ИНФОРМАТИКА И ИНФОРМАЦИОННЫЕ ПРОЦЕССЫ

УДК 004.42 Научная статья
DOI: 10.35330/1991-6639-2024-26-5-84-93
EDN: LEFHZL

Выбор архитектуры для мобильных приложений

А. К. Маринин

Московский финансово-промышленный университет «Синергия»
129090, Россия, Москва, ул. Мещанская, 9/14, стр. 1

Аннотация. Цель настоящей статьи заключается в анализе применимости шаблонов MVC, MVP и MVVM, VIPER и CLEAN architecture для мобильной разработки на ОС Android и iOS с выявлением подходящей программной архитектуры, чтобы разрабатывать мобильные приложения на базе указанных платформ, используя такие атрибуты, как уровень тестируемости, сопряженности и возможности внесения изменений и исправлений. С точки зрения методологии исследование строится на методах синтеза, обобщения данных, которые получены при сравнении моделей, чтобы повысить эффективность разработки. Эти архитектуры – MVC, MVP и MVVM, VIPER и CLEAN architecture. Критерии, которые используются при сопоставительном анализе, связаны с тестируемостью, сопряженностью, способностью к изменениям и исправлениям и возможностями неоднократного применения. Специфика архитектуры важна для разработки мобильного приложения. Проведенный сравнительный анализ приводит к выводу, что возможности архитектуры MVVM – самый подходящий вариант, чтобы разрабатывать мобильные приложения Android. После проведения анализа с учетом перечисленных критериев можно остановиться на конкретной архитектуре. Каждый шаблон располагает разными свойствами, поэтому выбор комбинации MVVM с Clean Architecture является оптимальным, потому что она оказывает поддержку всем атрибутам, а с помощью шаблона Clean Architecture решаемы многие сложности, которые могут возникать при применении исключительно MVVM.

Ключевые слова: программное обеспечение, мобильная разработка, шаблон архитектуры, MVC, MVP, MVVM, VIPER, CLEAN architecture, тестируемость, пользовательский интерфейс

Поступила 11.09.2024, одобрена после рецензирования 19.09.2024, принята к публикации 04.10.2024

Для цитирования. Маринин А. К. Выбор архитектуры для мобильных приложений // Известия Кабардино-Балкарского научного центра РАН. 2024. Т. 26. № 5. С. 84–93. DOI: 10.35330/1991-6639-2024-26-5-84-93

MSC: 68 Original article

Choosing an architecture for mobile applications

A.K. Marinin

Moscow University for Industry and Finance "Synergy"
129090, Russia, Moscow, 9/14, building 1, Meshchanskaya street

Abstract. The purpose of the article is to analyze the applicability of the MVC, MVP and MVVM, VIPER and CLEAN architecture templates in mobile development for iOS and Android OS and identify the most suitable software architecture for developing mobile applications on these platforms based on attributes such as testability, connectivity and maintainability. The research methodology is based on methods of analysis, synthesis, and generalization of data obtained by comparing various architectural

models for mobile development. These architectures are MVC, MVP and MVVM, VIPER and CLEAN architecture. The criteria for comparative analysis are testability, maintainability, conjugacy, and reusability. Architecture plays an important role in the development of mobile applications. As a result of the comparative analysis of the templates, we conclude that the MVVM architecture is a suitable option for developing mobile applications on Android. We conducted an analysis based on criteria such as testability, maintainability and coupling. All architecture templates support different properties, but we chose a combination of MVVM with Clean Architecture because it supports all attributes, and the Clean Architecture template solves the problems that arise when using MVVM alone.

Keywords: software, mobile development, architecture template, MVC, MVP, MVVM, VIPER, CLEAN architecture, testability, user interface

Submitted 11.09.2024, approved after reviewing 19.09.2024, accepted for publication 04.10.2024

For citation. Marinin A.K. Choosing an architecture for mobile applications. *News of the Kabardino-Balkarian Scientific Center of RAS.* 2024. Vol. 26. No. 5. Pp. 84–93. DOI: 10.35330/1991-6639-2024-26-5-84-93

Введение

Рынок мобильных приложений является одним из самых быстрорастущих, в особенности это касается Android и iOS. Совокупно обе системы занимают на рынке 98,5 %. При этом первая из названных занимает 70,4 % на рынке, о чем свидетельствуют статистические данные [1]. Архитектурные принципы ПО приобретают все большую значимость. Это естественный этап в развитии методологии при разработке ПО в ИТ-сфере. Тем важнее понимать, какие потребности и ожидания стоят за той скоростью, с которой внедряются программные архитектуры в сферу мобильной разработки, какова природа у распространенных сегодня вариантов архитектуры, оценить, насколько усовершенствовалось их применение, продвижение на рынке ПО данной направленности.

Под архитектурой ПО следует понимать разновидность модели структуры ПО и выбор оптимально работающего ПО. В связи с чем появляется возможность повторно использовать модули для иного ПО. Все структуры имеют определенные элементы ПО с установленными взаимосвязями. Ее оценка необходима, чтобы принимать важные структурные решения, в противном случае внесение изменений становится дорогостоящим. Архитектура располагает определенными параметрами. Речь идет о структурных параметрах, которые значимы для разрабатываемого ПО. Если архитектура отсутствует, то, как показывает практика, сопровождение и доработка невозможны, что вызывает перерасход средств в процессе разработки.

На современном этапе указанные проблемы и вопросы рассматриваются на разном уровне, включая блоги, авторитетнее всего именно научные, экспертные. Архитектура, обсуждаемая в публикациях [2–6], обычно представляет собой частные случаи, возникающие при разработке мобильных приложений. В частности, Ambani рассматривает архитектуру MVC, использование шаблона, его преимущества, недостатки, детали и функции, используемые для разработки мобильных приложений [2]. Vennaro в своей работе [3] рассматривает MVC как базовую парадигму архитектуры мобильных приложений, однако указывает на то, что для масштабирования проектов и расширения функционала разрабатываемых мобильных приложений данной архитектуры недостаточно. Автор подчеркивает, что выбор более сложных архитектур является ключом к разработке более модульных, тестируемых и более приспособленных к масштабированию приложений. По мнению Epiloksa, Kusumo и Adrian, выбор архитектуры является основополагающим фактором,

обеспечивающим высокую производительность приложения, при этом в случае с ОС Android тестирование, проведенное авторами, показало большую эффективность при использовании архитектуры MVC в сравнении с MVVM [4]. В публикациях [7–10] рассматриваются шаблоны высокого уровня. Так, в работе Sokolova K. et al. [7] исследуются широко используемые шаблоны архитектуры для проектирования мобильных приложений, в результате чего авторы предлагают унифицированную архитектурную модель, адаптированную для разработки на ОС Android. М. П. Василевским также рассматривается проблема разграничения между архитектурой и дизайном в контексте разработки пользовательского интерфейса (UI) мобильных приложений [10]. Соммервилом [11] транслируется идея, что большой разницы между мобильными и веб-приложениями не наблюдается, только изредка есть необходимость специализированного подхода к первым из названных. Добреан и Диосан [12] известны сравнительным исследованием MVC, MVP, MVVM и VIPER (архитектурные решения для iOS в [13]). Салазар и Брамбилла [14] рассматривают высокоуровневый процесс, который важен при проектировании архитектурных решений для ПО. Статья [15] посвящена архитектуре VIPER для IOS, которая может выступать в качестве аналога предложенной Apple архитектуре MVC. Последняя, например, не всегда представляет собой наилучшее решение, несмотря на то, что считается простой. Со временем неизбежно расширение, а это значит, что код увеличится и усложнится. Статья [16] рассматривает архитектуру через призму скрипта Service Worker.

Итак, при оптимальности архитектуры ПО процесс тестирования приложения становится простым, инструменты могут использоваться неоднократно. Не имеется образца, эталона, по которому можно воссоздавать архитектуру в приложениях Android или iOS. Абстрактность шаблона при проектировании подразумевает, что разработка для каждого модуля происходит с учетом конкретных задач и ожиданий. При этом разработчиками используется ряд апробированных шаблонов, чтобы проектировать интерактивные приложения: MVC, MVP и MVVM, VIPER и CLEAN architecture. Они предоставляют возможности для структуры приложения, в результате решение упрощается, а процесс тестирования становится управляемым.

Методология

Целью статьи является анализ, насколько применимы шаблоны MVC, MVP и MVVM, VIPER и CLEAN architecture в рамках мобильной разработки для ОС Android и iOS, чтобы впоследствии установить, какая программная архитектура оптимальна, с учетом тестируемости, сопряженности и возможности внесения изменений и исправлений.

Методология в исследовании строится на методах анализа, синтеза, обобщения данных, которые получены при сравнении различных архитектурных моделей, чтобы разрабатывать мобильные приложения. Это такие архитектуры, как MVC, MVP и MVVM, VIPER и CLEAN architecture. Критерии, чтобы проводить сопоставительный анализ, связаны с такими параметрами, как

- уровень тестируемости, что подразумевает способность кода быть протестированным. Архитектуры, которые обеспечивают четкое разделение логики представления и бизнес-логики, облегчают тестирование кода;
- сопряженность определяется как степень взаимосвязи между компонентами системы, архитектуры с низкой сопряженностью позволяют изменять отдельные компоненты без влияния на другие части системы;
- способность системы к изменениям или исправлениям;
- способность использоваться повторно.

Результаты

Шаблон «Модель-представление-контроллер» (MVC). Способствует разделению контроллера так, что представление и активность непосредственно связываются между собой, в обход иных обязанностей, какими располагает контроллер.

Его компоненты:

Модель: уровень данных несет ответственность за то, каким образом управляется бизнес-логика и сетевой API или API базы данных.

Вид: уровень пользовательского интерфейса выражается в визуализации данных из модели.

Контроллер: логический уровень информируется о том, как ведет себя пользователь, и может обновить модель, если это потребуется.

Архитектура MVC отличается полной совместимостью с проблемой разделения задач. Это способствует улучшению тестируемости кода, упрощению его расширения, что облегчает внесение новых функций. Многие разработчики считают ее идеальной, если есть потребность в полном контроле приложений. Шаблон не пользуется серверными форм-факторами, а прибегает к возможностям front controller. Это лучшая тактика, если нужно обработать входящие запросы и пользоваться одним контроллером на сервере. Он также эффективно разделяет модель и внешний вид. Тестирование модели при этом существенно упрощается. Доступно изменение как самой модели, так и вида, при этом влияния контроллера на процесс в целом не наблюдается. Отображаемое на экране – под полной ответственностью контроллера. Если расширяется функционал, наблюдается дальнейший рост файла, при этом деятельность ОС увязывается с работой пользовательского интерфейса, механизмов доступа к данным. Подобные аспекты являются ограничениями для рефакторинга и изменений. Это не означает, что контроллер и логика представления должны оказаться в Activity / Fragment. В противном случае получится слишком сильная связка компонентов. Fragments, Activities и Views – это представления для изучаемого шаблона.

Шаблон «Модель-представление-презентатор» (MVP). Альтернативное решение для моделей MVC с разделением приложения на аспекты (рис. 1).

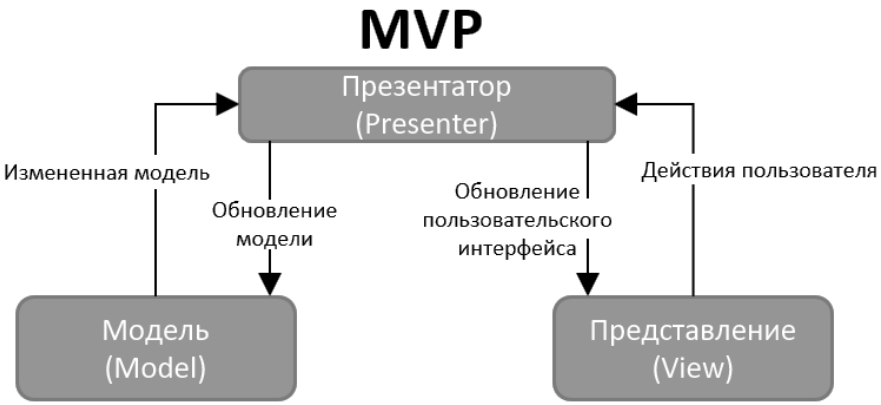

Рис. 1. *Шаблон «Модель-представление-презентатор» (составлено автором)*

Fig. 1. *Model-View-Presenter template (compiled by the author)*

Модель: связка данных и бизнес-логики в приложении, как это происходит в ранее рассмотренных моделях MVC. Отвечает за то, как обрабатываются, хранятся, управляются данные, за то, как реализуются все нужные бизнес-правила. Прямого взаимодействия не наблюдается, модель, представление и презентатор не взаимодействуют.

Представление: связано с пользовательским интерфейсом и уровнем представления. Занимается отображением информации, что транслирует модель, взаимодействует с ней активнее, чем в варианте с MVP, который более пассивен: там презентатор занимается обновлением и обрабатывает ввод.

Презентатор: связывает модель и представление, в известной степени может выступать как контроллер для MVC. Происходит извлечение данных с последующим обновлением представления, которое начинает работать более корректно. У презентатора есть и иные функции, он занимается обработкой ввода, становится мостиком, который соединяет представление и модель.

MVP является моделью, содействует упрощению автоматизированного модульного тестирования. Она также предоставляет чистый код. Разработчикам становится проще работать, любое представление не имеет привязки с источником данных. Никакого последовательного создания представления не потребуется, когда приложение создается на той или иной платформе. То есть представление становится доступнее, что и представляет собой логику и преимущество данной архитектуры. Представление и презентатор однозначно связаны, тем проще заниматься разделением приложения и модулей. В результате возникает двусторонняя связь View и Presenter. Но есть и недостатки MVP, особенно, если меняется логика в пользовательском интерфейсе. У контроллера и модели появляется ссылка от представления. Нет ограничений, чтобы обрабатывать логику интерфейса в одном классе. В результате контроллер и представление отвечают за обработку в равных частях. С помощью архитектурного шаблона MVP обеспечивается простота работы, кроме того, код может быть использован повторно для Android и iOS, что положительно сказывается на тестируемости.

Шаблон «Модель-Представление-Модель представления» (MVVM). Его состав: пользовательский интерфейс (UI), действие и фрагмент, способен взаимодействовать с презентатором. Структура также способствует изоляции логики программы и интерфейса. Ключевые компоненты (рис. 2).

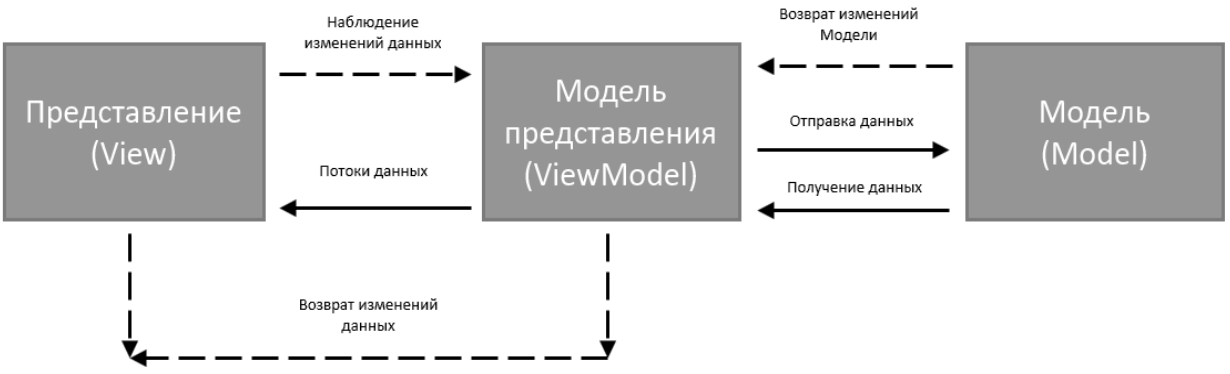

Рис. 2. *Шаблон «Модель-Представление-Модель представления» (MVVM) (составлено автором)*

Fig. 2. *Model-View-ViewModel (MVVM) template (author's compilation)*

Модели, которые содержат информацию о приложении, являются простыми классами, которые не имеют зависимости с иными компонентами. Отображение происходит с помощью визуальных и управляющих элементов на экранах. Зачастую это подклассы UIView.

Информация о модели преобразуется, все данные можно отслеживать через представление, а возникшие в процессе значения связывают и модель, и представление. При этом возможна передача в виде ссылки.

Шаблон осуществляет поддержку двустороннего сопряжения, которое возникает у представления и модели представления в соотношении «много-к-одному». Наличие вспомогательной библиотеки MVVM – это базовые классы, чтобы повысить комфортность взаимодействия с шаблоном. Среди классов – те, что помогают при работе с Activity и Fragment, с соответствующей логикой привязки, которая им уже присуща. Параллельно определяется, какими методами нужно работать с обратными вызовами. Шаблон превосходит MVP ввиду того, что имеет частичную зависимость от вида, что заметно облегчает работу. Современная IDE «предпочитает» именно этот вариант, тем более что появляется возможность уменьшать объем кода, что позволяет синхронизировать представление с View Model. Однако возникает ограниченность в сопряжении данных, для чего может понадобиться дополнительная память.

Шаблон «Clean Architecture» используется для разделения компонентов для кольцевой архитектуры. Возможна простая передача условий hat-кода от внешнего уровня на внутренние. При этом слои последних могут не иметь информацию о том, какие возможности есть у внешних слоев. Очевидно, что это свидетельство сложности проекта. Особенности структуры шаблона отражены на рисунке 3.

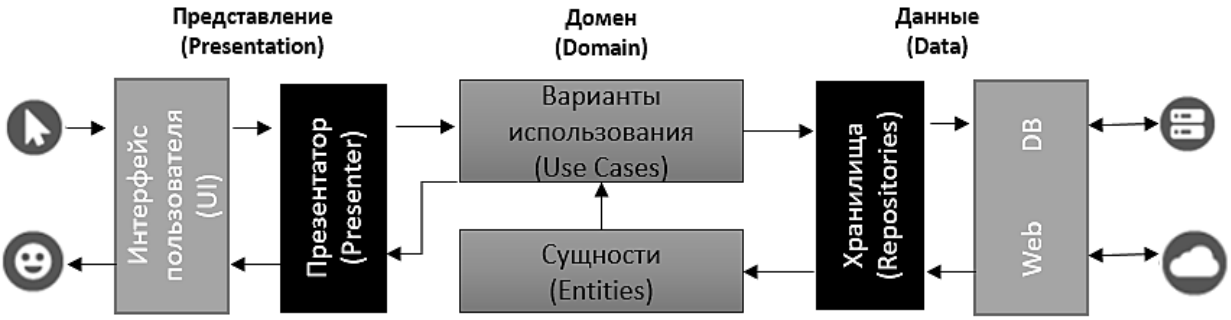

Рис. 3. *Шаблон «Clean Architecture» (составлено автором)*

Fig. 3. *Clean Architecture template (created by the author)*

Под сущностями понимается ряд общих бизнес-правил с содержанием также объектов передачи данных (DTOs). Внешние изменения на правилах практически не сказываются.

Варианты использования представлены как элементы, что могут взаимодействовать и являются обозначением бизнес-правил ПО, которые значимы для конкретных приложений. То есть на данном уровне наблюдается изоляция от изменения в базах данных, инструментах, интерфейсе.

Адаптеры интерфейса предназначены для преобразования данных. Они переводятся в форматы, которые совместимы с базами данных интернета.

Фреймворки и драйверы находятся на внешнем уровне, который состоит из фреймворка для веб-разработок, баз данных с интерфейсом и HTTP-клиентом.

Шаблон «Clean Architecture» занимается объединением слоев в модули. Речь идет о презентации с пользовательским интерфейсом, презентатором и ViewModels, домене с сущностями и организацией взаимодействия данных, с включением баз данных, там также содержатся Rest-клиенты.

Шаблон «VIPER». Расшифровка наименования свидетельствует о степени разделенности модуля. Образуются отдельные степени ответственности. Всем экранам отводится свой модуль. Таким образом, приложение разделяется на пять степеней. Шаблоны такого типа предназначены и перспективны для того, чтобы создавать чистую и модульную структуру. В результате сохраняются чистота и организация кода.

Представление в приложениях соотносится с Действием или Фрагментом. Задача – упрощение представления, насколько это возможно при имеющихся параметрах и аспектах. От представления в данных условиях ожидается, что оно продемонстрирует user interface. Взаимодействие организуется таким образом, чтобы действие выполнялось с учетом Презентатора. Последний может рассматриваться в качестве «руководителя» в отделе. От него поступают заказы Взаимодействующему, представление по его инициативе занимается отображением содержимого, а маршрутизатор – навигацией к тому или иному экрану. Сущность характеризуется данными, которые содержатся в приложении. То есть в действиях она схожа с моделью для уже рассмотренного выше шаблона MVP.

Задача маршрутизатора – обработка перехода к тому или иному экрану, пока развивается жизненный цикл приложения. Это обеспечивает коду многократность в использовании, его тестируемость повышается, отладка становится проще, как и понимание. Именно VIPER позволяет разработчикам работать с большей эффективностью, вывод кода для функций становится более предсказуемым. Это снимает большинство обсуждений в команде, экономит временные затраты на реализацию процесса.

Обсуждение

В настоящем разделе сравниваются предложенные выше шаблоны, среди которых MVC, MVP, MVVM Clean Architecture и VIPER, которые применимы для Android и iOS. Атрибуты становятся ключевыми критериями и индикаторами преимуществ. Речь идет о способности системы к изменениям и исправлениям, тестируемости и возможности повторного использования. Выясняется, что при добавлении функционала в MVC продолжается рост файла, а любое действие, выполняемое платформой, будет иметь различные связи, например, будут задействованы интерфейс, механизмы доступа к данным, а это является препятствием для осуществления рефакторинга и изменений. Размещение контроллера и логики представления в связку Действие / Фрагмент является негативным решением. Среди преимуществ архитектуры MVP – простота и многократность в использовании кода, отмечается лучшая тестируемость ПО в рамках заданного процесса под конкретный проект. Впрочем, специалисты отмечают, что при росте проектов может возникнуть сложность с презентатором. Он может начать функционировать как «большой класс», где применяются разнообразные методы, а это всегда вредит как пониманию, так и обслуживанию.

Эксперты считают, что VIPER способствует чистоте исходного кода, он также приобретает компактность, может быть использован повторно. Однако возникает иная проблема, связанная с недостаточным количеством ресурсов, чтобы действительно глубоко изучить дизайн данного шаблона. Если рассматривать его традиционный вариант, то именно докладчики занимаются передачей модулей.

MVVM особенно перспективен, если происходит работа над небольшими проектами, и может нивелировать недостатки ограничениями, которые наблюдаются в MVC, MVP.

Каждый компонент может быть протестирован по отдельности. При программировании стиль может быть более произвольным. Однако с ростом кодовой базы происходит чрезмерное «раздувание» ViewModel. Это приводит к тому, что обязанности разделить достаточно сложно. В этих случаях целесообразно воспользоваться возможностями MVVM с Clean Architecture. В результате появляется возможность для разделения обязанностей в наборе исходного кода. С его помощью можно гипотетически, абстрагированно рассматривать, какая логика действий будет приемлемой для приложения. То есть шаблон представляет собой сочетание преимуществ, которые есть в MVP (в частности, при разделении задач), и также позволяет одновременно применять сопряжение данных. На выходе специалисты получают архитектуру, где многочисленные операции участвуют в обработке модели. Также минимизируется логика внешнего вида. Итак, можно прийти к выводу, что MVVM совокупно с Clean Architecture – самая оптимальная архитектура, чтобы разрабатывать мобильные приложения на базе Android и iOS.

Выводы

Правильный выбор архитектуры способствует более эффективной разработке мобильного приложения. При состоявшемся сопоставительном анализе шаблонов можно прийти к следующим выводам. Если мобильные приложения разрабатываются на базе Android или iOS, то целесообразно применять MVVM. Проведение анализа сопровождалось оценкой тестируемости, сопряженности и возможности внесения изменений и исправлений. У каждого шаблона есть свои сильные стороны, однако в рамках исследования наиболее оптимальна комбинация MVVM и Clean Architecture. Она отличается поддержкой всех критериев, а участие шаблона Clean Architecture помогает справиться с различными проблемами, что невозможно, если используется лишь первый из названных шаблонов. Выбор данной комбинации считаем наиболее оптимальным и целесообразным для разработки приложений на основании того, что такая комбинация позволяет четко разделить логику представления и бизнес-логику, что упрощает поддержку и тестирование кода. MVVM обеспечивает эффективное связывание данных между моделью и представлением, что минимизирует логику внешнего вида и упрощает работу с пользовательским интерфейсом, в сочетании с Clean Architecture это позволяет создать гибкую и масштабируемую структуру приложения, что особенно важно при увеличении кодовой базы и сложности проекта. Это способствует более чистой и управляемой архитектуре приложения. Резюмируя, необходимо напомнить о том, что архитектурный шаблон MVVM не является универсальным решением, которое уместно и корректно в любой ситуации. Все проекты обладают известной индивидуальностью, и архитектурный шаблон должен отвечать требованиям разработчика. В дальнейшем целесообразно провести анализ и экспериментальное исследование, чтобы понять, как действуют иные атрибуты, например, как разные шаблоны сказываются на производительности.

СПИСОК ЛИТЕРАТУРЫ / REFERENCES

1. Mobile operating system market share worldwide. URL: https://gs.statcounter.com/os-market-share/mobile/worldwide/2023 (дата обращения: 12.03.2024).

2. Ambani D. Model view controller (MVC): A latest mobile & web application development approaches. *Vidhyayana-An International Multidisciplinary Peer-Reviewed E-Journal.* 2020. Vol. 6. No. 3. Pp. 1–12.

3. Vennaro E. VIPER. iOS Development at Scale: App Architecture and Design Patterns for Mobile Engineers. Berkeley, CA: Apress, 2023. Pp. 299–326.

4. Epiloksa H.A., Kusumo D.S., Adrian M. Effect Of MVVM Architecture Pattern on Pp. 1949–1955.

5. Sukarsa I.M., Piarsa I.N., Premana Putra I.G.B. et al. Application of MVP architecture in developing android-based seminar ticket booking applications. *Journal RESTI (Rekayasa Sistem dan Teknologi Informasi).* 2020. Vol. 4. No. 3. Pp. 513–520. DOI: 10.29207/resti.v4i3.1396

6. Capdepon Q., Hlad N., Serial A. et al. Migration process from monolithic to micro frontend architecture in mobile applications. *IWST 2023: International Workshop on Smalltalk Technologies*. Lyon, France; August 29th-31st, 2023. Pp. 1–10.

7. Sokolova K., Lemercier M. Towards high quality mobile applications: Android passive MVC architecture. *International Journal On Advances in Software*. 2014. Vol. 7. No. 2. Pp. 123–138.

8. Lombardi M., Pascale F., Santaniello D. Internet of things: A general overview between architectures, protocols and applications. *Information.* 2021. Vol. 12. No. 2. P. 87.

9. Nunkesser R. Using hexagonal architecture for mobile applications. *ICSOFT*. 2022. Pp. 113–120.

10. *Василевский М. П.* Выбор архитектуры Android приложения // Информационно-аналитические и интеллектуальные системы для производства и социальной сферы. 2022. С. 25–30.

Vasilevsky M.P. Selecting the architecture of an Android application. *Informatsionno-analiticheskiye i intellektual'nyye sistemy dlya proizvodstva i sotsial'noy sfery* [Information, analytical and intelligent systems for production and the social sphere]. 2022. Pp. 25–30. (In Russian)

11. Sommerville I. Engineering software products. London: Pearson, 2020. 355 p.

12. Dobrean D., Dioșan L. A comparative study of software architectures in mobile applications. *Studia Universitatis Babeș-Bolyai Informatica*. 2019. Pp. 49–64.

13. Andika M.R., Selviandro N., Wulandari G.S. Understanding the Impact of Modularity in iOS App Performance using VIPER Architecture Pattern. *2023 3rd International Conference on Intelligent Cybernetics Technology & Applications (ICICyTA). IEEE*, 2023. Pp. 358–363.

14. Salazar F.J.A., Brambilla M. Tailoring software architecture concepts and process for mobile application development. *Proceedings of the 3rd International Workshop on Software Development Lifecycle for Mobile*. 2015. Pp. 21–24.

15. *Курганова А. Г.* Процесс выбора архитектуры для мобильного приложения // StudNet. 2021. Т. 4. № 6. С. 386–397.

Kurganova A.G. The process of choosing an architecture for a mobile application. *StudNet.* 2021. Vol. 4. No. 6. Pp. 386–397. (In Russian)

16. *Бакшанский В. Д., Дунская Л. К., Замотайлова Д. А.* Организационные различия архитектуры мобильных приложений с использованием скрипта Service Worker // Цифровизация экономики: направления, методы, инструменты. 2021. С. 261–264.

Bakshansky V.D., Dunskaya L.K., Zamotailova D.A. Organizational differences in the architecture of mobile applications using the Service Worker script. *Tsifrovizatsiya ekonomiki: napravleniya, metody, instrumenty* [Digitalization of the economy: directions, methods, tools]. 2021. Pp. 261–264. (In Russian)

Финансирование. Исследование проведено без спонсорской поддержки.

Funding. The study was performed without external funding.

Информация об авторе

Маринин Алексей Константинович, магистрант, Московский финансово-промышленный университет «Синергия»;
129090, Россия, Москва, ул. Мещанская, 9/14, стр. 1;
aleksei.marinin247@gmail.com, ORCID: https://orcid.org/0009-0008-0242-8074, SPIN-код: 9423-4060

Information about the author

Alexey K. Marinin, Master's Student, Moscow University for Industry and Finance "Synergy";
129090, Russia, Moscow, 9/14, building 1, Meshchanskaya street;
aleksei.marinin247@gmail.com, ORCID: https://orcid.org/0009-0008-0242-8074, SPIN-code: 9423-4060