Analysis, and Testing and Validation.

A. Project Development

This section explains how the XP process is outlined, namely the exploration phase and the planning phase. This initial phase is important because it is used as the basis and main reference in implementing a better clean architecture.

1) Exploration Phase

This phase started with the requirement analysis. Requirement analysis is divided into several parts, namely user stories analysis and grouping user stories based on features.

The following image in Figure 5 is a use case diagram of the Rinjani Visitor application that has been agreed upon and created by the previous team. It can be noted that this mobile application only has one actor, namely the User, because this application is specifically designed as a product purchasing application. This diagram has several use cases, including displaying profiles, displaying wishlists, viewing products and their details, ordering products that have been ordered, booking products, and making payments. All of these use cases can only be accessed when the user actor has registered in the application and logged in to the application.

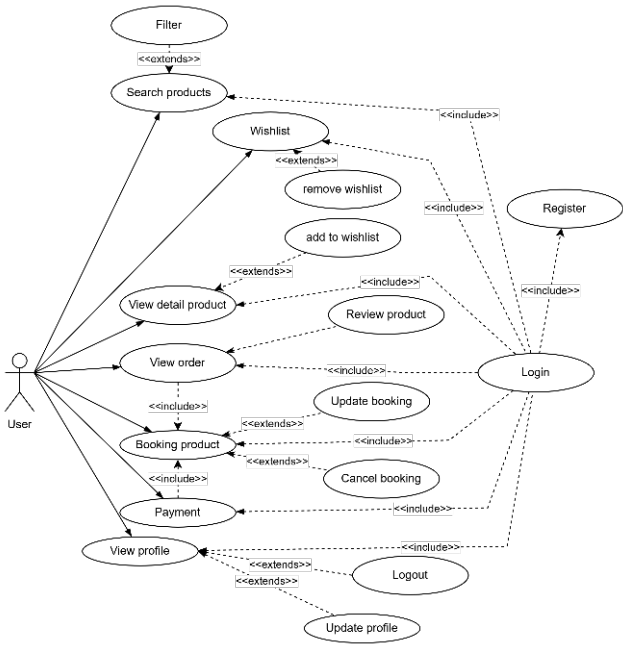

Fig. 5. Use Case Diagram Rinjani Visitor App

2) Planning Phase

a) Feature Grouping

Based on Figure 6, the required functionalities were grouped into the high-level features shown in Table I. Grouping based on features is necessary to prevent overlapping responsibilities between one feature and another. Each of these features encompasses a set of related user stories that will guide the development in the iteration phase. The result released in this phase is a feature table that can be used as a reference. Another feature, such as notification also been added as these features are essential for the Rinjani Visitor mobile application to handle notifications, such as notifying the user if the product order is accepted.

TABLE I. FEATURE LIST

Features	Description
Authentication	Set authentication functions such as register, login, edit profile, and logout.
Booking	Manage product orders (select number of visitors, add-ons, etc.)
Favourite *(wish list)*	Add and remove favourite products, and display products that have been favourited
Order	Purchase products that have been booked, select a payment method, and display a list of products that have been purchased.
Product	Displays the products for sale including their details (description, purchase time, list of images, etc.)
Review	Send review
Notification	Show notification

B. Clean Architecture Implementation and Analysis

1) Implement Separation of Concerns

This stage is the process of structuring the source code, which functions to tidy up the source code by separating it into features and creating an initial layer separation. Based on the previous table, there are about 6 main features that have been grouped based on the given case study diagram. Therefore, the code structuring process is grouped based on 6 main features.

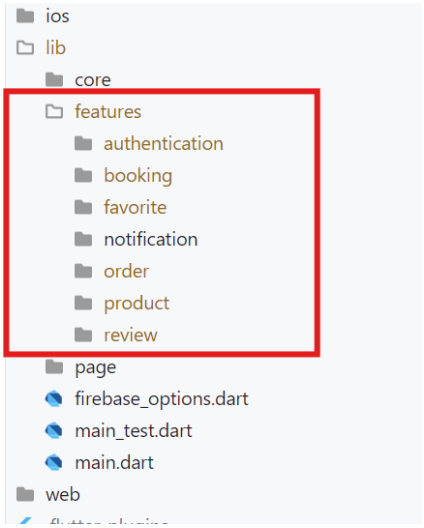

Fig. 6. Application Structuring Based on Features (Orange Highlight)

In Figure 6, the source code is structured into directories based on the features defined in the planning phase. This modular, feature-centric design is a key enabler of effective team collaboration. By creating distinct boundaries for each feature, such as 'authentication' and 'booking', the architecture allows for a parallel development workflow.

For instance, one developer can implement the user registration logic within the 'authentication' folder, while another simultaneously builds the booking functionality within its respective folder, without interfering with each other's work. This separation minimizes code interdependencies between features, drastically reducing the likelihood of complex merge conflicts and allowing team members to take clear ownership of their respective modules. Consequently, the team can develop faster and more efficiently, as developers only need to understand the inner workings of the specific feature they are assigned.

After this, the process continues to create initial layer separation for every feature. Because there are 7 features that need to be implemented, only the authentication feature will be explained in more detail.

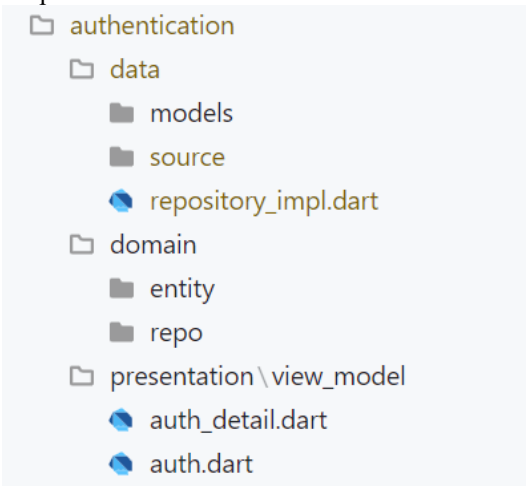

Fig. 7. Application Structuring Based on Layer

In Figure 7, the three main layer folders are created inside the authentication feature, namely domain, data, and presentation. Each folder contains code that matches its respective concentration based on the Clean Architecture principle. The domain folder stores code related to entities that present entities in the code, functioning as a basic foundation for code that will be used in the code in the outer layer. The data folder stores a collection of codes related to communication between the internal data source (database) and external data sources (server API) based on the repository entity that has been determined in the domain folder. The last is the presentation folder, which presents the code source in the presenter layer, functioning to process and display data in the application.

2) Enforcing the Dependency Rule

After source code structuring is finished, the implementation process is carried out, focusing on the authentication feature. The process begins with writing code first on the domain layer, followed by implementing the code in the data layer, and ends with integration into the view model in the presentation layer.

a) Domain Layer

The domain layer is a layer that handles communication between the presentation layer and the data layer. At this stage, the abstract class repository and the entity class is defined. The entity class is used as the basis of communication between the presentation layer and data

layer, while the repository class is used as the foundation of repository class implementation in the data layer.

```
class AuthEntity {
  String? userId;
  String? username;
  String? email;
  String? role;
  String? accessToken;
  String? refreshToken;
  DateTime? accessExpiredAt;
  AuthEntity(
      {this.userId,
      this.username,
      this.email,
      this.role,
      this.accessToken,
      this.refreshToken});

  copyWith({String? accessToken}) {
    this.accessToken = accessToken ?? this.accessToken;
  }

  String toAccessTokenAuthorization() {
    return "Bearer $accessToken";
  }

  String toRefreshTokenAuthorization() {
    return "Bearer $refreshToken";
  }
}
```

Fig. 8. *AuthEntity* Class

The class at Figure 8 is an entity class named *AuthEntity*. This class is used to communicate between the data layer and the presentation layer. This class also has several functions that are used for matters relating to the condition of user accounts stored in the application.

```
You, 10 months ago | 2 authors (You and one other)
abstract class AuthRepository {
  Future<AuthEntity?> register(
      {required String username,
      required String email,
      required String country,
      required String password});

  Future<AuthEntity?> logIn({required String email, required String password});
  Future<AuthDetailEntity?> getUserDetail(String accessToken, String userId);
  Future<void> updateUserDetail(
    String accessToken,
    String userId, {
    String? phoneNumber,
    String? name,
    String? country,
    String? email,
    String? password,
    String? confirmPassword,
  });
  Future<bool?> uploadAvatar(String accessToken, File file);

  Future<String?> resetPassword({required String email});

  Future<AuthEntity?> logout();
  Future<AuthEntity?> getSavedSession();
  Future<AuthEntity?> refresh(AuthEntity? authEntity);
}
```

Fig. 9. *AuthRepository* Class

The code in Figure 9 is an abstract class, *AuthRepository*. This class has a function as a foundation for implementing the repository class on the data layer. It can be seen that this class has several methods related to user account authentication, as well as several methods related to user profiles. The *AuthRepository* class will be implemented in the *AuthRepositoryImpl* class in the next Figure.

b) Data Layer

The data layer handles communication from the application to information outside the application environment. This communication consists of 2 types, namely local communication related to storage and databases, and remote communication related to communication between the application and the API server. At this stage, a repository class is created that is responsible for exposing data originating from external data to the presentation layer, by implementing the *AuthRepository* abstract class that has been created in the data layer.

```
final authRepositoryProvider = Provider((ref) => AuthRepositoryImpl(
    local: ref.read(authLocalSourceProvider),
    remote: AuthRemoteSource(ref.read(authDioServiceProvider)))); //

You, 5 months ago | 1 author (You)
class AuthRepositoryImpl implements AuthRepository {
  final AuthLocalSource local;
  final AuthRemoteSource remote;
  static const NAME = "AuthRepository";

  AuthRepositoryImpl({required this.local, required this.remote});

  @override
  Future<AuthEntity?> logout() async {
    await local.clearSession();
    return null;
  }
```

Fig. 10. *AuthRepositoryImpl* Class

Figure 10 shows the implementation from the *AuthRepository* class that comes from the domain layer. It can be seen that each existing function is implemented according to its function. For example, in the logout method, it can be seen that the author implements how the logout function works, starting by deleting authentication information stored in local storage, and returning a null value.

c) Presentation Layer

The presentation layer acts as an intermediary between the data layer and the application interface. This layer is responsible for managing the application state, such as variables displayed on the interface, and storing code related to the display, including the view model that connects the interface to the repository in the data layer. At this stage, only the process of integrating the repository class that has been created in the data layer into the view model developed by the development team is required.

```
class AuthViewModel extends AsyncNotifier<AuthEntity?> {
  // ignore: constant_identifier_names
  static const NAME = "AuthRiverpod";
  AuthRepository get repository => ref.read(authRepositoryProvider);

  FutureOr<void> logOut() async {
    if (state.hasValue && state.value?.refreshToken != null) {
      state = const AsyncLoading();
      developer.log("$NAME : Logout emitted");
      state = await AsyncValue.guard(() async => await repository.logout());
      developer.log("AuthEntity: ${state.asData.toString()}");
    }
  }
```

Fig. 11. View Model Integration with *AuthRepository*

The image at Figure 11 is the result of integration between the view model class related to account authentication with the repository that was created previously. It can be seen that this class calls the *AuthRepositoryImpl* class with the definition of the *AuthRepository* data type that comes from the domain layer by utilizing the Dependency Injection feature. The author only needs to call the *logOut()* method in the *AuthRepository* class without having to know how the *logOut()* method works. This successfully demonstrates the Dependency Rule, as the Presentation Layer is fully decoupled from the implementation details of the Data Layer, depending only on the abstraction provided by the Domain Layer.

C. Testing & Validation

The testing process uses the black box testing method by implementing the User Acceptance method, consisting of application workflow instructions. The testing process works by monitoring users as they run the application, following the instructions on the form provided. If the tester successfully runs the features on the application, it is declared successful on the form.

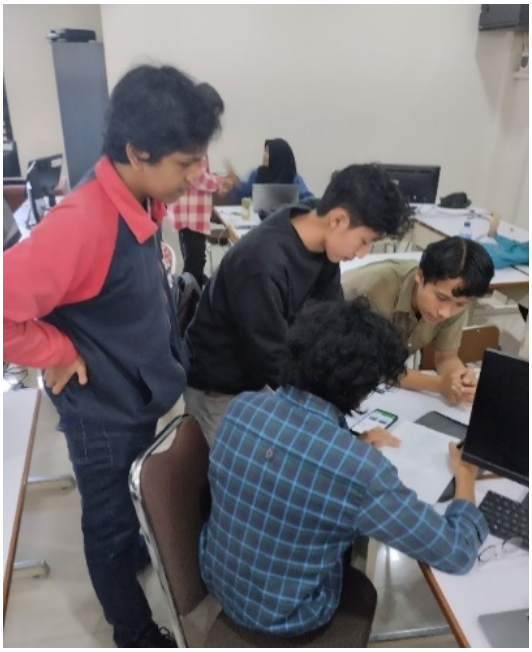

Fig. 12. Conditions for implementing User Acceptance Testing activities

In this testing process, around 5 people at Mataram University were selected to be part of the volunteer testers of this test. This is because in the testing process, the condition of the main Rinjani Visitor server is still in the development stage, so it is necessary to limit the number of people who volunteer for this test to prevent server overload.

TABLE II. USER ACCEPTANCE TESTING RESULT

No.	Test Case	Expected Result	Result
1.	Splash Screen	The splash screen is visible when first opening the app	Success
2.	Register	Users can create an account and get an email confirmation to activate an account	Success
3.	Login	Log in to the app using your registered email and password	Success
4.	Forgot Password	A new password will be sent to the user's email and they can log in with it	Success
5.	Profile page	The profile page will display the profile picture and account setting	Success
6.	Update profile picture	Users can upload a new profile picture and see it	Success
7.	Update profile	Users can change their name, phone number, and country	Success
8.	Change Password	Users can update their password	Success
9.	Home page	Users can see the home page, search products by title	Success
10.	Detail product	A product card can be clicked to see and display detailed information	Success
11.	Favorite	User can add a product to their wishlist by clicking the heart icon and remove it by clicking again	Success
12.	Wishlist	Users can see their favorite products through the wishlist page	Success
13.	Booking Product	When the product details, the user can set start-date, total person, add, and offering price. An email with booking information will be sent to the admin	Success
14.	See All Booking History	User can see all their booking history	Success
15.	Booking Email Confirmation	Users can receive an email confirmation after booking	Success
16.	Choose method payment	Users can choose the method of payment	Success
17.	Payment via Wise	Users can pay with Wise	Success
18.	Payment via Bank	Users can pay with a Bank	Success
19.	Payment Email Confirmation	Users can receive email confirmation after payment	Success
20.	Order History	Users can see the list of order history	Success
21.	Create Review	Users can write review	Success

Based on the test results above, it can be seen that the application can run well. This can be seen from the information data in the form of test results that have a success value. This successful validation of all 21 functional requirements confirms that the implementation of Clean Architecture effectively separated concerns and improved the codebase's structure without negatively impacting the application's stability or the end-user experience.

V. CONCLUSIONS

The implementation of Clean Architecture in the Rinjani Visitor application demonstrates its effectiveness in decoupling business logic from technical implementation details, by enhancing modularity and simplifying both development and maintenance processes. By isolating core components, Clean Architecture enables developers to work on individual features without compromising the overall stability of the application, fostering a more agile and scalable development environment. However, to sustain these benefits, it is recommended that the development team periodically review the adopted architectural structure to evaluate its efficacy in supporting long-term maintenance and future feature expansions. Additionally, iterative improvements should be made to the architecture to align it with evolving team requirements and technological advancements. This proactive approach will ensure the architecture remains robust, adaptable, and conducive to the collaborative growth of the application.

REFERENCES

[1] M. T. Sembiring and Carine, "Peninjauan Prospek Smartphone Global: Studi Pustaka," *TALENTA Conference Series: Energy & Engineering*, vol. 2, no. 3, 2019.

[2] J. Yoo, S. Choi, Y. Hwang, and M. Y. Yi, "The role of user resistance and social influences on the adoption of smartphone: Moderating effect of age," *Journal of Organizational and End User Computing*, vol. 33, no. 2, 2021, doi: 10.4018/JOEUC.20210301.oa3.

[3] D. Sugandini, A. Aprilivianto, and B. Y. Darasta, "Adopsi Aplikasi Berbasis Android," *Eksos LPPM*, vol. 2, no. 2, 2020.

[4] P. Noviati Sadikin, S. Mulatsih, B. Pramudya Noorachmat, and H. Susilo Arifin, "Analisis Willingness-To-Pay Pada Ekowisata Taman Nasional Gunung Rinjani," *Jurnal Analisis Kebijakan Kehutanan*, vol. 14, no. 1, 2017, doi: 10.20886/jakk.2017.14.1.31-46.

[5] T. M. Puspita, "Pesona keindahan Alam Taman Nasional Gunung Rinjani Lombok," *Atmosfer: Jurnal Pendidikan, Bahasa, Sastra, Seni, Budaya, dan Sosial Humaniora*, vol. 1, no. 2, 2023.

[6] B. H. S. Purwana, "Potensi Ekowisata Berbasis Budaya Masyarakat Di Desa Sanaru, Kabupaten Lombok Utara," *Kebudayaan*, vol. 13, no. 2, 2019, doi: 10.24832/jk.v13i2.199.

[7] A. Rosadi, I. G. A. Suwartane, S. Budilaksono, F. Nurzaman, E. P. Dewi, and F. Febriyanti, "Penerapan model kegiatan pembelajaran magang MBKM pada program matching fund kedaireka," *Jurnal Edukasi dan Multimedia*, vol. 1, no. 1, 2023.

[8] Google, "Flutter - Build apps for any screen." Accessed: Jan. 02, 2024. [Online]. Available: https://flutter.dev/

[9] J. Maylia Suhendro, M. Sudarma, and D. Care Khrisne, "Rancang Bangun Aplikasi Seluler Penyedia Jasa Perawatan dan Kecantikan Menggunakan Framework Flutter," *Jurnal Spektrum*, vol. 8, no. 2, 2021, doi: 10.24843/spektrum.2021.v08.i02.p9.

[10] F. Enggar Krisnada and R. Tanone, "Aplikasi Penjualan Tiket Kelas Pelatihan Berbasis Mobile menggunakan Flutter," *Jurnal Teknik Informatika dan Sistem Informasi*, vol. 5, no. 3, 2020, doi: 10.28932/jutisi.v5i3.1865.

[11] R. C. Martin, *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. 2017.

[12] A. C. Beltrão, F. D. A. Farzat, and G. H. Travassos, "Technical Debt: A Clean Architecture Implementation," 2021. doi: 10.5753/cbsoft_estendido.2020.14620.

[13] D. Sanchez, A. E. Rojas, and H. Florez, "Towards a Clean Architecture for Android Apps using Model Transformations," *IAENG Int J Comput Sci*, vol. 49, no. 1, 2022.

[14] A. R. Fajri, "Penerapan Design Pattern MVVM dan Clean Architecture pada Pengembangan Aplikasi Android (Studi Kasus: Aplikasi Agree)," *Official Scientific Journals of Universitas Islam Indonesia*, 2022.

[15] Moh. H. Badrudduja and R. E. Putra, "Penerapan Clean Architecture pada Aplikasi Pemesanan Makanan menggunakan Metode Slope One Algorithm," *Journal of Informatics and Computer Science (JINACS)*, vol. 3, no. 04, 2022, doi: 10.26740/jinacs.v3n04.p506-514.

[16] J. Wiratama, H. Santoso, and Clairence, "Developing a Class Scheduling Mobile Application for Private Campus in Tangerang with the Extreme Programming (XP) Model," *G-Tech: Jurnal Teknologi Terapan*, vol. 7, no. 2, 2023, doi: 10.33379/gtech.v7i2.2288.

[17] S. Al-Saqqa, S. Sawalha, and H. Abdelnabi, "Agile software development: Methodologies and trends," *International Journal of Interactive Mobile Technologies*, vol. 14, no. 11, 2020, doi: 10.3991/ijim.v14i11.13269.

[18] M. Yasvi, K. Yadav, and Sahendrasingh. S., "Review On Extreme Programming-XP," *International Conference on Robotics, Smart Technology and Electronics Engineering, At Delhi*, no. April, 2019.

[19] L. Setiyani, "Desain Sistem : Use Case Diagram," *Prosiding Seminar Nasional : Inovasi & Adopsi Teknologi 2021*, no. September, 2021.

[20] M. Syarif and E. B. Pratama, "Analisis Metode Pengujian Perangkat Lunak Blackbox Testing Dan Pemodelan Diagram Uml Pada Aplikasi Veterinary Services Yang Dikembangkan Dengan Model Waterfall," *Jurnal Teknik Informatika Kaputama (JTIK)*, vol. 5, no. 2, 2021.

[21] Umiringsih, M. Nur Ichsanudin, M. Yusuf, and S. Suraya, "Pengujian Fungsional Perangkat Lunak Sistem Informasi Perpustakaan Dengan Metode Black Box Testing Bagi Pemula," *Storage: Jurnal Ilmiah Teknik dan Ilmu Komputer*, vol. 1, no. 2, 2022, doi: 10.55123/storage.v1i2.270.

[22] S. Gordon *et al.*, "Best Practice Recommendations: User Acceptance Testing for Systems Designed to Collect Clinical Outcome Assessment Data Electronically," *Ther Innov Regul Sci*, vol. 56, no. 3, 2022, doi: 10.1007/s43441-021-00363-z.