УДК 004.4
DOI 10.17513/snt.39696

РАЗРАБОТКА ПРОГРАММНОГО КОМПЛЕКСА ДЛЯ МНОГОЯДЕРНОЙ РЕКОНФИГУРИРУЕМОЙ ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЫ

[1]Мартышкин А.И., [2]Семенов А.С., [2]Митрохин М.А., [2]Акифьев И.В., [2]Токарев А.Н.

[1]ФГБОУ ВО «Пензенский государственный технологический университет», Пенза, e-mail: mai@penzgtu.ru;
[2]ФГБОУ ВО «Пензенский государственный университет», Пенза, e-mail: andreysemenov2000@gmail.com, vt@pnzgu.ru, huntersu@yandex.ru, ant19@mail.ru

Основной целью данного исследования является сравнение способов разработки программного обеспечения для реконфигурируемых вычислительных систем на примере смартфона. Проведен анализ долей рынка мобильных операционных систем в мире. Кроссплатформенный способ и способ разработки под определенную платформу сравниваются по различным критериям (производительность итогового программного продукта, стоимость разработки, поддержка новейшего функционала ОС), на основе чего выявляются их преимущества и недостатки. Для сравнения производительности конечного программного продукта создано два приложения на Android и Flutter с идентичным функционалом. Все приложения запускаются на одном устройстве под ОС Android. Также в данной статье представлены рекомендации по выбору самого эффективного инструмента для разработки кроссплатформенного программного обеспечения. Достигнутые результаты исследования показывают, что при выборе способа разработки программного обеспечения для реконфигурируемой вычислительной системы, следует ориентироваться на некоторые приоритеты. Если главное – производительность и хватает финансовых ресурсов, то стоит выбирать способ разработки под определенную платформу. Если финансы ограничены и малые потери производительности требуемого программного обеспечения несущественны, следует обратить внимание на кроссплатформенный способ разработки, где по ряду критериев лидирует Flutter.

Ключевые слова: многоядерные реконфигурируемые вычислительные сети, смартфон, Android, Flutter, React Native, кроссплатформенность

DEVELOPMENT OF A SOFTWARE PACKAGE FOR A MULTI-CORE RECONFIGURABLE COMPUTING SYSTEM

[1]Martyshkin A.I., [2]Semenov A.S., [2]Mitrokhin M.A., [2]Akifev I.V., [2]Tokarev A.N.

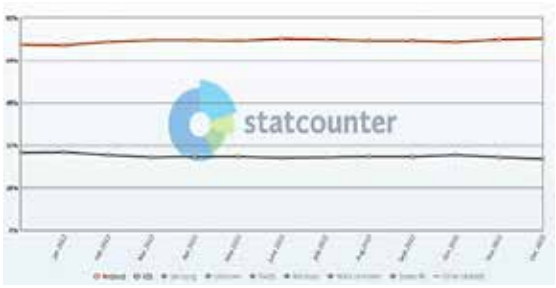
[1]Penza State Technological University, Penza, e-mail: mai@penzgtu.ru;
[2]Penza State University, Penza, e-mail: andreysemenov2000@gmail.com, vt@pnzgu.ru, huntersu@yandex.ru, ant19@mail.ru

The main purpose of this study is to compare the ways of software development for reconfigurable computing systems on the example of a smartphone. An analysis of the market shares of mobile operating systems in the world is carried out. The cross-platform method and the method of development for a particular platform are compared according to different criteria (performance of the final software product, development cost, support for the latest operating system functionality), on the basis of which their advantages and disadvantages are identified. To compare the performance of the final software product we created 2 applications on Android and Flutter with identical functionality. All applications are run on the same Android device. This article also presents recommendations on how to choose the most efficient tool for cross-platform software development. The results of the study show that when choosing a way to develop software for a reconfigurable computing system, you should be guided by some priorities. If the main thing is performance and enough financial resources, it is worth choosing the development method for a particular platform. If finances are limited and small performance losses of the required software are not significant, you should pay attention to cross-platform development method, where Flutter leads by a number of criteria.

Keywords: multi-core reconfigurable computing networks, smartphone, Android, Flutter, React Native, cross-platform

Реконфигурируемые вычислительные системы (РВС) – это системы, в которых есть возможность изменять аппаратную часть и тем самым менять модель производимых в ней вычислений [1–3]. Также РВС рассмотрены в [4, 5]. Разработка программного обеспечения для многоядерной реконфигурируемой вычислительной системы – это процесс, имеющий множество нюансов [6]. Если рассматривать этот вопрос на примере мобильных устройств, которые являются многоядерными реконфигурируемыми вычислительными системами, основной проблемой становится выбор платформы. За долгие годы развития мобильных устройств лишь две операционные системы (ОС) смогли остаться на плаву и занять лидирующие позиции [7]. Речь идет об *IOS*

и *Android*. Согласно графику, представленному на сайте *statcounter* [8] (рис. 1, а), наибольшая доля мирового рынка мобильных ОС принадлежит *Android*, она составляет 71,09%. И лишь 28,21% принадлежит *IOS*. На основании этих данных можно сделать вывод, что для привлечения большего количества пользователей стоит выбрать *Android*. Но, если обратить внимание на состояние рынка мобильных ОС в отдельных странах, например в США (рис. 1, б), можно увидеть, что там большая доля принадлежит IOS, а не *Android*, что можно увидеть на графике с сайта *statcounter*.



а

б

Рис. 1. График долей рынка мобильных ОС в мире (а) и в США (б)

Подобную ситуацию можно наблюдать и в других странах мира, например в Канаде, а это значит, что программный продукт (ПП), который предназначен для пользователей из разных стран, должен иметь возможность работать на двух платформах.

Создатели ПП могут пойти по пути разработки двух отдельных приложений для каждой ОС или по пути кроссплатформенной разработки. У каждого способа есть свои преимущества и недостатки. В рамках данного исследования они будут выявлены с помощью сравнения ПП, разработанными этими способами, стоимости разработки, скорости поддержки новейших функций платформы [9]. Также в исследовании будут проанализированы два популярных фреймворка для кроссплатформенной мобильной разработки – *Flutter* и *ReactJS*. Будут выделены их преимущества и недостатки, путем сравнения по критериям производительности, безопасности и сложности подбора компетентных специалистов. Также в статье даны рекомендации по выбору одного из фреймворков для создания нового ПП.

Материалы и методы исследования

Для сравнения производительности итогового ПП будет создано два приложения на *Android* и *Flutter* с одинаковым функционалом. Все приложения будут запускаться на одном устройстве под ОС *Android*. Приложение будет представлять собой экран, на котором одновременно вращаются, уменьшаются и увеличиваются 28 изображений. Результатом теста будут являться следующие зафиксированные данные: *FPS* (*Frames Per Second* или количество кадров в секунду); загрузка процессора; количества задействованной оперативной памяти. Для того чтобы выявить причину разницы в производительности двух приложений, будут рассмотрены процессы отрисовки графического интерфейса *Android* и *Flutter*. Расчет разницы в стоимости разработки ПП будет происходить в рамках выявления разницы затрат на создание мобильного приложения, но будет учитываться только оплата труда разработчиков, делая допущение, что остальные затраты идентичны. Сравнение скорости поддержки новейших функций платформы будет происходить на основании личного опыта. Сравнение *Flutter* и *React Native*, как конкурентов в кроссплатформенном способе разработки ПП, будет проходить на основании информации из открытых источников и личного опыта. Было разработано два тестовых приложения на *Flutter* и *Android*. На рис. 2 представлен снимок экрана тестового приложения. Он идентичен для приложения на *Android* и *Flutter*.

У приложения, написанного на *Android*, частота кадров в секунду стабильно держится в районе 60. Максимальная зафиксированная просадка – до 59 к/с. FPS приложения на *Flutter* крайне нестабилен, не может надолго зафиксироваться в одном положении. Среднее значение FPS – 50 к/с. Максимальная зафиксированная просадка – до 29 к/с. Приложение на *Flutter* затрачивает больше ресурсов процессора, нежели приложение на *Android*. В среднем приложение на *Flutter* требует 33% ресурсов *CPU*, а приложение на *Android* 26%. Пиковая загруженность процессора во время исполнения приложения на *Flutter* составила 39%, а на *Android* 30%. Приложение на *Flutter* во время выполнения затрачивает больше ОП, чем приложение на *Android*. В среднем разница равна 120 Мб.

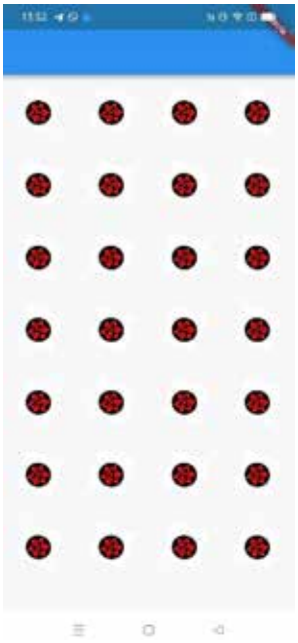


Рис. 2. Снимок экрана приложения

На основе указанных данных очевидно, что приложение, написанное под конкретную платформу, будет производительнее, чем кроссплатформенное решение. Этот факт является его главным преимуществом. Были рассмотрены процессы отрисовки графического интерфейса *Android* и *Flutter*. Для наглядности были составлены две модели в виде сетей Петри [10]. Для наглядности будут составлены две модели в виде сетей Петри. Сеть Петри, отражающую основные этапы рендеринга приложения, написанного на *Android,* можно видеть на рис. 3.

Результат тестирования (в виде зафиксированных данных) представлен в табл. 1.

Процесс, изображенный на модели, отражает следующее. После изменения в *View* (изменение цвета, размера, положения и т.д.) *Android* проходит по элементам, создавая *Display List* (он хранит информацию о рендеринге) и определяя область(и), где произошло изменение. Все эти действия выполнялись в потоке *UI*. Затем список передается в поток рендеринга. Там он превращается в *DLOps* (*Display List Operations*) – операции списка отображения (нарисуй, раскрась и т.д.), и из него убираются операции, которые происходят за пределами области изменения, которая была выявлена ранее. Параллельно с этим подготавливается буфер для *GL*-операций. После *DLOps* помещаются в этот буфер с предварительной трансформацией в *GL*-команды. Полученный буфер загружается в *GPU* и происходит отрисовка экрана [11, с. 81; 12].

Сеть Петри, отражающую основные этапы рендеринга приложения, написанного на *Flutter*, можно увидеть на рис. 4.

Процесс, изображенный на модели, отражает следующее. После изменения в *Widget Flutter* проходит по дереву элементов и вызывает *rebuild* для каждого, который помечен как *dirty* (грязный). Затем *Flutter* проходит по дереву *render objects* и запрашивает компоновку для каждого, который помечен как *dirty*. В результате формируется *Scene*. Она передается в *Flutter engine*, там из нее извлекается *Layer Tree*. Все ранее перечисленные действия происходили в потоке *UI*. Дальше запускается *Raster* поток, в него передается *Layer Tree*, там оно разворачивается, подготавливаются слои. Далее подготавливается очередь команд *Skia*. Параллельно с этим подготавливается буфер для *GL*-операций. Затем *Skia*-команды помещаются в этот буфер с предварительной трансформацией в *GL*- команды. Полученный буфер загружается в *GPU*, отрисовывается новый экран [13, с. 108; 14, с. 307].

На основании сравнения двух процессов можно сделать вывод, что они похожи, за исключением момента до формирования буфера *GL*-команд. Процесс рендеринга на *Flutter* содержит операции разворота дерева и подготовки его слоев, аналога которым нет в *Android*, там сразу формируется *DLOps*, который трансформируется в GL-команды. Скорее всего этим обусловлено более высокое потребление ресурсов *CPU*, *RAM* у приложения, написанного на *Flutter* (в сравнении с приложением, написанным на *Android*), чем и обусловлено меньшее количество кадров в секунду. Для сравнения стоимости разработки ПП были приведены средние зарплаты 2023 г. *Android*, *iOS* и *Flutter* разработчиков в Москве по данным сайта *HH.ru* [15] для разной квалификации программистов (табл. 2).

Таблица 1

FPS, *CPU*, Memory во время исполнения приложений

	Время работы приложения, мин	*FPS*, к/с	*CPU*,%	*Memory*, Mb
Android	2	60 к/с	26%	50–58 Мб
Flutter	2	55 к/с	33%	100–114 Мб

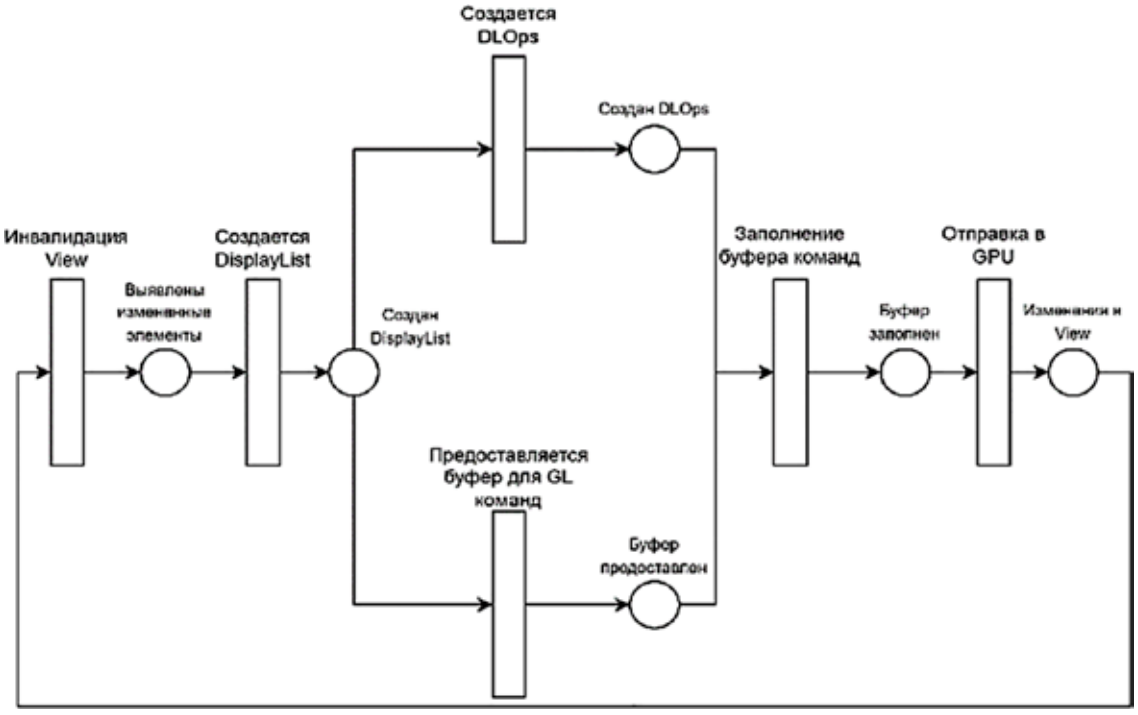


Рис. 3. Сеть Петри отрисовки графического интерфейса Android

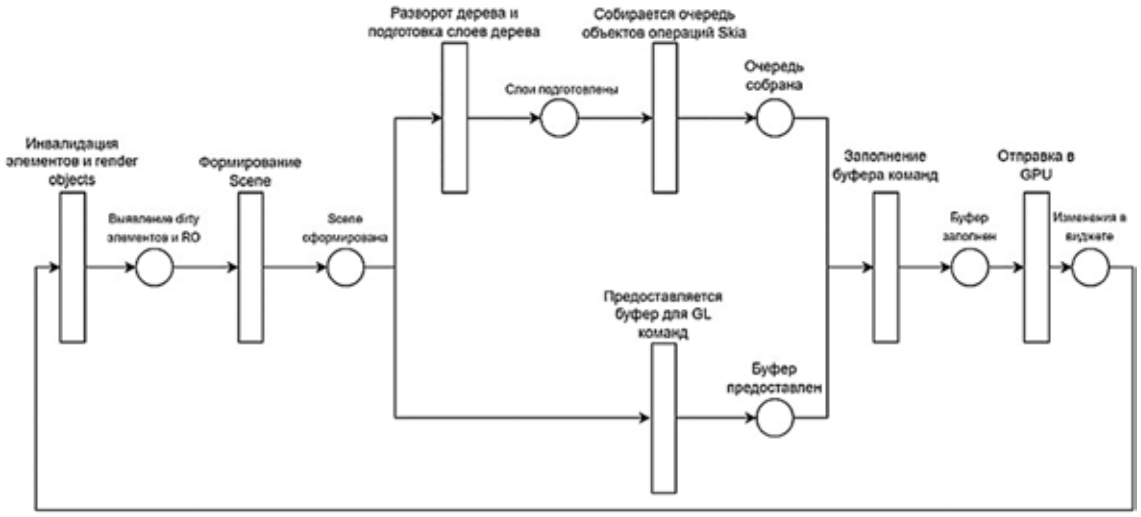


Рис. 4. Сеть Петри отрисовки графического интерфейса Flutter

Таблица 2

Средние зарплаты разработчиков мобильных приложений, руб./мес.

	Android	IOS	Flutter	React Native
Senior	290 000	320 000	280 000	260 000
Middle	180 000	200 000	150 000	150 000
Junior	90 000	100 000	80 000	80 000

Таблица 3

FPS, *CPU*, *Memory* во время исполнения приложений

	Время работы приложения, мин	*FPS*, к/с	*CPU*, %	*Memory*, Mb
React Native	2	7	8,5	424
Flutter	2	19	10,28	168

Важно, что кроссплатформенная мобильная разработка подразумевает создание единой кодовой базы для двух операционных систем: *iOS* и *Android* – то есть необходимо иметь одну команду разработчиков. Тогда как разработка отдельно под каждую операционную систему требует наличия нескольких команд – двух в случае разработки под две операционные системы.

Предположим, что для разработки программного продукта необходима команда из двух *Senior*, трех *Middle* и двух *Junior* разработчиков. Очевидно, дешевле выйдет разработка на *Flutter*/ *React Native* [16]. Месячные затраты на две команды разработчиков (*Android* и *IOS*) в среднем составят 2 млн 740 тыс. руб. Команда *Flutter* или *React* разработчиков в месяц требует в среднем 1 млн 170 тыс. руб. Разница составляет 1 млн 570 тыс. руб., что является весомым преимуществом кроссплатформенной разработки. Касательно скорости поддержки новейших функций платформы, инструменты для создания приложений под конкретную ОС получают доступ к ее новым функциям в день их выпуска. Фреймворки для кроссплатформенной разработки смогут пользоваться новыми функциями лишь спустя некоторое время. Задержка, как правило, небольшая, но она есть, а значит, по этому критерию первенство вновь берет разработка под определенную ОС.

Было произведено сравнение двух конкурирующих инструментов для разработки кроссплатформенных ПП – *Fluitter* и *React Native*. Производительность приложений на *Flutter* превосходит производительность приложений на *React Native*. В качестве основного языка программирования *Flutter* использует *Dart*, который в свою очередь компилируется в нативный код платформы, который в равных условиях всегда будет быстрее *JavaScript* от *React* Native. Также стоит отметить производительность рендеринга *Flutter*. *React Native* использует родные виджеты платформы и события передает через *JavaScript*, что сильно сказывается на производительности. Конкуренты от Google используют собственный встроенный 2D движок *Skia*, что позволяет рендерить объекты с частотой до 120 кадров в секунду [17, с. 10].

В статье «*Flutter vs React Native vs Native*: *Deep Performance Comparison*» [18] приводится сравнение производительности двух фреймворков в виде конкретного примера. На каждом из них создано приложение, состоящее из одного экрана, на котором 200 изображений подвержены различной анимации (вращение, увеличение, исчезновение). Результаты теста представлены в табл. 3.

Можно заметить, что приложение, написанное на *Flutter*, больше нагружает процессор, но показывает большее количество кадров в секунду и использует меньше оперативной памяти, что доказывает его превосходство в производительности над приложением, которое написано на *React Native*. Если говорить о безопасности, то конечное приложение на *React Native* – это *JavaScript* код. Любой может получить к нему доступ в конечной сборке релиза и изменить его логику. Итоговый нативный код приложения, написанного на *Flutter*, не читаем для человека, что делает практически невозможным обратную обработку. Кроме того, единая нативная библиотека со сложной структурой и меняющимися форматами данных, которая формируется после компиляции, не позволяет даже частично восстановить исходный код при реверс-инжиниринге. Приложения, написанные на *Flutter,* по безопасности приближаются к нативным.

Что касается подбора специалистов, *React Native* с 2015 г. используется для создания мобильных приложений. К тому же для разработки использует популярный язык программирования *JavaScript*. *Flutter* выпустил первую свою стабильную версию в 2018 г., а для разработки использует малоиспользуемый язык *Dart*. Из этих вводных данных ясно, что на рынке труда гораздо больше разработчиков, умеющих работать с *React Native*. Стоит заметить, что при желании можно без особых усилий переквалифицировать уже имеющихся сотрудников, работающих с приложениями для *Android* или *IOS*, во *Flutter*-разработчиков. Гораздо легче, чем в *React Native*-разработчиков. Это обусловлено тем, что, в отличие от *JavaScript*, *Dart* является типизированным языком программирования, так же как *Java*, *Kotlin*, *Objective-C* и *Swift*, и в принципе они имеют схожий синтаксис.

Заключение

Таким образом, результаты исследования показывают, что при выборе способа разработки ПП для реконфигурируемой вычислительной системы необходимо ориентироваться на поставленные приоритеты. Если главным является производительность и хватает финансовых ресурсов, стоит выбирать способ разработки под определенную платформу. Если финансовый ресурс ограничен и небольшие потери в производительности итогового ПП некритичны, следует обратить внимание на кроссплатформенный способ разработки. Среди инструментов кроссплатформенной разработки по ряду критериев лидирует *Flutter*.

Список литературы

1. Гузик В.Ф., Каляев И.А., Левин И.И. Реконфигурируемые вычислительные системы. Таганрог: Южный федеральный университет, 2016. 472 с.

2. Чернышев А.А., Борзов Д.Б., Асеев Д.А. Реконфигурируемые вычислительные системы как метод повышения производительности вычислительного комплекса // Провинциальные научные записки. 2021. № 2 (14). С. 52–55.

3. Каляев И.А., Левин И.И. Реконфигурируемые вычислительные системы на основе ПЛИС. Ростов-на-Дону: Южный научный центр РАН, 2022. 475 с.

4. Гонзалез М.Ф. Реконфигурируемые вычисления: обзор // Известия Тульского государственного университета. Технические науки. 2023. № 2. С. 42–49.

5. Евтихов В.Г., Евтихова Н.В., Евтихов М.В., Евтихов М.Г. Высокопроизводительные вычисления. М.: Федеральное государственное автономное образовательное учреждение высшего образования «Московский политехнический университет», 2023. 338 с.

6. Мартышкин А.И., Кирюткин И.А., Мереняшева Е.А. Автотестирование встраиваемой реконфигурируемой вычислительной системы // Известия Юго-Западного государственного университета. 2023. Т. 27, № 1. С. 140–152.

7. Таненбаум Э., Бос Х. Современные операционные системы. СПб.: Питер, 2015. 1120 с.

8. Statcounter Global Stats [Электронный ресурс]. URL: https://gs.statcounter.com (дата обращения: 25.03.2023).

9. Пащенко Д.В., Трокоз Д.А., Синев М.П., Мартышкин А.И., Кормишина В.В., Малини Д.Д. Моделирование автоматизированной информационной системы государственных услуг // XXI век: итоги прошлого и проблемы настоящего плюс. 2018. Т. 7, № 2 (42). С. 6–9.

10. Трокоз Д.А., Бикташев Р.А., Синев М.П., Федяшев М.С., Шеянов Н.Н. Методика преобразования темпорального конечного автомата в СП-модель // XXI век: итоги прошлого и проблемы настоящего плюс. 2020. Т. 9, № 3 (51). С. 45–49.

11. Марсикано К., Гарднер Б., Филлипс Б., Стюарт К. Android. Программирование для профессионалов / Пер. с англ. Е.А. Матвеев, С. Черников. СПб.: Питер. 2021. 704 с.

12. Глубокое понимание механизма рендеринга Android [Электронный ресурс]. URL: https://russianblogs.com/article/7623516090 (дата обращения: 25.03.2023).

13. Biessek A. Flutter for Beginners: An introductory guide to building cross-platform mobile applications with Flutter and Dart 2. M.: Packt Publishing, 2019. 512 p.

14. Napoli M. Beginning Flutter: A Hands on Guide to App Development / M. Biessek. M.: Packt Publishing, 2019. 512 p.

15. HeadHunter [Электронный ресурс]. URL: https://hh.ru (дата обращения: 25.03.2023).

16. Трокоз Д.А., Мартышкин А.И., Федяшев М.С., Карлыганов А.Д., Лысцов Н.А. Типовое решение задачи авторизации пользователя в информационной системе // XXI век: итоги прошлого и проблемы настоящего плюс. 2019. Т. 9, №. 4(48). С. 33-38.

17. Windmill E. Flutter in Action. M.: Manning, 2020. 368 p.

18. Flutter vs React Native vs Native: Deep Performance Comparison [Электронный ресурс]. URL: https://medium.com/swlh/flutter-vs-react-native-vs-native-deep-performance-comparison-990b90c11433 (дата обращения: 27.03.2023).