UDC 004.777:640.433.045

B.K. Berdaliyev*, P.A. Kozhabekova, A.T. Kalbayeva
Master student, M. Auezov South Kazakhstan University, Shymkent, Kazakhstan
Cand.Tech.Sci., Associate Professor, M. Auezov South Kazakhstan University, Shymkent, Kazakhstan
Cand.Tech.Sci., Associate Professor, M. Auezov South Kazakhstan University, Shymkent, Kazakhstan
***Corresponding Author's Email: beiybarys.berdaliyev@gmail.com**

BEST PRACTICE ARCHITECTURE AND STATE MANAGEMENT IN FLUTTER APPLICATIONS

Abstract

In the rapidly evolving landscape of mobile application development, Flutter has emerged as a robust framework for creating high-performance, cross-platform apps. This article explores best practices in architecture and state management within Flutter applications, with a particular focus on developing educational tools for Kazakh children to learn about their cultural heritage. We delve into architectural patterns that enhance code maintainability and scalability, such as MVVM and Clean Architecture. Additionally, we examine various state management solutions, including Provider, Bloc and Riverpod, comparing their strengths and use cases. The study highlights the collaborative efforts of the development team, demonstrating how effective teamwork and role allocation can lead to the successful implementation of a portable and user-friendly application. By eliminating the existing gap in educational resources adapted for children, this Flutter-based application offers a free and accessible platform for learning Kazakh traditions.

Keywords: Flutter, clean architecture, Bloc, Riverpod, state management, user experience, cross-platform.

Introduction

In today's digital age, mobile applications have become an integral part of our daily lives, providing solutions and services across various domains. Flutter, a UI toolkit developed by Google, has rapidly gained popularity among developers for its ability to create high-performance, visually appealing, and natively compiled applications for mobile, web, and desktop from a single codebase. This versatility makes Flutter an ideal choice for developing cross-platform applications efficiently.

This article focuses on the best practices in architecture and state management for Flutter applications, using the example of an educational app designed to teach Kazakh traditions to children. The app is intended for both Android and iOS users, addressing a notable gap in the availability of cultural and educational resources for Kazakh kids. It is crucial to build such applications with a solid architectural foundation and effective state management to ensure maintainability, scalability, and a smooth user experience.

We will explore various architectural patterns, such as Model-View-ViewModel (MVVM) and Clean Architecture, which help in organizing code and improving its readability and testability. Additionally, we will delve into state management solutions, including Provider, Bloc, and Riverpod, comparing their features and appropriate use cases. Effective state management is essential for handling the dynamic nature of mobile applications and ensuring a responsive and reliable user interface.

The development of this educational app also highlights the importance of collaboration within a development team. Efficient teamwork and clear role distribution contribute significantly to the successful implementation of complex projects. By examining the current systems in use and their integration with the app, we demonstrate how this Flutter-based application offers a free, accessible, and portable solution for learning about Kazakh traditions.

In this article, we aim to provide insights and guidelines for developers looking to adopt best practices in Flutter application development. The principles and methodologies discussed here are

not only applicable to educational apps but can also be generalized to a wide range of mobile applications, contributing to the advancement of the Flutter development ecosystem [1].

Problem Statement

The development of mobile applications that are both functional and maintainable poses significant challenges, particularly in the realm of cross-platform development. Flutter, with its ability to deliver high-performance apps from a single codebase for both Android and iOS, offers a compelling solution. However, the success of Flutter applications heavily relies on the adoption of best practices in architecture and state management. Poor architectural decisions can lead to unmanageable codebases, difficult maintenance, and scalability issues, while ineffective state management can result in inefficient UI updates, poor performance, and a suboptimal user experience.

In the context of educational applications aimed at preserving and teaching cultural heritage, such as an app designed to teach Kazakh traditions to children, these challenges are compounded. There is a notable scarcity of quality educational tools tailored to Kazakh kids that are both free and accessible. Existing applications often require memberships or are not designed with the unique educational needs of children in mind.

This study addresses the need for a robust architectural framework and efficient state management strategies in developing a culturally enriching, user-friendly, and maintainable educational application using Flutter. By focusing on best practices, this article aims to guide developers in creating applications that are not only effective in teaching but also sustainable in the long term.

The problem is twofold:

The lack of quality, accessible educational applications for Kazakh children that effectively teach cultural traditions.

The technical challenges in implementing best practices for architecture and state management in Flutter to ensure these applications are maintainable, scalable, and provide a seamless user experience.

Addressing these problems will help bridge the gap in educational resources available to Kazakh children and provide a blueprint for developers to follow in creating high-quality Flutter applications [2].

Importance of Architecture in Flutter Applications

A well-structured architecture serves as the backbone of any successful software project, providing a roadmap for developers to follow and ensuring that the codebase remains maintainable, scalable, and adaptable to changing requirements. In the context of Flutter, where applications can quickly grow in complexity, having a clear architectural design becomes paramount [3].

Why Architecture Matters:

- Modularity and Maintainability: A well-defined architecture breaks down the application into smaller, manageable components, each with its specific responsibility. This modularity makes it easier to understand, maintain, and extend the codebase over time.
- Scalability: As applications evolve and grow, a well-thought-out architecture provides a solid foundation for scaling the application. It allows for new features to be added seamlessly without causing disruption to existing functionality.
- Code Reusability: An architecture that promotes code reuse enables developers to leverage existing components across different parts of the application, reducing development time and minimizing code duplication.
- Testability: By separating concerns and dependencies, a good architecture facilitates unit testing, making it easier to verify the behavior of individual components in isolation and ensure the overall correctness of the application.

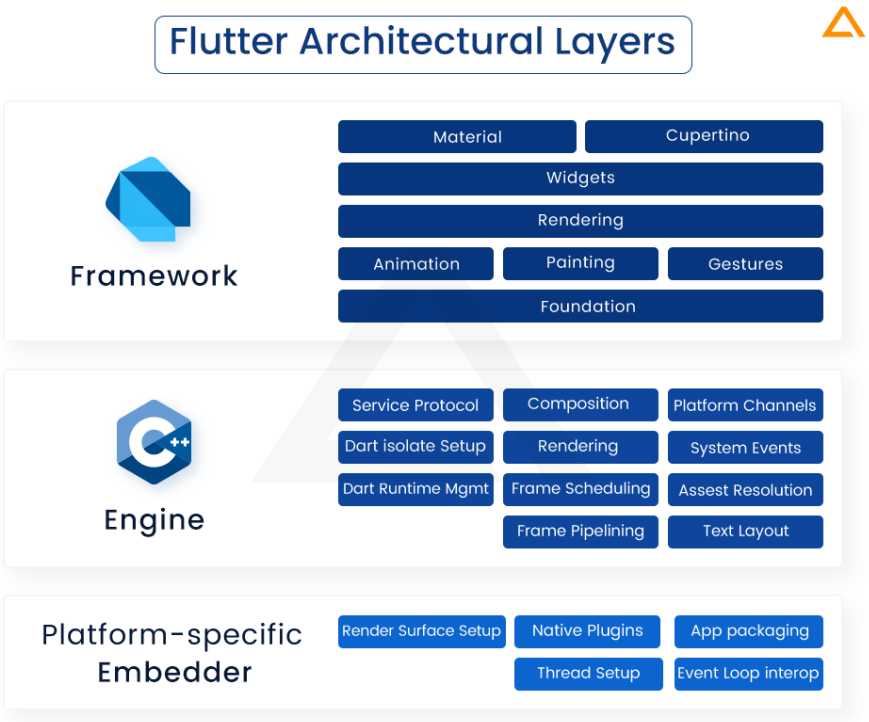

Figure 1. Basics of Flutter Architecture [4].

Architectural Patterns in Flutter: MVVM and Clean Architecture

Two widely adopted architectural patterns in Flutter are the Model-View-ViewModel (MVVM) and Clean Architecture.

MVVM (Model-View-ViewModel): MVVM separates an application into three main components: the Model, View, and ViewModel. The Model represents the data and business logic, the View represents the UI, and the ViewModel acts as an intermediary between the two, handling the presentation logic. This separation of concerns promotes code maintainability, testability, and scalability [5].

In Flutter, implementing MVVM involves creating Dart classes for each component:

- Model: Represents the data and business logic.
- View: Displays the UI and captures user interactions.
- ViewModel: Acts as a mediator between the Model and the View, handling the presentation logic and data transformation.

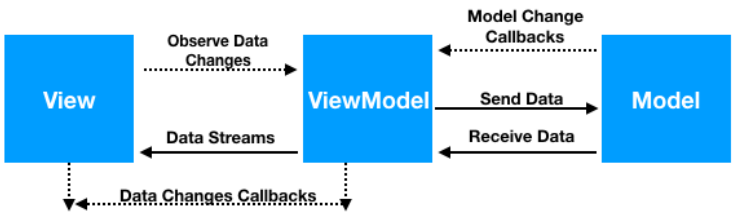

Figure 2. MVVM.

Clean Architecture: Clean Architecture, popularized by Robert C. Martin, emphasizes separation of concerns and dependency inversion. It divides an application into layers, with the innermost layer containing the business logic (use cases), surrounded by layers representing the

interface adapters and frameworks. Clean Architecture fosters independence from external frameworks, making applications easier to test and maintain.

In Flutter, Clean Architecture can be implemented by organizing code into the following layers:

- Entities: Represent the core data structures of the application.
- Use Cases: Contain the business logic and application-specific rules.
- Frameworks and Drivers: Interface adapters that connect the application to external frameworks and platforms [6].

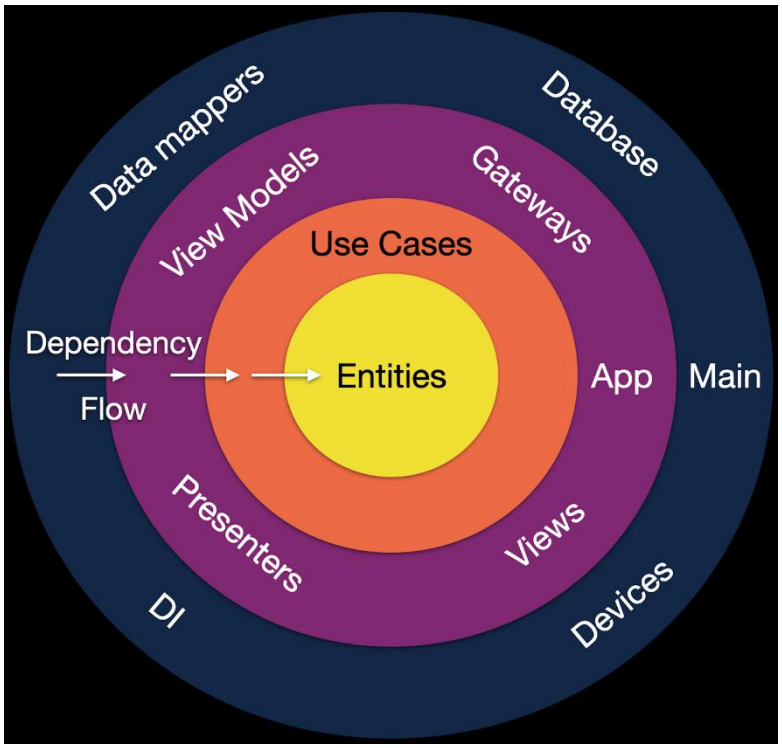

Figure 3. Clean Architecture.

State Management in Flutter

State management is a critical aspect of Flutter development, as it dictates how data is managed and shared across different parts of an application. Flutter offers various state management solutions, each catering to different use cases and preferences.

1. Provider

Provider is a lightweight and easy-to-use state management solution that leverages Flutter's inherited widget mechanism. It promotes a clear separation of concerns and is ideal for small to medium-sized applications [7].

Advantages: Simplicity, ease of use, and integration with Flutter's widget tree.

Use Cases: Ideal for applications with moderate state management requirements.

2. Bloc (Business Logic Component)

Bloc is a more complex but powerful state management solution that uses the reactive programming model. It helps separate business logic from the UI and encourages the use of streams [8].

Advantages: Scalability, testability, and a clear separation of concerns.

Use Cases: Suitable for larger applications with complex state management needs.

3. Riverpod

Riverpod builds on top of Provider but offers several improvements, such as better support for code reuse and testing. It provides a more robust and flexible way to manage state.

Advantages: Enhanced testability, flexibility, and a more intuitive API.

Use Cases: Can be used for both simple and complex applications, providing a scalable solution [9].

Best Practices in Architecture and State Management

Implementing best practices in architecture and state management is crucial for developing maintainable and scalable Flutter applications. Here are some key principles to follow:

1. Separation of Concerns: Ensure that your code is modular and that each component has a single responsibility. This makes the codebase easier to manage and test.
2. Use Dependency Injection: Implement dependency injection to decouple classes and promote code reuse. This can be achieved using packages like get_it or injectable.
3. Adopt a Consistent State Management Solution: Choose a state management solution that fits the needs of your application and stick with it throughout the project. Consistency helps maintain a clear and understandable codebase.
4. Leverage Flutter's Composition Model: Flutter's widget composition model allows you to build complex UIs from simple widgets. Use this to your advantage by breaking down your UI into smaller, reusable components.
5. Implement Unit and Integration Testing: Ensure that your application is well-tested. Write unit tests for your business logic and integration tests for your state management and UI components.
6. Use Linting and Code Analysis Tools: Employ tools like flutter_lints to maintain code quality and consistency across your project.

Conclusion

Implementing best practices in architecture and state management is essential for building robust, maintainable, and scalable Flutter applications. By adopting patterns like MVVM or Clean Architecture and choosing the right state management solution, developers can ensure their applications are well-structured and efficient. These practices not only improve the development process but also enhance the end-user experience, making applications more reliable and enjoyable to use.

In conclusion, the principles and guidelines discussed in this article provide a strong foundation for developers looking to harness the full potential of Flutter. Whether you're building a simple app or a complex, feature-rich application, following these best practices will help you achieve success in your Flutter development journey.

References

1. GSM Association, the Mobile Society Research Institute within NTT DOCOMO. Children's use of mobile phones. An international comparison 2011, pp. 6-25
2. Rideout V,J., Foehr U.G., Roberts D.F. Generation M2: Media in the Lives of 8-to 18-Year-Olds. Kaiser Family Foundation, 2010, 6p.
3. Flutter Documentation. (n.d.). Flutter Architecture Guide. Available at: https://flutter.dev/docs/development/data-and-backend/architecture
4. Saurabh Barot. Flutter best practices to follow in 2024. Available at https://aglowiditsolutions.com/blog/flutter-best-practices/
5. Microsoft. (n.d.). Model-View-ViewModel (MVVM) pattern. Available at: https://docs.microsoft.com/en-us/xamarin/xamarin-forms/enterprise-application-patterns/mvvm
6. Martin R.C. Clean Architecture. A Craftsman's Guide to Software Structure and Design. New York, 2017, 39p.
7. Flutter Team. (n.d.). Provider package. Available at: https://pub.dev/packages/provider
8. Flutter Community. Flutter Bloc package. Available at: https://pub.dev/packages/flutter_bloc
9. Riverpod documentation. (n.d.). Available at: https://riverpod.dev/

Б.Қ. Бердалиев*, П.А. Қожабекова, А.Т. Қалбаева
магистрант, М. Әуезов атындағы Оңтүстік Қазақстан университеті, Шымкент, Қазақстан
т.ғ.к., доцент, М. Әуезов Атындағы Оңтүстік Қазақстан университеті, Шымкент, Қазақстан
т.ғ.к., доцент, М. Әуезов Атындағы Оңтүстік Қазақстан университеті, Шымкент, Қазақстан
***Корреспондент авторы:** beiybarys.berdaliyev@gmail.com

ҮЗДІК ТӘЖІРИБЕ СӘУЛЕТ ЖӘНЕ МЕМЛЕКЕТТІК БАСҚАРУ FLUTTER ҚОСЫМШАЛАРЫНДА

Түйін

Мобильді қосымшаларды әзірлеудің қарқынды дамып келе жатқан ландшафтында Flutter өнімділігі жоғары кросс-платформалық қосымшаларды құрудың сенімді негізіне айналды. Бұл мақалада Қазақ балаларының мәдени мұралары туралы білуі үшін білім беру құралдарын әзірлеуге ерекше назар аудара отырып, Flutter қолданбалары аясында сәулет және мемлекеттік басқару саласындағы озық тәжірибелер қарастырылады. БІЗ mvvm Және Clean Architecture Сияқты кодтың сақталуы мен масштабталуын жақсартатын архитектуралық үлгілерді зерттейміз. Сонымен қатар, Біз Мемлекеттік басқарудың әртүрлі шешімдерін, соның ішінде Жеткізушілерді, Блоктарды және Riverpod-ты олардың күшті жақтары мен пайдалану жағдайларын салыстыра отырып қарастырамыз. Зерттеу әзірлеушілер тобының бірлескен күш-жігерін көрсетеді, бұл топтық жұмыс пен рөлдерді бөлудің портативті және пайдаланушыға ыңғайлы қолданбаны сәтті енгізуге қаншалықты тиімді әкелетінін көрсетеді. Балаларға бейімделген білім беру ресурстарындағы бар олқылықты жою арқылы Бұл Flutter негізіндегі қолданба қазақ дәстүрлерін үйренуге арналған тегін және қолжетімді платформаны ұсынады.

Кілттік сөздер: Flutter, таза сәулет, Блок, Riverpod, мемлекеттік басқару, пайдаланушы тәжірибесі, кросс-платформа.

Б.К. Бердалиев*, П.А. Кожабекова, А.Т. Калбаева
магистрант, Южно-Казахстанский университет им. М. Ауэзова, Шымкент, Казахстан
к.т.н., доцент, Южно-Казахстанский университет им. М. Ауэзова, Шымкент, Казахстан
к.т.н., доцент, Южно-Казахстанский университет им. М. Ауэзова, Шымкент, Казахстан
***Автор для корреспонденции:** beiybarys.berdaliyev@gmail.com

ЛУЧШАЯ ПРАКТИКА АРХИТЕКТУРА И ГОСУДАРСТВЕННОЕ УПРАВЛЕНИЕ В ПРИЛОЖЕНИЯХ FLUTTER

Аннотация

В быстро меняющемся мире разработки мобильных приложений Flutter стал надежной платформой для создания высокопроизводительных кроссплатформенных приложений. В этой статье рассматриваются лучшие практики в области архитектуры и государственного управления в приложениях Flutter, с особым акцентом на разработку образовательных инструментов, позволяющих казахстанским детям узнать о своем культурном наследии. Мы исследуем архитектурные шаблоны, которые улучшают удобство сопровождения и масштабируемость кода, такие как MVVM и чистая архитектура. Кроме того, мы изучаем различные решения для управления состоянием, включая Provider, Block и Riverpod, сравнивая их сильные стороны и варианты использования. В исследовании освещаются совместные усилия команды разработчиков, демонстрирующие, как эффективная командная работа и распределение ролей могут привести к успешной реализации портативного и удобного в использовании приложения. Устраняя существующий пробел в образовательных ресурсах, адаптированных для детей, это приложение на основе Flutter предлагает бесплатную и доступную платформу для изучения казахских традиций.

Ключевые слова: Flutter, чистая архитектура, Block, Riverpod, управление состоянием, пользовательский опыт, кроссплатформенность.