МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

Кафедра Прикладная математика и информатика
(наименование)

09.03.03 Прикладная информатика
(код и наименование направления подготовки/специальности)

Разработка программного обеспечения
(направление (профиль)/специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему Разработка мобильного приложения для педагогов учреждений образования

Обучающийся Е.С. Мяделко _____
(Инициалы Фамилия) (личная подпись)

Руководитель д.т.н., доцент, С.В. Мкртычев
(учёная степень (при наличии), учёное звание (при наличии), Инициалы Фамилия

Тольятти 2024

Аннотация

Тема бакалаврской работы – «Разработка мобильного приложения для педагогов учреждений образования».

Структура дипломной работы представлена введением, тремя главами, заключением, списком использованной литературы и приложениями.

В введении представлены актуальность выбранной темы, объект и предмет исследования, основные цели и задачи исследования, используемые методики и практическая значимость полученных результатов.

В первой главе рассматриваются объект и предмет исследования, формулируется постановка задач на разработку мобильного приложения.

Во второй главе рассматриваются методологии и технологии проектирования информационных систем и моделирование бизнес-процессов для предмета исследования.

В третьей главе описывается процесс прототипирования и отладки мобильного приложения.

Заключение отражает основные проектные решения, разработанные методики и модели, общие выводы проведённых исследований.

Ключевые слова: педагог, программное обеспечение, прототип, информационная система.

Бакалаврская работа представлена на 69 страницах и включает в себя 12 рисунков, 11 таблиц и приложение.

Список используемых источников содержит 21 наименование.

Оглавление

Введение

Актуальность темы автоматизации и цифровизации деятельности педагогов обусловлена стремительными изменениями в образовательной сфере и широким внедрением информационных технологий в повседневную жизнь. В современных условиях педагоги сталкиваются с большими объемами административной работы, необходимостью организации учебного процесса, оценки успеваемости и взаимодействия с родителями и учениками. Автоматизация этих процессов позволяет значительно сократить затраты времени на рутинные задачи, что дает возможность педагогам сосредоточиться на основной деятельности – обучении и развитии учеников.

Цифровизация образовательных процессов способствует улучшению качества обучения благодаря более точному анализу данных, персонализации учебных планов и мониторингу прогресса учащихся в реальном времени. Информационные системы могут автоматизировать такие задачи, как планирование уроков, ведение журналов успеваемости, управление расписанием и подготовка отчетности. Это помогает улучшить организацию образовательного процесса, повысить его прозрачность и предоставить педагогам эффективные инструменты для управления обучением.

Объектом исследования бакалаврской работы является автоматизация деятельности педагогов учреждения образования.

Предметом исследования бакалаврской работы является мобильное приложение для педагогов учреждений образования.

Цель работы заключается в разработке мобильного приложения, которое позволит автоматизировать некоторые аспекты деятельности педагога, сократив затрачиваемое время на рутинные операции.

Для достижения поставленной цели нужно решить следующие задачи:

- анализ предметной области;
- разработка требований к программному обеспечению;
- проектирование структуры приложения;

– выбор технологий и инструментов разработки;

– разработка пользовательского интерфейса;

– реализация функционала приложения;

– тестирование приложения.

Работа основана на положениях теоретического и частного методах исследования, а именно анализа и практического программирования.

В первой главе рассматриваются объект и предмет исследования, а также формулируются задачи по разработке мобильного приложения для педагогов учреждений образования. Вторая глава посвящена методологиям и технологиям проектирования мобильных приложений, а также моделированию бизнес-процессов, связанных с предметом исследования. Описываются сценарии использования мобильного приложения и приводится их детальное описание. Значительное внимание уделяется выбору методологии проектирования и логическому проектированию мобильного приложения, что включает создание схемы базы данных и структуры приложения, необходимых для эффективной работы всех модулей мобильного приложения. Третья глава фокусируется на реализации и отладке мобильного приложения. В ней описывается выбор средств разработки, включая языки программирования, фреймворки и инструменты, а также процесс создания логической структуры базы данных и проектирования пользовательского интерфейса. Особое внимание уделено тестированию, которое проводится на различных уровнях — от модульного до системного, что позволяет выявить ошибки и внести улучшения на основе анализа результатов тестирования [2].

В заключении описываются результаты выполнения выпускной квалификационной работы.

Глава 1 Постановка задачи на разработку мобильного приложения для педагогов учреждений образования

1.1 Функциональные и структурные особенности информационных систем учреждений образования

Программное обеспечение информационной системы должно строиться на следующих ключевых принципах:

- модульность;
- гибкость и масштабируемость;
- интероперабельность;
- безопасность;
- удобство использования.

Цель информационной системы образовательного учреждения заключается в поддержке и оптимизации образовательного процесса, обеспечении автоматизации административных и учебных задач, а также организации эффективного взаимодействия между всеми участниками образовательного процесса. Основными задачами системы являются управление учебным планом, составление и корректировка расписания, учет успеваемости и посещаемости студентов, автоматизация процессов отчетности и контроля, а также поддержка дистанционного обучения и взаимодействие с внешними системами для интеграции образовательных ресурсов и отчетов.

Функциональные особенности информационной системы образовательного учреждения включают автоматизацию ключевых процессов учебной и административной деятельности. Система обеспечивает управление учебным процессом, включая составление расписаний, регистрацию студентов на курсы, учет успеваемости и посещаемости, а также ведение электронных журналов и дневников. Преподаватели могут вносить оценки, следить за прогрессом студентов, а администрация – получать оперативную отчетность и

статистику по учебному процессу. Для студентов и родителей предоставляется возможность удаленного доступа к информации об успеваемости и посещаемости, что повышает прозрачность и удобство взаимодействия. Также система поддерживает дистанционное обучение, интеграцию с платформами для онлайн-курсов, тестирования и видеолекций, что расширяет возможности образовательного процесса. Важной функциональной особенностью является обеспечение безопасности данных и контроль доступа, чтобы защитить персональные и учебные данные от несанкционированного доступа.

Архитектурные особенности мобильного приложения образовательного учреждения основаны на многослойной или модульной архитектуре, которая обеспечивает гибкость, масштабируемость и устойчивость системы. Чаще всего используется трехуровневая архитектура: на уровне клиентской части находятся интерфейсы для взаимодействия пользователей с системой, на уровне логики – бизнес-правила и обработка данных, а на уровне базы данных – централизованное хранилище, которое обеспечивает управление и сохранение данных, таких как информация о студентах, успеваемости, преподавателях, курсах и т.д. Такая архитектура позволяет разделять логику приложения от представления и данных, что упрощает поддержку, обновление и масштабирование системы. Важным аспектом является использование REST API или других интерфейсов для интеграции с внешними платформами и государственными системами, а также микросервисный подход, позволяющий распределить функциональность на независимые сервисы, которые могут быть развернуты, масштабированы и обновлены отдельно друг от друга. Также в архитектуре уделяется особое внимание безопасности, с внедрением механизмов шифрования данных, защиты от несанкционированного доступа и многоуровневого контроля доступа.

Далее перечислены архитектурные компоненты:

- клиентский интерфейс;
- сервер приложений;
- база данных;

– API;

– микросервисы;

– система безопасности;

Принципы безопасности в информационных системах включают несколько ключевых аспектов: аутентификация и авторизация, обеспечивающие проверку личности пользователей и предоставление им соответствующих прав доступа; шифрование данных для защиты информации при передаче и хранении; контроль доступа, позволяющий разграничить доступ к различным данным и функциям системы; регулярное обновление и патчинг системы для устранения уязвимостей; мониторинг и ведение логов для отслеживания подозрительных действий и возможных инцидентов; а также использование резервного копирования для защиты данных в случае сбоев или атак. Таким образом, основная цель мобильного приложения заключается в автоматизации и оптимизации процессов, обеспечении эффективного решения задач преподавателями.

1.2 Разработка требований к мобильному приложению для педагогов учреждений образования

Waterfall – это традиционная, последовательная методология разработки программного обеспечения, в которой проект проходит через ряд четко определенных фаз: сбор требований, проектирование, реализация, тестирование, внедрение и поддержка. Каждая фаза должна быть завершена перед началом следующей, что делает процесс строго линейным. Важной особенностью является то, что все требования фиксируются на начальном этапе, и изменения на более поздних стадиях либо не предусмотрены, либо их реализация обходится дорого. Этот подход подходит для проектов с четко определенными требованиями и минимальными изменениями по ходу разработки, но может быть менее гибким для динамично изменяющихся условий.

Преимущество методологии Waterfall заключается в ее структурированности и предсказуемости. Благодаря четко определенным этапам и фиксированным требованиям на раннем этапе, команда может планировать проект с высокой точностью и иметь ясное представление о сроках и ресурсах. Это особенно полезно для проектов с четко определенными и стабильными требованиями, где минимальные изменения ожидаются в процессе разработки. Waterfall также обеспечивает четкую документацию на каждом этапе, что упрощает процесс передачи знаний и последующую поддержку проекта [6]. Выделенные требования представлены в таблице 1.

Таблица 1 – Требования к ПО ИС для преподавателей

Категория	Требование	Описание	Приоритет	Комментарии
Управление курсами	Создание и редактирование курсов	Преподаватели могут создавать курсы, редактировать их содержание и управлять расписанием	Высокий	Курсы должны поддерживать разные форматы контента (видео, тесты)
Учет успеваемости	Ведение электронного журнала	Преподаватели могут добавлять оценки, просматривать успеваемость студентов	Высокий	Возможность экспорта данных в различные форматы (PDF, Excel)
Отчетность	Генерация отчетов по успеваемости и посещаемости студентов	Возможность создания отчетов за выбранный период с фильтрами по группам, предметам и датам	Средний	Отчеты должны быть настраиваемыми под разные требования
Уведомления	Уведомления преподавателям о важных событиях	Всплывающие уведомления о предстоящих событиях	Средний	Настройки уведомлений с возможностью отключения

Продолжение таблицы 1

Категория	Требование	Описание	Приоритет	Комментарии
Управление расписанием	Возможность редактирования расписания занятий	Преподаватели могут просматривать, изменять и обновлять расписание своих занятий	Высокий	Интеграция с системой расписаний учебного заведения
Пользовательский интерфейс	Удобный интерфейс для преподавателей	Простой и интуитивный интерфейс, позволяющий преподавателям легко управлять данными студентов	Высокий	Интеграция с существующими образовательными системами
Поддержка мобильных устройств	Доступ к системе через мобильные приложения	Преподаватели могут управлять данными и расписанием через мобильные устройства	Средний	Мобильная версия с адаптивным интерфейсом для iOS и Android
Система помощи	Встроенная система справки и поддержки	Возможность получения преподавателями справки по функционалу и инструкций	Низкий	Включает текстовые руководства и видеоуроки по работе с системой
Производительность	Быстрая обработка данных	Система должна обеспечивать высокую производительность при обработке большого количества данных	Высокий	Особенно в пиковые периоды, такие как сессии или экзамены
Надежность	Минимальное время простоя	Система должна быть надежной и доступной 24/7 для преподавателей и студентов	Высокий	Важно для беспрерывного учебного процесса

Выделенные требования к программному обеспечению для преподавателей образовательных учреждений включают широкий спектр функциональных и нефункциональных аспектов, направленных на

оптимизацию учебного процесса и повышение удобства работы [7]. Функциональные требования охватывают основные задачи, такие как управление курсами, учет успеваемости, генерация отчетов, уведомления, что обеспечивает преподавателям полный контроль над учебными данными. Нефункциональные требования акцентируют внимание на безопасности, удобстве интерфейса, производительности, надежности системы и доступности через мобильные устройства, что способствует созданию безопасной, стабильной и легко масштабируемой системы [11]. Этот набор требований позволяет удовлетворить как повседневные потребности преподавателей, так и обеспечить высокий уровень качества и надежности программного обеспечения.

1.3 Обзор и анализ аналогов приложений для преподавателей учреждений образования

Рынок программного обеспечения для преподавателей активно развивается, предлагая широкий спектр решений, предназначенных для автоматизации учебного процесса, улучшения взаимодействия с учениками и управления образовательными данными. Такие системы позволяют преподавателям эффективно организовывать курсы, управлять учебными материалами, вести электронные журналы, отслеживать успеваемость и посещаемость студентов, а также генерировать отчеты. Кроме того, программы для преподавателей интегрируются с системами управления обучением, что помогает реализовывать дистанционные курсы и проводить онлайн-оценку знаний. Современные решения часто включают облачные сервисы, что позволяет преподавателям работать с любой точки и на различных устройствах [8].

Разнообразие программного обеспечения варьируется от специализированных платформ для высшего образования до решений для школ и частных образовательных учреждений. Важными критериями выбора

такого ПО являются удобство использования, поддержка различных форматов контента, интеграция с существующими системами, безопасность данных и возможность адаптации под разные образовательные модели.

Далее перечислены три наиболее популярных решения на рынке программного обеспечения для преподавателей:

- Google Classroom – бесплатная система управления обучением от Google, предлагающая инструменты для создания курсов, общения с учениками, задания и оценки;
- Moodle – открытая платформа для создания онлайн-курсов с широким функционалом для дистанционного обучения и управления образовательным процессом;
- Blackboard – мощная система для управления обучением, используемая в основном в высших учебных заведениях, с функциями для создания курсов, ведения журналов, тестирования и проведения онлайн-уроков.

Рассмотрим данные программные средства более подробно.

Google Classroom – это облачная платформа для управления учебным процессом, разработанная Google. Она предназначена для преподавателей и студентов с целью упрощения организации учебного процесса, создания и распространения заданий, а также предоставления обратной связи. Google Classroom интегрирована с другими сервисами Google, такими как Google Drive, Google Docs и Gmail, что делает её удобной для работы с документами, презентациями и другими учебными материалами в единой экосистеме. Преподаватели могут создавать курсы, назначать задания, отслеживать прогресс студентов и взаимодействовать с ними в онлайн-формате, что делает платформу особенно полезной для дистанционного и гибридного обучения.

Однако, несмотря на множество преимуществ, Google Classroom имеет свои ограничения. Например, её функциональность может быть недостаточной для более сложных образовательных потребностей. Система не поддерживает некоторые продвинутые функции, доступные в более

специализированных LMS, такие как расширенные аналитические инструменты и детальная настройка интерфейса. Кроме того, интеграция с внешними системами ограничена, что может стать преградой для крупных учебных заведений с существующей инфраструктурой.

Основные достоинства:

– простота и удобство использования;

– бесплатность и интеграция с другими сервисами Google;

– доступность для всех типов учебных заведений;

– поддержка дистанционного обучения и совместной работы.

Основные недостатки:

– ограниченные возможности для настройки и персонализации;

– недостаток продвинутых аналитических инструментов;

– ограниченная интеграция с внешними системами и другими LMS.

Интерфейс программного средства представлен на рисунке 1.

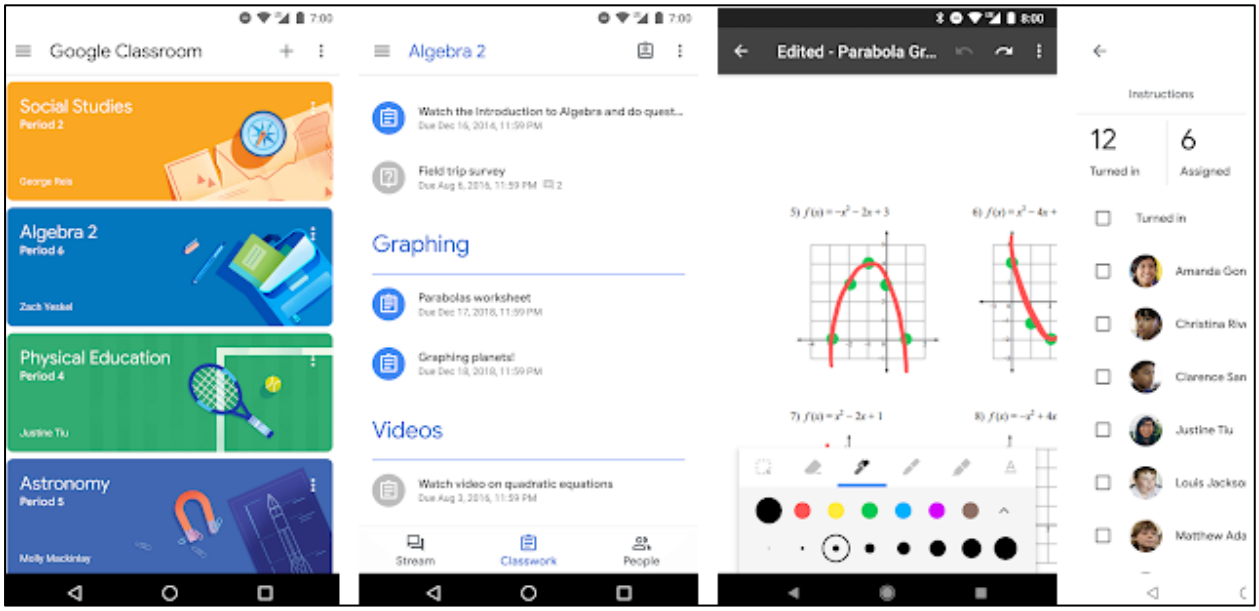

Рисунок 1 – Интерфейс программного средства

Moodle – это открытая система управления обучением, которая используется для создания онлайн-курсов и организации учебного процесса.

Разработанная в 2002 году, Moodle стала одной из самых популярных платформ для дистанционного образования благодаря своей гибкости и возможностям персонализации. Будучи бесплатной с открытым исходным кодом, Moodle позволяет учебным заведениям и организациям адаптировать платформу под свои конкретные нужды, настраивая её функционал и внешний вид. Платформа предоставляет преподавателям широкий набор инструментов для создания учебных материалов, проведения тестов, оценки студентов, а также организации форумов и общения.

Однако использование Moodle требует определённых технических навыков, особенно если необходимо настроить или интегрировать её с другими системами. Несмотря на её гибкость, Moodle может показаться сложной для новых пользователей из-за широкого набора функций и возможностей. Важно учитывать, что для полного использования её потенциала может потребоваться помощь квалифицированных специалистов, особенно если образовательная организация не имеет опыта работы с открытым ПО.

Основные достоинства:

- полностью настраиваемая и адаптируемая благодаря открытому коду;
- поддержка множества плагинов и интеграций;
- гибкость в создании курсов и управлении учебным процессом;
- подходит для любого типа образовательных учреждений.

Основные недостатки:

- сложность в освоении для новых пользователей;
- необходимость технической поддержки для настройки и управления;
- более тяжелая инфраструктура по сравнению с простыми системами.

Интерфейс программного средства представлен на рисунке 2.

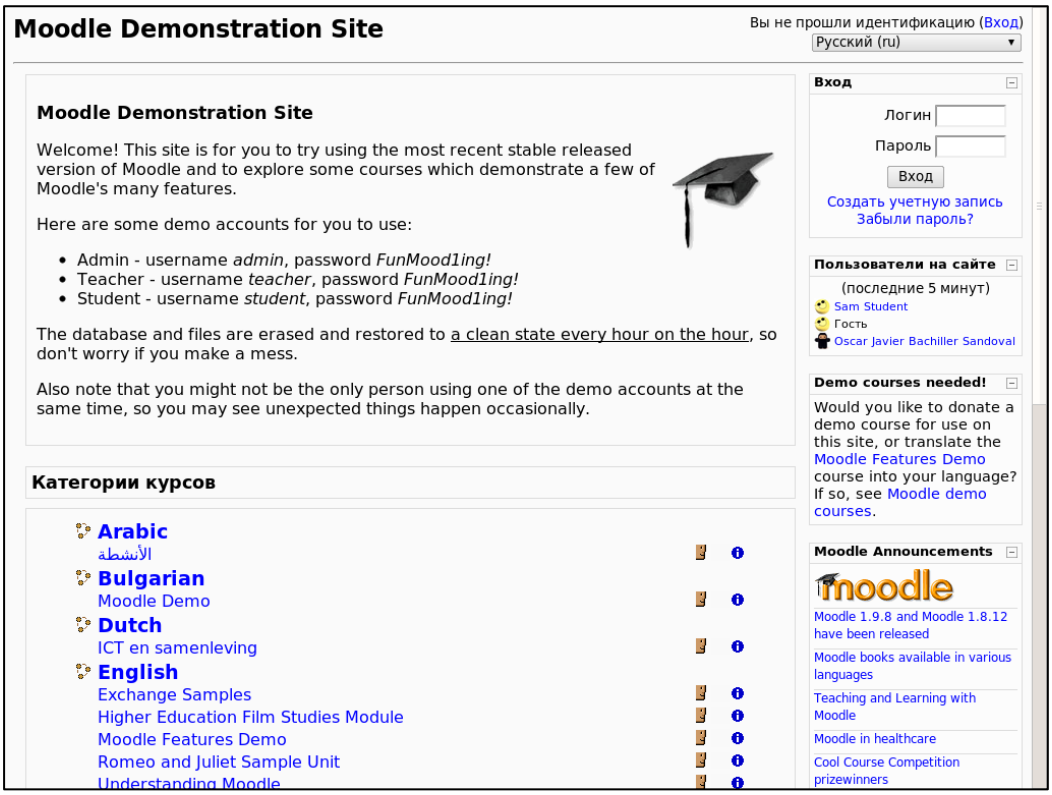

Рисунок 2 – Интерфейс программного средства

Blackboard – это одна из наиболее широко используемых систем управления обучением, особенно популярная в высших учебных заведениях и университетах. Платформа предоставляет полный набор инструментов для управления учебным процессом, включая создание курсов, взаимодействие с учащимися, проведение тестов и контроль успеваемости. Blackboard также поддерживает интеграцию с различными системами, что делает её мощным инструментом для образовательных учреждений с существующей ИТ-инфраструктурой. Платформа предназначена для как очного, так и дистанционного обучения, обеспечивая гибкость и возможность преподавателям и студентам взаимодействовать независимо от их местоположения.

Однако Blackboard имеет некоторые недостатки, связанные с её сложностью и стоимостью. Внедрение и поддержка Blackboard могут потребовать значительных финансовых затрат, что делает её менее доступной

для небольших учреждений. Кроме того, из-за обширного функционала система может показаться сложной в освоении, как для преподавателей, так и для студентов, особенно если нет должной технической поддержки. Несмотря на свою мощность, Blackboard иногда критикуется за громоздкость интерфейса и необходимость в дополнительных обучающих ресурсах для пользователей.

Основные достоинства:

– мощная функциональность для управления учебными процессами;
– поддержка видеоконференций и интеграция с внешними системами;
– аналитические инструменты для мониторинга успеваемости студентов;
– мобильное приложение для удобного доступа.

Основные недостатки:

– высокая стоимость внедрения и поддержки;
– сложность в освоении для новых пользователей;
– громоздкость интерфейса и необходимость в технической поддержке.

Интерфейс программного средства представлен на рисунке 3.

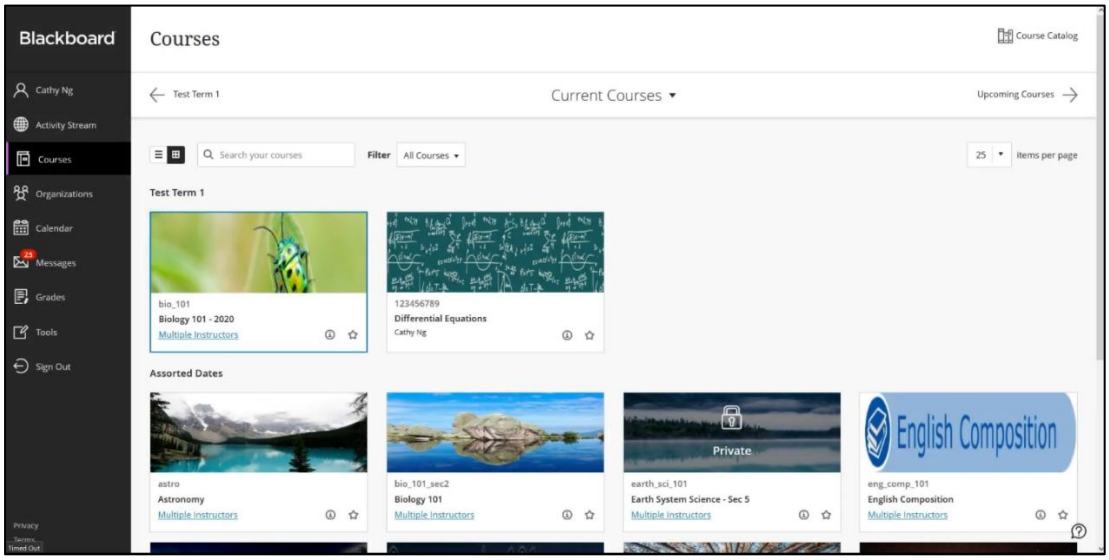

Рисунок 3 – Интерфейс программного средства

Рассмотренные программные средства для управления обучением – Google Classroom, Moodle и Blackboard – представляют собой мощные платформы, каждая из которых удовлетворяет различные потребности образовательных учреждений. Google Classroom выделяется своей простотой, доступностью и легкостью интеграции с другими сервисами Google, что делает его идеальным для школ и небольших учебных заведений. Moodle предлагает гибкость и высокую степень персонализации благодаря открытому исходному коду, что привлекает учебные заведения с уникальными потребностями и желанием настроить систему под себя. Blackboard, в свою очередь, предоставляет наиболее полную функциональность для управления сложными образовательными процессами, однако требует значительных ресурсов для внедрения и поддержки. Каждая из этих платформ имеет свои сильные и слабые стороны, и выбор между ними зависит от бюджета, технических требований и уровня сложности образовательного процесса.

В таблице 2 представлена сравнительная таблица программного обеспечения.

Таблица 2 – Сравнительная таблица функциональности

Критерий	Google Classroom	Moodle	Blackboard
Управление курсами	Базовое управление курсами и заданиями. Простая структура.	Высокая гибкость.	Мощное управление курсами.
Оценивание и аналитика	Базовые функции для оценки, минимальные аналитические возможности.	Продвинутые возможности оценки.	Самые развитые инструменты аналитики и оценивания, гибкие отчеты.
Интеграция с внешними системами	Интеграция с продуктами Google.	Широкая поддержка плагинов.	Мощные возможности интеграции

Продолжение таблицы 2

Критерий	Google Classroom	Moodle	Blackboard
Мобильная поддержка	Мобильное приложение с базовым функционалом.	Официальное мобильное приложение с поддержкой большинства функций.	Полноценное мобильное приложение с поддержкой всех функций.
Безопасность и конфиденциальность	Соответствует стандартам FERPA и COPPA, привязка к экосистеме Google.	Высокий уровень безопасности, контроль на стороне организации.	Корпоративные решения с усиленной безопасностью для крупных учреждений.
Масштабируемость	Подходит для небольших и средних учебных заведений.	Очень масштабируемая, подходит как для малых, так и для крупных организаций.	Максимальная масштабируемость, поддержка крупных учебных заведений и корпораций.

Рассмотренные программные средства имеют свои ограничения, несмотря на их популярность и широкое применение [9]. Google Classroom, хотя и прост в использовании, предоставляет ограниченные возможности для сложных образовательных процессов и интеграции с внешними системами, что может быть недостаточно для крупных учебных заведений. Moodle, будучи мощным и гибким, требует значительных технических навыков для настройки и поддержки, что может стать препятствием для менее технически подготовленных пользователей. Blackboard, хотя и предлагает максимальные возможности для управления сложными образовательными процессами, является дорогостоящим решением и часто критикуется за сложность освоения и громоздкость интерфейса.

Выводы по главе 1:

В рамках анализа предметной области были освещены основные функциональные и архитектурные особенности программных средства рассматриваемого толка.

Глава 2 Проектирование мобильного приложения для педагогов учреждений образования

2.1 Разработка диаграммы вариантов использования мобильного приложения

Диаграмма вариантов использования является важным инструментом в процессе разработки программного обеспечения. Она наглядно демонстрирует взаимодействия между пользователями и самой системой, отображая, какие функции доступны пользователям и как система должна реагировать на их запросы. Это позволяет упростить процесс проектирования, анализа и тестирования, обеспечивая лучшее понимание системы для всех участников проекта.

Основные функции диаграммы вариантов использования:

- визуализация функциональных возможностей;
- определение границ системы;
- документирование требований;
- упрощение коммуникации;
- поддержка планирования разработки;
- поддержка тестирования системы.

При разработке диаграммы вариантов использования необходимо обратить внимание на несколько ключевых аспектов. Прежде всего, важно четко определить границы системы, чтобы отделить функции, которые будут реализованы, от тех, которые выходят за ее рамки. Следует правильно идентифицировать всех участников, взаимодействующих с системой, и их роли, чтобы точно отразить, какие действия они могут выполнять. Необходимо также детализировать каждый вариант использования, обеспечив полное описание сценариев взаимодействия, включая нормальные и альтернативные потоки событий. Важно учесть исключения и нештатные ситуации, чтобы предусмотреть поведение системы в случае ошибок или

непредвиденных условий. Наконец, диаграмму следует сделать понятной для всех участников проекта, чтобы упростить коммуникацию и согласование требований.

Для разработки диаграммы будет использоваться язык моделирования UML.

UML – это унифицированный язык моделирования, который используется для визуального представления и документирования структуры и поведения программных систем [10]. UML включает в себя различные типы диаграмм, такие как диаграммы вариантов использования, классов, последовательностей, активности и другие, которые помогают разработчикам и аналитикам описывать архитектуру и динамику системы. Он широко используется в объектно-ориентированном программировании и поддерживает процессы разработки, от анализа требований до проектирования и реализации системы. UML стандартизирован и поддерживается различными инструментами, что делает его универсальным языком для описания сложных систем.

Диаграмма вариантов использования представлена на рисунке 4.

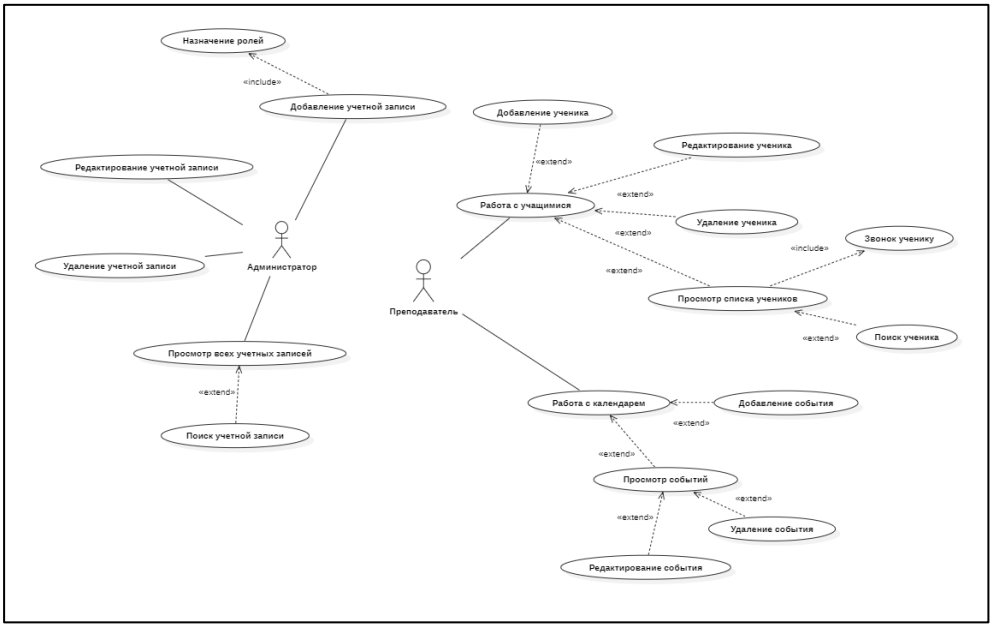

Рисунок 4 – Диаграмма вариантов использования

Таким образом, функционал программного средства выглядит полным и хорошо структурированным для выполнения всех необходимых задач, обеспечивая поддержку различных ролей пользователей и гибкость в управлении данными.

2.2 Описание вариантов использования

Описание вариантов использования необходимо для того, чтобы систематически задокументировать все возможные взаимодействия пользователей с системой. Оно предоставляет четкое представление о том, какие цели пользователь стремится достичь и каким образом система должна реагировать на его действия. Такое описание помогает детализировать функциональные требования к системе, что особенно важно для разработчиков и тестировщиков на всех этапах жизненного цикла проекта.

В описании указываются акторы, участвующие в сценарии, и их роли в процессе взаимодействия, а также различные потоки событий, включая основной и альтернативные.

Это позволяет учитывать как нормальное поведение системы, так и возможные отклонения или ошибки, которые могут возникнуть в процессе выполнения сценария.

Предусловия и постусловия помогают определить контекст начала и окончания варианта использования, что способствует лучшему пониманию условий выполнения задачи. Итоговое описание служит основой для проектирования, реализации и тестирования системы, а также для создания документации, которая может использоваться в дальнейшем для обучения пользователей или поддержки системы.

Описание варианта использования «Добавление ученика» представлено в таблице 3.

Таблица 3 – Описание варианта использования «Добавления ученика»

Краткое описание	Этот сценарий описывает процесс добавления нового ученика в систему, включая ввод личных данных и контактной информации.
Основные потоки сценария	Учитель выбирает опцию «Добавить ученика». Система открывает форму для ввода данных. Учитель вводит данные ученика: имя, отчество, фамилию и номер телефона. Учитель подтверждает добавление, нажав кнопку «Добавить». Система проверяет корректность введённых данных. Если данные корректны, система сохраняет данные ученика в базе данных и отображает уведомление о успешном добавлении.
Альтернативные потоки	Если введённые данные некорректны (например, неверный формат номера телефона), система выводит сообщение об ошибке и просит исправить данные. Учитель вносит исправления и снова подтверждает добавление.
Исключения и ошибки	Ошибки могут возникнуть при вводе некорректных данных, таких как пустые поля или неверный формат контактной информации. В случае ошибки система уведомляет пользователя и предлагает внести исправления.
Предусловия	Учитель авторизован в системе.
Постусловия	Ученик успешно добавлен в систему, данные сохраняются в базе данных, и ученик появляется в общем списке учеников.
Краткое описание	Этот сценарий описывает процесс добавления нового ученика в систему, включая ввод личных данных и контактной информации.

Диаграмма активности необходима для того, чтобы визуально представить поток управления или последовательность действий внутри системы или процесса. Она помогает детализировать логику выполнения операций, показывая, какие шаги выполняются последовательно, параллельно или условно. Диаграмма активности позволяет наглядно отобразить сложные процессы, включая ветвления, синхронизацию, параллельное выполнение действий и точки принятия решений. Она полезна для анализа, проектирования и документирования поведения системы, помогая выявить потенциальные узкие места, ошибки в логике и возможности для оптимизации процесса. Диаграмма активности особенно эффективна для моделирования рабочих процессов, операций внутри бизнес-процессов, а также для описания

алгоритмов в программном обеспечении. Она облегчает понимание сложных процессов, как для разработчиков, так и для аналитиков, менеджеров и других заинтересованных сторон, повышая прозрачность проекта и улучшая коммуникацию между участниками команды.

Диаграмма активности для варианта использования «Добавление ученика» представлена на рисунке 5.

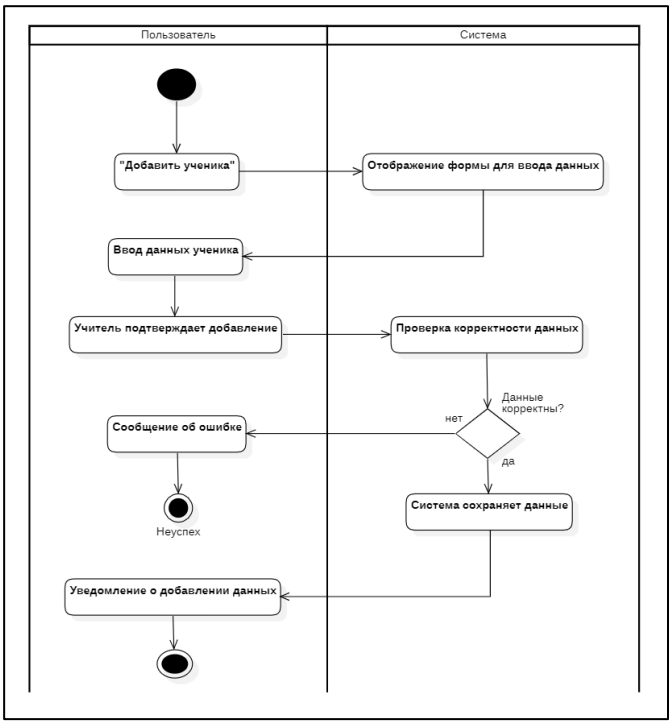

Рисунок 5 – Диаграмма активностей для варианта использования «Добавление ученика»

Диаграмма активности для варианта использования «Добавление учащегося» показывает процесс добавления нового ученика в систему, начиная с того, как учитель выбирает опцию «Добавить учащегося». Диаграмма отображает последовательные шаги, включая ввод личных данных, подтверждение добавления, проверку системы на корректность данных, и, в случае успешной проверки, сохранение информации в базе данных. Если данные введены некорректно, система возвращает пользователя к этапу исправления ошибок. В итоге, диаграмма заканчивается успешным

добавлением ученика в систему или повторным вводом данных при возникновении ошибок.

2.3 Выбор методологии проектирования программного обеспечения

Методология разработки – это структурированный набор принципов, практик и процессов, которые определяют, как программное обеспечение должно быть разработано, управляемо и внедрено. Она задает подход к проектированию, разработке, тестированию, внедрению и поддержке программных продуктов, охватывая весь жизненный цикл разработки. Методология определяет, какие этапы должны быть выполнены, как распределяются задачи между членами команды, и каким образом контролируется выполнение проекта. Методология разработки необходима для того, чтобы упорядочить и стандартизировать процесс разработки, обеспечивая более предсказуемый и управляемый результат. Она помогает команде эффективно работать вместе, минимизировать риски, соблюдать сроки и бюджеты, а также поддерживать высокое качество разрабатываемого программного обеспечения. Методологии также важны для улучшения взаимодействия между разработчиками, заказчиками и другими заинтересованными сторонами, обеспечивая прозрачность и понимание процессов [1].

Далее перечислены наиболее популярные методологии разработки [3]:

- Водопадная модель;
- Agile;
- Scrum;
- Kanban;
- Экстремальное программирование;
- Lean;
- Спиральная модель;
- DevOps.

Рассмотрим некоторые из них более подробно.

Водопадная модель – это традиционная и одна из самых ранних методологий разработки программного обеспечения, которая представляет собой последовательный процесс, разделённый на несколько фиксированных этапов: анализ требований, проектирование, разработка, тестирование, внедрение и поддержка. Каждый этап должен быть завершён перед переходом к следующему, без возврата к предыдущим шагам, что создаёт «каскадный» эффект. Водопадная модель удобна для проектов с чётко определёнными требованиями и стабильной спецификацией, но она может быть недостаточно гибкой в случае изменения требований на более поздних стадиях разработки, что делает её менее подходящей для проектов, где могут возникать частые изменения.

Agile – это гибкая методология разработки программного обеспечения, ориентированная на итеративный подход и постоянную адаптацию к изменениям. В Agile разработка делится на короткие циклы, называемые итерациями или спринтами, в конце которых выпускается работающая версия продукта. Основное внимание уделяется тесному взаимодействию с заказчиком, быстрой реакции на изменения требований и постоянному улучшению продукта. Agile способствует активной коммуникации внутри команды, непрерывной интеграции и тестированию, что позволяет быстрее достигать результатов и адаптироваться к изменяющимся условиям проекта.

Экстремальное программирование – это методология разработки, направленная на повышение качества программного обеспечения и гибкости в ответ на меняющиеся требования. XP основывается на таких инженерных практиках, как парное программирование, частые релизы, написание автоматизированных тестов до кода, постоянный рефакторинг, короткие циклы разработки и тесное взаимодействие с заказчиком.

Спиральная модель – это гибридная методология разработки программного обеспечения, которая сочетает элементы итеративного подхода и управления рисками.

Разработка в этой модели проходит через несколько циклов или витков, каждый из которых включает в себя фазы планирования, анализа рисков, проектирования, разработки и тестирования. Спиральная модель особое внимание уделяет оценке и минимизации рисков на каждом этапе проекта, что делает её подходящей для сложных и дорогостоящих проектов с высокой степенью неопределенности. Каждый виток спирали включает разработку прототипов, детальный анализ требований, оценку рисков и постепенное улучшение продукта на основе полученных результатов и обратной связи от заказчиков. На каждом этапе происходит корректировка целей и требований, что позволяет адаптировать проект к изменениям в бизнес-процессах или технологиях.

DevOps – это методология, которая объединяет процессы разработки программного обеспечения и эксплуатации с целью ускорения выпуска продукта, улучшения его качества и повышения эффективности взаимодействия между командами.

Основное внимание уделяется автоматизации и оптимизации всего жизненного цикла программного обеспечения – от разработки и тестирования до развертывания и мониторинга в производственной среде. DevOps внедряет практики непрерывной интеграции и непрерывной доставки, что позволяет командам быстро вносить изменения в код, автоматически тестировать их и оперативно развертывать на серверах.

Это сокращает время выхода на рынок и снижает вероятность ошибок. Ключевым элементом DevOps является культура совместной работы между разработчиками и операционными специалистами, что способствует улучшению процессов и минимизации простоев.

В таблице 4 представлены достоинства и недостатки методологий.

Таблица 4 – Достоинства и недостатки методологий

Методология	Описание
Водопадная модель	Последовательная модель разработки, в которой каждый этап завершается перед началом следующего, что создает каскадный эффект.
Agile	Гибкая методология разработки, основанная на итеративном подходе. Продукт разрабатывается в короткие итерации с постоянной обратной связью от заказчика и быстрыми адаптациями к изменениям.
Scrum	Фреймворк Agile, в котором работа команды организуется в короткие спринты. Важную роль играют постоянные ретроспективы, планирование и ежедневные встречи.
Kanban	Визуальная методология управления процессами разработки, где задачи организуются на канбан-доске. Цель – улучшить поток задач, обеспечив их плавное движение от начала до завершения.
Экстремальное программирование	Методология, фокусирующаяся на инженерных практиках, таких как парное программирование, тестирование до написания кода, частые релизы и постоянный рефакторинг для поддержания качества.
Спиральная модель	Итеративная модель разработки, которая включает в себя элементы управления рисками и повторяет фазы проектирования, разработки и тестирования в нескольких циклах, постепенно улучшая продукт.
DevOps	Методология, объединяющая процессы разработки и эксплуатации, с акцентом на автоматизацию и оптимизацию всего жизненного цикла разработки через практики непрерывной интеграции и доставки.

Для реализации мобильного приложения для педагогов учреждений образования была выбрана методология Waterfall, потому что этот проект, вероятно, имеет четко определенные требования и фиксированный набор функциональных возможностей. В образовательных учреждениях, как правило, существует необходимость в строгом соблюдении стандартов, нормативных актов и требований к отчетности, что делает последовательный и предсказуемый подход Waterfall особенно эффективным. В этой методологии каждый этап разработки – от анализа и проектирования до внедрения и тестирования – выполняется последовательно, что обеспечивает полное и тщательно документированное выполнение требований. Такой подход минимизирует риск того, что что-то важное будет упущено на ранних

стадиях, и обеспечивает возможность тщательной проверки всех аспектов системы до её внедрения. Waterfall также способствует созданию детализированной документации на каждом этапе, что критично для образовательной сферы, где качественная документация может быть обязательным требованием для аккредитации и отчетности перед государственными органами [4].

2.4 Логическое проектирование мобильного приложения

Архитектура программного средства – это высокоуровневая структура системы, которая описывает основные компоненты, их взаимодействие между собой и с внешней средой, а также принципы и стандарты, используемые для проектирования системы. Архитектура включает в себя различные аспекты, такие как логическая организация, модульная структура, распределение функций, принципы масштабируемости и безопасности. Она определяет общие решения, связанные с выбором технологий, способов хранения данных, коммуникации между компонентами, управлением потоками данных и обработкой запросов. Архитектура программного средства является фундаментом для его разработки и играет ключевую роль в достижении таких целей, как эффективность, поддерживаемость, расширяемость и надежность системы.

Модульная структура проекта – это способ организации программного обеспечения, при котором система разбивается на отдельные модули, каждый из которых является самостоятельным и отвечает за определённую функцию или набор функций. Модули работают как независимые блоки, которые могут взаимодействовать друг с другом через строго определённые интерфейсы, что позволяет минимизировать зависимости между различными частями системы. Такой подход облегчает разработку, поскольку модули можно разрабатывать, тестировать и внедрять поэтапно, не затрагивая другие компоненты системы. Модульная структура упрощает поддержку и сопровождение проекта: если

необходимо внести изменения или улучшения в один из модулей, это можно сделать без риска нарушить работу всей системы, при условии сохранения интерфейсов. Кроме того, модульность способствует повторному использованию кода. Один и тот же модуль может быть интегрирован в различные проекты или применён в разных частях одного проекта, что снижает затраты на разработку и тестирование.

Модульная структура также улучшает управляемость сложными проектами: каждый модуль можно распределить между различными командами разработчиков, что повышает производительность и ускоряет выполнение проекта.

В целом, модульная структура делает систему более гибкой и масштабируемой, позволяя легче адаптироваться к изменениям требований, быстрее исправлять ошибки и внедрять новые функции.

MVVM – это архитектурный паттерн, используемый для разделения логики пользовательского интерфейса и бизнес-логики в приложении.

В этой модели View отвечает за отображение данных и подписывается на изменения, которые поступают из ViewModel, в то время как ViewModel управляет данными и взаимодействует с Model, обеспечивая поток данных между ними. Model, в свою очередь, содержит бизнес-логику и взаимодействует с источниками данных, оставаясь независимой от интерфейса.

Архитектура программного средства представлена на рисунке 6.

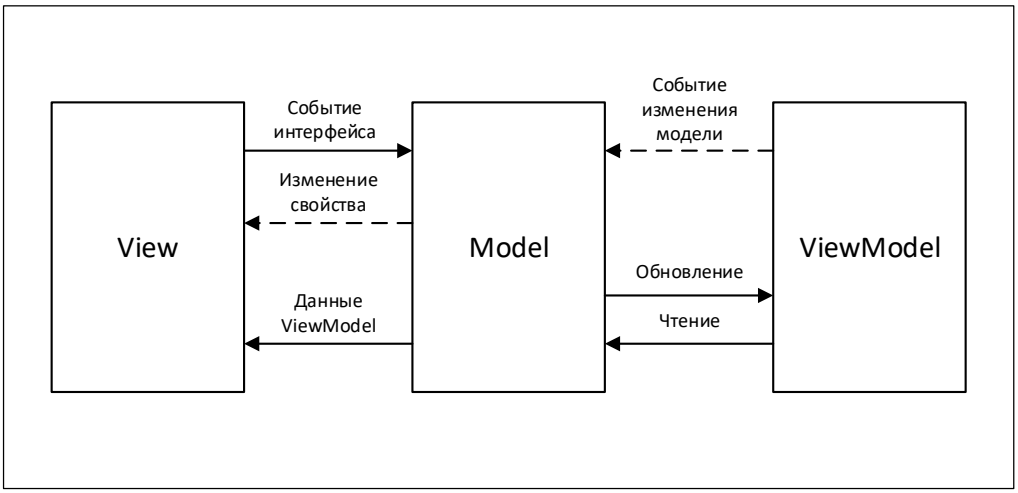

Рисунок 6 – Архитектура программного средства

Одним из главных достоинств паттерна является четкое разделение ответственности между компонентами. View отвечает только за отображение данных и взаимодействие с пользователем, ViewModel – за логику представления данных и их обработку, а Model – за управление данными и бизнес-логику. Это упрощает разработку, делает код более структурированным и облегчает его сопровождение. Еще одним плюсом MVVM является улучшенная тестируемость. Так как логика представления отделена от пользовательского интерфейса, ViewModel можно тестировать независимо от View, используя юнит-тесты. Это позволяет быстрее находить и исправлять ошибки на ранних этапах разработки. Кроме того, MVVM поддерживает лучшее управление состоянием данных и жизненным циклом. ViewModel не зависит от жизненного цикла Activity или Fragment, что помогает сохранять данные при изменениях конфигурации устройства. Это позволяет избегать потери данных и делает интерфейс более устойчивым к изменениям.

Диаграмма пакетов представлена на рисунке 7.

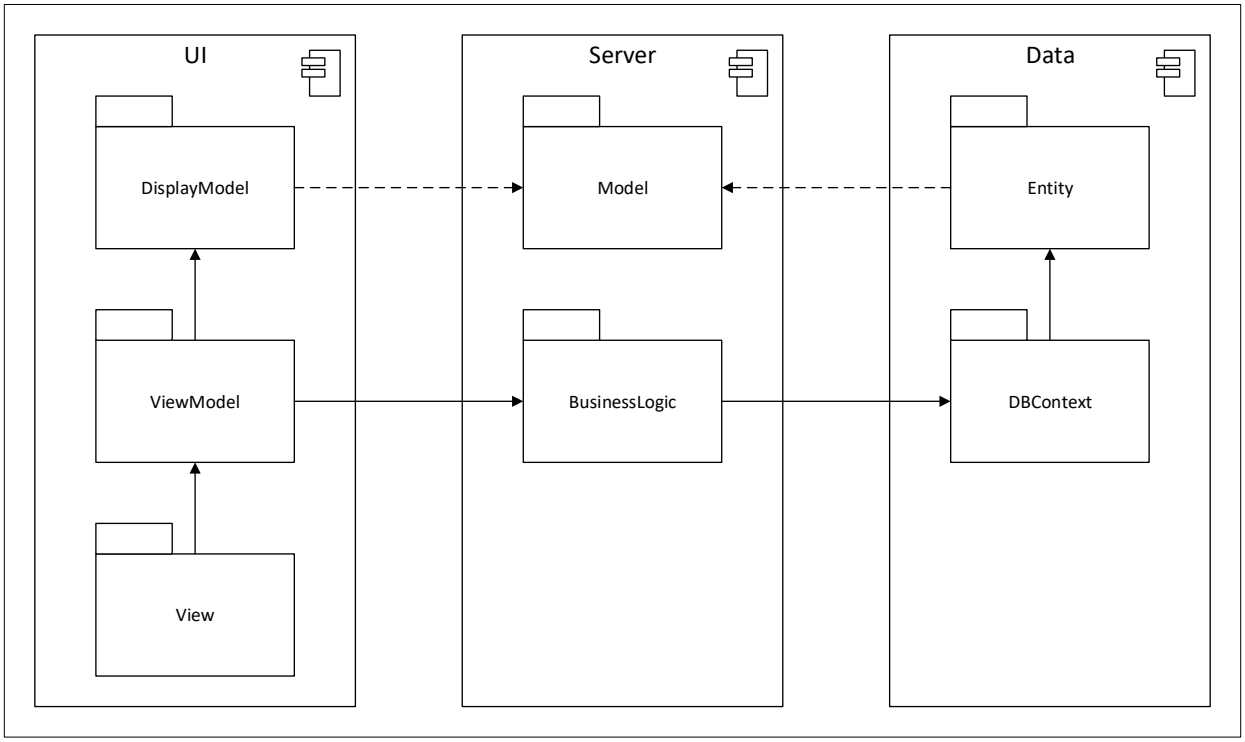

Рисунок 7 – Диаграмма пакетов программного средства

Диаграмма пакетов с разделением на три основные области – UI, Server и Data – помогает структурировать приложение по функциональным слоям. В пакете UI находятся компоненты, отвечающие за пользовательский интерфейс: View отображает данные, ViewModel управляет логикой взаимодействия с данными, а DisplayModel служит для передачи данных между View и ViewModel. Пакет Server обрабатывает бизнес-логику: BusinessLogic отвечает за правила и логику работы приложения, а Model управляет данными, которые поступают в бизнес-логику. Пакет Data отвечает за взаимодействие с базой данных: DBContext обеспечивает доступ к данным, а Entity представляет собой объекты данных, которые хранятся и управляются в базе данных. Такое разделение помогает улучшить модульность, поддерживаемость и масштабируемость системы.

Диаграмма компонентов необходима для того, чтобы наглядно показать, как различные части системы реализуются и взаимодействуют друг с другом на уровне физической реализации. Она фокусируется на представлении

компонентов, которые могут быть модулями, библиотеками, сервисами или микросервисами, и на их взаимосвязях через интерфейсы, протоколы и каналы связи. Диаграмма компонентов помогает разработчикам, архитекторам и всем членам команды понять структуру системы, ее модульность и зависимость между компонентами. Это особенно важно при работе над сложными или распределенными системами, где множество компонентов должны правильно взаимодействовать для обеспечения корректной работы. Диаграмма компонентов также упрощает интеграцию различных модулей, тестирование, улучшает поддержку и помогает управлять изменениями в архитектуре, так как она четко определяет зоны ответственности и интерфейсы, через которые компоненты обмениваются данными и запросами.

Диаграмма компонентов программного средства представлена на рисунке 8.

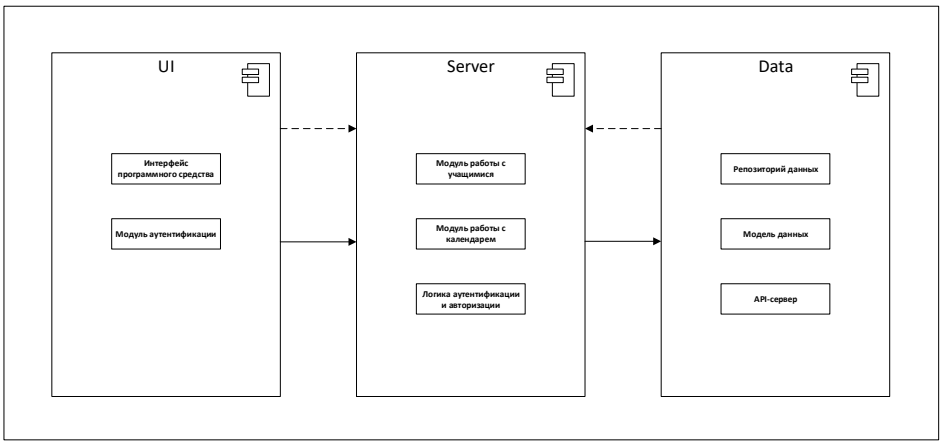

Рисунок 8 – Диаграмма компонентов программного средства

Таким образом, архитектура программного средства была рассмотрена с точки зрения четкого разделения на слои и компоненты, что свидетельствует о продуманности и хорошей структуризации системы. Такое разделение обеспечивает модульность, легкость в поддержке и масштабировании, а также четкую зону ответственности для каждого компонента. Организация системы по слоям, а именно: интерфейс пользователя, бизнес-логика и управление

данными – позволяет эффективно управлять сложностью приложения, улучшает тестируемость и облегчает интеграцию различных модулей. Эта архитектура способствует гибкости и надежности системы, обеспечивая долгосрочное развитие и поддержание программного обеспечения.

2.5 Проектирование модели данных

Реляционные базы данных являются одним из самых популярных и эффективных инструментов для хранения данных, особенно когда информация организована в виде взаимосвязанных таблиц. Они используют модель на основе реляционной алгебры, где данные хранятся в виде строк и столбцов, что позволяет легко определять связи между различными сущностями через ключи и отношения. Эта структура упрощает управление сложными данными, их запрос и обновление, обеспечивая целостность и непротиворечивость данных в системе. Одним из ключевых преимуществ реляционных баз данных является поддержка языка SQL. SQL позволяет эффективно управлять данными, выполняя сложные запросы для выборки, вставки, обновления или удаления информации. Благодаря мощным возможностям SQL, разработчики могут легко извлекать нужные данные, объединять таблицы, фильтровать результаты и проводить агрегацию данных, что делает реляционные базы данных удобными для работы с большими объемами информации. Реляционные базы данных также обеспечивают строгую целостность данных через механизмы транзакций. Такие свойства, как атомарность, согласованность, изоляция и устойчивость, гарантируют, что все изменения данных будут либо полностью завершены, либо отменены в случае ошибки, что помогает предотвратить потерю или повреждение данных. Это особенно важно для критически важных систем, где надежность хранения данных является приоритетом. Кроме того, реляционные базы данных легко масштабируются и поддерживают сложные отношения между таблицами с помощью внешних ключей и ограничений целостности. Это позволяет

эффективно управлять связями между сущностями, минимизируя дублирование данных и обеспечивая точное их обновление. С помощью индексов и других инструментов оптимизации производительность базы данных может быть существенно улучшена, что делает реляционные базы данных идеальными для обработки больших и сложных наборов данных.

Для организации эффективного хранения данных в базе данных необходимо учитывать ряд факторов, которые напрямую влияют на производительность, масштабируемость, надежность и безопасность системы. Эти факторы помогают оптимизировать доступ к данным, минимизировать задержки, обеспечить целостность и защиту данных, а также подготовить систему к потенциальному росту нагрузки. Рассмотрение этих факторов на этапе проектирования базы данных значительно упрощает последующее управление и поддержку системы.

Далее перечислены основные факторы эффективного хранения данных:

– тип базы данных;
– нормализация данных;
– индексирование;
– кэширование данных;
– оптимизация запросов;
– управление транзакциями.

Сущность – это объект или понятие из реального мира, которое обладает уникальными характеристиками и которое нужно моделировать в базе данных или в системе. Сущность может быть физическим объектом, таким как человек или абстрактным понятием, таким как событие, оценка или расписание. Каждая сущность описывается набором атрибутов, которые определяют её уникальные черты и значения. В контексте разработки программного обеспечения сущности служат основой для представления данных и структурирования предметной области, обеспечивая взаимодействие с различными компонентами системы.

Далее необходимо выделить сущности предметной области:

– ученик;

– учитель;

– учетная запись;

– событие;

– расписание;

– предмет;

– класс.

Далее необходимо представить логическую модель данных (рисунок 9).

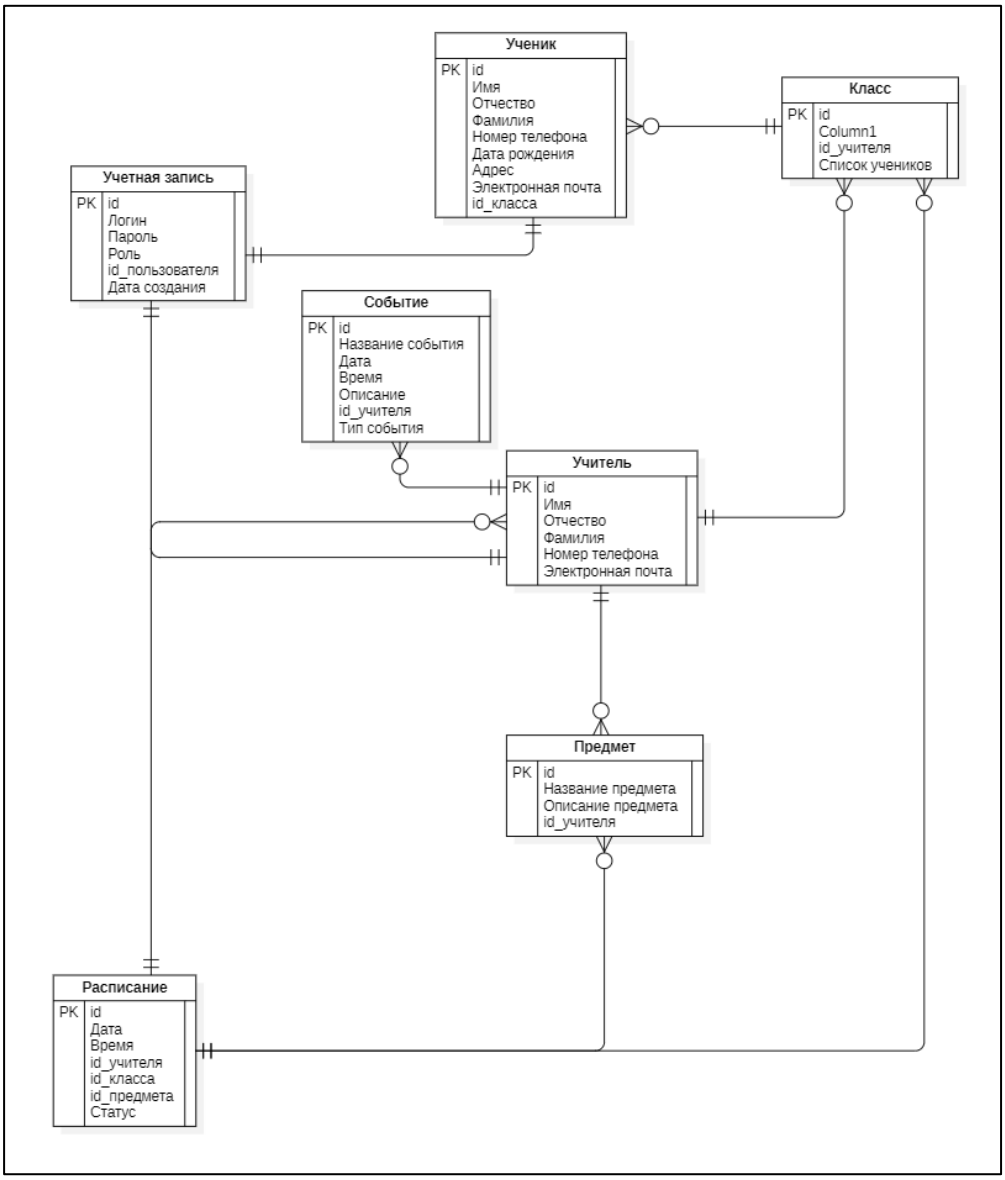

Рисунок 9 – Логическая модель данных программного средства

Эффективная организация хранения данных основывается на правильном выборе типа базы данных, продуманной структуре сущностей и оптимальных связях между ними.

Также важно учитывать индексацию, кэширование и регулярное резервное копирование данных для повышения скорости доступа и надежности системы.

В результате такая организация данных способствует стабильной работе приложения, его масштабируемости и упрощению сопровождения, обеспечивая качественное управление и защиту информации.

Выводы по главе 2

Благодаря логическому проектированию программного обеспечения можно создать качественный и надёжный в использовании продукт, который соответствует требованиям пользователей, легко расширяется, обновляется и поддерживается, а также обеспечивает удобный и интуитивно понятный интерфейс.

Глава 3 Реализация и тестирование мобильного приложения

3.1 Выбор средств разработки ПО

Этап выбора программных технологий и платформ для разработки информационной системы играет ключевую роль в успехе проекта. В качестве первого шага необходима оценка существующих технологий и платформ, которые могут поддержать разработку и эксплуатацию системы.

Рассматриваются такие параметры, как стабильность и актуальность технологий, совместимость с другими системами, а также наличие профессиональной поддержки и документации.

Например, при выборе между десктопным и веб-приложением следует учесть, как преподаватели планируют использовать систему, и какие устройства предпочтительны для работы [17].

Далее проводится выбор инструментов разработки, а именно языков программирования, фреймворков и баз данных.

Здесь учитываются требования к производительности, предпочтения команды разработчиков, а также возможные ограничения, такие как доступные ресурсы и бюджет.

Завершающий этап – выбор архитектуры и интеграционных решений. Определение архитектурного подхода должно основываться на том, как будет организовано взаимодействие между различными компонентами системы.

Интеграционные решения, такие как API, также играют важную роль, особенно если система будет взаимодействовать с внешними образовательными платформами.

В конечном итоге, выбор технологий и платформ должен учитывать долгосрочную поддержку, гибкость и возможность масштабирования системы в будущем.

Таким образом, на этапе анализа технологий будут выбраны:

– программная платформа;

– язык программирования;
– СУБД;
– архитектурные решения.

Последовательно приступим к анализу технологий.

Для разработки мобильных приложений существует несколько популярных платформ, каждая из которых предлагает уникальные возможности и особенности.

Рассмотрим основные платформы, которые обычно выбираются для создания мобильных систем [12]:

– Xamarin: кроссплатформенная среда разработки от Microsoft, позволяющая создавать приложения для iOS и Android с использованием языка C# и платформы .NET;
– Xamarin поддерживает значительную часть нативного функционала и предоставляет доступ к аппаратному обеспечению устройства;
– Flutter: фреймворк от Google, основанный на языке программирования Dart, который также позволяет создавать кроссплатформенные приложения для Android и iOS с использованием одного кода;
– Flutter популярен за свою производительность и возможность быстрого прототипирования благодаря гибкому виджетному подходу;
– React Native: фреймворк от Facebook, который позволяет писать кроссплатформенные приложения на языке JavaScript с использованием компонентов React. React Native дает возможность быстро разрабатывать приложения с производительностью, близкой к нативной, и обширной поддержкой библиотек.

Сравнение программных платформ представлено в таблице 5:

Таблица 5 – Сравнение программных платформ

Платформа	Язык разработки	Поддержка платформ	Кроссплатформенность	Доступ к нативным функциям
Xamarin	C#	iOS, Android	Высокая	Полный доступ
Flutter	Dart	iOS, Android	Высокая	Ограниченный через плагины
React Native	JavaScript	iOS, Android	Средняя	Доступ через библиотеки

На основании представленных данных, Xamarin является оптимальным выбором для кроссплатформенной разработки, поскольку он сочетает полную поддержку нативных функций и высокий уровень кроссплатформенности, что делает его эффективным для создания мобильных приложений, требующих глубокую интеграцию с устройством [13].

Для реализации мобильного приложения важно выбрать подходящую систему управления базами данных, которая будет эффективно работать на мобильных устройствах, поддерживать основные операции с данными и обеспечивать необходимый уровень производительности. Рассмотрим несколько популярных СУБД для мобильных приложений [14]:

- SQLite: легковесная и встраиваемая реляционная СУБД, которая широко используется в мобильной разработке. SQLite не требует настройки сервера и позволяет хранить данные непосредственно на устройстве, что делает её идеальной для локальных баз данных [18];
- Realm: современная СУБД, ориентированная на мобильные устройства и разработанная для высокой производительности и простоты использования;
- Firebase Realtime Database: облачная база данных от Google, разработанная для синхронизации данных в реальном времени. Firebase хорошо подходит для приложений, которым необходимы обновления данных на всех подключенных устройствах.

Сравнение программных платформ представлено в таблице 6.

Таблица 6 – Сравнительная таблица СУБД

СУБД	Тип	Хранилище	Синхронизация	Размер и ресурсопотребление
SQLite	Реляционная	Локальное	Не поддерживается	Легковесная, минимальное потребление
Realm	Объектная	Локальное/ облако	Поддерживается	Средний размер, высокая производительность
Firebase Realtime Database	Облачная	Облако	Поддерживается	Требуется интернет, высокий ресурс

SQLite является оптимальным выбором для реализации проекта, так как он легковесный, не требует серверной установки и идеально подходит для локального хранения данных на мобильных устройствах.

Для разработки мобильного приложения важен выбор подходящей интегрированной среды разработки, которая поддерживает необходимые языки программирования, упрощает процесс разработки и отладки, а также предоставляет функционал для кроссплатформенной разработки.

Рассмотренных распространенных IDE:

- Visual Studio: мощная IDE от Microsoft, поддерживающая C#, .NET, и Xamarin, что делает её идеальным выбором для кроссплатформенной разработки приложений для iOS и Android. Visual Studio обладает богатым набором инструментов для отладки, эмуляции и тестирования, а также интеграцией с Azure [20];
- Android Studio: официальная IDE для разработки Android-приложений от Google, которая поддерживает Java, Kotlin и XML. Android Studio предоставляет эмуляторы, анализ производительности и специфические для Android инструменты;
- Xcode: IDE от Apple для разработки приложений на iOS и macOS, поддерживающая Swift и Objective-C. Xcode предоставляет

инструменты для создания интерфейсов, отладки, тестирования и работы с симуляторами, но работает только на устройствах с macOS.

Сравнение программных платформ представлено в таблице 7.

Таблица 7 – Сравнительная таблица IDE

IDE	Языки	Платформы	Кроссплатформенная разработка	Дополнительные возможности
Visual Studio	C#, .NET, Xamarin	iOS, Android	Высокая	Интеграция с Azure, мощная отладка
Android Studio	Java, Kotlin	Android	Низкая	Специфичные инструменты для Android
Xcode	Swift, Objective-C	iOS, macOS	Низкая	Работа с интерфейсом, тестирование на iOS

Для реализации мобильного приложения выбраны технологии, которые обеспечивают высокую производительность, кроссплатформенность и удобство в разработке. Visual Studio с поддержкой Xamarin была выбрана в качестве основной IDE, что позволяет создавать единый код для iOS и Android на языке C#. Для хранения данных была выбрана SQLite – легковесная и быстрая СУБД, подходящая для локального хранения информации на устройстве. Эти технологии обеспечат стабильную работу приложения, эффективное использование ресурсов устройства и доступность данных в офлайн-режиме, что особенно важно для информационных приложений, ориентированных на широкий круг пользователей [15].

Правильный выбор средств разработки на этапе проектирования является ключевым моментом в процессе разработки программного обеспечения. Это важно по многим причинам. Правильный выбор средств разработки может значительно повысить эффективность процесса разработки, уменьшить время, необходимое для создания программного продукта, а также снизить затраты на разработку. Также, правильный выбор средств разработки может повысить качество программного обеспечения. Некоторые инструменты могут быть более подходящими для определенных задач, что

может повысить надежность и безопасность продукта. Правильный выбор средств разработки может облегчить сопровождение и поддержку программного обеспечения в будущем. Некоторые инструменты и технологии могут быть более подходящими для дальнейшего развития и сопровождения программного продукта [8].

3.2 Разработка логической структуры базы данных

Логическая структура базы данных необходима для упорядочивания данных и обеспечения целостности информации в рамках системы. Она представляет собой абстрактное описание того, как данные взаимосвязаны и организованы в рамках базы данных. Логическая структура определяет, какие таблицы, связи, индексы и ограничения будут существовать в базе данных, как они будут взаимодействовать друг с другом и какие правила будут применяться для поддержания согласованности данных. Это критично для того, чтобы все данные хранились систематизировано и могли быть извлечены быстро и эффективно при помощи запросов. Логическая структура обеспечивает понимание того, как информация будет обрабатываться в системе, помогает избегать избыточности данных и поддерживать оптимальную производительность при работе с большими объемами информации.

Еще одним ключевым аспектом логической структуры базы данных является поддержание целостности и корректности данных на протяжении всего жизненного цикла системы. С помощью ограничений, таких как первичные и внешние ключи, уникальные индексы и проверки, логическая структура позволяет контролировать, чтобы данные были правильными и согласованными между разными таблицами. Это особенно важно в сложных информационных системах, где данные могут поступать из разных источников или обрабатываться параллельно. Кроме того, логическая структура облегчает масштабирование системы, позволяя администраторам

базы данных и разработчикам вносить изменения, добавлять новые элементы или модифицировать существующие, не нарушая общую логику и консистентность базы данных.

SQLite предоставляет мощные инструменты для проектирования и разработки логического уровня данных, начиная с возможностей моделирования данных и структурирования таблиц. В SQLite поддерживается использование базовых типов данных, что позволяет создавать гибкие и лёгкие структуры для хранения информации. Разработчик может определять таблицы с необходимыми столбцами и типами данных, а также устанавливать связи между ними с помощью внешних ключей. Это способствует созданию согласованных и взаимосвязанных баз данных, которые легко адаптируются к различным задачам. Важную роль в проектировании играет структура базы данных, которая может быть организована таким образом, чтобы улучшить её читаемость и управление. Эти инструменты помогают создавать базы данных, которые остаются стабильными и эффективными даже при увеличении объёма данных. Кроме того, SQLite поддерживает обеспечение целостности данных на уровне логического проектирования благодаря встроенной поддержке ограничений и триггеров. Ограничения, такие как первичные и внешние ключи, уникальность, проверки условий и запрет на пустые значения, обеспечивают корректность и согласованность данных, исключая возможность ввода некорректных или противоречивых записей. Триггеры, которые могут автоматически выполнять заданные действия при изменении данных в таблицах, позволяют автоматизировать сложные правила и процессы, упрощая их реализацию и снижая вероятность ошибок. Эти функции помогают разработчикам формировать устойчивую и управляемую логическую структуру базы данных, обеспечивая надёжную работу приложения и высокую производительность.

Таким образом, информационную поддержку приложению будет осуществлять база данных следующей структуры:

Учётная запись представляет собой сущность, содержащую информацию о пользователях системы, включая логин, пароль и роль, что позволяет различным категориям пользователей (учителя, ученики, администраторы) входить в систему и взаимодействовать с ней. Она также включает идентификатор пользователя и дату создания учетной записи.

Рассмотрим свойства данной сущности:

- id: уникальный идентификатор записи пользователя;
- login: логин пользователя для входа в систему;
- password: пароль пользователя для авторизации;
- role: роль пользователя в системе (например, администратор, учитель, ученик);
- user_id: идентификатор связанного пользователя в других таблицах;
- creation_date: дата создания учетной записи.

Ученик является ключевой сущностью, представляющей школьников. В этой сущности хранится информация о персональных данных учеников, таких как имя, фамилия, отчество, контактные данные, адрес и электронная почта, а также связь с классом, к которому они прикреплены.

Рассмотрим свойства данной сущности:

- id: уникальный идентификатор записи ученика;
- first_name: имя ученика;
- middle_name: отчество ученика;
- last_name: фамилия ученика;
- phone_number: контактный номер телефона ученика;
- birth_date: дата рождения ученика;
- address: адрес проживания ученика;
- email: электронная почта ученика;
- class_id: идентификатор класса, к которому прикреплен ученик.

Класс представляет группу учеников и связан с учителем, который отвечает за этот класс. Эта сущность также включает список учеников, что

позволяет отслеживать, какие ученики обучаются в данном классе, и устанавливать связи с расписанием и событиями.

Рассмотрим свойства данной сущности:

– id: уникальный идентификатор записи класса;

– teacher_id: идентификатор учителя, закрепленного за классом;

– student_list: список учеников, прикрепленных к классу.

Событие отражает различные школьные мероприятия и уроки, привязанные к конкретной дате, времени и учителю. Событие может представлять урок, встречу или другое мероприятие, с указанием его типа, названия и краткого описания.

Рассмотрим свойства данной сущности:

– id: уникальный идентификатор записи события;

– event_name: название события;

– date: дата проведения события;

– time: время проведения события;

– description: краткое описание события;

– teacher_id: идентификатор учителя, ответственного за событие;

– event_type: тип события.

Учитель является сущностью, в которой хранится информация о преподавателях, включая их имя, фамилию, отчество, контактные данные и электронную почту. Эта сущность связана с предметами, которые ведет учитель, а также с классами, за которые он отвечает.

Рассмотрим свойства данной сущности:

– id: уникальный идентификатор записи учителя;

– first_name: имя учителя;

– middle_name: отчество учителя;

– last_name: фамилия учителя;

– phone_number: контактный номер телефона учителя;

– email: электронная почта учителя.

Предмет представляет собой сущность, хранящую информацию о различных дисциплинах, преподаваемых в учебном заведении. Она связана с учителями, которые отвечают за проведение занятий по этим предметам, и содержит название и описание предмета.

Рассмотрим свойства данной сущности:

– id: уникальный идентификатор записи предмета;

– subject_name: название предмета;

– subject_description: краткое описание предмета;

– teacher_id: идентификатор учителя, преподающего данный предмет.

Расписание управляет информацией о проведении занятий, включая дату и время, а также связи с учителями, классами и предметами. Это позволяет организовать и структурировать учебный процесс, отслеживать статус занятий и учитывать их изменения.

Рассмотрим свойства данной сущности:

– id: уникальный идентификатор записи расписания;

– date: дата проведения занятия;

– time: время проведения занятия;

– teacher_id: идентификатор учителя, проводящего занятие;

– class_id: идентификатор класса, которому назначено занятие;

– subject_id: идентификатор предмета, преподаваемого на занятии;

– status: статус занятия (например, подтверждено, отменено).

Разработанная логическая модель данных демонстрирует высокую эффективность за счет четкого разделения сущностей и установления однозначных связей между ними. Такая структура позволяет организованно хранить данные об учениках, учителях, классах, событиях и расписании, обеспечивая целостность и согласованность данных. Использование внешних ключей и привязка записей к конкретным ролям и событиям способствует надежности системы и упрощает дальнейшее масштабирование и модификацию. В результате, модель данных отвечает требованиям гибкости и

надежности, необходимым для информационной системы образовательного учреждения.

3.3 Проектирование и разработка пользовательского интерфейса

При проектировании и разработке пользовательского интерфейса необходимо учитывать несколько важных подходов, которые обеспечат удобство, функциональность и эстетику интерфейса. Эти подходы помогут создать интерфейс, который будет не только визуально привлекательным, но и интуитивно понятным для пользователей.

Ориентация на пользователя. В центре проектирования должен находиться пользователь и его задачи. Интерфейс должен быть интуитивным и доступным, с четкими и понятными элементами управления. Это достигается через исследования пользовательского опыта, тестирование с реальными пользователями и анализ их поведения. Проектирование интерфейса с учетом пользовательских ожиданий и сценариев использования помогает уменьшить барьеры при работе с приложением и повысить удовлетворенность пользователей.

Принципы модульности и гибкости. Интерфейс должен быть разработан таким образом, чтобы элементы могли легко адаптироваться к изменениям. Это особенно важно для поддержания кроссплатформенных приложений и создания интерфейсов, которые корректно отображаются на различных устройствах. Использование адаптивного дизайна и компонентов с переиспользуемыми элементами помогает поддерживать единообразие интерфейса на разных платформах и разрешениях экрана.

Минимализм и когнитивная нагрузка. Чем меньше действий и усилий требуется от пользователя для выполнения задачи, тем лучше. Простота и минимализм в дизайне помогают пользователям быстрее ориентироваться в интерфейсе и избегать путаницы. Важно избегать избыточной информации и чрезмерного количества элементов на экране. Концентрация на главных

функциях приложения с возможностью быстрого доступа к ним улучшает взаимодействие и уменьшает вероятность ошибок со стороны пользователей.

Консистентность и предсказуемость. Интерфейс должен быть согласованным во всех частях приложения. Например, кнопки одного типа должны выглядеть и вести себя одинаково во всех разделах приложения. Консистентность помогает пользователям предсказуемо взаимодействовать с интерфейсом, что снижает обучение и повышает комфорт работы. Этого можно добиться с помощью общих стилей, паттернов взаимодействия и соблюдения платформенных гайдлайнов.

В среде разработки на Xamarin проектирование и разработка пользовательского интерфейса также требует учета особенностей платформы и доступных инструментов. Xamarin активно используется для создания кроссплатформенных приложений, где пользовательский интерфейс может разрабатываться с использованием XAML и программных методов [19].

Однако с развитием Xamarin.Forms появились новые подходы и возможности, упрощающие создание UI и его управление. Xamarin.Forms предлагает мощный инструмент для декларативного создания интерфейсов, аналогичный Jetpack Compose, с использованием XAML. Этот подход позволяет разработчикам описывать интерфейс декларативно, задавая структуру и поведение элементов прямо в разметке [21].

Когда данные изменяются, интерфейс обновляется автоматически благодаря механизму привязки данных. Это позволяет упростить разработку динамических интерфейсов и уменьшить количество ошибок, связанных с состоянием UI. Использование XAML делает код более читаемым и поддерживаемым, а также снижает объем рутинного программного кода.

В Xamarin разработчики могут использовать возможности языка C# и платформы .NET, такие как LINQ, асинхронные операции и расширенные методы, для упрощения взаимодействия с пользовательским интерфейсом.

Асинхронные методы особенно полезны для обработки задач, таких как загрузка данных из API или взаимодействие с локальной базой данных, без

блокировки основного потока интерфейса. Это обеспечивает плавный пользовательский опыт и высокую отзывчивость приложения. Xamarin.Forms поддерживает паттерн MVVM, который легко интегрируется благодаря таким инструментам, как BindableObject и Command.

MVVM разделяет логику приложения и представление, где ViewModel отвечает за управление состояниями и взаимодействие с данными, а View автоматически обновляется при изменении привязанных данных.

Это позволяет достичь реактивного поведения и делает код более модульным, что значительно упрощает поддержку и масштабирование приложения.

Дополнительно Xamarin предлагает возможности кастомизации для каждой платформы через рендереры, что позволяет создавать уникальный дизайн для iOS и Android. Такой подход даёт разработчикам гибкость, позволяя использовать нативные элементы управления, когда это необходимо.

Xamarin.Forms также поддерживает высокоуровневые API для работы с анимациями, стилями и адаптивным дизайном, что делает разработку пользовательского интерфейса быстрой и эффективной.

Для проектирования макета пользовательского интерфейса будет использовано программное средство Figma [16].

На рисунке 10 представлена форма для заполнения данных об ученике.

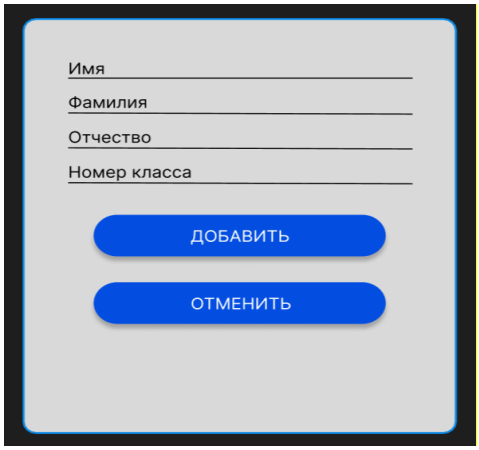

Рисунок 10 – Форма заполнения данных об учащемся

Исходя из представленного интерфейса, основными принципами, заложенными в программное средство, будут простота и интуитивность использования. Форма добавления данных содержит минимальное количество полей для ввода, что упрощает взаимодействие с системой и снижает вероятность ошибок ввода. Визуально интерфейс разделен на четкие блоки, а большие и понятные кнопки для действий делают интерфейс доступным для всех категорий пользователей, минимизируя когнитивную нагрузку.

Этот интерфейс ориентирован на быстроту выполнения задач и логичное, последовательное представление данных, что делает его удобным и эффективным в использовании.

3.4 Маршрутизация страниц приложения

Механизм маршрутизации в Xamarin.Forms обеспечивает управление переходами между страницами приложения, создавая интуитивно понятную навигацию для пользователя. Этот процесс организуется с использованием встроенных инструментов, таких как NavigationPage, TabbedPage, MasterDetailPage и современный Shell.

Каждый из этих подходов предоставляет свои возможности и нацелен на реализацию различных сценариев навигации, начиная от простой линейной структуры до сложных иерархий с боковыми меню и вкладками. Одним из самых популярных инструментов маршрутизации является NavigationPage, который использует стек для управления переходами.

Этот механизм добавляет новые страницы в стек при навигации вперед и удаляет их при возврате. Такой подход упрощает управление состоянием приложения, а встроенная кнопка «Назад» делает навигацию естественной для пользователя.

Навигация через стек хорошо подходит для линейных сценариев, например, последовательного отображения деталей выбранного элемента. Для приложений с несколькими разделами, такими как вкладки, используется

TabbedPage. Этот инструмент позволяет организовать навигацию в виде переключаемых вкладок, где каждая вкладка представляет отдельную страницу или группу страниц. Вкладки обеспечивают быстрый доступ к основным разделам приложения, не требуя сложных переходов.

Связь функциональных возможностей и маршрутизации интерфейса приложения должна быть логичной, последовательной и четко отражать пользовательские сценарии работы. Основная задача – сделать так, чтобы каждая функция приложения была доступна пользователю через интуитивно понятные переходы и действия в интерфейсе.

Функциональные возможности приложения должны определять структуру маршрутизации. Это значит, что пользовательские действия должны вести к соответствующим страницам, где реализуются эти функции. Например, при нажатии кнопки «Добавить» пользователь должен быть перенаправлен на страницу, где происходит ввод необходимых данных, а при завершении этого действия – возвращен на основную страницу с обновленным списком.

Каждая функция должна иметь четкий и однозначный путь в приложении, без ненужных шагов и сложных переходов. Маршрутизация должна учитывать контекст выполнения каждой функции. Это включает передачу и сохранение состояния данных между страницами.

Если пользователь заполнил часть формы и перешел на другую страницу, система должна корректно передать введенные данные в новый экран или сохранить их при возврате.

Важно обеспечить, чтобы пользователи могли плавно перемещаться между различными функциями приложения без потери данных или необходимости повторного ввода.

Таким образом, программное средство будет иметь следующую маршрутизацию:

Стартовый экран: Вход в систему с проверкой роли пользователя. После успешного входа пользователь будет перенаправлен на соответствующую страницу.

Экран администратора:

- просмотр всех учетных записей: Маршрутизация на страницу с таблицей всех пользователей, где администратор может управлять учетными записями. Возможность перейти на страницу редактирования учетной записи. Возможность удаления учетной записи с соответствующей страницей подтверждения. Поиск учетной записи с фильтрацией в списке или переходом на отдельную страницу поиска;
- назначение ролей: Страница назначения ролей пользователям или создания новых учетных записей;
- добавление учетной записи: Форма для добавления новой учетной записи с возможностью указания роли;
- редактирование учетной записи: Страница для внесения изменений в существующую учетную запись.

Экран преподавателя:

Работа с учащимися (рисунок 11) – главная страница преподавателя с доступом к списку учеников:

- переход на страницу добавления ученика;
- возможность редактирования данных ученика на отдельной странице;
- страница подтверждения для удаления ученика;
- просмотр списка учащихся с возможностью поиска и фильтрации.

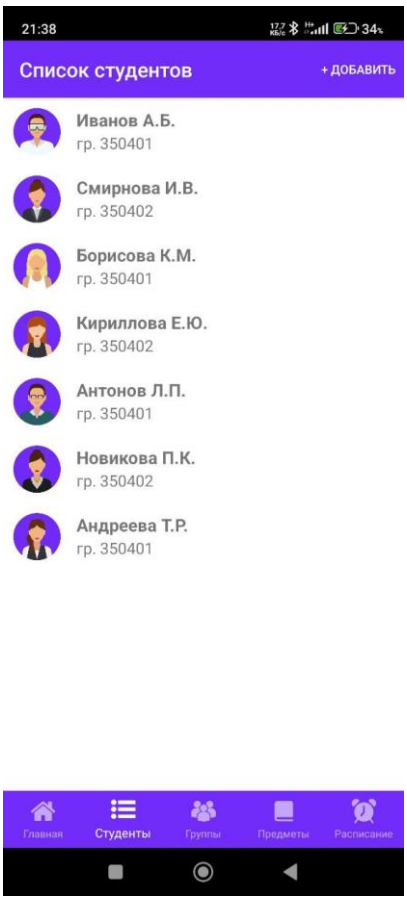

Рисунок 11 – Форма «Список студентов»

Переходы между страницами осуществляются на основе взаимодействия пользователя с элементами интерфейса, такими как календарь, кнопки или списки. Например, при выборе даты в календаре приложение может перенаправить пользователя на страницу с подробным списком событий или на страницу создания нового события для выбранной даты. Также возможен переход на страницу редактирования события, если оно уже существует.

Важным аспектом является сохранение состояния календаря и отображение данных, что позволяет пользователю возвращаться к предыдущему экрану без потери информации.

Экранные формы приложения представлены в приложении А.

3.5 Тестирование мобильного приложения

Тестирование с помощью тест-кейсов – это структурированный процесс проверки функциональности программного обеспечения на соответствие ожидаемым требованиям. Тест-кейсы представляют собой детализированные сценарии, описывающие шаги, которые нужно выполнить, чтобы убедиться в правильной работе функции или системы. Каждый тест-кейс включает в себя набор условий, действий и ожидаемых результатов. Цель использования тест-кейсов заключается в том, чтобы покрыть как можно больше аспектов работы системы, выявить возможные ошибки и удостовериться, что система ведет себя корректно в различных ситуациях. Создание тест-кейсов начинается с анализа требований системы. На основании этих требований определяется, какие функции и сценарии использования нужно протестировать. Тест-кейсы должны быть понятными и конкретными, с четко определенными шагами, чтобы любой тестировщик мог их выполнить независимо от того, был ли он вовлечен в разработку системы. Важная часть тест-кейса – это ожидаемый результат, который позволяет тестировщику сравнить фактическое поведение системы с тем, как система должна работать в соответствии с требованиями [5].

Исполнение тест-кейсов помогает выявить дефекты, ошибки или некорректное поведение системы. Если результат выполнения тест-кейса соответствует ожидаемому, тест считается успешным, если нет – это указывает на проблему, требующую исправления. Каждый тест-кейс фиксируется, и результаты сохраняются для дальнейшего анализа и отчетности. Тест-кейсы могут быть как ручными, когда тестировщик выполняет все шаги вручную, так и автоматизированными, когда для выполнения сценариев используются специальные инструменты автоматизации тестирования. Регулярное выполнение тест-кейсов важно для обеспечения качества программного обеспечения на протяжении всего жизненного цикла разработки. Тест-кейсы позволяют убедиться, что система

корректно работает после внесения изменений или добавления новой функциональности, обеспечивая регрессионное тестирование.

В таблицах 8 – 11 представлены тест-кейсы для тестирования программного средства.

Таблица 8 – Тестирование добавления учетной записи

Тест	Название тест-кейса	Предусловия	Шаги теста	Ожидаемый результат
ТС-001	Добавление новой учетной записи	Пользователь авторизован как администратор	1.Нажать на кнопку "Добавить учетную запись". 2.Ввести данные пользователя (логин, пароль, роль). 3.Нажать на кнопку «Сохранить».	Учетная запись успешно добавлена. В списке пользователей отображается новая запись с корректными данными.

Таблица 9 – Тестирование редактирования учетной записи

Тест	Название тест-кейса	Предусловия	Шаги теста	Ожидаемый результат
ТС-002	Редактирование существующей учетной записи	Пользователь авторизован как администратор; учетная запись существует	1.Открыть список учетных записей. 2.Найти учетную запись для редактирования. 3.Нажать на «Редактировать». 4.Изменить данные учетной записи. 5.Нажать «Сохранить».	Учетная запись обновлена. В списке пользователей отображаются обновленные данные для этой записи.

Таблица 10 – Тестирование удаления учетной записи

Тест	Название тест-кейса	Предусловия	Шаги теста	Ожидаемый результат
ТС-003	Удаление учетной записи	Пользователь авторизован как администратор; учетная запись существует	1.Открыть список учетных записей. 2.Найти учетную запись для удаления. 3.Нажать на «Удалить». 4.Подтвердить удаление.	Учетная запись успешно удалена. Она больше не отображается в списке пользователей.

Таблица 11 – Тестирование работы с календарем

Тест	Название тест-кейса	Предусловия	Шаги теста	Ожидаемый результат
ТС-004	Добавление события в календарь	Пользователь авторизован как преподаватель	1.Открыть календарь. 2.Выбрать дату события. 3.Нажать «Добавить событие». 4.Ввести информацию о событии. 5.Нажать «Сохранить».	Событие добавлено в календарь. При выборе этой даты в календаре отображается созданное событие.
ТС-005	Редактирование события в календаре	Пользователь авторизован как преподаватель; событие существует	1.Открыть календарь. 2. Найти событие на нужной дате. 3.Нажать «Редактировать». 4. Изменить данные события. 5. Нажать «Сохранить».	Данные события обновлены. При выборе даты в календаре отображается обновленное событие.
ТС-006	Удаление события из календаря	Пользователь авторизован как преподаватель; событие существует	1.Открыть календарь. 2. Найти событие на нужной дате. 3.Нажать «Удалить». 4.Подтвердить удаление.	Событие удалено из календаря. При выборе даты событие больше не отображается.

В Xamarin возможность юнит-тестирования реализуется с использованием популярных библиотек и инструментов, которые позволяют тестировать как логику приложения, так и определенные функциональные аспекты. Тестирование организуется на уровне платформы .NET, что предоставляет доступ к стандартным библиотекам для тестирования, а также к специализированным инструментам, адаптированным для Xamarin-приложений.

Основой для юнит-тестирования в Xamarin является использование библиотек, таких как NUnit или xUnit, которые широко применяются в .NET-экосистеме. Эти библиотеки позволяют создавать и выполнять тесты для бизнес-логики, моделей и вспомогательных классов. Тесты пишутся на C#, и разработчики могут использовать уже знакомые конструкции и подходы. NUnit особенно популярен благодаря интеграции с инструментами разработчиков, такими как Visual Studio. В Xamarin поддерживается структура разделения приложения на слои, например, при использовании архитектуры MVVM. Это позволяет изолировать бизнес-логику и модели данных от пользовательского интерфейса, что делает их удобными для юнит-тестирования. Например, ViewModel может быть протестирован отдельно от View, что гарантирует корректное поведение логики, не зависимой от платформы.

Для выполнения тестов Xamarin предоставляет встроенную интеграцию с Visual Studio Test Explorer, который упрощает написание и выполнение тестов прямо в среде разработки. Это позволяет разработчикам проверять корректность своих решений и получать быстрые результаты тестирования. Тесты можно запускать локально или в облачных решениях, таких как Azure DevOps, что делает процесс масштабируемым и удобным. Если в приложении требуется тестирование специфичного для платформы кода, можно использовать платформо-зависимые тесты. Для этого разработчики используют интерфейсы, чтобы подменить реальные реализации тестовыми.

Например, для тестирования взаимодействия с SQLite можно использовать библиотеку Moq, которая позволяет имитировать поведение базы данных.

Юнит тестирование представлено на рисунке 12.

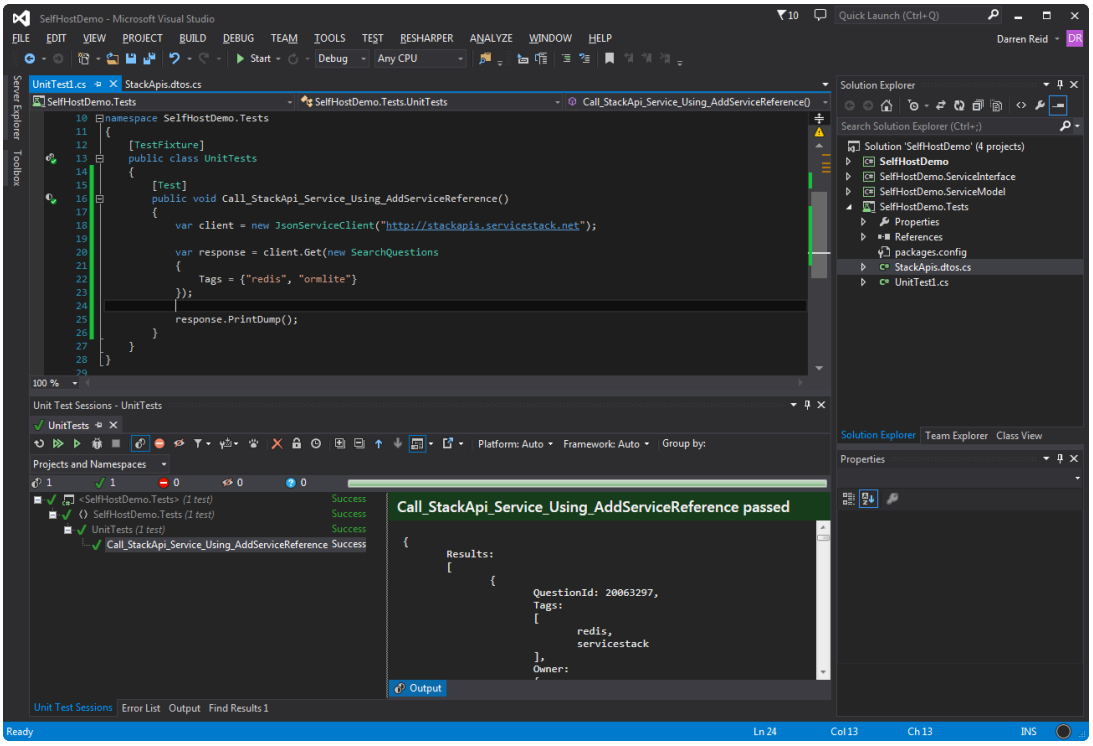

Рисунок 12 – Юнит тестирование приложение

Тестирование показало, что код программного продукта имеет недостаточную структурированность, что затрудняет его поддержку и дальнейшее развитие. Отсутствие четкого разделения задач на логически независимые компоненты приводит к высокой связности кода, где различные функциональные элементы переплетаются между собой, что снижает модульность. Это делает код более сложным для понимания и тестирования, так как разработчику приходится работать с большими фрагментами логики вместо отдельных, четко очерченных блоков. Кроме того, выявлены признаки избыточности в коде, что выражается в дублировании некоторых элементов и действий. Использование однотипных конструкций и повторяющихся проверок увеличивает объем кода без добавления новой функциональности,

что негативно сказывается на его эффективности. Это создает ненужную нагрузку при внесении изменений, так как приходится редактировать несколько участков кода одновременно, увеличивая вероятность появления ошибок. Также тестирование показало, что структура кода снижает его читаемость. Непрозрачная логика, недостаток комментариев и отсутствие четких границ между различными функциональными задачами создают сложности для быстрого восприятия и анализа кода. Это особенно критично при работе в команде, где другие разработчики должны оперативно ориентироваться в коде и поддерживать его без длительного погружения в контекст выполнения программы.

3.6 Доработка мобильного приложения

Для устранения недостатков в структурированности кода были приняты меры по разделению логики на отдельные, независимые блоки. Это включало выделение ключевых функций в отдельные методы, каждый из которых отвечает за конкретную задачу, например, за формирование сообщений для событий или за отображение уведомлений пользователю. Такой подход позволил уменьшить связность кода и повысить его модульность, что значительно упростило процесс тестирования и внесения изменений. Также были устранены проблемы с избыточностью кода. Все повторяющиеся элементы и дублирующиеся проверки были сведены к единому методу, который используется в разных частях программы. Это сократило объем кода, сделало его более лаконичным и улучшило его поддерживаемость. Внесение изменений теперь затрагивает только один участок кода, что уменьшает вероятность ошибок и ускоряет процесс модификаций. Для улучшения читаемости были добавлены комментарии к ключевым частям кода, объясняющие его функциональность. Это позволило сделать код более понятным для других разработчиков, снизить время на погружение в логику программы и повысить эффективность командной работы. Четкое именование

переменных и методов также помогло сделать код более понятным на первый взгляд, что особенно важно в долгосрочной поддержке программного продукта.

По итогам тестирования можно заключить, что система демонстрирует стабильную работу, но имела недостаточную структурированность, избыточность кода и сложности с читаемостью.

Вывод по главе 3

В главе 3 была подробно рассмотрена реализация и тестирование мобильного приложения. На начальном этапе был обоснован выбор средств разработки, что позволило определить инструменты для эффективного создания системы. Далее была разработана логическая структура базы данных, обеспечившая оптимальное хранение и обработку данных. Проектирование и разработка пользовательского интерфейса были направлены на создание интуитивно понятной и функциональной системы для конечных пользователей, а маршрутизация страниц приложения обеспечила удобную навигацию.

Заключение

Основной целью работы стало создание мобильного приложения, которое позволило бы сократить время, затрачиваемое на рутинные административные задачи, такие как планирование уроков, ведение журналов успеваемости и управление расписанием.

В ходе работы был проведен анализ актуальности автоматизации и цифровизации образовательного процесса, выбраны подходящие методологии проектирования и средства разработки, а также выполнена реализация и тестирование системы.

Особое внимание уделено логическому проектированию базы данных и пользовательскому интерфейсу, что обеспечило надежность и удобство использования системы.

Результаты работы подтвердили, что внедрение информационной системы способствует повышению эффективности работы педагогов, позволяет автоматизировать ключевые процессы управления учебным процессом и улучшить организацию образовательной деятельности.

Проведенные тестирования показали стабильную работу системы, а внесенные на основе анализа тестов улучшения повысили ее производительность и надежность.

Для устранения недостатков в структурированности кода были приняты меры по разделению логики на отдельные, независимые блоки. Это включало выделение ключевых функций в отдельные методы, каждый из которых отвечает за конкретную задачу, например, за формирование сообщений для событий или за отображение уведомлений пользователю.

Таким образом, разработанное программное обеспечение не только отвечает поставленным задачам, но и предоставляет педагогам мощный инструмент для управления образовательной деятельностью, что делает его актуальным и востребованным в условиях современных цифровых преобразований в образовательной сфере.

Список используемых источников

1. Архитектурные решения информационных систем: учебник для вузов / А. И. Водяхо, Л. С. Выговский, В. А. Дубенецкий, В. В. Цехановский. - 3-е изд., стер. – Санкт-Петербург: Лань, 2022. – 356 с. – ISBN 978-5-507-44710-7. – Текст: электронный // Лань: электронно-библиотечная система. – URL: https://e.lanbook.com/book/254624 (дата обращения: 20.04.2024). – Режим доступа: для авториз. пользователей.

2. Богомолов, А. В. Методические рекомендации по подготовке, выполнению и оформлению курсовых работ по дисциплине «Проектирование информационных систем» для бакалавров, обучающихся по направлению подготовки 09.03.03 «Прикладная информатика»: методические рекомендации / А. В. Богомолов, Э. А. Игнатьева, К. Н. Фадеева. - Чебоксары: ЧГПУ им. И. Я. Яковлева, 2022. – 48 с. – Текст: электронный // Лань: электронно-библиотечная система. – URL: https://e.lanbook.com/book/354065 (дата обращения: 13.04.2024). – Режим доступа: для авториз. пользователей.

3. Восемь лучших методологий разработки ПО в 2024 году. URL: https://www.purrweb.com/ru/blog/metodologii-dlya-razrabotki-po/ (дата обращения: 14.04.2024).

4. Зубкова, Т. М. Технология разработки программного обеспечения: учебное пособие / Т. М. Зубкова. – Оренбург: ОГУ, 2017. – ISBN 978-5-7410-1785-2. – Текст: электронный // Лань: электронно-библиотечная система. – URL: https://e.lanbook.com/book/110632 (дата обращения: 13.04.2024). – Режим доступа: для авториз. пользователей.

5. Игнатьев, А. В. Тестирование программного обеспечения / А. В. Игнатьев. - 3-е изд., стер. - Санкт-Петербург: Лань, 2023. – 56 с. – ISBN978-5-507-45425-9. – Текст: электронный // Лань: электронно-библиотечная система. – URL: https://e.lanbook.com/book/269873 (дата обращения: 27.04.2024). - Режим доступа: для авториз. пользователей.

6. Классификация требований к программным системам. URL: https://dit.isuct.ru/Publish_RUP/core.base_rup/guidances/concepts/requirements_62E28784.html (дата обращения 13.04.2024).

7. Машкин, А. В. Технология разработки программного обеспечения: учебное пособие / А. В. Машкин. - Вологда: ВоГУ, 2014. – 75 с. – ISBN 978-5-87851-526-9. – Текст: электронный // Лань: электронно-библиотечная система. – URL: https://e.lanbook.com/book/93087 (дата обращения: 13.04.2024). – Режим доступа: для авториз. пользователей.

8. Методологии управления проектами: 12 популярных подходов. URL: https://asana.com/ru/resources/project-management-methodologies (дата обращения: 14.04.2024).

9. Нафикова, А. Р. Объектно-ориентированный анализ и проектирование программного обеспечения на языке UML: учебное пособие / А. Р. Нафикова. – Уфа: БГПУ имени М. Акмуллы, 2022. - 118 с. – ISBN978-5-907475-48-9. – Текст: электронный // Лань: электронно-библиотечная система. – URL: https://e.lanbook.com/book/219221 (дата обращения: 14.04.2024). – Режим доступа: для авториз. пользователей.

10. Петрова, О. Б. Разработка и анализ требований проектирования программного обеспечения: практикум: учебное пособие / О. Б. Петрова. – Санкт-Петербург: СПбГУТ им. М.А. Бонч-Бруевича, 2022. - 37 с. – Текст: электронный // Лань: электронно-библиотечная система. – URL: https://e.lanbook.com/book/279218 (дата обращения: 14.04.2024). – Режим доступа: для авториз. пользователей.

11. Семенова, И. И. SQL стандарт в современных СУБД: манипулирование данными: учебное пособие / И. И. Семенова, Е. О. Шершнева. – 2-е изд., дериватив., испр. и доп. – Омск: СибАДИ, 2023. – 54 с. – ISBN 978-5-00113-242-4. – Текст: электронный // Лань: электроннобиблиотечная система. – URL: https://e.lanbook.com/book/407393 (дата обращения: 21.04.2024). – Режим доступа: для авториз. пользователей.

12. Синицын, И. В. Разработка мобильных приложений: учебное пособие / И. В. Синицын, Е. А. Чернов, Ю. А. Воронцов. – Москва: РТУ МИРЭА, 2023 - Часть 1 – 2023. – 162 с. – ISBN 978-5-7339-1799-3. – Текст: электронный // Лань: электронно-библиотечная система. – URL: https://e.lanbook.com/book/368735 (дата обращения: 20.04.2024). – Режим доступа: для авториз. пользователей.

13. Соснин, П. И. Архитектурное моделирование автоматизированных систем: учебник для вузов / П. И. Соснин. – 2-е изд., стер. - Санкт-Петербург: Лань, 2024. - 180 с. – ISBN 978-5-507-49488-0. – Текст: электронный // Лань: электронно-библиотечная система. – URL: https://e.lanbook.com/book/393065 (дата обращения: 20.04.2024). – Режим доступа: для авториз. пользователей.

14. Токмаков, Г. П. Базы данных: Модели и структуры данных, язык SQL, программирование баз данных: учебное пособие / Г. П. Токмаков. – Ульяновск: УлГТУ, 2021. – 362 с. – ISBN 978-5-9795-2184-8. – Текст: электронный // Лань: электронно-библиотечная система. – URL:

https://e.lanbook.com/book/259706 (дата обращения: 21.04.2024). – Режим доступа: для авториз. пользователей.

15. Туманова, М. Б. Проектирование программных систем: учебное пособие / М. Б. Туманова, Е. К. Михайлова, Е. А. Муравьева. – Москва: РТУ МИРЭА, 2023. – 138 с. – ISBN 978-5-7339-2050-4. – Текст: электронный // Лань: электронно-библиотечная система. – URL: https://e.lanbook.com/book/398273 (дата обращения: 20.04.2024). – Режим доступа: для авториз. пользователей.

16. Figma: The Collaborative Interface Design Tool. URL: https://www.figma.com/ (дата обращения 21.04.2024).

17. Microsoft Learn: Xamarin Essentials [Электронный ресурс]. – URL: https://learn.microsoft.com/en-us/xamarin/ (дата обращения: 22.06.2024).

18. SQLite Documentation [Электронный ресурс]. – URL: https://www.sqlite.org/docs.html (дата обращения: 12.07.2024).

19. MVVM Design Pattern for Xamarin.Forms [Электронный ресурс]. – URL: https://www.syncfusion.com/blogs/post/mvvm-design-pattern-for-xamarin-forms.aspx (дата обращения: 01.07.2024).

20. Visual Studio IDE Documentation [Электронный ресурс]. – URL: https://learn.microsoft.com/en-us/visualstudio/ (дата обращения: 21.07.2024).

21. Xamarin.Forms Performance Optimization [Электронный ресурс]. – URL: https://montemagno.com/xamarin-forms-performance-tips/ (дата обращения: 03.08.2024).

Приложение А

Основные формы интерфейса программного обеспечения

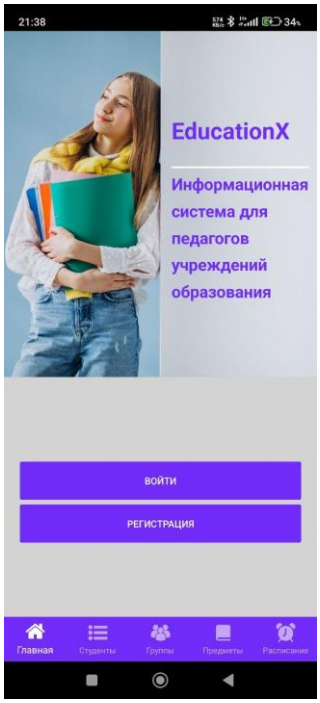

Рисунок А.1 – Стартовая страница мобильного приложения

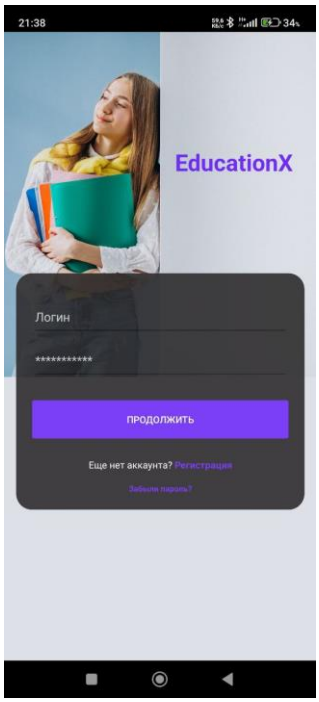

Рисунок А.2 – Форма авторизации

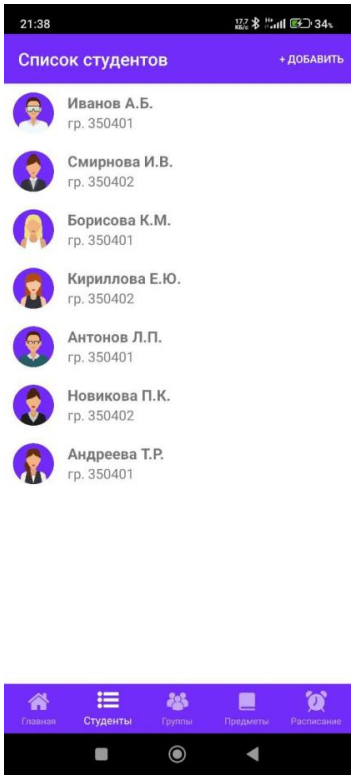

Рисунок А.3 – Список студентов

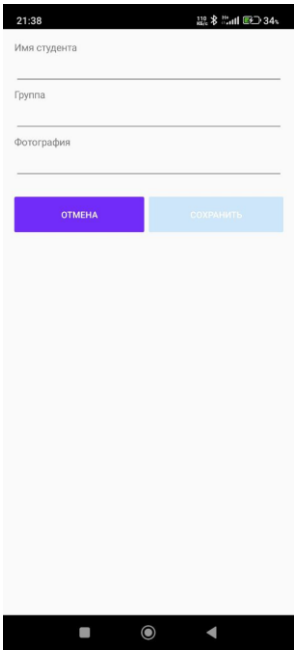

Рисунок А.4 – Форма добавления учащегося

Рисунок А.5 – Список учебных групп

Рисунок А.6 – Список предметов

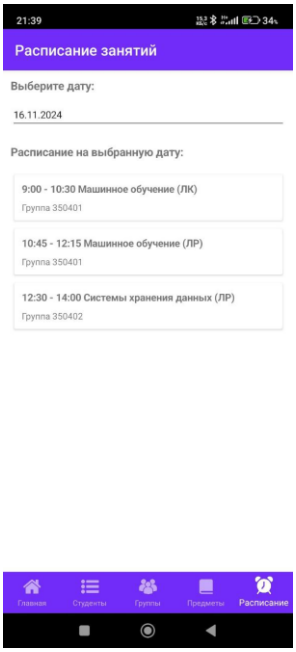

Рисунок А.7 – Расписание занятий

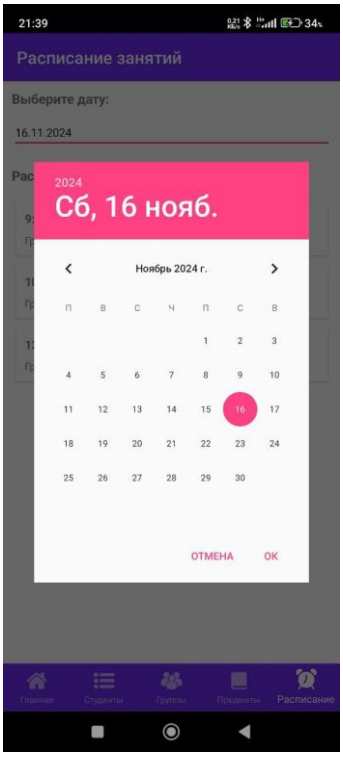

Рисунок А.8 – Работа с календарем