МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
**«Южно-Уральский государственный университет
(национальный исследовательский университет)»**
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой, д.ф.-м.н.,
профессор

_____ Л.Б. Соколинский

«___»_____ 2024 г.

Разработка мобильного приложения для обучения новых членов стипендиальных комиссий

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 09.03.04.2024.308-448.ВКР

Научный руководитель,
доцент кафедры СП, к.э.н.
_____Т.Ю. Шабанов

Автор работы,
студент группы КЭ-404
_____ Я.А. Иванов

Ученый секретарь
(нормоконтролер)
_____ И.Д. Володченко
«___»_____ 2024 г.

Челябинск, 2024 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

УТВЕРЖДАЮ
Зав. кафедрой СП
_____ Л.Б. Соколинский
29.01.2024 г.

ЗАДАНИЕ
на выполнение выпускной квалификационной работы бакалавра
студенту группы КЭ-404
Иванову Ярославу Андреевичу,
обучающемуся по направлению
09.03.04 «Программная инженерия»

1. **Тема работы** (утверждена приказом ректора от 22.04.2024 г. № 764-13/12)
Разработка мобильного приложения для обучения новых членов стипендиальных комиссий.
2. **Срок сдачи студентом законченной работы:** 03.06.2024 г.
3. **Исходные данные к работе**
3.1. Android Mobile App Developer Tools. [Электронный ресурс] URL: https://developer.android.com (дата обращения: 29.01.2024 г.).
3.2. Expo Documentation. [Электронный ресурс] URL: https://docs.expo.dev (дата обращения: 29.01.2024 г.).
3.3. Getting Started With GraphQL.js | GraphQL. [Электронный ресурс] URL: https://graphql.org/graphql-js (дата обращения: 29.01.2024 г.).
3.4. Документы – СКС Профсоюза. [Электронный ресурс] URL: https://sksrf.ru/documents (дата обращения: 29.01.2024 г.).
4. **Перечень подлежащих разработке вопросов**
4.1. Выполнить анализ предметной области.
4.2. Спроектировать мобильное приложение.

4.3. Реализовать мобильное приложение.

4.4. Провести тестирование разработанного приложения.

5. Дата выдачи задания: 29.01.2024 г.

Научный руководитель,
доцент кафедры СП, к.э.н. Т.Ю. Шабанов

Задание принял к исполнению Я.А. Иванов

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность

В настоящее время сфера онлайн-образования продолжает демонстрировать растущий интерес российских граждан. Так, согласно данным опроса аналитической компании Data Insight, среди 1600 респондентов в возрасте от 18 до 64 лет 85% планируют учиться онлайн в будущем [1]. При этом 53% отмечают, что главным плюсом онлайн-обучения является возможность учиться в удобное время. Подобный спрос влияет и на рынок мобильных приложений: аналитики Statista отмечают, что в Google Play категория Education располагается на втором месте по популярности, уступая только играм [2], а в App Store – на третьем, уступая играм и бизнесу [3].

Законы и нормативные акты, регулирующие сферу высшего образования и стипендиального обеспечения, периодически изменяются и дополняются. В связи с этим, в России организуются различные мероприятия и инициативы, позволяющие представителям образовательных учреждений и специалистам области быть в курсе последних изменений. Одним из примеров является всероссийская школа-семинар «Стипком 2023», прошедшая в Челябинске при поддержке федерального агентства по делам молодежи и губернатора Челябинской области [4]. Сотрудничество с органами исполнительной власти показывает заинтересованность государства в развитии профессиональных компетенций стипендиальных комиссий, администраций вузов и органов студенческого самоуправления.

Создание образовательной мобильной платформы может повысить удобство и доступность обучения, благодаря чему станет возможным получать необходимую информацию и развивать навыки в любом месте и в любое время, что особенно важно для занятых людей. Кроме этого, приложение позволит стандартизировать процесс обучения, обеспечивая согласованность знаний и подходов у всех членов стипендиальных комиссий. Это особенно важно для обеспечения справедливости и прозрачности в принятии решений о внесении изменений в локальные нормативные акты.

Постановка задачи

Целью выпускной квалификационной работы является разработка мобильного приложения для обучения новых членов стипендиальных комиссий. Для достижения поставленной цели необходимо решить следующие задачи:

1) выполнить анализ предметной области;
2) спроектировать мобильное приложение;
3) реализовать мобильное приложение;
4) провести тестирование разработанного приложения.

Структура и содержание работы

Работа состоит из введения, четырех глав, заключения и списка литературы. Объем работы составляет 54 страницы, объем списка литературы – 28 источников.

В первой главе проводится анализ предметной области, обзор аналогичных проектов, а также существующих инструментов и технологий для реализации поставленной цели.

Вторая глава посвящена проектированию мобильного приложения: определению функциональных и нефункциональных требований, вариантов использования и архитектуры. Помимо этого, в главе представлены структура хранилища контента и макеты экранов.

Третья глава описывает выбранный технологический стек и реализацию компонентов системы.

В четвертой главе представлен набор тестов для проверки функциональности и пользовательского интерфейса разработанного приложения.

Приложение А содержит спецификацию вариантов использования.

Приложение Б содержит диаграмму компонентов системы.

Приложение В содержит листинги с исходным кодом экранов и компонентов, отвечающих за отображение контента курсов и глав.

Приложение Г содержит диаграмму деятельности, описывающую алгоритм прохождения главы курса.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Описание предметной области

В целях стимулирования и социальной поддержки, обучающимся в России предоставляются соответствующие меры, такие как получение стипендий, материальной помощи и других денежных выплат, предусмотренных законодательством об образовании [5]. Существуют несколько видов стипендий, включая государственную академическую стипендию студентам, государственную социальную стипендию студентам и государственные стипендии аспирантам, ординаторам, ассистентам-стажерам. Размеры выплат устанавливаются исходя из размера стипендиального фонда – средств федерального бюджета, выделяемых на стипендиальное обеспечение обучающихся [6].

Для всестороннего и объективного рассмотрения вопросов назначения всех видов стипендий и других форм материальной поддержки, в образовательных организациях создаются стипендиальные комиссии – коллегиальные органы, в состав которых входят представители как администрации, так и студенчества. Интересы и права последних, как правило, защищают члены органов студенческого самоуправления или первичной профсоюзной организации.

Координацию деятельности первичных, региональных и межрегиональных организаций профсоюза работников народного образования и науки Российской Федерации в работе со студенческой молодежью осуществляет студенческий координационный совет общероссийского профсоюза образования «СКС Профсоюза» – постоянно действующий представительный орган первичных профсоюзных организаций студентов [7]. В число его основных целей также входит совершенствование условий для повышения эффективности работы данных организаций по представительству и защите прав и профессиональных интересов обучающихся.

Для решения вышеописанной проблемы «СКС Профсоюза» принимает участие в реализации просветительских проектов и площадок для обмена опытом, в число которых входят всероссийские форумы и образовательные семинары разной направленности. Спектр мероприятий также включает программы, позволяющие студентам узнать больше о теоретических аспектах распределения стипендиального фонда в соответствии с законодательством России.

Приложение, разработанное в ходе выполнения выпускной квалификационной работы, предназначено для обучения новых членов стипендиальных комиссий, которые могут быть недостаточно знакомы с процедурами, критериями и правилами, определяющими процесс принятия решений по назначению денежных выплат обучающимся. Оно повысит доступность знаний о нормативных правовых актах и практике их применения.

1.2. Анализ аналогичных проектов

В настоящее время существует не так много интернет-ресурсов, относящихся к рассматриваемой предметной области. Как правило, люди, желающие ознакомиться с нормативным регулированием стипендиального обеспечения, ищут соответствующую информацию в справочно-правовых системах, посещают тематические мероприятия в университете или обращаются к членам профсоюзного комитета по соответствующему вопросу.

Исходя из этого, в качестве аналогов на рассмотрение были взяты не только тематические платформы, но и конструкторы онлайн-курсов, преимущества и недостатки которых были учтены на этапах проектирования и реализации.

Приложение «СКС РФ»

«СКС РФ» – мобильное приложение для студентов, предоставляющее возможность получать скидки и специальные предложения от партнеров собственной дисконтной программы, а также узнавать новости университета и профсоюза [8]. Кроме этого, в нем содержится правовой справочник

по важным вопросам, включая информацию о стипендиях, материальной помощи и проживания в общежитии. Интерфейс соответствующих разделов приложения представлен на рисунке 1.

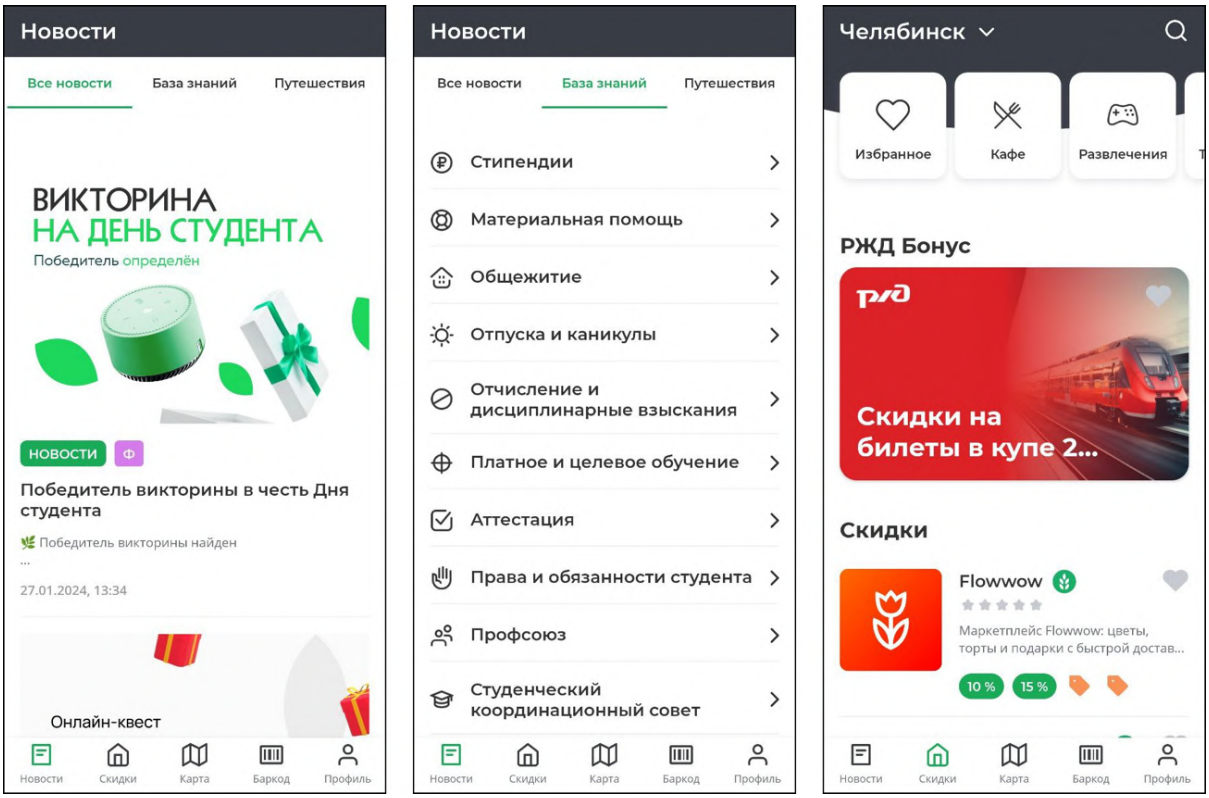

Рисунок 1 – Скриншоты приложения «СКС РФ»

Несмотря на обширную базу знаний по различным аспектам образовательной деятельности вузов, в проекте не реализованы интерактивные элементы обучения, что не позволяет осуществить проверку пользователей на предмет осведомленности в сфере правового обеспечения студентов. К недостаткам также можно отнести тот факт, что не все разделы справочника наполнены контентом: например, отсутствует информация о первичной профсоюзной организации вуза.

Stepik

Stepik – образовательная платформа с возможностью создания и публикации собственных курсов в каталоге для 7 миллионов пользователей [9]. Онлайн-конструктор включает множество инструментов для работы с текстом, загрузки видеоматериалов и использования функций автоматической

проверки. Также платформа предоставляет авторам общую статистику: о числе просмотров, регистраций и прогресса выполнения заданий. В свою очередь студенты курса могут вести обсуждения между собой и задавать вопросы преподавателю на форуме.

Платформа имеет мобильные приложения под операционные системы iOS и Android, а также веб-интерфейс. На рисунке 2 изображены экраны главной страницы Android-приложения, каталога курсов и интерактивной среды обучения.

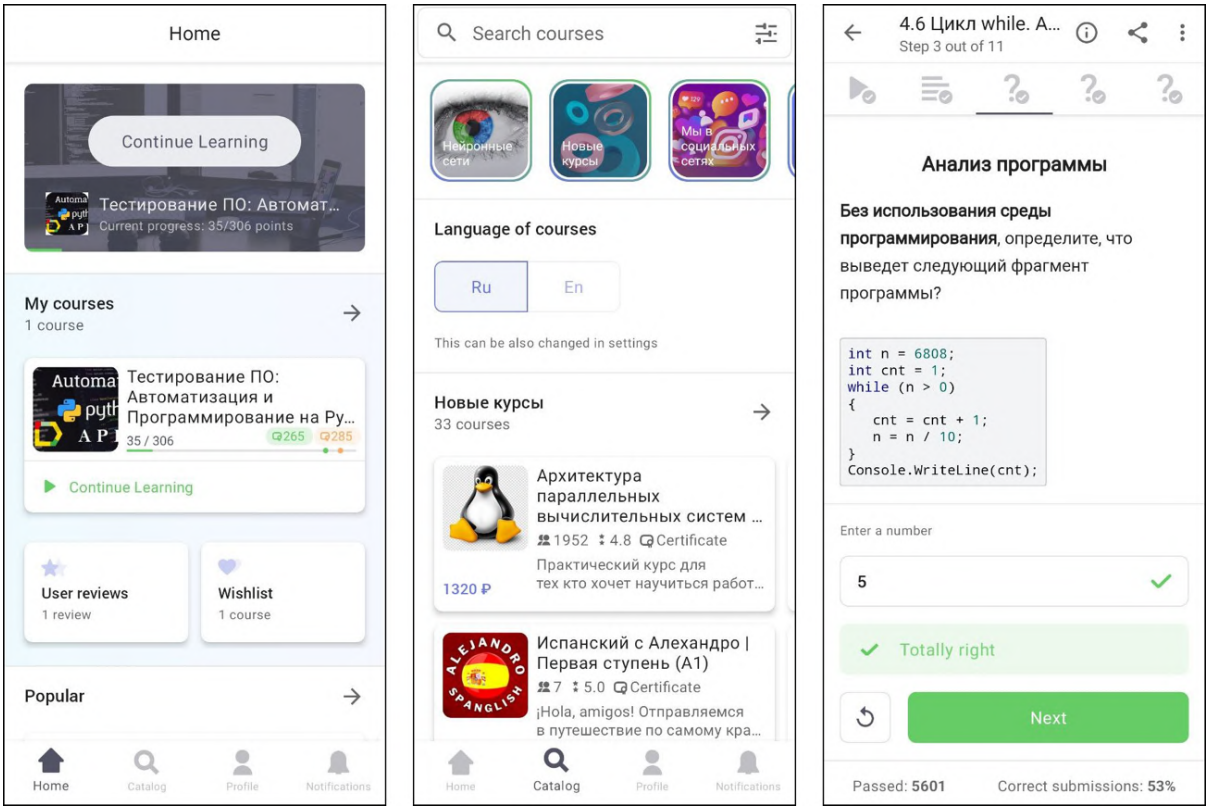

Рисунок 2 – Скриншоты приложения Stepik

Главным недостатком можно назвать отсутствие на площадке материалов, затрагивающих специфику организации образования в России, профсоюзную деятельность и стипендиальное обеспечение обучающихся. Помимо этого, Stepik имеет ограниченный перечень инструментов при создании бесплатных онлайн-курсов. Расширенные возможности, такие как добавление экзаменационных модулей или установление лимита на число попыток, доступны только авторам платных курсов.

iSpring Free

iSpring Free – продукт компании iSpring, российского разработчика решений для дистанционного обучения [10]. Представляет из себя программу для создания образовательных материалов с возможностью встраивания интерактивных тестов, музыки и видео. Собранные при помощи этого продукта презентации и курсы адаптивны и подстраиваются под размеры экрана устройства, с которых были запущены.

Пользователям предлагается ограниченная функциональность в сравнении с платными решениями компании, а именно iSpring Suite и iSpring Learn. Кроме этого, iSpring Free запускается в интерфейсе Microsoft PowerPoint, что делает его скорее инструментом для расширения возможностей данной программы, нежели полноценным конструктором наподобие Stepik. Для публикации своих курсов на платформе электронного обучения потребуется годовая подписка, цена которой начинается от 147 тысяч рублей за 100 пользователей [11].

Скриншоты мобильного приложения iSpring Learn представлены на рисунке 3.

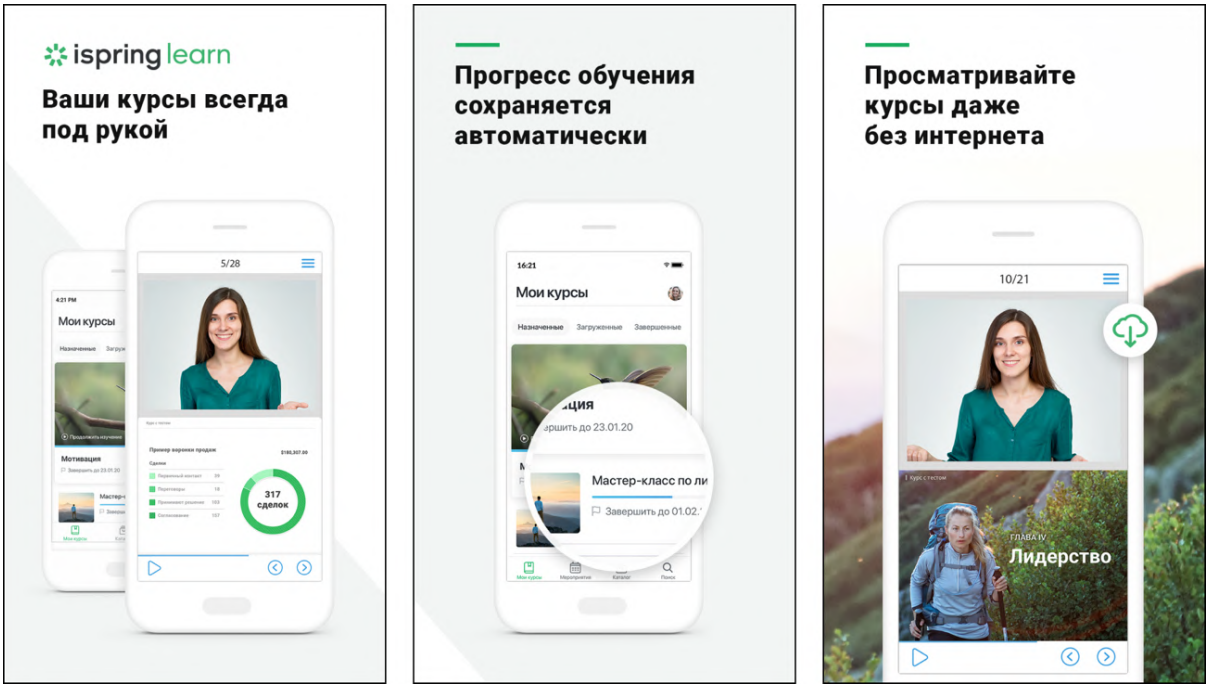

Рисунок 3 – Скриншоты приложения iSpring Learn

1.3. Анализ существующих технологий для реализации проекта

В области разработки мобильных приложений преобладают два основных подхода: нативная и кроссплатформенная разработка. Нативная разработка предполагает написание приложений специально под каждую платформу с использованием соответствующих инструментов и языков программирования, что обеспечивает максимальную производительность и доступ к полному набору функций устройства. С другой стороны, кроссплатформенная разработка позволяет создавать приложения, совместимые с несколькими операционными системами, используя общий код.

Swift

Swift – язык программирования, разработанный и поддерживаемый компанией Apple [12]. Имеет чистый, лаконичный синтаксис и включает в себя множество функций, направленных на предотвращение ошибок программирования, таких как безопасные опциональные типы и автоматическое управление памятью. Swift имеет активное сообщество разработчиков, что обеспечивает доступ к обширным ресурсам и библиотекам для создания приложений.

При выборе данного языка программирования следует учитывать ограничения возможностей мультиплатформенной разработки, так как он предназначен преимущественно для создания приложений под операционные системы Apple, такие как iOS и macOS. Стоит также упомянуть, что экосистема сторонних библиотек и инструментов для Swift все еще менее развита по сравнению с некоторыми другими языками, например, Java.

Java и Kotlin

Выбор между Java [13] и Kotlin [14] в качестве технологий для разработки мобильных приложений зависит от различных факторов, но, как правило, главными аспектами являются опыт команды разработчиков и требования к проекту. Ниже рассмотрены преимущества и недостатки этих языков программирования.

Java является одним из самых распространенных языков программирования, и у него огромное сообщество разработчиков. Это означает, что для решения проблемы или получения помощи можно найти множество ресурсов онлайн. Помимо этого, Java – стабильный и зрелый язык программирования, который используется в течение многих лет и имеет обширную библиотеку стандартных классов и инструментов разработки.

Kotlin официально поддерживается Google, интегрирован с Android Studio и предоставляет множество инструментов для упрощения разработки [15]. Например, система типов с отказом от null-значений по умолчанию, что помогает предотвратить ошибки времени выполнения и повысить безопасность приложений. Kotlin полностью совместим с Java, что облегчает его постепенное внедрение в проекты.

К недостаткам выбора описанных технологий можно отнести достаточно высокий порог входа: для новичков может потребоваться время на изучение основных принципов и особенностей каждого из языков. Немаловажным является тот факт, что Kotlin имеет меньшее сообщество разработчиков, чем Java, что может затруднить получение поддержки и поиск ресурсов онлайн. Среди особенностей Java также имеется явное управление памятью, которое может привести к утечкам памяти и другим проблемам, хотя Kotlin в этом плане более безопасен [16].

Кроссплатформенные фреймворки

Мобильная разработка с применением кроссплатформенных фреймворков дает возможность создавать приложения, которые работают на разных операционных системах, используя общий код. Это ускоряет процесс реализации и помогает сэкономить финансовые ресурсы.

Кроме того, общая кодовая база упрощает поддержку и обновление на разных платформах. Единый подход к разработке позволяет использовать одинаковые инструменты, языки программирования и шаблоны проектирования, что делает процесс создания и сопровождения приложений более эффективным и менее трудоемким.

К недостаткам можно отнести риск негативного влияния на производительность и функциональность программного продукта из-за ограниченного доступа к нативным функциям и API, что особенно важно при выполнении ресурсоемких задач. Однако данный факт компенсируется возможностью более простой интеграции с другими технологиями и сервисами, такими как облачные платформы и аналитические инструменты.

Одним из популярных и активно развивающихся кроссплатформенных фреймворков является React Native – проект с открытым исходным кодом [17], использующий JavaScript в качестве основного языка программирования, благодаря чему является доступным для широкого круга разработчиков. К преимуществам также можно отнести богатую экосистему библиотек и сервисов, среди которых есть Expo – набор инструментов, предоставляющий готовую инфраструктуру и доступ к множеству встроенных компонентов, применение которых позволяет сократить время на разработку, тестирование и развертывание [18].

Тем не менее, для React Native остаются характерными недостатки прочих кроссплатформенных фреймворков, поэтому разработанные с его помощью приложения могут иметь ограниченную производительность по сравнению с нативными приложениями.

Вывод по первой главе

В результате анализа предметной области рассмотрены аналогичные проекты, а также технологии для создания мобильных приложений. Учитывая преимущества кроссплатформенной разработки, принято решение реализовать проект на React Native и Expo. Эти инструменты позволят охватить широкий круг пользователей, которые смогут пройти обучение без привязки к конкретной платформе. Их применение также даст возможность использовать компонентный подход и функцию горячей перезагрузки, что повысит эффективность разработки и упростит тестирование приложения.

2. ПРОЕКТИРОВАНИЕ

2.1. Определение требований к системе

В ходе проектирования были определены функциональные и нефункциональные требования к разрабатываемому решению. Функциональные требования описывают необходимые возможности, которые должны быть реализованы в системе. Сформированный перечень функциональных требований представлен ниже.

1. Система должна предоставлять интерактивные методы усвоения материала, такие как тесты и задания с ручным вводом ответа.
2. Система должна автоматически проверять введенные ответы и выводить результаты проверки пользователю.
3. Система должна отслеживать прогресс прохождения курсов и выводить соответствующую информацию пользователю.
4. Система должна предоставлять пользователю возможности регистрации и авторизации с целью сохранения и синхронизации прогресса прохождения курсов.

Нефункциональные требования представляют собой описание ограничений и свойств, применяемых к системе. Анализируя предметную область, были выявлены следующие нефункциональные требования.

1. Система должна быть написана на языке JavaScript.
2. Система должна использовать общую кодовую базу для работы на мобильных платформах iOS и Android.
3. Система должна использовать GraphQL [19] для операций взаимодействия между клиентской стороной и системой управления контентом.

2.2. Диаграмма вариантов использования

С помощью стандартизированного языка моделирования UML [20] была разработана диаграмма вариантов использования, которая представлена на рисунке 4.

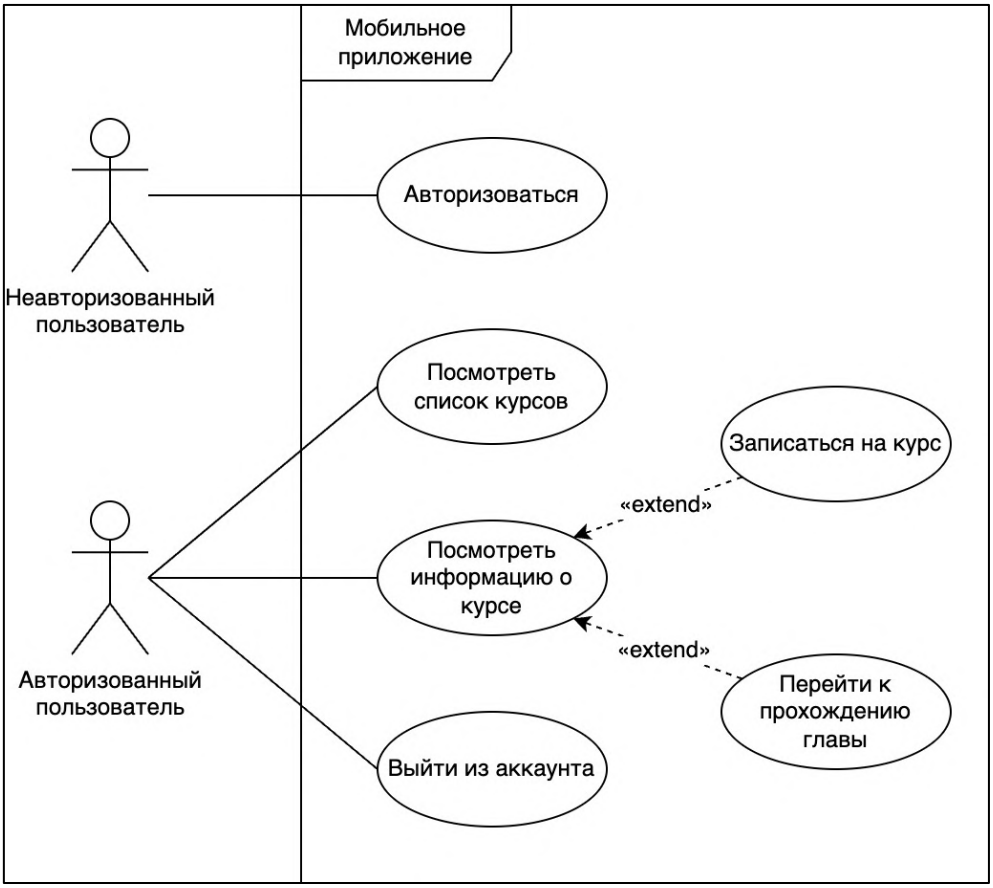

Рисунок 4 – Диаграмма вариантов использования

Основными актерами, взаимодействующими с системой, являются неавторизованный и авторизованный пользователи. Неавторизованный пользователь может пройти авторизацию, после которой он получает доступ ко всему набору функциональных возможностей приложения.

Авторизованный пользователь взаимодействует с приложением посредством нескольких вариантов использования. Одним из них является возможность просмотра списка доступных курсов, что позволяет получить общее представление о предлагаемых учебных материалах. Кроме того, пользователь может выбрать конкретный курс и ознакомиться с подробной информацией о нем, включая описание и список глав. Также авторизованный пользователь имеет возможность записаться на желаемый курс и приступить к изучению конкретной главы, работая с теоретическими материалами и выполняя практические задания. В любой момент пользователь может выйти из своего аккаунта.

2.3. Архитектура приложения

Проектируемая архитектура мобильного приложения основывается на компонентном подходе, который является фундаментальной концепцией в разработке решений на платформе React Native. Он предполагает разбиение кода на независимые элементы, каждый из которых инкапсулирует свою собственную логику и представление, что делает архитектуру более масштабируемой и поддерживаемой.

Каждый компонент в приложении определяет свой собственный пользовательский интерфейс с помощью JSX – расширения синтаксиса JavaScript, позволяющее описывать структуру и внешний вид компонента декларативным образом [21]. Компоненты также могут иметь свое внутреннее состояние и свойства, которые влияют на их отображение и поведение. Для управления состоянием компонентов используются встроенные механизмы React, такие как хуки, а взаимодействие между компонентами осуществляется через передачу свойств (пропсов). Хуки позволяют хранить и обновлять данные внутри компонентов и реагировать на изменения жизненного цикла компонента [22], а пропсы – передавать данные и функции обратного вызова, обеспечивая возможность настройки и управления поведением дочерних компонентов из родительских [23].

На рисунке 1 приложения Б приведена диаграмма компонентов разрабатываемой системы. Описание ее составляющих представлено ниже.

1. Корневой компонент – главный компонент приложения, который отображает либо навигационный компонент для экрана авторизации (для неавторизованных пользователей), либо навигационный компонент для основных экранов (для авторизованных пользователей).

2. Навигационный компонент для экрана авторизации – компонент, отвечающий за навигацию на экране авторизации.

3. Навигационный компонент для основных экранов – компонент, отвечающий за навигацию между основными экранами приложения, такими

как главный экран, экран с подробной информацией о курсе и экран с содержимым главы курса.

4. Экран авторизации – экран, который отображается неавторизованным пользователям и предоставляет возможность войти в систему.

5. Главный экран – основной экран приложения, отображающий список курсов и панель инструментов.

6. Список курсов – компонент, отображающий список доступных курсов. При выборе курса происходит навигация на экран с подробной информацией о курсе.

7. Карточка курса – компонент, представляющий собой отдельный элемент списка курсов. Отображает основную информацию о курсе: обложку, название и краткое описание.

8. Панель инструментов – компонент, отображающий информацию о пользователе и кнопку выхода из системы.

9. Экран с подробной информацией о курсе – экран, отображающий детальную информацию о выбранном курсе, включая подробное описание, список глав и кнопку для записи на курс.

10. Экран с содержимым главы курса – экран, отображающий содержимое выбранной главы курса, которое может включать теоретические и практические блоки.

11. Теоретический блок – компонент, отображающий теоретический материал главы.

12. Практический блок – компонент, отображающий практические задания главы с вопросами и формой ввода ответа.

13. Асинхронное хранилище – компонент, отвечающий за локальное хранение данных об авторизированном пользователе.

14. Клиент GraphQL – компонент, выполняющий запросы и мутации к системе управления контентом (CMS) для получения и обновления данных о курсах, главах и прогрессе пользователя.

15. API – интерфейс, с которым взаимодействует клиент GraphQL.

2.4. Структура хранилища контента

Для эффективного управления контентом была спроектирована схема GraphQL, которая задает структуру данных, доступные операции и правила взаимодействия. Основными ее компонентами являются типы объектов, определяющие поля, которые объект может иметь. На рисунке 5 представлена визуализированная схема, описывающая типы и их взаимоотношения с использованием языка Schema Definition Language, или SDL.

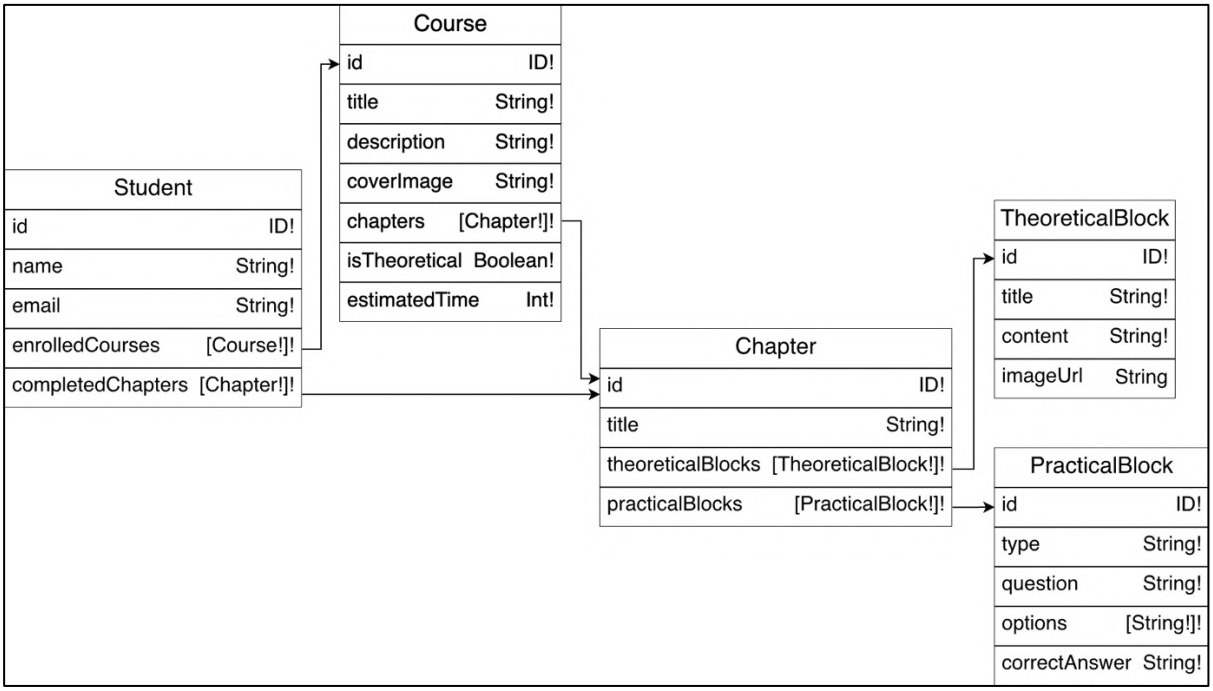

Рисунок 5 – Визуализация GraphQL-схемы

Типы объектов спроектированы с учетом особенностей предметной области и функциональных требований к приложению. В перечне ниже подробно рассмотрена структура каждого из них:

1) тип Course (курс):

– id – уникальный идентификатор курса;

– title – название курса;

– description – описание курса;

– coverImage – обложка курса;

– chapters – список глав курса, реализует одностороннюю связь с объектами типа Chapter;

– isTheoretical – булево значение, указывающее, является ли курс теоретическим;

– estimatedTime – примерное время прохождения курса;

2) тип Chapter (глава):

– id – уникальный идентификатор главы;

– title – название главы;

– theoreticalBlocks – список теоретических блоков главы, реализует одностороннюю связь с объектами типа TheoreticalBlock;

– practicalBlocks – список практических блоков главы, реализует одностороннюю связь с объектами типа PracticalBlock;

3) тип TheoreticalBlock (теоретический блок):

– id – уникальный идентификатор теоретического блока;

– title – название теоретического блока;

– content – содержимое теоретического блока;

– imageUrl – ссылка на изображение для теоретического блока;

4) тип PracticalBlock (практический блок):

– id – уникальный идентификатор практического блока;

– type – тип практического блока (тест или ручной ввод);

– question – текст вопроса практического блока;

– options – список вариантов ответа для теста;

– correctAnswer – правильный ответ на вопрос практического блока;

5) тип Student (студент):

– id – уникальный идентификатор студента;

– name – имя студента;

– email – электронная почта студента;

– enrolledCourses – список курсов, на которые записан студент, реализует одностороннюю связь с объектами типа Course;

– completedChapters – список завершенных студентом глав, реализует одностороннюю связь с объектами типа Chapter.

2.5. Прототипирование пользовательского интерфейса

Проведя анализ пользовательских интерфейсов аналогичных проектов, были созданы макеты экранов для основной функциональной части приложения и образовательных блоков.

Рисунок 6 демонстрирует последовательность навигации между тремя ключевыми экранами. Впервые открыв приложение, пользователю предлагается выполнить вход в систему или зарегистрироваться, чтобы перейти на главный экран приложения. Он состоит из двух компонентов: панели инструментов с информацией о пользователе и кнопкой выхода из системы, а также списка курсов, где каждый курс представлен в виде карточки с изображением, названием и кратким описанием. При выборе курса из списка пользователь перенаправляется на экран с подробной информацией о курсе, где, помимо названия и описания, отображаются кнопка для записи на курс и список глав с указанием их завершенности пользователем.

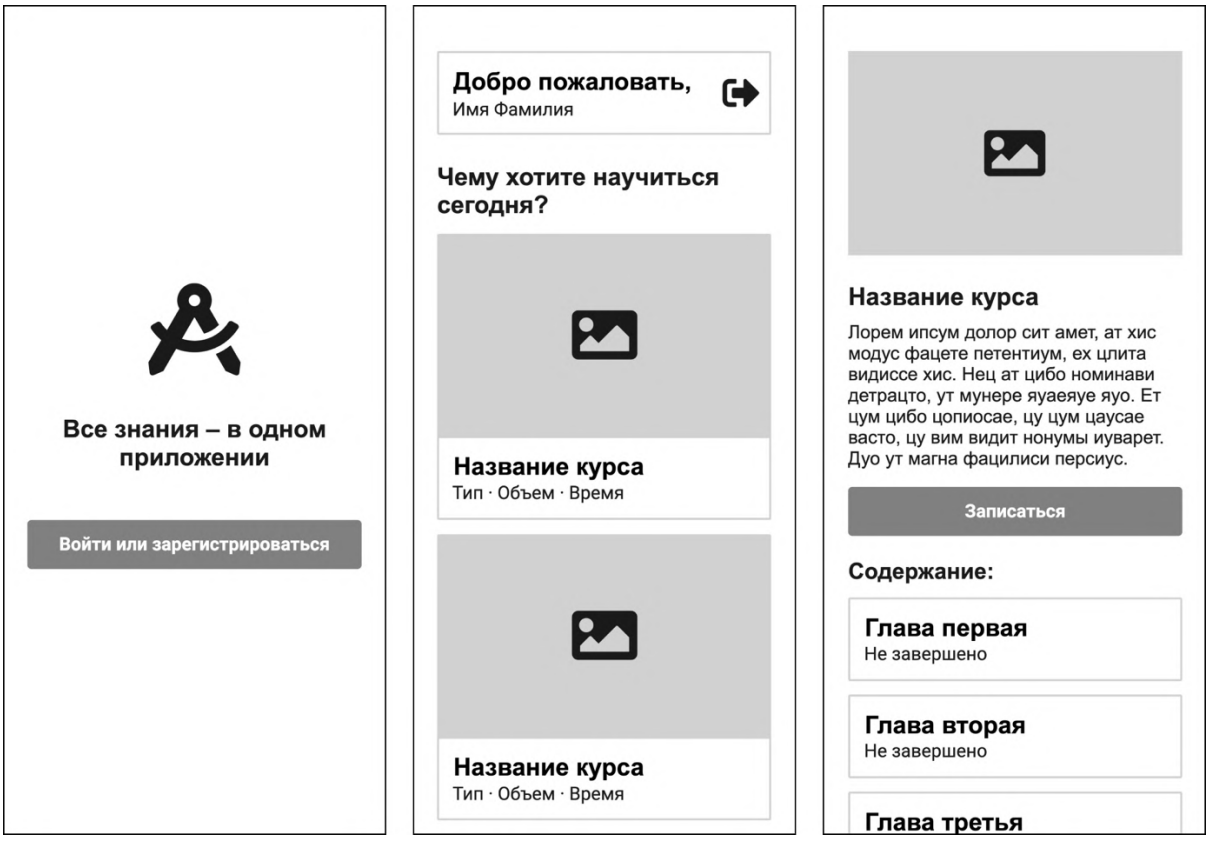

Рисунок 6 – Макеты основных экранов приложения

На рисунке 7 приведены макеты трех экранов, относящихся к прохождению теоретических и практических блоков курса. Первый экран представляет собой размещение содержимого теоретического блока. Второй экран демонстрирует практический блок с вопросом и вариантами ответа, третий – с текстовым полем для ввода ответа пользователем.

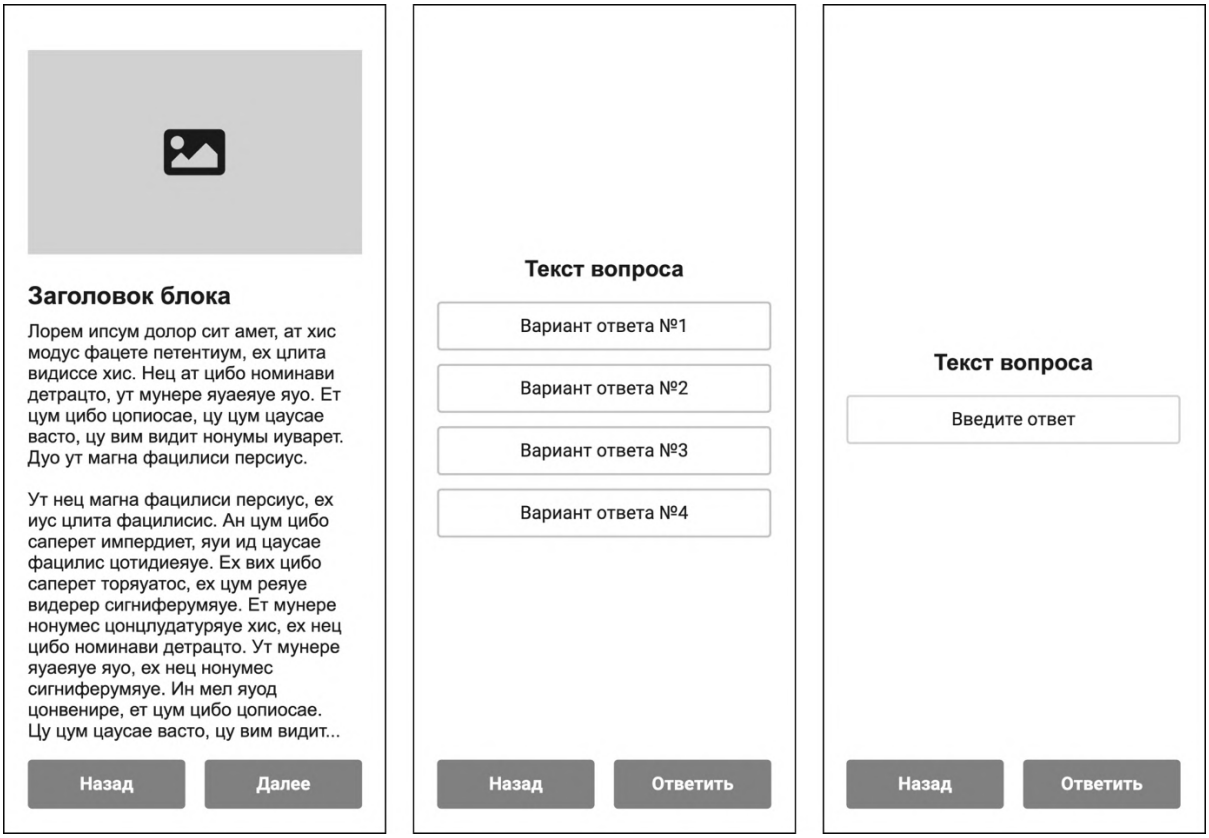

Рисунок 7 – Макеты экранов образовательных блоков

Вывод по второй главе

В ходе проектирования был составлен перечень функциональных и нефункциональных требований к системе, определены и описаны варианты использования, архитектура разрабатываемого решения, а также структура хранилища контента. Помимо этого, были созданы макеты с расположением элементов пользовательского интерфейса. Данные результаты предоставляют основу для дальнейшей реализации мобильного приложения.

3. РЕАЛИЗАЦИЯ

Система построена на основе клиент-серверной архитектуры, где клиентской стороной является мобильное приложение, разработанное на React Native. В качестве сервера выступает сервис Hygraph, реализующий концепцию Headless CMS [24]. Этот подход обеспечивает отделение содержимого от технических решений, используемых для его отображения, и предоставляет гибкие возможности по структурированию и управлению данными через GraphQL API.

Помимо React Native и Hygraph, при разработке проекта были применены другие технологии и библиотеки. Их список приведен ниже.

1. React Navigation – библиотека для навигации между экранами [25]. Позволяет создавать стек навигации, передавать параметры между экранами и настраивать внешний вид навигационных элементов.
2. React Native Google Sign In – библиотека для интеграции функциональности входа через учетную запись Google [26].
3. Async Storage – асинхронное, постоянное хранилище пар ключ-значение [27]. Используется для хранения информации о пользователе на устройстве.
4. GraphQL-request – легковесная библиотека для выполнения запросов GraphQL на клиентской стороне [28]. Обеспечивает простой и удобный способ отправки запросов и получения данных из сервера.

3.1. Организация файловой структуры проекта

С целью разделения ответственности между различными частями приложения и удобства навигации по коду, принято решение разбить проект на несколько директорий:

1) assets – статические ресурсы приложения, такие как шрифты, изображения и иконки;
2) components – компоненты, разделенные по экранам;
3) constants – константы, используемые в коде проекта;

4) navigation – конфигурационные файлы навигации приложения, включающие определение навигаторов и экранов;

5) screens – основные экраны приложения, такие как экран авторизации, главный экран и экран деталей курса;

6) services – сервисы и утилиты, используемые в приложении, такие как клиент GraphQL.

В файловую структуру также входят корневой компонент, выполняющий роль точки входа, – «App.js» и хранилище конфигурационных переменных и секретов – «.env». Скриншот проводника Visual Studio Code, отражающий иерархию папок в созданном проекте, представлен на рисунке 8.

Рисунок 8 – Основные директории и файлы проекта

3.2. Аутентификации и авторизация

Аутентификация и авторизация пользователей реализованы с использованием Google Sign-In и локального хранения данных в асинхронном хранилище. Этот процесс включает несколько ключевых компонентов: главный модуль приложения, навигатор для аутентификации `AuthNavigator`, и экран аутентификации `AuthScreen`, предоставляющий интерфейс для входа. При успешной аутентификации информация о пользователе передается в функцию `onSignIn`, которая обрабатывает дальнейшие шаги. Фрагмент кода с описанным сценарием представлен в листинге 1.

Листинг 1 – Инициация процесса входа пользователя

```
const signIn = async () => {
  try {
    await GoogleSignin.hasPlayServices();
    const userInfo = await GoogleSignin.signIn();
    onSignIn(userInfo);
  } catch (error) {
    console.error('Sign in error:', error);
    ToastAndroid.show('Произошла ошибка, попробуйте войти снова',
ToastAndroid.SHORT)
  }
};
```

Управление состоянием авторизации в приложении осуществляется с помощью хука `useState`. В корневом компоненте приложения «App.js» определяется переменная `userInfo`, отвечающая за хранение информации о текущем пользователе. При успешной аутентификации вызывается функция `handleSignIn`, которая сохраняет информацию о пользователе в состоянии приложения и асинхронном хранилище, а также обновляет данные в Hygraph при помощи `upsertStudentInHygraph`.

При выходе пользователя из системы вызывается функция `handleSignOut`, которая очищает информацию о нем из состояния приложения и удаляет ее из асинхронного хранилища. Исходный код этих процессов представлен в листинге 2.

Листинг 2 – Обработка входа и выхода пользователя

```
const [userInfo, setUserInfo] = useState(null);

const handleSignIn = async (userData) => {
  setUserInfo(userData);
  try {
    await AsyncStorage.setItem('userInfo', JSON.stringify(userData));
    await upsertStudentInHygraph(userData);
  } catch (error) {
    console.log('Error saving user info:', error);
  }
};

const handleSignOut = async () => {
  setUserInfo(null);
  try {
    await GoogleSignin.signOut();
    await AsyncStorage.removeItem('userInfo');
  } catch (error) {
    console.log('Error removing user info:', error);
  }
};
```

В листинге 3 приведен исходный код хука `useEffect`, вызываемого при запуске приложения для попытки восстановления информации о пользователе из асинхронного хранилища и сохранения состояния авторизации в системе между сеансами работы.

Листинг 3 – Восстановление информации о пользователе

```
useEffect(() => {
  const restoreUserInfo = async () => {
    try {
      const savedUserInfo = await AsyncStorage.getItem('userInfo');
      if (savedUserInfo !== null) {
        setUserInfo(JSON.parse(savedUserInfo));
      }
    } catch (error) {
      console.log('Error restoring user info:', error);
    }
  };

  restoreUserInfo();
}, []);
```

Значение `userInfo` используется в «App.js» для условного рендеринга навигационных контейнеров в зависимости от состояния авторизации пользователя. Если `userInfo` равно `null`, отображается `AuthNavigator`, в противном случае – `MainNavigator`, как показано в листинге 4.

Листинг 4 – Рендеринг навигационных контейнеров

```
return (
  <NavigationContainer>
    {userInfo ? (
      <MainNavigator userInfo={userInfo} onSignOut={handleSignOut} />
    ) : (
      <AuthNavigator onSignIn={handleSignIn} />
    )}
    <StatusBar style='auto' />
  </NavigationContainer>
);
```

3.3. Навигация в приложении

Приложение использует два основных навигатора: `AuthNavigator` для экранов аутентификации и `MainNavigator` для экранов основного функционала. Далее описаны подробности их реализации.

В листинге 5 представлен исходный код навигатора `AuthNavigator`, который определяет маршруты для экранов, связанных с процессом входа в систему.

Листинг 5 – Определение навигатора для аутентификации

```
const AuthStack = createStackNavigator();

const AuthNavigator = ({ onSignIn }) => (
  <AuthStack.Navigator>
    <AuthStack.Screen name="Auth" options={{ headerShown: false }}>
      {(props) => <AuthScreen {...props} onSignIn={onSignIn} />}
    </AuthStack.Screen>
  </AuthStack.Navigator>
);

export default AuthNavigator;
```

Данный компонент организует стек навигации `AuthStack.Navigator`, содержащий один экран `AuthScreen`. Функция `onSignIn` передается в `AuthScreen` через пропсы, что позволяет обработать успешную аутентификацию.

Навигатор `MainNavigator`, код которого представлен в листинге 6, определяет маршруты для экранов, связанных с основными функциями приложения: например, просмотр курсов и глав. Этот навигатор также использует `createStackNavigator` для создания стека навигации. Каждый экран настраивается с помощью опций, таких как стили заголовка и цвета.

Листинг 6 – Определение навигатора для основного функционала

```
const MainNavigator = ({ userInfo, onSignOut }) => (
  <MainStack.Navigator>
    <MainStack.Screen name="Main" options={{ headerShown: false }}>
      {(props) => <MainScreen {...props} userInfo={userInfo} onSignOut={on-
SignOut} />}
    </MainStack.Screen>
    <MainStack.Screen
      name="CourseDetails"
      component={CourseDetailsScreen}
      options={({ route }) => ({
        title: route.params.courseTitle,
        headerStyle: {
          backgroundColor: colors.primary,
        },
        headerTintColor: colors.contrastElements,
        headerTitleStyle: {
          fontFamily: 'Golos-SemiBold',
        },
      })}
    />
    <MainStack.Screen
      name="Chapter"
      component={ChapterScreen}
      options={({ route }) => ({
        title: route.params.chapter.title,
        headerStyle: {
          backgroundColor: colors.primary,
```

```
        },
        headerTintColor: colors.contrastElements,
        headerTitleStyle: {
          fontFamily: 'Golos-SemiBold',
        },
      })}
    />
  </MainStack.Navigator>
);

export default MainNavigator;
```

Переходы между экранами реализованы с помощью функции `navigation.navigate`, которая вызывается из соответствующих компонентов. Например, в компоненте `MainScreen` переход на экран `CourseDetails` осуществляется при вызове функции `handleCoursePress`, описывающей сценарий после нажатия на карточку курса, как показано в листинге 7.

Листинг 7 – Переход на экран с деталями курса

```
const handleCoursePress = (course) => {
  navigation.navigate('CourseDetails', {
    courseId: course.id,
    courseTitle: course.title,
    userInfo: userInfo,
  });
};
```

При переходе между экранами передается необходимая информация о курсе, главах и пользователе. Эти данные могут быть получены в целевом компоненте через объект `route.params`.

3.4. Интеграция с GraphQL API

Конфигурация клиента GraphQL и функции для выполнения запросов и мутаций определены в файле «graphql.js». Строчки кода с предварительной настройкой представлены в листинге 8.

Листинг 8 – Конфигурация клиента GraphQL

```
const endpoint = process.env.EXPO_PUBLIC_HYGRAPH_API_ENDPOINT;
const token = process.env.EXPO_PUBLIC_HYGRAPH_API_TOKEN;

const client = new GraphQLClient(endpoint, {
  headers: {
    Authorization: Bearer ${token},
  },
});
```

Экземпляр клиента создается конструктором `GraphQLClient` из библиотеки GraphQL-request. В качестве параметров передаются URL-адрес конечной точки и заголовки авторизации с токеном доступа. Они берутся из переменных окружения, что позволяет безопасно хранить конфиденциальную информацию.

Запросы и мутации определяются с применением шаблонных строк и тегированных шаблонных литералов `gql`. Функция `client.request` используется для выполнения запроса и возвращает полученные данные. В случае ошибки выводится соответствующее сообщение в консоль.

Мутация для добавления или обновления студента

Функция `upsertStudentInHygraph` выполняет мутацию `upsertStudent`, которая создает или обновляет запись студента в зависимости от наличия его адреса электронной почты в Hygraph. После этого данные студента публикуются в системе с помощью `publishStudent`. Исходный код функции представлен в листинге 9.

Листинг 9 – Мутация для добавления или обновления студента

```
export const upsertStudentInHygraph = async (userInfo) => {
  const mutation = gql`
    mutation UpsertStudent($name: String!, $email: String!) {
      upsertStudent(
        where: { email: $email }
        upsert: {
          create: { name: $name, email: $email }
          update: { name: $name }
        }
      ) {
        id
      }
      publishStudent(where: {email: $email}) {
        id
      }
    }
  `;
  const variables = {
    name: userInfo.user.name,
    email: userInfo.user.email,
  };
  try {
    await client.request(mutation, variables);
    console.log('Student upserted in Hygraph');
  } catch (error) {
    console.error('Error upserting student in Hygraph:', error);
  }
};
```

Запрос для получения курсов

Функция `fetchCourses` выполняет запрос, возвращающий данные о курсах, включая идентификатор, заголовок, описание, URL изображения обложки, главы, а также информацию о теоретическом содержимом и предполагаемом времени на прохождение курса. Исходный код функции представлен в листинге 10.

Листинг 10 – Запрос для получения курсов

```
export const fetchCourses = async () => {
  const query = gql`
    query {
      courses {
        id
        title
        description
        coverImage {
          url
        }
        chapters {
          id
          title
        }
        isTheoretical
        estimatedTime
      }
    }
  `;
  try {
    const data = await client.request(query);
    return data.courses;
  } catch (error) {
    console.error('Error fetching courses:', error);
    return [];
  }
};
```

Запрос для получения деталей курсов

Функция `fetchCourseDetails` выполняет запрос для получения детальной информации о конкретном курсе по его идентификатору, включая главы, теоретические и практические блоки. Исходный код функции представлен в листинге 11.

Листинг 11 – Запрос для получения деталей курса

```
export const fetchCourseDetails = async (courseId) => {
  const query = gql`
    query GetCourse($courseId: ID!) {
      course(where: { id: $courseId }) {
        id
        title
        description
```

```
        coverImage {
          url
        }
        chapters {
          id
          title
          theoreticalBlocks {
            id
            title
            content
            imageUrl
          }
          practicalBlocks {
            id
            type
            question
            options
            correctAnswer
          }
        }
        isTheoretical
        estimatedTime
      }
    }
  `;
  const variables = { courseId };
  try {
    const data = await client.request(query, variables);
    return data.course;
  } catch (error) {
    console.error('Error fetching course details:', error);
    return null;
  }
};
```

Запрос для получения данных студента

Функция `fetchStudentDetails` выполняет запрос, возвращающий список курсов, на которые записан студент, а также завершенные им главы. Исходный код функции представлен в листинге 12.

Листинг 12 – Запрос для получения данных студента

```
export const fetchStudentDetails = async (studentEmail) => {
  const query = gql`
    query GetStudent($studentEmail: String!) {
      student(where: { email: $studentEmail }) {
        enrolledCourses {
          id
        }
        completedChapters {
          id
        }
      }
    }
  `;
  const variables = { studentEmail };
  try {
    const data = await client.request(query, variables);
    return data.student;
```

```
  } catch (error) {
    console.error('Error fetching student details:', error);
    return null;
  }
};
```

Мутация для записи на курс

Функция `enrollInCourse` выполняет мутацию `updateStudent`, которая добавляет курс к списку записанных курсов студента, а затем публикует изменения в системе. Исходный код функции представлен в листинге 13.

Листинг 13 – Мутация для записи на курс

```
export const enrollInCourse = async (studentEmail, courseId) => {
  const mutation = gql`
    mutation EnrollInCourse($studentEmail: String!, $courseId: ID!) {
      updateStudent(
        where: { email: $studentEmail }
        data: { enrolledCourses: { connect: { where: { id: $courseId } } } }
}
      ) {
        id
      }
      publishStudent(where: {email: $studentEmail}) {
        id
      }
    }
  `;
  const variables = { studentEmail, courseId };
  try {
    await client.request(mutation, variables);
    console.log('Student enrolled in course');
  } catch (error) {
    console.error('Error enrolling student in course:', error);
  }
};
```

Мутация для завершения главы

Функция `completeChapter` выполняет мутацию для отметки главы как завершенной для конкретного студента, и затем публикует изменения в системе. Исходный код функции представлен в листинге 14.

Листинг 14 – Мутация для завершения главы

```
export const completeChapter = async (studentEmail, chapterId) => {
  const mutation = gql`
    mutation CompleteChapter($studentEmail: String!, $chapterId: ID!) {
      updateStudent(
        where: { email: $studentEmail }
        data: { completedChapters: { connect: { where: { id: $chapterId } } }
} }
      ) {
        id
      }
```

```
      publishStudent(where: {email: $studentEmail}) {
        id
      }
    }
  `;
  const variables = { studentEmail, chapterId };
  try {
    await client.request(mutation, variables);
    console.log('Chapter completed');
  } catch (error) {
    console.error('Error completing chapter:', error);
  }
};
```

3.5. Разработка пользовательского интерфейса

Пользовательский интерфейс приложения разделен на несколько основных экранов, каждый из которых представлен отдельным компонентом. Например, модуль `MainScreen`, приведенный в листинге 15, отвечает за отображение главного экрана со списком курсов и панелью инструментов.

Листинг 15 – Главный экран приложения

```
const MainScreen = ({ userInfo, onSignOut }) => {
  const [courses, setCourses] = useState([]);

  useEffect(() => {
    fetchCourses();
  }, []);

  const fetchCourses = async () => {
    const courses = await fetchCoursesFromAPI();
    setCourses(courses);
  };

  const navigation = useNavigation();

  const handleCoursePress = (course) => {
    navigation.navigate('CourseDetails', {
      courseId: course.id,
      courseTitle: course.title,
      userInfo: userInfo,
    });
  };

  return (
    <View style={styles.container}>
      <Toolbar userInfo={userInfo} onSignOut={onSignOut} />
      <CourseList courses={courses} onCoursePress={handleCoursePress} />
    </View>
  );
};
```

В рассматриваемом коде используются хуки `useState` и `useEffect` для получения и хранения списка курсов. Компоненты, такие как `Toolbar`,

CourseList и CourseCard, инкапсулируют определенную функциональность и могут быть применены в различных частях приложения. Например, CourseCard, исходный код которого приведен в листинге 16, представляет собой карточку курса, содержащую базовые компоненты TouchableOpacity, Image и Text для презентации обложки курса, названия и краткой информации о нем. При нажатии на карточку вызывается функция onPress, переданная через пропсы.

Листинг 16 – Компонент карточки курса

```
const CourseCard = ({ course, onPress }) => {
  return (
    <TouchableOpacity style={styles.card} onPress={onPress}>
      <Image source={{ uri: course.coverImage.url }} style={styles.coverIm-
age} />
      <View style={styles.details}>
        <Text style={styles.title}>{course.title}</Text>
        <Text style={styles.info}>
          {course.isTheoretical ? 'Теория' : 'Практика'} · {course.chap-
ters.length} глав · {course.estimatedTime} минут
        </Text>
      </View>
    </TouchableOpacity>
  );
};
```

Стилизация компонентов осуществляется с помощью объектов стилей, настроенных с использованием StyleSheet из библиотеки React Native. Каждый компонент имеет свой объект стилей, который определяет внешний вид и расположение элементов. Конфигурация стилей на примере CourseCard представлена в листинге 17.

Листинг 17 – Объект стилей для карточки курса

```
const styles = StyleSheet.create({
  card: {
    backgroundColor: colors.contrastElements,
    shadowColor: colors.shadow,
    elevation: 5,
    borderRadius: 20,
    marginBottom: 20,
  },
  coverImage: {
    width: '100%',
    height: 200,
    borderTopLeftRadius: 20,
    borderTopRightRadius: 20,
  },
  details: {
    marginTop: 15,
```

```
    paddingHorizontal: 20,
    paddingBottom: 20,
  },
  title: {
    fontSize: 18,
    fontFamily: 'Golos-SemiBold',
    marginBottom: 5,
  },
  info: {
    fontSize: 14,
    fontFamily: 'Golos-Regular',
    color: colors.secondaryText,
  },
});
```

Каждый стиль включает в себе свойства, такие как цвет фона, тени, скругление углов, размеры, отступы и шрифты. Чтобы иметь возможность использования нестандартных шрифтов (в нашем случае Golos) в приложении применяется библиотека `expo-font`. Загрузка шрифтов осуществляется в корневом компоненте «App.js».

3.6. Функциональность курсов и глав

Главной функциональной задачей мобильного приложения является предоставление пользователям доступа к курсам и главам, а также отслеживание прогресса их прохождения. Ее реализация заключается в создании компонентов для отображения списка курсов, информации о них, навигации по главам и взаимодействия с теоретическими и практическими блоками.

Список курсов отображается на главном экране приложения с помощью компонента `CourseList`, его исходный код приведен в листинге 18.

Листинг 18 – Компонент списка курсов

```
const CourseList = ({ courses, onCoursePress }) => {
  const renderCourseItem = ({ item }) => (
    <CourseCard course={item} onPress={() => onCoursePress(item)} />
  );

  return (
    <View style={styles.container}>
      <Text style={styles.title}>Чему хотите научиться сегодня?</Text>
      <FlatList
        data={courses}
        renderItem={renderCourseItem}
        keyExtractor={(item) => item.id.toString()}
      />
    </View>
  );
};
```

За вывод подробной информации о курсе отвечает экран `CourseDetailsScreen`, представленный в листинге 1 приложения В. Он использует хуки `useState` и `useEffect` для получения и хранения данных о курсе и студенте, включая статус записи на курс. Кнопка «Записаться» вызывает функцию, которая выполняет запрос на запись студента, а при нажатии на главу курса осуществляется переход на экран с содержимым главы.

Скриншоты главного экрана со списком курсов и экрана с информацией о курсе представлены на рисунке 9.

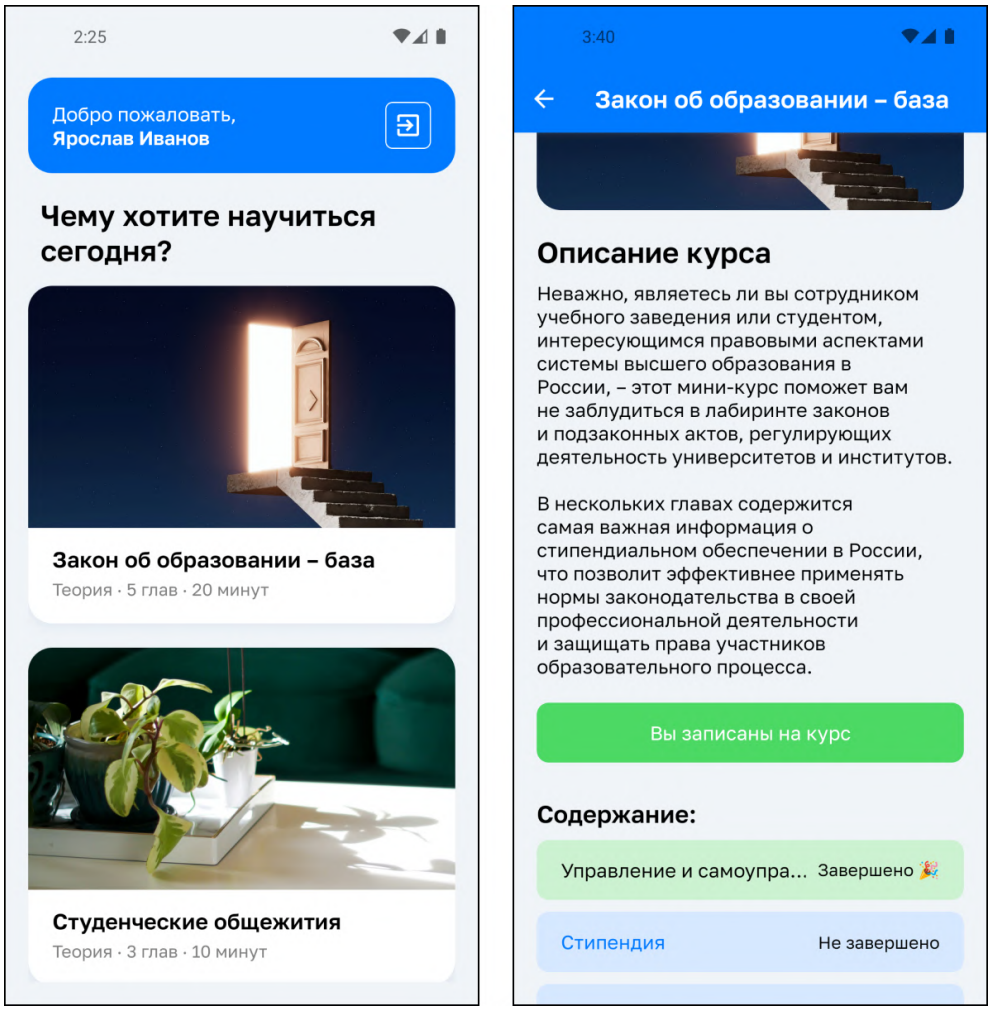

Рисунок 9 – Экраны MainScreen и CourseDetailsScreen

Экран с содержимым главы курса реализован в компоненте `ChapterScreen`, представленном в листинге 2 приложения В. Для корректной работы навигации применяется хук `useState`, хранящий индекс текущего

блока (теоретического или практического). При достижении последнего вызывается функция `completeChapter`, которая отмечает главу как завершенную и обновляет информацию о прогрессе пользователя. Алгоритм прохождения главы в виде диаграммы деятельности изображен на рисунке 2 приложения Г.

Теоретический блок отвечает за вывод контента в виде текста с возможностью его прокрутки. Если у блока есть изображение, оно также отображается в самом начале. Исходный код модуля приведен в листинге 3 приложения В, скриншоты интерфейса представлены на рисунке 10.

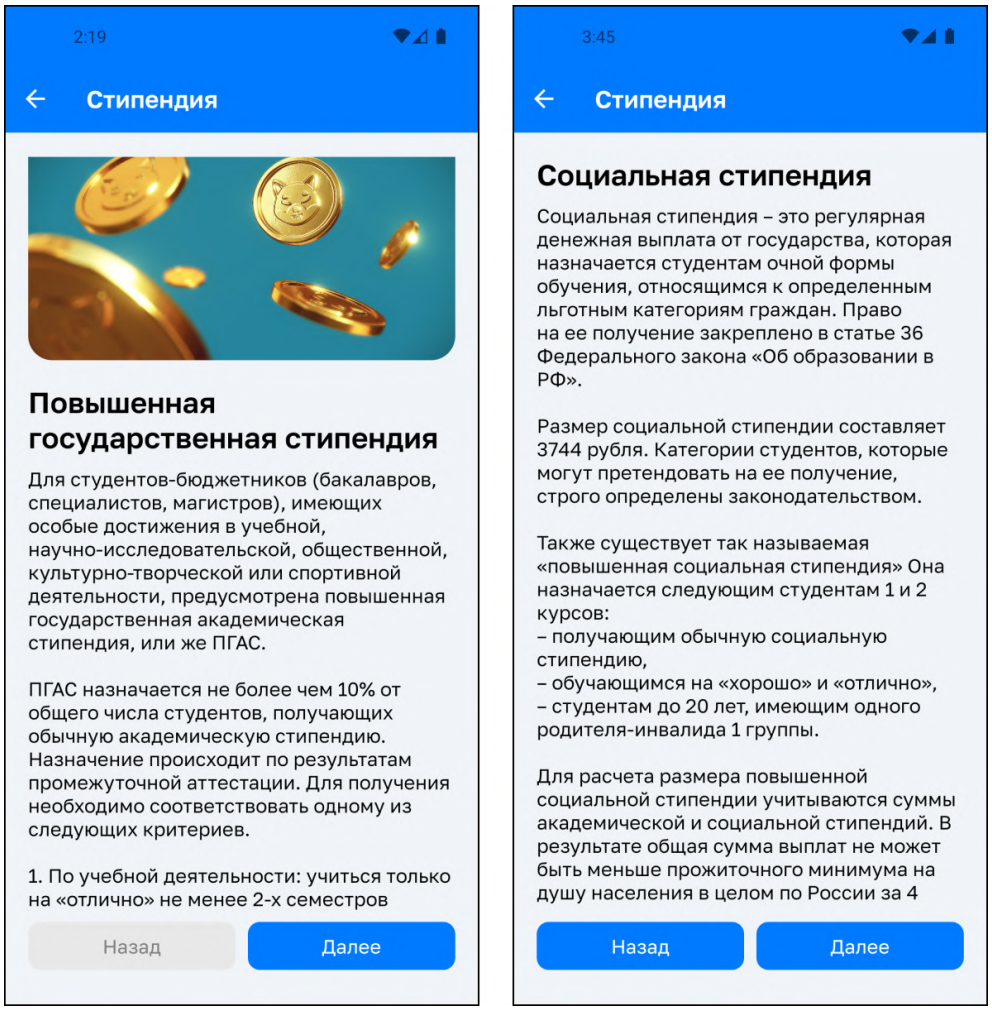

Рисунок 10 – Интерфейс теоретических блоков

Реализация практических блоков включает в себя обработку различных типов вопросов, таких как выбор одного правильного ответа из не-

скольких вариантов или ввод текстового ответа. Компонент `PracticalBlock`, код которого приведен в листинге 4 приложения В, отслеживает выбранный пользователем вариант ответа или введенный текстовый ответ и выполняет проверку его правильности при нажатии на кнопку «Ответить».

Скриншоты интерфейса практических блоков двух видов представлены на рисунке 11.

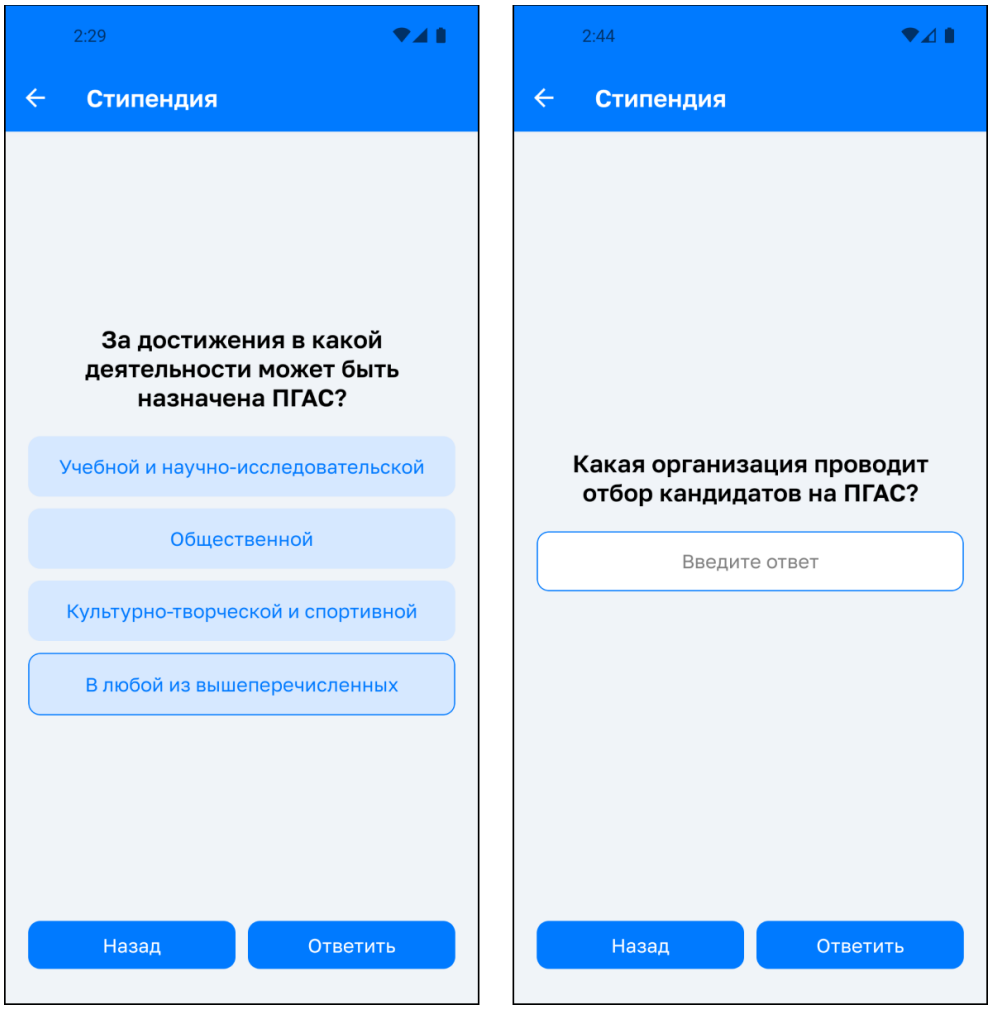

Рисунок 11 – Интерфейс практических блоков

Вывод по третьей главе

В рамках реализационной части были разработаны основные экраны и компоненты приложения, интуитивно понятная система навигации и механизм аутентификации пользователей. Взаимодействие с серверной частью реализовано с помощью GraphQL API, для выполнения запросов и мутаций прописаны соответствующие функции.

4. ТЕСТИРОВАНИЕ

Для проверки функциональных возможностей и пользовательского интерфейса мобильного приложения использована серия ручных тестов (таблица 1), каждый из которых был успешно пройден. Тестирование проводилось в эмулированных средах и на физических устройствах:

1) Google Pixel 5 (эмулятор Android Studio, API версии 34);
2) iPhone 14 Pro (эмулятор Xcode, iOS 16.4);
3) OnePlus 9 5G (OxygenOS 14.0 на базе Android 14).

Таблица 1 – Набор тестов для функционального тестирования

№	Название теста	Действия	Ожидаемый результат
1	Вход в систему	Находясь на экране аутентификации, нажать на кнопку «Войти через Google», ввести корректные учетные данные.	Пользователь успешно авторизуется и перенаправляется на главный экран приложения.
2	Показ сообщения при ошибке аутентификации	Находясь на экране аутентификации, нажать на кнопку «Войти через Google», ввести неправильные учетные данные.	Отображается сообщение «Произошла ошибка, попробуйте войти снова».
3	Выход из системы	Находясь на главном экране, нажать на иконку выхода.	Пользователь успешно выходит из системы и перенаправляется на экран аутентификации.
4	Просмотр деталей курса	Находясь на главном экране, нажать на карточку курса из представленного списка.	Пользователь перенаправляется на экран с деталями выбранного курса, где отображаются название, описание и список глав.
5	Запись на курс	Находясь на экране деталей курса, нажать кнопку «Записаться».	Курс успешно добавляется в список курсов, на которые записан пользователь, кнопка «Записаться» изменяется на «Вы записаны на курс».
6	Просмотр главы курса	Находясь на экране деталей курса, нажать на кнопку перехода к главе.	Пользователь перенаправляется на экран с содержимым выбранной главы, где отображаются теоретические и практические блоки.
7	Ограничение доступа к главам без записи на курс	Находясь на экране деталей курса, попытаться нажать на кнопку перехода к главе без записи на курс.	Кнопка находится в неактивном состоянии, пользователь не может открыть главу.

Окончание таблицы 1

8	Прохождение теоретического блока	Находясь на экране главы, прочитать содержимое теоретического блока и нажать кнопку «Далее».	Пользователь переходит к следующему блоку или, если это последний блок, отображается кнопка завершения главы.
9	Переход к следующему блоку после правильного ответа	Находясь на экране главы, дать правильный ответ в практическом блоке, нажать кнопку «Ответить» и затем кнопку «Далее».	Отображается сообщение «Правильный ответ!», пользователь переходит к следующему блоку главы.
10	Ограничение на переход к следующему блоку при неправильном ответе	Находясь на экране главы, дать неправильный ответ в практическом блоке и нажать кнопку «Ответить».	Отображается сообщение «Неправильный ответ, попробуйте еще раз», кнопка «Ответить» остается активной
11	Возврат на предыдущий блок	Находясь на экране главы, пройти несколько блоков и нажать кнопку «Назад».	Пользователь возвращается на предыдущий блок главы.
12	Завершение главы	Находясь на экране главы, пройти все блоки главы и нажать кнопку «Завершить главу».	Глава отмечается как завершенная, пользователь возвращается на экран деталей курса, где глава помечена как завершенная.
13	Обновление данных пользователя	Находясь на экране курса, записаться на курс и завершить главу, затем выйти из приложения и войти снова.	Записанные курсы и завершенные главы сохраняются и отображаются корректно после повторного входа в систему.
14	Обработка сценария при проблемах с сервером	Находясь на экране деталей курса, нажать на кнопку перехода к главе, перед этим временно ограничив доступ на чтение информации о содержимом глав через API.	Отображается сообщение «Загрузка...»

Вывод по четвертой главе

По итогам проверки все тесты из подготовленного набора были успешно пройдены. Следовательно, функциональные возможности и пользовательский интерфейс приложения работают корректно.

ЗАКЛЮЧЕНИЕ

В рамках выпускной квалификационной работы было разработано мобильное приложение для обучения новых членов стипендиальных комиссий. При этом были решены следующие задачи.

1. Выполнен анализ предметной области.
2. Спроектировано мобильное приложение.
3. Реализовано мобильное приложение.
4. Проведено тестирование разработанного приложения.

В процессе выполнения работы был получен опыт кроссплатформенной разработки на языке JavaScript с применением библиотеки React Native и набора инструментов Expo, а также настройки эмулированных сред в Android Studio и Xcode. Помимо этого, были изучены особенности взаимодействия между клиентской стороной и системой управления контентом Hygraph через GraphQL API.

Дальнейшие улучшения могут быть направлены на расширение функциональности системы: например, добавление офлайн-доступа, чтобы пользователи имели возможность скачивать курсы и учиться без подключения к интернету. Кроме того, в уже разработанную систему отслеживания прогресса можно внедрить геймификацию, повысив таким образом мотивацию на достижение целей обучения.

Другим направлением развития может стать расширение платформы на прочие области знаний и интеграция с дополнительными источниками обучающего контента. В целях масштабирования проекта можно рассмотреть сотрудничество с образовательными учреждениями и экспертами в сфере профсоюзной деятельности для создания уникальных курсов и широкой базы методических материалов.

ЛИТЕРАТУРА

1. «Рынок онлайн-образования в России»: исследование Data Insight и Нетологии – маркетинговое исследование от агентства Data Insight. [Электронный ресурс] URL: https://datainsight.ru/russian_education_market (дата обращения: 20.02.2024 г.).

2. Google Play most popular app categories 2022 | Statista. [Электронный ресурс] URL: https://www.statista.com/statistics/279286/google-play-android-app-categories (дата обращения: 20.02.2024 г.).

3. Apple: most popular app store categories 2022 | Statista. [Электронный ресурс] URL: https://www.statista.com/statistics/270291/popular-categories-in-the-app-store (дата обращения: 20.02.2024 г.).

4. Подведены итоги «Стипкома 2023» в Челябинске – Южно-Уральский государственный университет. [Электронный ресурс] URL: https://www.susu.ru/ru/news/2023/11/21/podvedeny-itogi-stipkoma-2023-v-chelyabinske (дата обращения: 20.02.2024 г.).

5. Федеральный закон от 29.12.2012 г. № 273-ФЗ. [Электронный ресурс] URL: http://www.kremlin.ru/acts/bank/36698 (дата обращения: 20.02.2024 г.).

6. Постановление Правительства РФ от 17.12.2016 N 1390 «О формировании стипендиального фонда» (с изменениями и дополнениями) | ГАРАНТ. [Электронный ресурс] URL: https://base.garant.ru/71570302 (дата обращения: 20.02.2024 г.).

7. Об организации – СКС Профсоюза. [Электронный ресурс] URL: http://sksrf.ru/about (дата обращения: 20.02.2024 г.).

8. Приложение СКС РФ. [Электронный ресурс] URL: https://sksbonus.ru (дата обращения: 20.02.2024 г.).

9. Stepik – образовательная платформа и маркетплейс онлайн-курсов. [Электронный ресурс] URL: https://welcome.stepik.org/ru (дата обращения: 20.02.2024 г.).

10. Программа для создания интерактивных презентаций – iSpring Free. [Электронный ресурс] URL: https://www.ispring.ru/ispring-free (дата обращения: 20.02.2024 г.).

11. LMS iSpring Learn | Платформа для онлайн-обучения. [Электронный ресурс] URL: https://www.ispring.ru/ispring-learn (дата обращения: 20.02.2024 г.).

12. Swift.org – Documentation. [Электронный ресурс] URL: https://www.swift.org/documentation (дата обращения: 20.02.2024 г.).

13. Java Documentation. [Электронный ресурс] URL: https://docs.oracle.com/en/java (дата обращения: 20.02.2024 г.).

14. Kotlin Documentation. [Электронный ресурс] URL: https://kotlinlang.org/docs/home.html (дата обращения: 20.02.2024 г.).

15. Add Kotlin to an existing app | Android Developers. [Электронный ресурс] URL: https://developer.android.com/kotlin/add-kotlin (дата обращения: 20.02.2024 г.).

16. Native memory management | Kotlin Documentation. [Электронный ресурс] URL: https://kotlinlang.org/docs/native-memory-manager.html (дата обращения: 20.02.2024 г.).

17. Get Started with React Native. [Электронный ресурс] URL: https://reactnative.dev/docs/environment-setup (дата обращения: 20.02.2024 г.).

18. Expo Documentation. [Электронный ресурс] URL: https://docs.expo.dev (дата обращения: 20.02.2024 г.).

19. GraphQL | A query language for your API. [Электронный ресурс] URL: https://graphql.org (дата обращения: 20.02.2024 г.).

20. Буч Г. Язык UML. Руководство пользователя: руководство / Г. Буч, Д. Рамбо, И. Якобсон. // Москва: ДМК Пресс, 2008. – 496 с.

21. Writing Markup with JSX – React. [Электронный ресурс] URL: https://react.dev/learn/writing-markup-with-jsx (дата обращения: 20.02.2024 г.).

22. Built-in React Hooks – React. [Электронный ресурс] URL: https://react.dev/reference/react/hooks (дата обращения: 20.02.2024 г.).

23. Passing Props to a Component – React. [Электронный ресурс] URL: https://react.dev/learn/passing-props-to-a-component (дата обращения: 20.02.2024 г.).

24. Hygraph Documentation. [Электронный ресурс] URL: https://hygraph.com/docs (дата обращения: 20.02.2024 г.).

25. Getting started | React Navigation. [Электронный ресурс] URL: https://reactnavigation.org/docs/getting-started (дата обращения: 20.02.2024 г.).

26. React Native Google Sign In. [Электронный ресурс] URL: https://react-native-google-signin.github.io (дата обращения: 20.02.2024 г.).

27. Async Storage Documentation. [Электронный ресурс] URL: https://react-native-async-storage.github.io/async-storage/docs/usage (дата обращения: 20.02.2024 г.).

28. GitHub – jasonkuhrt/graphql-request: Minimal GraphQL client. [Электронный ресурс] URL: https://github.com/jasonkuhrt/graphql-request (дата обращения: 20.02.2024 г.).

ПРИЛОЖЕНИЯ

Приложение А. Спецификация вариантов использования

Спецификация вариантов использования (ВИ) системы приведена в таблицах 1–6.

Таблица 1 – Спецификация ВИ «Авторизоваться»

Прецедент: Авторизоваться
ID: 1
Краткое описание: Вход неавторизованного пользователя в систему
Главные актеры: Неавторизованный пользователь
Второстепенные актеры: Нет
Предусловия: 1. Пользователь открыл приложение.
Основной поток: 1. Прецедент начинается, когда неавторизованный пользователь нажимает кнопку входа на экране авторизации. 2. Пользователь вводит свои учетные данные и подтверждает вход. 3. Система сохраняет данные пользователя и устанавливает статус авторизации. 4. Система перенаправляет пользователя на главный экран приложения.
Постусловия: 1. Пользователь авторизован в системе и имеет доступ к основным функциям приложения.
Альтернативные потоки: I. Ошибка входа. 1. Система отображает сообщение об ошибке на экране авторизации с предложением повторить попытку входа.

Таблица 2 – Спецификация ВИ «Посмотреть список курсов»

Прецедент: Посмотреть список курсов
ID: 2
Краткое описание: Отображение списка доступных курсов для авторизованного пользователя
Главные актеры: Авторизованный пользователь
Второстепенные актеры: Нет
Предусловия: 1. Пользователь авторизован в системе
Основной поток: 1. Прецедент начинается, когда авторизованный пользователь открывает главный экран приложения. 2. Система отправляет запрос к серверу для получения списка доступных курсов.

Окончание таблицы 2 приложения А

3. Сервер возвращает список курсов. 4. Система отображает список курсов на главном экране.
Постусловия: 1. На главном экране отображается список доступных курсов.
Альтернативные потоки: I. Ошибка получения списка курсов. 1. Система логирует ошибку в консоль. 2. Система отображает пустой список курсов на главном экране.

Таблица 3 – Спецификация ВИ «Посмотреть информацию о курсе»

Прецедент: Посмотреть информацию о курсе
ID: 3
Краткое описание: Отображение подробной информации о выбранном курсе для авторизованного пользователя
Главные актеры: Авторизованный пользователь
Второстепенные актеры: Нет
Предусловия: 1. Пользователь авторизован в системе. 2. На главном экране отображается список доступных курсов.
Основной поток: 1. Прецедент начинается, когда авторизованный пользователь выбирает курс из списка на главном экране. 2. Система отправляет запрос к серверу для получения подробной информации о выбранном курсе. 3. Сервер возвращает информацию о курсе. 4. Система отображает экран с подробной информацией о курсе, включая описание, список глав и кнопку для записи на курс.
Постусловия: 1. На экране отображается подробная информация о выбранном курсе.
Альтернативные потоки: I. Ошибка получения информации о курсе. 1. Система отображает сообщение «Загрузка...» вместо подробной информации о курсе, пока данные не будут успешно получены.

Таблица 4 – Спецификация ВИ «Записаться на курс»

Прецедент: Записаться на курс
ID: 4
Краткое описание: Запись авторизованного пользователя на выбранный курс
Главные актеры: Авторизованный пользователь

Второстепенные актеры: Нет
Предусловия: 1. Пользователь авторизован в системе. 2. На экране отображается подробная информация о выбранном курсе.
Основной поток: 1. Прецедент начинается, когда авторизованный пользователь нажимает кнопку «Записаться» на экране с подробной информацией о курсе. 2. Система отправляет запрос к серверу для записи пользователя на выбранный курс. 3. Сервер обрабатывает запрос и добавляет пользователя в список участников курса. 4. Система отображает сообщение об успешной записи на курс.
Постусловия: 1. Пользователь записан на выбранный курс. 2. На экране с подробной информацией о курсе отображается сообщение об успешной записи.
Альтернативные потоки: I. Ошибка записи на курс. 1. Система логирует ошибку в консоль, состояние записи остается неизменным.

Таблица 5 – Спецификация ВИ «Перейти к прохождению главы»

Прецедент: Перейти к прохождению главы
ID: 5
Краткое описание: Переход авторизованного пользователя к прохождению выбранной главы курса
Главные актеры: Авторизованный пользователь
Второстепенные актеры: Нет
Предусловия: 1. Пользователь авторизован в системе. 2. Пользователь записан на выбранный курс. 3. На экране отображается подробная информация о курсе со списком глав.
Основной поток: 1. Прецедент начинается, когда авторизованный пользователь выбирает главу из списка на экране с подробной информацией о курсе. 2. Система отправляет запрос к серверу для получения содержимого выбранной главы. 3. Сервер возвращает содержимое главы, включая теоретические и практические блоки. 4. Система отображает экран с содержимым выбранной главы.
Постусловия: 1. На экране отображается содержимое выбранной главы курса.
Альтернативные потоки: I. Ошибка получения содержимого главы. 1. Система отображает сообщение «Загрузка...» вместо содержимого главы, пока данные не будут успешно получены.

Таблица 6 – Спецификация ВИ «Выйти из аккаунта»

Прецедент: Выйти из аккаунта
ID: 6
Краткое описание: Выход авторизованного пользователя из системы
Главные актеры: Авторизованный пользователь
Второстепенные актеры: Нет
Предусловия: 1. Пользователь авторизован в системе.
Основной поток: 1. Прецедент начинается, когда авторизованный пользователь нажимает кнопку «Выйти» на панели инструментов. 2. Система отправляет запрос к серверу для выхода пользователя из системы. 3. Сервер обрабатывает запрос и завершает сеанс пользователя. 4. Система очищает локальные данные пользователя и перенаправляет его на экран авторизации.
Постусловия: 1. Пользователь вышел из системы и перенаправлен на экран авторизации.
Альтернативные потоки: I. Ошибка выхода из системы. 1. Система логирует ошибку в консоль, пользователь остается авторизованным в системе, локальные данные не очищаются.

Приложение Б. Диаграмма компонентов

Диаграмма компонентов, отражающая статическую структуру системы, представлена на рисунке 1.

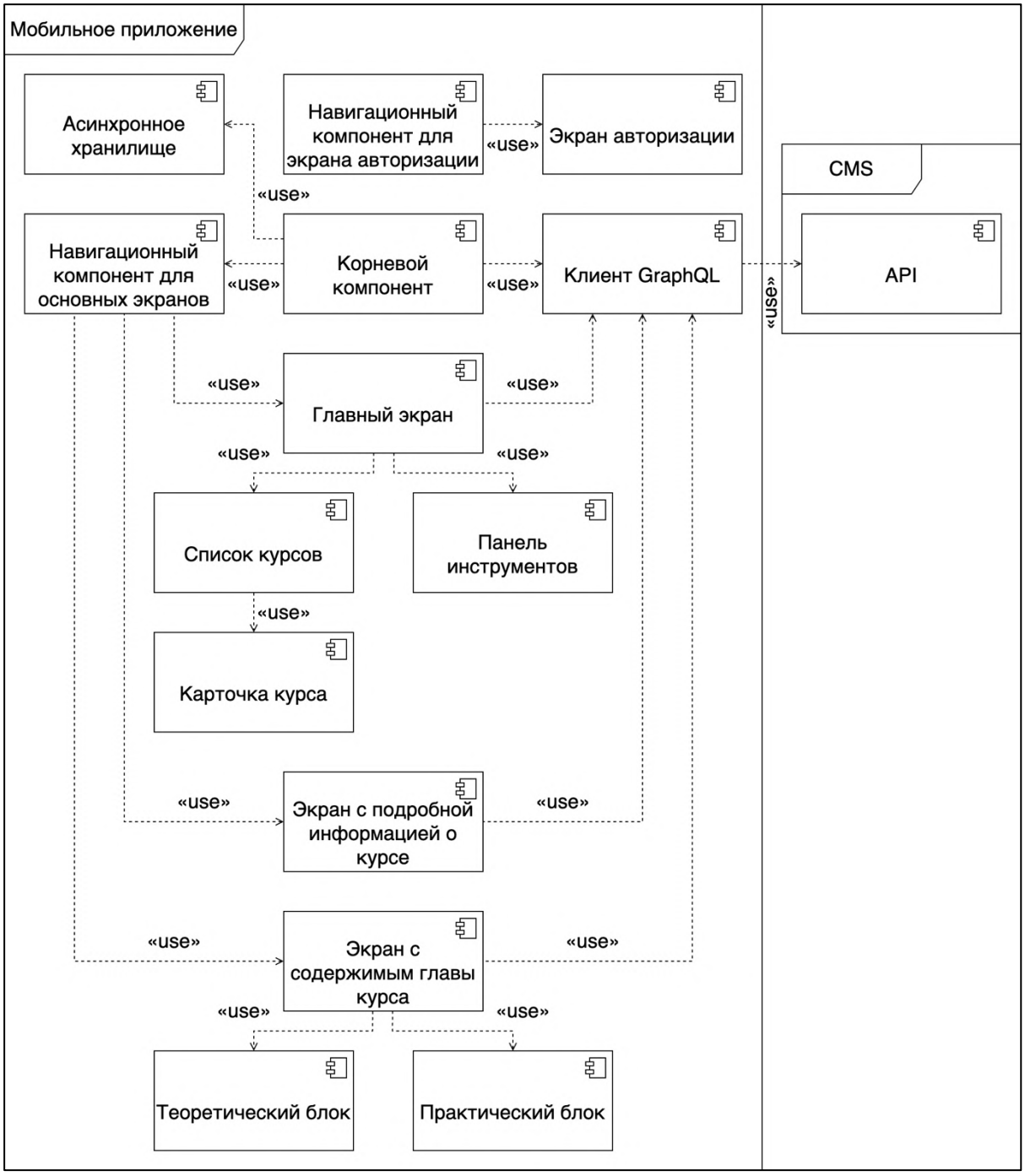

Рисунок 1 – Диаграмма компонентов

Приложение В. Исходный код экранов и компонентов

Исходный код экранов и компонентов, отвечающих за отображение контента курсов и глав, приведен в листингах 1–4.

Листинг 1 – Экран с подробной информацией о курсе

```
const CourseDetailsScreen = ({ route }) => {
  const { courseId, userInfo, refreshKey } = route.params;
  const [course, setCourse] = useState(null);
  const [student, setStudent] = useState(null);
  const [isEnrolled, setIsEnrolled] = useState(false);

  useEffect(() => {
    fetchCourseDetails();
    fetchStudentDetails();
  }, [refreshKey]);

  const fetchCourseDetails = async () => {
    const course = await fetchCourseDetailsFromAPI(courseId);
    setCourse(course);
  };

  const fetchStudentDetails = async () => {
    const student = await fetchStudentDetailsFromAPI(userInfo.user.email);
    setStudent(student);
    setIsEnrolled(student.enrolledCourses.some((enrolledCourse) => en-
rolledCourse.id === courseId));
  };

  const handleEnrollPress = async () => {
    await enrollInCourse(userInfo.user.email, course.id);
    setIsEnrolled(true);
  };

  const navigation = useNavigation();

  const handleChapterPress = (chapter) => {
    navigation.navigate('Chapter', {
      chapter,
      courseId: course.id,
      courseTitle: course.title,
      userInfo: userInfo,
    });
  };

  if (!course) {
    return <Text>Загрузка...</Text>;
  }

  return (
    // Отображение подробной информации о курсе
  );
};
```

Листинг 2 – Экран с содержимым главы курса

```
const ChapterScreen = ({ route }) => {
  const { chapter, courseId, courseTitle, userInfo } = route.params;
  const [currentBlockIndex, setCurrentBlockIndex] = useState(0);
  const navigation = useNavigation();
```

```
  const completeChapter = async () => {
    await completeChapterInAPI(userInfo.user.email, chapter.id);
    navigation.navigate('CourseDetails', { courseId, courseTitle, userInfo,
refreshKey: Date.now() });
  };

  const handleNextBlock = () => {
    if (currentBlockIndex < chapter.theoreticalBlocks.length + chap-
ter.practicalBlocks.length - 1) {
      setCurrentBlockIndex(currentBlockIndex + 1);
    } else {
      completeChapter();
    }
  };

  const handlePreviousBlock = () => {
    if (currentBlockIndex > 0) {
      setCurrentBlockIndex(currentBlockIndex - 1);
    }
  };

  const theoreticalBlocks = chapter.theoreticalBlocks || [];
  const practicalBlocks = chapter.practicalBlocks || [];
  const currentBlock = [...theoreticalBlocks, ...practicalBlocks][cur-
rentBlockIndex];

  if (!currentBlock) {
    return <Text>Загрузка...</Text>;
  }

  const isTheoreticalBlock = theoreticalBlocks.some((block) => block.id ===
currentBlock.id);
  const isLastBlock = currentBlockIndex === theoreticalBlocks.length +
practicalBlocks.length - 1;

  return (
    <View style={styles.container}>
      {isTheoreticalBlock ? (
        <TheoreticalBlock
          block={currentBlock}
          onNext={handleNextBlock}
          onPrevious={handlePreviousBlock}
          isFirst={currentBlockIndex === 0}
          isLast={isLastBlock}
          onComplete={isLastBlock ? completeChapter : undefined}
        />
      ) : (
        <PracticalBlock
          block={currentBlock}
          onNext={handleNextBlock}
          onPrevious={handlePreviousBlock}
          isFirst={currentBlockIndex === 0}
          isLast={isLastBlock}
          onComplete={isLastBlock ? completeChapter : undefined}
        />
      )}
    </View>
  );
};
```

Листинг 3 – Компонент теоретического блока

```
const TheoreticalBlock = ({ block, onNext, onPrevious, isFirst, isLast, on-
Complete }) => {
  return (
    <SafeAreaView style={styles.safeArea}>
      <ScrollView contentContainerStyle={styles.scrollViewContent}>
        {block.imageUrl && <Image source={{ uri: block.imageUrl }}
style={styles.image} />}
        <Text style={styles.title}>{block.title}</Text>
        <Text style={styles.content}>{block.content}</Text>
      </ScrollView>
      <View style={styles.buttonsContainer}>
        <TouchableOpacity
          style={[styles.button, styles.leftButton, isFirst && styles.disa-
bledButton]}
          onPress={onPrevious}
          disabled={isFirst}
        >
          <Text style={[styles.buttonText, isFirst && styles.disabledBut-
tonText]}>Назад</Text>
        </TouchableOpacity>
        {!isLast ? (
          <TouchableOpacity style={[styles.button, styles.rightButton]} on-
Press={onNext}>
            <Text style={styles.buttonText}>Далее</Text>
          </TouchableOpacity>
        ) : (
          <TouchableOpacity style={[styles.button, styles.rightButton]} on-
Press={onComplete}>
            <Text style={styles.buttonText}>Завершить главу</Text>
          </TouchableOpacity>
        )}
      </View>
    </SafeAreaView>
  );
};
```

Листинг 4 – Компонент практического блока

```
const PracticalBlock = ({ block, onNext, onPrevious, isFirst, isLast, on-
Complete }) => {
  const [selectedOption, setSelectedOption] = useState(null);
  const [manualInput, setManualInput] = useState('');
  const [isAnswerCorrect, setIsAnswerCorrect] = useState(false);
  const [isAnswerSubmitted, setIsAnswerSubmitted] = useState(false);

  useEffect(() => {
    // Сбрасываем состояние при изменении блока
    setSelectedOption(null);
    setManualInput('');
    setIsAnswerCorrect(false);
    setIsAnswerSubmitted(false);
  }, [block]);

  const handleOptionPress = (option) => {
    setSelectedOption(option);
  };

  const handleSubmit = () => {
```

```
    const isCorrect =
      (block.type === 'MULTIPLE_CHOICE' && selectedOption === block.cor-
rectAnswer) ||
      (block.type === 'MANUAL_INPUT' && manualInput.trim().toLowerCase()
=== block.correctAnswer.toLowerCase());
    setIsAnswerCorrect(isCorrect);
    setIsAnswerSubmitted(true);
    if (!isCorrect) {
      setSelectedOption(null);
      setManualInput('');
    }
  };

  const handleNext = () => {
    if (isAnswerCorrect) {
      onNext();
    }
  };

  const handleComplete = () => {
    if (isAnswerCorrect && onComplete) {
      onComplete();
    }
  };

  return (
    <SafeAreaView style={styles.safeArea}>
      <View style={styles.contentContainer}>
        <Text style={styles.question}>{block.question}</Text>
        {block.type === 'MULTIPLE_CHOICE' && (
          // Отображение вариантов ответа для вопросов в формате теста
        )}
        {block.type === 'MANUAL_INPUT' && (
          // Отображение текстового поля для ввода ответа
        )}
        {isAnswerSubmitted && (
          <>
            {isAnswerCorrect ? (
              <Text style={styles.feedbackText}>Правильный ответ!</Text>
            ) : (
              <Text style={styles.feedbackText}>Неправильный ответ, попро-
буйте еще раз</Text>
            )}
          </>
        )}
      </View>
      <View style={styles.buttonsContainer}>
        {/* Кнопки навигации и проверки ответа */}
      </View>
    </SafeAreaView>
  );
};
```

Приложение Г. Алгоритм прохождения главы курса

Диаграмма деятельности, описывающая алгоритм прохождения главы курса, представлена на рисунке 2.

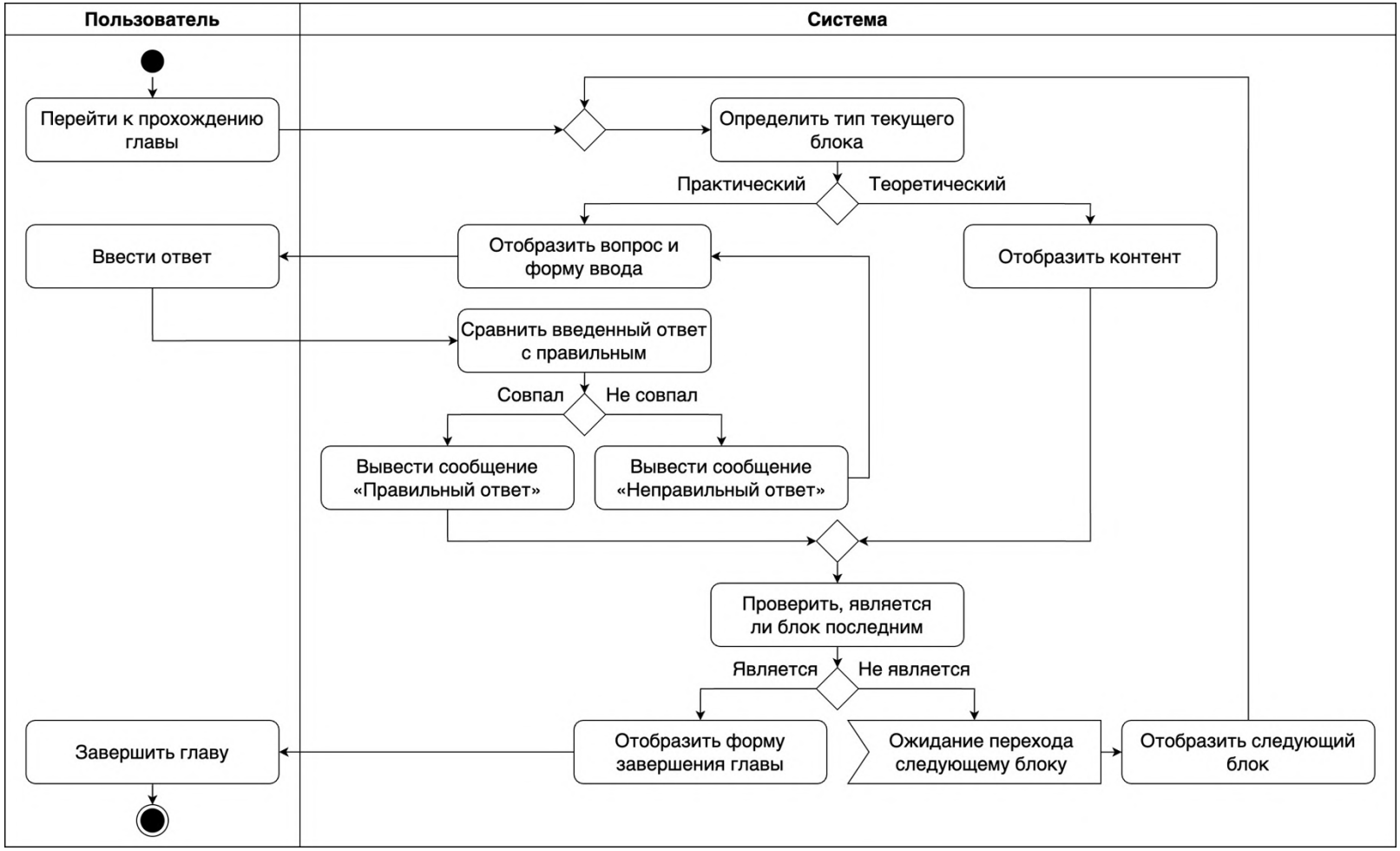

Рисунок 2 – Алгоритм прохождения главы курса