

Титульный лист

Баранов Д.А. ИВТ 2.1. Санкт-Петербург. 2026

Содержание

Титульный лист.....	1
Содержание.....	2
Введение.....	3
Глава 1. Теоретические основы разработки мобильного приложения для гибкого планирования и приоритизации задач с возможностью визуализации прогресса.....	9
1.1. Особенности планирования задач в современной цифровой среде.....	9
1.2. Теоретические подходы к планированию и приоритизации задач.....	12
1.3. Анализ существующих мобильных приложений для управления задачами...	16
1.4. Обоснование направления разработки мобильного приложения.....	20
Глава 2. Разработка мобильного приложения для гибкого планирования и приоритизации задач.....	25
2.1 Обоснование выбора средств разработки.....	25
2.2 Проектирование архитектуры приложения.....	29
2.3 Реализация структуры базы данных.....	34
2.4 Проектирование и реализация пользовательского интерфейса (UI/UX).....	38
2.5 Реализация функциональных возможностей.....	42
2.6 Реализация совместной работы.....	46
2.7 Тестирование разработанного приложения.....	50
2.8 Анализ результатов.....	58
Заключение.....	64
Список использованных источников.....	67

Введение

Информатизация современного общества детерминирует переход к мобильным технологиям как основному инструменту организации профессиональной и личной деятельности. Смартфон интегрирует функции коммуникации, непрерывного обучения и оперативного управления ресурсами. Ускорение ритма жизни и экспоненциальный рост объемов обрабатываемой информации приводят к необходимости внедрения строгих систем тайм-менеджмента [1, с. 3]. Практика доказывает, что системное управление временем минимизирует когнитивную перегрузку и предотвращает масштабное снижение результативности при многозадачности [2, с. 24]. Программные решения в данной профильной сфере автоматизируют рутинные процессы составления расписаний, высвобождая интеллектуальные ресурсы для аналитической работы [3, с. 109].

Ежедневное функционирование современного пользователя сопряжено с обработкой гетерогенных потоков задач: академических заданий, рабочих поручений, встреч и долгосрочных личных проектов. Попытки удержания неструктурированных списков дел исключительно в рабочей памяти неизбежно повышают уровень психологического напряжения и перманентно снижают способность к глубокой концентрации [4, с. 55]. Отсутствие профильного цифрового инструмента нарушает логическую последовательность выполняемых операций, прямо провоцируя пропуск критических дедлайнов. Научные исследования подтверждают, что оптимизация распределения временных ресурсов напрямую зависит от применения специализированного программного обеспечения, способного автоматически структурировать входящую информацию [5, с. 48]. Автоматизация тайм-менеджмента выступает не просто элементом удобства, но критическим фактором поддержания профессиональной и академической успешности [3, с. 112].

Рынок программного обеспечения предлагает обширный спектр систем управления проектами и персональных планировщиков. Системный анализ существующих платформ выявляет их полярную сегментацию. Корпоративные системы обладают избыточным функционалом, ориентированным на командную разработку и интеграцию сложных IT-процессов, что усложняет их использование в индивидуальном повседневном формате [6, с. 312]. Высокий порог входа и визуально перегруженный интерфейс отталкивают рядовых пользователей. Базовые приложения-списки, напротив, ограничиваются примитивной логикой добавления и удаления пунктов. В них полностью отсутствуют механизмы

гибкого ранжирования, декомпозиции комплексных целей на подзадачи и встроенного аналитического контроля [7, с. 299]. Выявлен явный дисбаланс: доступные инструменты либо требуют неоправданных затрат времени на поддержку собственной иерархии, либо не покрывают базовых потребностей в приоритизации [7, с. 301].

Критическим недостатком большинства индивидуальных планировщиков признается отсутствие развитой системы визуализации прогресса. Представление результатов через графики, календарные тепловые карты и диаграммы Ганта позволяет объективировать субъективное восприятие личной продуктивности [8, с. 735; 9, с. 101]. Визуальная аналитика трансформирует абстрактный перечень выполненных действий в измеримый цифровой показатель. Наглядное отображение динамики выполнения задач активно стимулирует мотивацию, формирует устойчивую дисциплину и существенно упрощает выявление периодов пиковой активности [10, с. 124]. Интеграция графических модулей переводит процесс управления временем из пассивной фиксации в активный анализ [11, с. 580].

Современная архитектура мобильного приложения обязана обеспечивать высокую гибкость настроек рабочей среды. Инструментарий должен включать поддержку матрицы Эйзенхауэра для разделения задач по критериям срочности и важности, а также методы ABC-анализа для строгого ранжирования приоритетов [12, с. 74; 13, с. 56]. Эффективность планированиякратно повышается при внедрении циклических таймеров, чек-листов и повторяющихся напоминаний [14, с. 254]. Фоновые механизмы push-уведомлений гарантируют своевременное оповещение о дедлайнах, а облачная синхронизация обеспечивает целостность данных при попеременном использовании нескольких устройств [15, с. 1084]. Формируется объективная потребность в разработке легковесного, но алгоритмически мощного кроссплатформенного приложения.

Реализация подобного программного комплекса требует аргументированного выбора оптимального технологического стека. Использование фреймворка Flutter и языка программирования Dart позволяет создавать высокопроизводительные кроссплатформенные приложения с единой кодовой базой для операционных систем Android и iOS [16, с. 23]. Данная технология обеспечивает нативную производительность отрисовки интерфейсов за счет встроенного графического движка рендеринга [16, с. 25]. Применение Flutter сокращает общие временные затраты на разработку и тестирование пользовательского интерфейса на 25–30%, гарантируя абсолютную визуальную идентичность приложения на различных мобильных платформах [17, с. 138].

Актуальность темы выпускной квалификационной работы «Разработка мобильного приложения для гибкого планирования и приоритизации задач с возможностью визуализации прогресса» обусловлена ростом потребности в удобных и функциональных мобильных инструментах управления задачами, способных повысить личную и коллективную продуктивность пользователей. Значимость исследования определяется также тем, что современные программные решения нередко либо слишком упрощены, либо, напротив, чрезмерно сложны, что создает предпосылки для разработки более сбалансированного приложения, сочетающего необходимые функции планирования, контроля и визуального анализа результатов.

Объектом исследования является процесс планирования, организации и контроля выполнения задач в мобильной цифровой среде.

Предметом исследования выступают методы, модели и программные средства реализации мобильного приложения, обеспечивающего гибкое планирование задач, их приоритизацию и визуализацию прогресса выполнения.

Целью выпускной квалификационной работы является разработка мобильного приложения для гибкого планирования и приоритизации задач с возможностью визуализации прогресса выполнения.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Изучить предметную область и проанализировать существующие мобильные приложения, предназначенные для управления задачами и планирования деятельности.
2. Выявить основные достоинства и недостатки имеющихся решений, а также определить требования, предъявляемые пользователями к подобным системам.
3. Сформулировать функциональные и нефункциональные требования к разрабатываемому мобильному приложению.

4. Разработать структуру и архитектуру программного решения, определить состав его основных модулей и принципы их взаимодействия.
5. Спроектировать пользовательский интерфейс приложения с учетом требований удобства, наглядности и доступности.
6. Реализовать основные функциональные возможности приложения, включая создание и редактирование задач, чек-листы, приоритизацию, календарное представление и визуализацию прогресса.
7. Реализовать механизмы уведомлений, синхронизации данных и, при необходимости, средства совместной работы пользователей.
8. Провести тестирование разработанного программного продукта и оценить корректность его функционирования.
9. Проанализировать результаты разработки и определить возможные направления дальнейшего совершенствования приложения.

Гипотеза исследования заключается в предположении о том, что использование мобильного приложения, объединяющего средства гибкого планирования, механизмы приоритизации задач, систему напоминаний и наглядную визуализацию прогресса, позволит повысить эффективность управления задачами и улучшить организацию деятельности пользователя по сравнению с применением традиционных способов планирования или простых приложений-списков задач.

Для решения поставленных задач в работе применяются следующие методы исследования: анализ научной и учебной литературы по теме исследования; сравнительный анализ существующих программных решений; системный анализ предметной области; моделирование структуры и логики программного продукта;

проектирование программного обеспечения и пользовательского интерфейса; тестирование разработанного приложения; аналитическая обработка полученных результатов.

Информационную основу исследования составляют научные публикации и учебные издания, посвященные разработке мобильных приложений, организации пользовательского интерфейса и вопросам повышения продуктивности; официальная документация используемых технологий и платформ; материалы, связанные с проектированием цифровых сервисов; аналитические обзоры существующих приложений для управления задачами; а также иные специализированные источники, необходимые для обоснования выбранных решений и методов разработки.

Практическая значимость выпускной квалификационной работы заключается в создании мобильного приложения, которое может быть использовано для организации личной и совместной деятельности, планирования задач, контроля сроков и наглядного отслеживания результатов. Разрабатываемое решение может служить основой для последующего расширения функциональности, внедрения в реальную практику и дальнейшего развития как самостоятельного программного продукта.

В рамках работы предполагается реализация таких функциональных возможностей, как создание и редактирование задач, установка разовых и повторяющихся напоминаний, формирование чек-листов и подзадач, сортировка задач по степени важности, срочности и категориям, визуализация прогресса с использованием графиков, диаграмм и статистики, отображение задач в календаре, поддержка push-уведомлений, облачная синхронизация, элементы совместной работы пользователей и персонализация интерфейса. Перечень указанных функций может быть уточнен и расширен в процессе выполнения работы.

Результатом выполнения выпускной квалификационной работы должна стать разработка мобильного приложения, обеспечивающего эффективное планирование и управление задачами, а также описание этапов его проектирования, реализации и тестирования. Полученные результаты могут представлять интерес как с практической точки зрения, так и в контексте дальнейшего изучения подходов к созданию мобильных приложений для повышения продуктивности пользователя.

Глава 1. Теоретические основы разработки мобильного приложения для гибкого планирования и приоритизации задач с возможностью визуализации прогресса

1.1. Особенности планирования задач в современной цифровой среде

В условиях тотальной цифровизации и высокодинамичной информационной среды управление временем претерпевает фундаментальные трансформации. Размытие пространственно-временных границ между профессиональной деятельностью, непрерывным обучением и частной жизнью обуславливает экспоненциальный рост числа параллельно выполняемых задач [1, с. 418]. Анализ эмпирических данных доказывает, что неспособность субъектов адаптироваться к сверхвысоким скоростям информационного обмена выступает ключевым фактором снижения производительности труда и развития синдрома профессионального выгорания [2, с. 164]. Традиционные методы фиксации поручений на бумажных носителях утрачивают релевантность, поскольку исключают возможность оперативной актуализации данных, алгоритмизации действий и гибкой реорганизации приоритетов. Современное планирование трансформируется в комплексную систему управления жизненным циклом задачи, требующую обязательной экстериоризации процессов структурирования деятельности во внешнюю цифровую среду [3, с. 397].

Проблема адаптации к высокоинтенсивному цифровому взаимодействию неразрывно связана с физиологическими ограничениями мозга, концептуализированными в теории когнитивной нагрузки. Установлено, что емкость рабочей памяти строго лимитирована, позволяя одновременно удерживать в активном состоянии от трех до пяти единиц информации [4, с. 115; 5, с. 84]. Непрерывное воздействие разнородных потоков данных формирует избыточную внешнюю когнитивную нагрузку, значительно превышающую пропускную способность перцептивного аппарата [5, с. 85]. Хроническое пребывание в состоянии информационной перегрузки провоцирует снижение точности

принимаемых решений, эмоциональное истощение и паралич выбора [6, с. 1082]. Следовательно, цифровая система планирования призвана выполнять функцию когнитивного фильтра, минимизирующего излишнее напряжение рабочей памяти посредством дискретизации проектов.

Специфика познавательной активности в современной цифровой среде характеризуется формированием феномена непрерывного частичного внимания и иллюзией многозадачности. Попытки параллельной обработки множественных потоков фактически представляют собой высокоскоростное переключение фокуса, неизбежно сопровождающееся когнитивной задержкой и накоплением нейрофизиологического переутомления [3, с. 398; 7, с. 8]. Архитектура существующих платформ алгоритмически поощряет поведение поиска новизны, что приводит к развитию клипового мышления, препятствующего глубокому аналитическому погружению [3, с. 400]. Повсеместная доступность сетевых ресурсов индуцирует эффект цифровой амнезии, при котором хранение фактических данных делегируется внешним носителям [3, с. 401]. Описанные деструктивные тенденции диктуют необходимость применения в системах тайм-менеджмента механизмов жесткой приоритизации и искусственного ограничения отвлекающих факторов.

Теоретические модели управления задачами базируются на интеграции процессов целеполагания, декомпозиции, организации выполнения и рефлексии [8, с. 11]. Главным условием эффективности признается полная выгрузка намерений в цифровое хранилище, что предотвращает истощение ментального резерва, вызванное необходимостью постоянного внутреннего мониторинга статуса незавершенных дел [2, с. 166]. Сложная единица деятельности в программной среде структурируется набором обязательных атрибутов: наименованием, сроком, статусом, приоритетом и подзадачами. Метод декомпозиции снижает уровень неопределенности, а реалистичное нормирование времени минимизирует риски каскадного срыва графиков [2, с. 167; 8, с. 12].

Избыточное усложнение структурной модели задачи отрицательно сказывается на эргономике продукта, требуя поддержания баланса между функциональной полнотой и скоростью взаимодействия. Сводный анализ влияния технологических компонентов на когнитивную нагрузку представлен в таблице 1.1.

Таблица 1 — Компоненты системы планирования и их влияние на когнитивную

Этап управления задач	Возникающая когнитивная проблема	Технологическое решение в мобильной среде
Фиксация намерения	Эффект Зейгарник, перегрузка кратковременной памяти	Экстериоризация данных, быстрый ввод интерфейса
Целеполагание	«Паралич выбора», высокая степень неопределенности	Декомпозиция задач, иерархическая структура
Выполнение работы	Фрагментация внимания, иллюзия многозадачности	Ограничение отвлекающих факторов, push-уведомления
Анализ результатов	Искажение субъективного восприятия затраченного времени	Визуализация прогресса (графики, цветовое кодирование)

Доминирующим инструментом реализации современных систем планирования выступают портативные вычислительные устройства. Сценарии взаимодействия с мобильными планировщиками отличаются предельной степенью фрагментарности [9, с. 72]. Длительность рабочих сессий зачастую исчисляется секундами, что предполагает ввод данных «на ходу». Ключевыми требованиями к мобильным интерфейсам признаются минимизация количества сенсорных транзакций для фиксации задачи, высокая скорость отклика и интуитивная понятность [9, с. 74]. Мощнейшим механизмом управления вниманием служат push-уведомления. Выступая в качестве внешнего поведенческого триггера, выверенное оповещение принудительно возвращает фокус пользователя к приоритетной задаче, однако отсутствие

персонализированных настроек и избыточная частота уведомлений мгновенно генерируют деструктивный информационный шум [9, с. 76].

Неотъемлемым компонентом архитектуры мобильного приложения для планирования выступает визуализация прогресса. Перевод текстовых массивов в графические форматы (календарные сетки, шкалы прогресса) существенно снижает когнитивную нагрузку при перцептивной обработке больших объемов данных [5, с. 87; 9, с. 78]. Визуальное отображение текущего состояния дел ускоряет когнитивный анализ загруженности, способствует своевременному выявлению периодов переработки и формирует положительную обратную связь. Наглядная фиксация достигнутых результатов обеспечивает подкрепление, стимулирует мотивацию и поддерживает ритмичность выполнения долгосрочных проектов [9, с. 79]. Разработка программного обеспечения в рассматриваемой предметной области требует синтеза инженерно-технических решений с актуальными принципами нейроэргономики и доказанными закономерностями функционирования когнитивного аппарата.

1.2. Теоретические подходы к планированию и приоритизации задач

Проектирование систем управления задачами опирается на теоретическое обоснование алгоритмов распределения ресурсов и когнитивной нагрузки. Анализ моделей управления временем выявляет необходимость применения системного подхода, способного компенсировать ограничения памяти в условиях высокой информационной плотности [11, с. 56]. Разработка методологического базиса для мобильных систем планирования требует детального рассмотрения концепций: Getting Things Done (GTD), матрицы Эйзенхауэра, метода MoSCoW, канбан-модели, принципа Парето и фреймворка Scrum.

Методология Getting Things Done (GTD) представляет собой систему процессуального управления, ориентированную на снижение уровня стресса за счет внешнего хранения информации [13, с. 354]. Теоретический постулат

основывается на преодолении эффекта Зейгарник: перенос незавершенных действий на цифровые носители высвобождает интеллектуальный ресурс исключительно для аналитической работы. Алгоритм формализуется через стадии сбора, обработки, организации, обзора и выполнения. Особое значение имеет эвристический механизм «двух минут», который предотвращает засорение базы микрозадачами [2, с. 20]. Интеграция GTD в архитектуру цифрового планировщика диктует необходимость создания минималистичных интерфейсов быстрого ввода и реляционной структуры тегов.

Матрица Эйзенхауэра классифицирует рабочие элементы в двумерной системе координат на основе векторов «важность» (влияние на стратегические цели) и «срочность» (близость дедлайна) [8, с. 107]. Пересечение векторов формирует четыре квадранта, определяющих поведенческие стратегии: немедленное выполнение, проактивное планирование, делегирование и исключение. Теоретическое обоснование подхода направлено на нейтрализацию когнитивного искажения, инстинктивно смещающего фокус на срочные задачи в ущерб важным. Оптимизация планирования достигается путем превентивного перераспределения ресурсов во второй квадрант [18, с. 52]. Программная реализация требует параметрической архитектуры баз данных для автоматического вычисления индекса приоритетности.

Закон Парето (правило 80/20) описывается степенным законом распределения, постулируя, что 20% критически важных действий генерируют 80% итогового успеха [9, с. 327]. В системах гибкого управления данный принцип дополняется методом семантической категоризации MoSCoW, ориентированным на принятие решений в условиях жестких ресурсных ограничений [4, с. 85]. Задачи разделяются по степени критичности: Must have (обязательные), Should have (важные), Could have (желательные) и Won't have (исключенные из текущего цикла). В контексте мобильного приложения данный алгоритм выступает

инструментом таймбоксинга, программно ограничивая количество обязательных элементов в рамках одного дня.

Теория Канбан представляет собой методологию управления процессами, основанную на визуализации потока создания ценности и жестком лимитировании нагрузки. Принципиальным отличием выступает переход к системе «вытягивания» (pull system), при которой задачи переносятся на следующий этап исключительно при освобождении ресурса. Ключевым аналитическим регулятором выступает ограничение незавершенной работы (WIP). Данное ограничение математически обосновано законом Литтла: сокращение количества параллельных задач пропорционально уменьшает среднее время производственного цикла [12, с. 108]. Пространственное перемещение карточек по виртуальной доске снижает барьер восприятия, а программные WIP-лимиты защищают пользователя от когнитивного истощения [20, с. 37].

Методология Scrum представляет собой эмпирический фреймворк для структурирования работы в условиях неопределенности на основе итеративно-инкрементального подхода [6, с. 18]. Организационная структура регламентируется строгими артефактами: бэклогом продукта (глобальный приоритизированный реестр) и спринтом (изолированный временной интервал). Фундаментальным правилом выступает неприкосновенность бэклога спринта, что предотвращает размытие фокуса. В мобильных планировщиках это трансформируется в концепцию «персональных спринтов» с ограниченным пулом задач и предоставлением аналитики продуктивности для коррекции планов на следующий цикл [5, с. 103]. Изолированное применение рассмотренных концепций сопровождается специфическими ограничениями (таблица 1).

Таблица 2 — Сравнительная характеристика теоретических моделей управления задачами [14, с. 478]

Методология	Ключевой	Главное	Системное	Область
-------------	----------	---------	-----------	---------

	механизм контроля	преимущество	ограничение	применения в приложении
GTD	Внешнее хранение	Разгрузка оперативной памяти	Отсутствие строгих правил выбора	Архитектура Inbox, теги
Эйзенхауэр	Двумерная оценка	Фокус на стратегии	Падение наглядности при росте баз	Модуль автоматической сортировки
MoSCoW	Семантическая категоризация	Исключение не критичных элементов	Субъективное завышение статуса	Фильтрация задач при планировании
Канбан	Визуализация, WIP-лимиты	Снижение издержек на переключение	Требует отфильтрованного потока	Основной интерфейс (виртуальная доска)
Scrum	Жесткие итерации	Защита фокуса от прерываний	Высокие издержки на администрирование	Временные рамки, аналитика периодов

Исследование моделей приоритизации подтверждает, что максимальная эффективность достигается при использовании гибридных адаптивно-цифровых систем [10, с. 321]. Архитектура современного планировщика базируется на многоуровневом синтезе: сбор данных осуществляется по GTD, фильтрация — на основе матрицы Эйзенхауэра и метода MoSCoW (с учетом принципа Парето), а выполнение — в формате Канбан-доски с рамками персональных спринтов Scrum. Интеграция комплекса моделей формирует надежный теоретический базис для разработки программного продукта, выступающего в роли активной системы поддержки принятия решений [19, с. 25].

1.3. Анализ существующих мобильных приложений для управления задачами

Анализ существующих программных решений необходим для определения актуального состояния предметной области и выявления функциональных особенностей, востребованных пользователями. В условиях цифровизации и перехода к гибридным форматам занятости управление процессами трансформировалось из применения простых списков в использование кроссплатформенных систем. Современные решения обеспечивают автоматизацию рутинных процедур, распределение ресурсов, мониторинг прогресса и интеграцию с внешними сервисами [1, с. 41]. Приложения данного класса предназначены для структурирования рабочих процессов, снижения когнитивной нагрузки и повышения индивидуальной продуктивности за счет алгоритмизации данных [2, с. 35].

Оценка функциональности современных мобильных таск-трекеров требует применения многофакторного подхода. В качестве ключевых критериев для проведения сравнительного анализа выделяются: функциональные возможности (создание иерархии задач, настройка дедлайнов и напоминаний), эргономика интерфейса, встроенные методы приоритизации, поддержка механизмов совместной работы, уровень кроссплатформенности и надежность синхронизации. Дополнительным фактором выступает способность системы работать автономно с последующим автоматическим разрешением конфликтов при синхронизации данных [1, с. 42]. На основе данных параметров проводится исследование четырех распространенных на рынке решений: Todoist, Microsoft To Do, Trello и TickTick.

Программный продукт Todoist классифицируется как инструмент планирования и самоконтроля. Архитектура выстроена вокруг списков с поддержкой вложенных подзадач. Отличительной чертой системы выступает

автоматическое распознавание дат и приоритетов из текстового ввода. Поддерживаются четыре уровня приоритизации, кодируемые цветами, что упрощает фильтрацию задач. Совместная работа реализована через общие проекты, делегирование и обмен комментариями [4, с. 348]. Синхронизация данных осуществляется в реальном времени. К преимуществам относятся высокая скорость ввода и интуитивный интерфейс. Недостатки проявляются в ограниченности инструментов глубокой визуальной аналитики прогресса и жестком фокусе на списочном представлении, снижающем эффективность контроля длительных проектов.

Приложение Microsoft To Do глубоко интегрировано в глобальную корпоративную экосистему. Ключевой парадигмой выступает список «Мой день», ежедневно обнуляющий фокус пользователя и предлагающий перечень задач на текущие сутки на основе анализа просроченных дедлайнов [1, с. 43]. Система поддерживает создание списков, добавление этапов, файлов и заметок. Приоритизация ограничена отметкой задачи как важной для ее переноса в смарт-список. Совместная работа включает базовый обмен доступом к спискам. Сильными сторонами признаются бесшовная интеграция с корпоративными сервисами Microsoft и высокий уровень безопасности данных [5, с. 122]. Недостатки кроются в ограниченности визуализации прогресса, отсутствии пользовательских фильтров на основе тегов и сложных алгоритмов сортировки многоуровневых задач.

Программный продукт Trello основан на методологии канбан и представляет собой систему визуального управления процессами. Интерфейс организован в виде досок, колонок и карточек, пространственно отображающих жизненный цикл задачи. Это обеспечивает высокую наглядность, снижает нагрузку и способствует быстрому считыванию статуса проекта [4, с. 349]. Функционал включает настройку карточек посредством цветных меток, чек-листов и дедлайнов. Поддержка совместной работы выступает центральным элементом: система

позволяет распределять роли, назначать исполнителей и вести обсуждения [2, с. 35]. Преимуществом является гибкость командного взаимодействия. К недостаткам в мобильном формате относится сложность навигации по масштабным доскам на экранах смартфонов. Логика системы ориентирована на проекты, что делает ее избыточной для быстрого личного планирования.

Приложение TickTick позиционируется как гибридный планировщик, объединяющий управление списками задач, встроенный календарь и инструменты тайм-менеджмента. Отличительной особенностью является интеграция техник контроля времени прямо в приложение, что позволяет фиксировать длительность работы. Функционал включает списки, подзадачи и напоминания с привязкой к геолокации. Мобильная версия поддерживает распределение задач в сетке календаря. Совместная работа позволяет делиться списками и назначать исполнителей. Сильные стороны TickTick заключаются в интеграции разнообразных инструментов, включая трекер привычек, в едином интерфейсе [6, с. 307]. Недостатком выступает информационная перегруженность интерфейса дополнительными функциями, повышающая порог входа для пользователей.

Для систематизации результатов исследования сформирована сравнительная таблица на основе ключевых характеристик рассматриваемых мобильных приложений.

Таблица 3 — Сравнительный анализ мобильных приложений для управления задачами

Критерий	Todoist	Microsoft To Do	Trello	TickTick
Парадигма	Иерархические списки	Плоские списки, смарт-папки	Канбан-доски, карточки	Списки, календарная сетка
Приоритизация	4 уровня,	Отметка	Цветовые метки	4 уровня,

	фильтры	«Важное»	(ярлыки)	матрица
Визуализация	Индикаторы выполнения	Вычеркивание, списки	Перемещение карточек	Статистика, трекер
Совместная работа	Делегирование, проекты	Совместный доступ	Распределение ролей	Совместные списки
Мобильный ввод	Естественный язык	Стандартный текст	Карточки в колонках	Голосовой ввод, драг-энд-дроп
Тайм-менеджмент	Отсутствует	Отсутствует	Доступно через плагины	Встроенный таймер
Оффлайн-режим	Полная поддержка	Базовое кэширование	Локальное кэширование	Полная поддержка
Сегмент	Личная продуктивность	Индивидуальный	Командные проекты	Личное планирование

Сравнительный анализ подтверждает строгую сегментацию рынка в зависимости от выбранной разработчиками архитектурной парадигмы. Todoist и TickTick демонстрируют наибольшую эффективность в сценариях личного планирования за счет высокой скорости ввода данных и встроенных механизмов приоритизации. Trello обладает высокой эффективностью в области визуализации потоков задач, однако его применение в мобильной среде ограничивается рамками командных проектов. Microsoft To Do выступает нишевым решением для пользователей, нуждающихся в строгой интеграции с корпоративной программной экосистемой.

Анализ тенденций развития рынка свидетельствует о растущей потребности в программном обеспечении, адаптирующемся под индивидуальные паттерны планирования при строгой минимизации когнитивной нагрузки [6, с. 308]. Современные требования пользователей включают наличие кроссплатформенной архитектуры с поддержкой оффлайн-режима и фоновой синхронизации. Ни одно

из рассмотренных приложений не предлагает сбалансированного сочетания алгоритмов гибкой приоритизации и глубокого визуального контроля прогресса на уровне отдельных задач. Ограничение визуализации простой статистикой либо жесткой привязкой к колонкам формирует технологическую нишу. Обосновывается целесообразность разработки мобильного приложения, объединяющего алгоритмы гибкой сортировки и многоуровневой приоритетности с инновационными форматами визуализации прогресса в рамках минималистичного интерфейса.

1.4. Обоснование направления разработки мобильного приложения

Проведенный теоретический анализ предметной области доказывает, что архитектура эффективной мобильной системы управления задачами базируется на интеграции комплекса методологических принципов. Специфика мобильного взаимодействия характеризуется высокой степенью фрагментарности, что диктует необходимость обеспечения минимальной задержки при создании элементов планирования. Процесс сбора и структурирования информации опирается на алгоритмическую базу методологии Getting Things Done (GTD). Для реализации гибкой приоритизации разрабатываемая система применяет синтез матрицы Эйзенхауэра, распределяющей задачи по векторам важности и срочности, а также метода MoSCoW для определения категориальной критичности. Представление агрегированных данных предполагает внедрение вариативных интерфейсных паттернов: линейных списков, календарных сеток и элементов карточного представления. Использование канбан-подхода обеспечивает интуитивно понятную пространственную организацию статусов выполнения с применением механики перетаскивания (drag-and-drop).

Фундаментальным функциональным модулем проектируемой системы выступает подсистема визуализации прогресса. Внедрение глубокой графической аналитики трансформирует мобильное приложение в инструмент оценки динамики продуктивности, позволяющий прогнозировать будущую нагрузку на основе исторических данных. Функционирование платформы неразрывно связано

с использованием фоновых процессов. В условиях прерывистого взаимодействия с устройством push-уведомления применяются для контроля критических сроков. Архитектура приложения гарантирует своевременность оповещений, опираясь на системные алгоритмы энергосбережения. Техническая реализация заявленного функционала требует строгого обоснования выбора технологического стека.

Определение архитектурного направления разработки базируется на сравнении нативной и кроссплатформенной парадигм. Нативный подход предполагает использование двух независимых кодовых баз под операционные системы Android и iOS. Практические тестирования подтверждают, что нативные приложения характеризуются высокой скоростью выполнения вычислительных операций, а потребление оперативной памяти (ОЗУ) в нативных сборках в среднем в 2,5 раза ниже по сравнению с кроссплатформенными аналогами [1, с. 687]. Однако поддержка параллельной разработки увеличивает суммарное время создания приложения в среднем до 2400 часов для охвата доминирующих платформ [2, с. 1301]. Для проектов категории систем управления задачами подобные экономические издержки признаются нецелесообразными. Внедрение кроссплатформенных технологий формирует единую кодовую базу, обеспечивая до 70–80 % переиспользования программной логики [2, с. 1302]. Для объективного обоснования выбора проведен структурный анализ ключевых параметров актуальных кроссплатформенных технологий (Таблица 1).

Таблица 4 — Сравнительная архитектурная характеристика кроссплатформенных технологий

Параметр оценки	React Native	Kotlin Multiplatform	Flutter
Основной язык	JavaScript / TypeScript	Kotlin	Dart
Компиляция в релизе	Интерпретация + JIT	AOT	AOT
Рендеринг интерфейса	Нативные виджеты (Bridge)	Нативный UI / Compose	Внутренний движок

Доступ к системным API	Ограниченный	Прямой (байт-код)	Каналы платформы
Консистентность UI	Ограничена системной оболочкой	Высокая	Пиксельная точность на всех ОС

Архитектурный анализ фреймворка React Native выявляет зависимость от среды выполнения JavaScript и асинхронного моста (Bridge), транслирующего виртуальные компоненты в нативные элементы интерфейса. Пропускная способность данного моста выступает серьезным техническим ограничением. Операции сериализации данных в процессе отрисовки сложных анимационных последовательностей или обработки жестов вызывают задержки, приводящие к пропуску кадров [3, с. 560]. Технология Kotlin Multiplatform предлагает трансляцию бизнес-логики непосредственно в нативный байт-код для виртуальной машины Java и в машинный код через компилятор LLVM, исключая промежуточные мосты. В рамках фреймворка реализуется концепция выделения ожидаемых интерфейсов в общем модуле с их последующей реализацией на платформенном уровне [4, с. 710]. Несмотря на технологическое совершенство в слое бизнес-логики, инструментарий для построения кроссплатформенного пользовательского интерфейса находится в стадии активной эволюции. Указанный фактор сопряжен с дефицитом специализированных сторонних библиотек для реализации нестандартной интерактивной графики, критически важной для подсистемы визуализации прогресса [6, с. 415].

Фреймворк Flutter функционирует как портативный инструментарий со встроенным графическим конвейером. Основным инструментом разработки выступает язык Dart, архитектура которого глубоко оптимизирована под задачи создания пользовательских интерфейсов. Инфраструктура языка поддерживает два независимых режима компиляции. В процессе разработки применяется JIT-компиляция, обеспечивающая работу механизма горячей перезагрузки (Hot Reload) для итеративного прототипирования [5, с. 23]. В процессе подготовки релизных сборок активируется AOT-компиляция напрямую в оптимизированный

ARM-машинный код, исключая избыточную нагрузку на интерпретатор [5, с. 24]. Механизм рендеринга базируется на высокопроизводительном движке, отрисовывающем пиксели интерфейса на аппаратном холсте с использованием мощностей графического процессора [2, с. 1303]. Полный отказ от платформенных OEM-виджетов обеспечивает стопроцентную консистентность интерфейса на любых операционных системах. Управление интенсивными вычислительными процессами реализуется посредством механизма изолятов — независимых рабочих потоков с собственной областью памяти, обмен данными между которыми осуществляется через асинхронную передачу сообщений [5, с. 26].

Выбор технологического стека Flutter и Dart для проектирования мобильного приложения строго обоснован. Комплекс требований, включающий построение высокоинтерактивного интерфейса с элементами канбан-доски, глубокую аналитическую визуализацию прогресса и стабильную работу с локальными базами данных, наиболее эффективно реализуется благодаря архитектурным особенностям рассматриваемого фреймворка. Прямая компиляция в машинный код, независимый движок рендеринга и развитая парадигма управления состояниями предоставляют надежный инструментарий для воплощения теоретических концепций гибкого планирования.

Выводы по главе 1

В первой главе были рассмотрены теоретические основы разработки мобильного приложения для гибкого планирования и приоритизации задач. Установлено, что в современной цифровой среде задача управления делами выходит за рамки простого составления списков и требует комплексного подхода, включающего структурирование, контроль сроков, изменение приоритетов, визуализацию прогресса и адаптацию к мобильному сценарию использования.

Проведенный анализ показал, что при проектировании подобного приложения целесообразно опираться на сочетание нескольких методических

подходов: GTD, матрицы Эйзенхауэра, метода MoSCoW и канбан-модели. Каждый из них решает отдельную часть проблемы, а их совместное использование позволяет сформировать более гибкую и практически применимую модель работы с задачами.

Анализ существующих приложений подтвердил, что современные сервисы обладают значительным функциональным потенциалом, однако делают акцент на разных аспектах: личной продуктивности, командной работе, интеграции с внешними сервисами или визуальном управлении процессами. На основании этого сделан вывод о целесообразности разработки мобильного приложения, которое объединяет ключевые функции планирования, приоритизации, визуализации прогресса, синхронизации и персонализации в единой системе.

Полученные теоретические выводы служат основой для практической части выпускной квалификационной работы, в которой будут определены требования к приложению, разработана его архитектура, спроектирован интерфейс и реализованы основные функциональные модули.

Глава 2. Разработка мобильного приложения для гибкого планирования и приоритизации задач

2.1 Обоснование выбора средств разработки

Практическая реализация программного продукта требует научно обоснованного выбора технологического стека, поскольку ошибки на этапе проектирования детерминируют накопление технического долга на последующих стадиях жизненного цикла информационной системы. Проектируемое мобильное приложение классифицируется как система гибкого планирования и управления задачами (task-трекер). Специфика систем данного класса определяет жесткие нефункциональные требования: высокую частоту пользовательских взаимодействий, обеспечение стабильной частоты кадров при рендеринге сложных элементов графического интерфейса и гарантированную отказоустойчивость в условиях нестабильного сетевого соединения. На основании этих критериев архитектурный базис приложения выстроен вокруг парадигмы Offline-First.

В современных исследованиях архитектур мобильных систем [5, с. 321; 7, с. 26] концепция Offline-First описывается не просто как наличие механизма кэширования, а как фундаментальный подход к проектированию, при котором локальное хранилище устройства выступает в роли единственного суверенного источника истины (Source of Truth). Сетевое подключение в данной парадигме рассматривается как опциональная возможность, предназначенная исключительно для фоновой синхронизации данных, а не как обязательное условие функционирования бизнес-логики.

Определение стратегии реализации клиентской части приложения базировалось на сравнительном анализе нативного и кроссплатформенного подходов. Исторически нативная разработка считалась эталоном в контексте производительности и интеграции с аппаратным обеспечением. Однако

необходимость параллельной поддержки изолированных кодовых баз для платформ iOS и Android требует кратного увеличения ресурсных затрат. В исследованиях Е. А. Можаровского [1, с. 152] и М. К. Lodhi [2, с. 45] проводится сравнительный анализ метрик производительности современных фреймворков, результаты которого подтверждают, что актуальные кроссплатформенные решения практически полностью нивелировали отставание от нативных инструментов по скорости выполнения и отзывчивости интерфейса.

При выборе конкретного инструмента кроссплатформенной разработки оценивались архитектурные особенности лидеров сегмента: React Native и Flutter. Фреймворк React Native функционирует на базе асинхронного моста (JS Bridge), отвечающего за сериализацию и трансляцию команд из среды выполнения JavaScript в нативные OEM-компоненты операционной системы. При высокой интенсивности взаимодействий или отрисовке сложных анимаций данный процесс сериализации провоцирует задержки рендеринга [10, с. 1083]. В противоположность этому подходу, архитектура Flutter исключает использование платформенных виджетов. Процесс рендеринга осуществляется с помощью собственного высокопроизводительного графического движка (Impeller/Skia), взаимодействующего непосредственно с графическим API устройства (Metal/Vulkan) [4, с. 84].

Таблица 5 – Сравнительная характеристика технологий кроссплатформенной разработки (нативная, React Native, Flutter)

Характеристика	Нативная разработка	React Native	Flutter
Единая кодовая база	Отсутствует	Да (JavaScript / TypeScript)	Да (Dart)
Принцип рендеринга интерфейса	Вызов OEM-компонентов ОС	Асинхронный JS Bridge + OEM	Независимый движок (Skia/Impeller)

Производительность UI	Эталонная (до 120 fps)	Зависит от пропускной способности моста	Близкая к эталонной (до 120 fps)
Механизм доступа к API платформы	Прямой вызов	Через модули трансляции	Через платформенные каналы

Прямой контроль над каждым пикселем дисплея и отсутствие накладных расходов на межпроцессное взаимодействие делают Flutter оптимальным выбором для реализации интерактивных графиков прогресса, круговых диаграмм и физики перемещения карточек задач (drag-and-drop), заложенных в спецификацию проектируемого приложения.

Выбор фреймворка Flutter детерминировал использование объектно-ориентированного языка программирования Dart. В научных публикациях [3, с. 24] отмечается, что языковая среда Dart оптимизирована специально для разработки клиентских интерфейсов, сочетая строгую статическую типизацию с реактивными паттернами программирования. Ключевым технологическим преимуществом Dart является применение двойственной модели компиляции. На этапе разработки применяется JIT-компиляция (Just-In-Time), обеспечивающая функционирование механизма «горячей перезагрузки» (Hot Reload) — мгновенной инъекции измененного исходного кода в работающую виртуальную машину без потери текущего состояния интерфейса. На этапе сборки релизных версий задействуется AOT-компиляция (Ahead-Of-Time), транслирующая исходный код в оптимизированные бинарные инструкции ARM-архитектуры, что исключает стадию интерпретации на устройстве пользователя и гарантирует высокую скорость запуска [3, с. 26; 4, с. 85].

Дополнительным аргументом в пользу Dart стала его модель параллелизма, основанная на концепции изолятов (isolates). В отличие от классических потоков (threads) с общей памятью, каждый изолят обладает собственным изолированным пулом памяти и независимым циклом обработки событий (Event Loop). Изоляция памяти на уровне виртуальной машины предотвращает возникновение состояний гонки (race conditions) и исключает потребность в механизмах блокировки (mutex). Данная архитектура позволяет выносить ресурсоемкие операции — такие как криптографическое шифрование локальной базы данных или десериализацию объемных JSON-ответов — в фоновые процессы, гарантируя бесперебойную работу основного потока пользовательского интерфейса [1, с. 154].

Наиболее ответственным этапом проектирования Offline-First системы является выбор локальной системы управления базами данных (СУБД). Традиционным решением для мобильных платформ выступает реляционная СУБД SQLite, гарантирующая строгую транзакционную целостность (ACID). Однако реляционная модель накладывает существенные алгоритмические ограничения при работе с глубоко вложенными структурами данных. Базовая предметная сущность разрабатываемого приложения («Задача») обладает сложной топологией: она инкапсулирует списки подзадач, массивы пользовательских тегов, чек-листы и хронометраж сессий фокусировки. Хранение подобных объектов в реляционной базе данных требует нормализации таблиц и выполнения ресурсоемких JOIN-запросов на этапе чтения, а также использования тяжеловесных ORM-библиотек (Object-Relational Mapping) для маппинга структур [6, с. 25].

В качестве оптимальной альтернативы интегрирована документоориентированная СУБД Isar, спроектированная специально для экосистемы Dart. Isar сериализует объекты напрямую в бинарный формат и использует технологию проецирования файлов в оперативную память (Memory-Mapped Files). Сравнительные тестирования производительности

показывают, что при массовой записи вложенных структур документоориентированные NoSQL-решения демонстрируют значительное преимущество по скорости выполнения транзакций по сравнению с классическими ORM-инструментами. Важной инженерной особенностью Isar является встроенная реактивность: любой запрос (Query) к базе данных может быть преобразован в асинхронный поток (Stream), автоматически генерирующий события при мутации данных. Подобная функциональность исключает необходимость написания императивной логики обновления графических виджетов.

Для обеспечения синхронизации данных между устройствами пользователя и управления процессом аутентификации в архитектуру интегрирована облачная платформа Firebase (модули Authentication и Cloud Firestore). При этом в работе строго разграничиваются зоны ответственности локального и удаленного хранилищ. Использование встроенных алгоритмов кэширования Firestore в качестве основного механизма автономной работы признано нецелесообразным, поскольку облачный кэш функционирует как непрозрачная подсистема, алгоритмы вытеснения данных из которой не контролируются бизнес-логикой приложения. В соответствии с методологией, описанной S. Н. Pothineni [5, с. 323], гарантированная доступность данных обеспечивается исключительно при использовании выделенной локальной базы данных в качестве первичного источника. Локальная СУБД Isar сохраняет полную объектную модель приложения, тогда как облачная инфраструктура Firestore выполняет функцию распределенной коммуникационной шины, используемой исключительно для передачи дельта-обновлений в фоновом режиме.

2.2 Проектирование архитектуры приложения

Проектирование архитектурного каркаса программного обеспечения базируется на принципе строгого разделения ответственности (Separation of Concerns). Отсутствие четко очерченных границ между компонентами системы

приводит к высокой связности кода (High Coupling), при которой логика обработки пользовательского ввода, сетевые запросы и правила валидации данных объединяются в монолитные классы. Подобная структура делает код невосприимчивым к масштабированию и исключает возможность проведения изолированного модульного тестирования.

Анализ современных архитектурных методологий выявил два доминирующих подхода: Clean Architecture (Чистая архитектура) и Feature-Sliced Design (FSD). В то время как FSD ориентирован на разбиение проекта на независимые бизнес-фичи и демонстрирует высокую эффективность в масштабных веб-системах, для мобильных приложений со сложной перекрестной моделью данных оптимальным решением признана классическая многослойная структура [8, с. 45]. В рамках проектируемой системы выделены три жестко изолированных уровня:

1. Уровень предметной области (Domain Layer). Представляет собой ядро системы, полностью абстрагированное от внешних фреймворков и библиотек. На данном уровне формализованы бизнес-сущности (модели задач, категорий, пользовательских целей) и сценарии использования (Use Cases). Слой устанавливает абстрактные контракты (интерфейсы) для взаимодействия с данными, диктуя правила работы всей системы.
2. Уровень данных (Data Layer). Выполняет функцию технического провайдера информации. Включает в себя модели передачи данных (DTO — Data Transfer Objects), источники данных (Data Sources) и реализации репозитория. Репозиторий инкапсулирует логику маршрутизации запросов: алгоритм самостоятельно определяет, следует ли извлечь данные из локального хранилища Isar для обеспечения мгновенного отклика, или инициировать обращение к облачной инфраструктуре Firestore [7, с. 28].

3. Уровень представления (Presentation Layer). Ограничен исключительно задачами рендеринга графического интерфейса и управления визуальными состояниями. Слой не содержит бизнес-логики и взаимодействует с системой только посредством вызова контрактов доменного уровня.

Взаимодействие между изолированными слоями реализуется на базе паттерна внедрения зависимостей (Dependency Injection). Для этих целей задействован механизм Service Locator (через пакет GetIt). Использование данного паттерна обеспечивает инверсию управления: высокоуровневые компоненты запрашивают необходимые сервисы из единого реестра, что позволяет осуществлять прозрачную подмену реальных репозиторий на тестовые заглушки (mocks/stubs) в процессе автоматизированного тестирования.

Управление состоянием динамического пользовательского интерфейса реализовано посредством паттерна BLoC (Business Logic Component). В сравнении с более легковесными менеджерами состояний, BLoC формализует потоки данных на основе строгой парадигмы конечного автомата (Finite State Machine). Внешние триггеры — действия пользователя или системные прерывания — трансформируются в объекты Событий (Events). Компонент BLoC обрабатывает входящий поток событий, выполняет необходимую бизнес-логику и эмитирует имутабельные объекты Состояний (States). Графический интерфейс пассивно реагирует на изменения состояния, перерисовывая только обновленные узлы дерева виджетов. Несмотря на необходимость создания большого объема шаблонного кода, детерминированность паттерна BLoC гарантирует абсолютную предсказуемость поведения приложения при обработке сложных асинхронных операций (например, при одновременном обновлении таймеров задач и изменении статусов синхронизации).

Специфика Offline-First архитектуры требует разработки специализированного модуля синхронизации (Sync Engine). Тривиальная перезапись полного снимка базы данных при каждом восстановлении сетевого соединения является ресурсоемким процессом, приводящим к неоправданному расходу заряда аккумулятора и сетевого трафика. В приложении реализована оптимизированная модель дельта-синхронизации (Delta Sync) [6, с. 27]. Для ее обеспечения в локальные модели данных интегрированы служебные метаданные: уникальный идентификатор, флаг состояния isSynced, флаг мягкого удаления isDeleted и серверная метка времени updatedAt.

Синхронизационный процесс вынесен в независимый фоновый изолят и разделен на две фазы. При обнаружении стабильного соединения запускается алгоритм передачи (Push): менеджер формирует пакетный запрос из локальных записей со статусом isSynced == false и отправляет их в Firestore единой транзакцией (batch write). После подтверждения записи сервером локальные флаги обновляются. На второй фазе (Pull) приложение запрашивает из облачного хранилища исключительно те документы, временная метка updatedAt которых превышает время последней успешной синхронизации устройства.

Функционирование любой распределенной системы, подверженной разрывам связи, регламентируется теоремой CAP, согласно которой система вынуждена балансировать между согласованностью (Consistency), доступностью (Availability) и устойчивостью к разделению (Partition tolerance). Offline-First архитектура детерминирует выбор в пользу доступности и переходит к модели строгой конечной согласованности (Eventual Consistency). Это обуславливает неизбежность возникновения коллизий при параллельном редактировании одних и тех же записей на разных устройствах в автономном режиме [5, с. 324].

В современной инженерной практике применяются различные алгоритмы разрешения конфликтов совместного доступа.

Таблица 6 – Сравнительная характеристика стратегий разрешения конфликтов (OT, CRDT, LWW)

Стратегия разрешения конфликтов	Принцип обработки данных	Вычислительная сложность	Сфера применения в мобильной разработке
OT (Operational Transformation)	Трансформация атомарных операций в реальном времени	Высокая	Системы совместного редактирования текстов
CRDT (Conflict-Free Replicated Data Types)	Математически детерминированное слияние графов	Средняя (требует значительного объема памяти)	Распределенные базы данных реального времени
LWW (Last-Write-Wins)	Разрешение коллизий на базе вектора серверного времени	Низкая	CRUD-приложения, task-трекеры, кэши

Анализ структур CRDT (Conflict-Free Replicated Data Types) выявил их математическую безупречность в контексте бесконфликтного слияния. Однако реализация данного алгоритма требует постоянного хранения полного дерева истории всех изменений (tombstones), что приводит к экспоненциальному росту объема локальной базы данных. Учитывая контекст разрабатываемого task-трекера, где гранулярность редактирования ограничивается заменой строковых значений или статусов, принято прагматичное решение об использовании стратегии LWW (Last-Write-Wins) [5, с. 325]. В качестве арбитра коллизий выступает серверная метка времени Firestore. Алгоритм обеспечивает надежную конвергенцию данных без усложнения бизнес-логики и излишнего потребления ресурсов постоянной памяти смартфона.

Смещение вычислительной нагрузки на клиентское устройство (Edge Computing) формирует дополнительные угрозы информационной безопасности. Локальное хранение конфиденциальных данных требует обеспечения защиты от

несанкционированного доступа при физической компрометации файловой системы смартфона. Данная уязвимость нивелирована внедрением аппаратных механизмов шифрования.

Интегрированная СУБД Isar поддерживает прозрачное шифрование страниц базы данных с использованием криптографического стандарта AES с длиной ключа 256 бит. Для обеспечения стойкости криптосистемы уникальный симметричный ключ генерируется при первичной инициализации приложения и никогда не сохраняется в локальных директориях файловой системы. Вместо этого ключ помещается в доверенную зону выполнения (Trusted Execution Environment) операционной системы — аппаратные хранилища Android Keystore или iOS Keychain [9, с. 11]. Доступ приложения к ключу дешифрования блокируется на уровне операционной системы до момента успешного прохождения пользователем процедуры биометрической аутентификации (FaceID, TouchID). Выстроенная многоуровневая архитектура не только удовлетворяет строгим требованиям производительности графического интерфейса, но и формирует надежный инженерный фундамент для безопасной эксплуатации системы в гетерогенных сетевых условиях.

2.3 Реализация структуры базы данных

Функционирование мобильных информационных систем с индивидуальным планированием и командным взаимодействием требует применения гибридных подходов к моделированию структур данных. В условиях нестабильного сетевого покрытия классические клиент-серверные архитектуры демонстрируют недостатки (задержки отклика интерфейса, потеря данных). Спроектирована архитектура на базе интеграции локального хранилища (Isar) и облачной инфраструктуры (Cloud Firestore). Данный симбиоз обеспечивает бесперебойную работу в парадигме Offline-First. Отказ от реляционных моделей (и ресурсоемких операций JOIN) в пользу концепции NoSQL значительно повышает пропускную способность подсистемы ввода-вывода и скорость десериализации [7, с. 52].

Локальная база данных Isar выполняет функцию базового инфраструктурного слоя для автономной работы приложения [1, с. 16]. СУБД реализует хранение объектов в формате типизированных документов, обеспечивая извлечение глубоко вложенных древовидных структур за константное время $O(1)$.

Центральным узлом информационной модели выступает сущность «Задача» (Task). Структура документа включает типизированные атрибуты:

- `id` — скалярный числовой идентификатор (автоинкремент) для оптимизации поиска по локальным B-деревьям.
- `remoteId` — строковый идентификатор UUID для синхронизации с облачным двойником.
- `title` и `description` — текстовые атрибуты с поддержкой токенизированного полнотекстового поиска (Full-text search), не блокирующего основной поток рендеринга.
- `createdAt` и `deadline` — временные метки (Unix Timestamp) для абсолютного исключения коллизий часовых поясов.
- `status` — перечислимый тип данных (enum: `ToDo`, `InProgress`, `Review`, `Done`).
- `progress` — вычисляемый параметр (от 0.0 до 1.0) степени завершенности вложенных подзадач.
- `syncStatus` — служебное мета-поле (`Synced`, `PendingCreate`, `PendingUpdate`, `PendingDelete`) для инициализации фоновых запросов [2, с. 114].

Реализация чек-листов базируется на паттерне внедренных объектов (Embedded Objects). Массив структур Subtask физически располагается внутри родительского документа Task. Исключение внешних ключей позволяет выгружать объект в оперативную память единым блоком, снижая фрагментацию

памяти и нагрузку на Garbage Collector. Изменение логического флага в Subtask через локальный реактивный триггер инициирует автоматический перерасчет атрибута progress в изолированном фоновом потоке.

Справочник категорий (Category) реализован через механизм реактивных ссылок (IsarLink). Изменение визуальных параметров категории (цветового кода, иконки) автоматически генерирует каскадные сигналы инвалидации кэша ассоциированных задач, перерисовывая элементы интерфейса. Сравнительный анализ моделей данных представлен в таблице 1.

Таблица 7 — Сравнение характеристик реляционной и документоориентированной моделей

Критерий	SQLite (реляционная)	Isar NoSQL (документоориентированная)
Организация хранения	Строго нормализованные таблицы	Гибкие типизированные документы
Извлечение сущностей	Процессорозависимые операции JOIN	Единовременное чтение вложенных объектов ($O(1)$)
Реактивность данных	Ресурсоемкие сторонние надстройки	Встроенная поддержка потоков (Streams)
Полнотекстовый поиск	Ограниченная поддержка, задержки	Нативная токенизация на уровне ядра
Миграция схем данных	Выполнение скриптов ALTER TABLE	Динамическое разрешение полей, автомиграция

Обеспечение персистентности в распределенной среде реализовано посредством сервиса Cloud Firestore. СУБД оперирует масштабируемыми иерархиями гибких документов без жесткой схемы [10, с. 103]. Интеграция технологии клиент-сервер на базе Firebase минимизирует задержки

маршрутизации пакетов, предоставляя защищенный канал взаимодействия и встроенную систему аутентификации без промежуточного REST API [6, с. 24].

Облачная база данных денормализована для радикального сокращения количества сетевых запросов. Пространство имен сегментировано на изолированные коллекции `users` и `shared_tasks` [8, с. 32].

Коллекция `users` агрегирует профили авторизованных субъектов (идентификатор, хэшированный email, имя, массив рабочих пространств). Хранение учетных данных делегировано Firebase Authentication [8, с. 32], что соответствует стандартам OAuth2 / OpenID Connect [2, с. 115].

Коллекция `shared_tasks` дублирует семантику локальной модели Task, расширяя ее метаданными: `ownerId` (права на жесткое удаление), `participants` (массив UID для интеграции с Firestore Security Rules [2, с. 116]) и `updatedAt` (серверная метка времени Server Timestamp). Денормализация исключает извлечение профилей при обращении к списку задач [7, с. 53].

Обеспечение отказоустойчивости при фрагментарном сетевом покрытии базируется на кэшировании мутаций в локальной базе [1, с. 15]. При отсутствии соединения транзакции маршрутизируются в Isar с присвоением статуса в `syncStatus`. При восстановлении связи фоновый сервис (Worker Isolate) инициализирует выгрузку накопленных мутаций [2, с. 114].

Разрешение конфликтов параллельной модификации [3, с. 5] реализовано стратегией LWW (Last-Write-Wins) на основе строгого хронологического упорядочивания [3, с. 8]. Алгоритм включает транзакцию обновления с запрашиваемой генерацией новой метки сервером. Облачный кластер осуществляет атомарное сравнение: если временная метка физически более поздней транзакции превосходит текущую, изменения фиксируются [3, с. 8]. При

отклонении транзакции проигравший клиент получает легитимную версию документа через WebSocket-канал [6, с. 25], восстанавливая консистентность в Isar. Использование CRDT признано вычислительно избыточным для обновления бинарных статусов и коротких строк [2, с. 114]. Изменение состояний синхронизации представлено в таблице 2.

Таблица 8 — Изменение состояний жизненного цикла синхронизации

Локальное состояние	Действие без сети	Новое состояние syncStatus	Реакция фонового сервиса при подключении
Данные отсутствуют	Создание задачи	PendingCreate	Выполнение HTTP POST/PUT, смена на Synced
Synced	Изменение атрибутов	PendingUpdate	Выполнение HTTP PATCH, проверка метки LWW
PendingCreate	Изменение атрибутов	PendingCreate	Агрегация изменений, ожидание пакетной отправки
Synced	Жесткое удаление	PendingDelete	Выполнение HTTP DELETE, локальная очистка
PendingCreate	Жесткое удаление	Физическое удаление	Отмена накопленных сетевых транзакций

2.4 Проектирование и реализация пользовательского интерфейса (UI/UX)

Разработка интерфейса для систем оперативно-календарного планирования [4, с. 46] сопряжена с необходимостью обеспечения баланса между высокой информационной плотностью экранов и минимизацией когнитивной нагрузки.

Элиминация избыточных декоративных компонентов снижает зрительное утомление и рассеивание внимания оператора.

Цветовая схема визуальной оболочки приложения подчинена правилу колористики «60-30-10» [9, с. 89]:

60% площади отводится под нейтральный фоновый тон (реализована тема Dark Mode с глубоким темно-серым оттенком, физически снижающим интенсивность свечения OLED-матриц).

30% пространства формируется элементами второго порядка (карточки, навигационные панели). Псевдотрехмерная иерархия слоев обеспечивается генерацией мягких теней (Elevation).

10% площади выделено под акцентные индикаторы со строгим семантическим кодированием (красный — дедлайны, синий — элементы Floating Action Button, зеленый — прогресс-бары).

Типографика откалибрована по международному стандарту инклюзивности WCAG 2.1. Коэффициент контрастности текста к фону зафиксирован на уровне 4.5:1, что гарантирует считывание глифов векторного шрифта под прямыми солнечными лучами.

Центральным узлом взаимодействия выступает интерактивный дашборд [9, с. 88], разделенный на аналитический и операционный секторы. Аналитический блок формирует контекст продуктивности посредством математических репрезентаций: круговой диаграммы (Pie Chart) завершенности задач и линейных графиков (Sparklines) недельной динамики.

Операционный сектор обеспечивает манипуляцию объектами баз данных с возможностью мгновенного переключения (без перезагрузки) между двумя режимами отображения:

Режим линейного списка (List View) — упорядоченная последовательность записей, характеризующаяся высокой скоростью вертикального визуального сканирования.

Режим Канбан-доски (Kanban Board) — пространственная и процессная визуализация статусов. Методология Kanban, основанная на принципе вытягивания [4, с. 46; 9, с. 89], репрезентирует этапы потока операций в виде вертикальных столбцов, а задачи — в виде интерактивных карточек, перемещаемых между этапами [9, с. 90]. Систематизация элементов интерфейса приведена в таблице 3.

Таблица 9 — Систематизация элементов интерфейса и влияние на опыт

Элемент UI	Идентифицированная эргономическая проблема	Психологический и когнитивный эффект
Канбан-доска	Утрата контроля над сложными проектами	Формирование ясного пространственного восприятия [9, с. 90]
Drag-and-Drop	Временные затраты на изменение статусов	Возникновение ощущения прямого тактильного контроля
Пропорция 60-30-10	Зрительное переутомление от плотности данных	Существенное снижение когнитивной нагрузки
Микроанимации	Дефицит внутренней мотивации при рутине	Формирование дофаминового подкрепления итераций

Ключевым интерактивным паттерном изменения статусов выступает пространственное перетаскивание объектов (Drag-and-Drop), исключающее вызов модальных диалоговых окон. Для обеспечения производительности в приложении интегрирован паттерн реактивного управления состоянием BLoC (Business Logic Component), гарантирующий стабильность задержек отрисовки (latency stability) [5, с. 11].

Взаимодействие при перетаскивании карточки формализуется следующим образом:

Действие аппаратно генерирует типизированное событие (Event) в графическом потоке системы.

Слой BLoC валидирует операцию изменения статуса, инициирует асинхронный вызов к СУБД Isar и помещает задачу в очередь на синхронизацию.

Контроллер генерирует новое иммутабельное состояние (State).

Графический фреймворк [5, с. 12] реактивно перерисовывает исключительно зависимые узлы дерева виджетов, гарантируя стабильную частоту в 60 кадров в секунду.

Механизм перетаскивания синхронизирован с подсистемой тактильной обратной связи (haptic feedback) для подтверждения регистрации команды на нейрофизиологическом уровне.

Интерфейс детализации процессов реализован посредством всплывающей нижней панели (Bottom Sheet) с визуальным затемнением (dimming) фонового слоя. Данный оптический прием формирует эффект искусственной глубины резкости: фокус оператора концентрируется на активной задаче, а сохраненный

силуэт Канбан-доски на заднем плане психологически облегчает возврат к макро-планированию, исключая потерю контекста глобального проекта (в отличие от классического стека навигации).

Инструментарий управления чек-листами функционирует в режиме потокового inline-редактирования. Текстовый курсор и виртуальная клавиатура активируются непосредственно в строке списка, исключая вызов блокирующих окон (Alert Dialogs). Структурирование информации обеспечивается управлением весом и контрастностью шрифтов без засечек (sans-serif), а служебные метаданные редуцируются в серый спектр.

Управление параметрами отображения интегрировано в виде горизонтальной скроллируемой панели быстрого выбора (Choice Chips), расположенной под графическим блоком диаграмм. Представление фильтров в форме интерактивных таблеток позволяет применять составные критерии сортировки в одно касание, исключая необходимость вызова тяжеловесных боковых меню (Navigation Drawer). Внедрение микроинтеракций переводит процесс управления потоками операций в интуитивное русло и сокращает время выполнения типовых задач.

2.5 Реализация функциональных возможностей

Проектирование функционального слоя программного комплекса требует точной балансировки между технической производительностью мобильного устройства и когнитивной эргономикой графического интерфейса. Архитектурный базис, сформированный на этапах проектирования, выполняет исключительно утилитарную функцию, в то время как основная практическая ценность цифрового продукта раскрывается через его интерактивные модули. В данном подразделе детально рассматривается программная реализация ключевых пользовательских сценариев: системы локальных уведомлений, управления

вложенными чек-листами, пространственного календарного планирования и визуализации аналитических метрик.

Эффективность современных систем планирования напрямую зависит от стабильности модуля напоминаний. Информирование пользователя о приближающихся сроках выполнения задач является критическим требованием к приложениям категории тайм-менеджмента, что подтверждается анализом пользовательского опыта [6, с. 312]. Разработка фоновых процессов в мобильной среде сопряжена с жесткими платформенными ограничениями. Алгоритмы энергосбережения в ОС Android и строгие лимиты фонового выполнения в iOS инициируют принудительное завершение процессов, находящихся в режиме ожидания. Анализ существующих подходов к проектированию архитектуры планировщиков [1, с. 74] выявил, что применение стандартных библиотек маршрутизации пуш-уведомлений часто приводит к необходимости написания избыточного платформенно-зависимого кода для управления бейджами и сложными расписаниями. В качестве оптимального решения интегрирован пакет, инкапсулирующий низкоуровневую логику планирования и предоставляющий детерминированное управление каналами связи. Внедрение данного инструмента потребовало модификации конфигурационных правил компилятора (в частности, для Android Gradle Plugin 8), чтобы защитить локальную базу данных SQLite, применяемую для кэширования событий, от удаления на этапе обфускации кода.

Реализованный модуль оповещений поддерживает гибкую настройку триггеров: одноразовые вызовы, циклические повторения по дням недели и интервальные напоминания. В ходе системного тестирования была зафиксирована программная уязвимость, связанная с автоматическим удалением запланированных интенгов (PendingIntent) при перезагрузке устройства под управлением ОС Android. Для обеспечения непрерывности работы системы и парирования данной уязвимости внедрен нативный компонент BroadcastReceiver, перехватывающий системное событие инициализации BOOT_COMPLETED. При

запуске операционной системы приложение активирует изолированный фоновый процесс, извлекает из локальной нереляционной базы Isar актуальный список невыполненных задач и перестраивает очередь в планировщике [1, с. 76]. Указанный алгоритм выполняется полностью в фоновом режиме, не требуя ручного вмешательства пользователя.

Избыточная концентрация элементов в линейных списках увеличивает когнитивную нагрузку и снижает скорость принятия решений. Пространственное восприятие времени требует наличия визуальных якорей, функцию которых выполняет календарный модуль [6, с. 314]. Для реализации интерфейса интегрирован компонент `table_calendar`, предоставляющий широкие возможности адаптации состояний. Модуль функционирует в режиме динамической подгрузки данных: контроллер состояния ограничивает объем потребляемой оперативной памяти, загружая события исключительно для текущего активного месяца. При наличии задач с критическим приоритетом или приближающимся дедлайном программный метод `dayBuilder` генерирует соответствующий цветовой маркер под датой, что значительно ускоряет процесс считывания информации.

Особое внимание при разработке уделено механике декомпозиции задач с использованием чек-листов. Разделение комплексной задачи на серию атомарных этапов базируется на принципах когнитивно-поведенческой психологии. Взаимодействие с элементами чек-листа инициирует инвертирование булевых флагов в локальной базе данных, сопровождаясь каскадным пересчетом общего прогресса выполнения родительской сущности. В исследованиях эффективности применения игровых механик в неигровых контекстах [3, с. 122; 5, с. 442; 11, с. 1094] установлено, что внедрение микро-наград в формате немедленного визуального подтверждения прогресса способствует значительному снижению уровня прокрастинации. Дробление сложного задания формирует модель «быстрого успеха», стимулируя вовлеченность пользователя [7, с. 12].

Трансформация числовых статистических показателей в мотивирующий стимул реализована посредством интеграции аналитического дашборда. Как отмечается в профильной литературе, дашборд представляет собой информационно-графическую модель, помогающую сфокусироваться на конкретных показателях продуктивности [4, с. 384]. Выбор графической библиотеки оказывает критическое влияние на производительность приложения при рендеринге элементов интерфейса. Оптимальным решением стала интеграция открытой библиотеки `fl_chart`, использующей плавную математическую интерполяцию кривых Безье. Данный подход обеспечивает высокую отзывчивость графического интерфейса по сравнению с тяжеловесными коммерческими аналогами.

Аналитический экран выполняет агрегацию локальных данных и формирует несколько типов визуальных репрезентаций. Влияние выбранных методов визуализации на когнитивное восприятие сведено в сравнительную таблицу.

Таблица 10 — Применяемые методы визуализации прогресса и их когнитивное воздействие

Тип визуализации	Реализуемая метрика	Психологический эффект	Особенности интерфейсной реализации
Круговая диаграмма (Pie Chart)	Доля завершенных задач за день	Концентрация внимания на текущем моменте, формирование чувства завершенности [4, с. 385]	Анимационное заполнение кольца при достижении 100% выполнения.
Линейный график (Line Chart)	Недельная динамика продуктивности	Выявление паттернов активности,	Применение плавных кривых, исключение

		визуализация периодов спада и подъема [2, с. 16]	жестких координатных сеток для минимизации визуального шума.
Горизонтальная гистограмма	Распределение нагрузки по категориям (работа, личные задачи)	Объективная оценка баланса распределения времени (work-life balance) [7, с. 14]	Использование скрытых легенд, активация точных числовых значений по долгому нажатию.

При проектировании графического интерфейса учитывались современные стандарты инклюзивности и доступности. Анализ методов визуализации педагогической и статистической информации [2, с. 14; 8, с. 5] доказывает, что перегруженные графические элементы создают барьеры для пользователей с нарушениями зрения при работе с экранными лупами. В связи с этим графики очищены от избыточных осей координат и мелких текстовых легенд. Интерактивность интерфейса обеспечена системой сенсорных событий: удержание элемента активирует контрастную всплывающую панель (tooltip) с точным значением. Завершение всех запланированных на день задач сопровождается плавным заполнением кругового графика и легким тактильным откликом устройства (Haptic Feedback). Системное внедрение подобных микроинтеракций повышает общий уровень вовлеченности в использование программного продукта.

2.6 Реализация совместной работы

Индивидуальное планирование охватывает лишь часть функциональных потребностей целевой аудитории. Возникновение необходимости делегирования бытовых задач или координации командных проектов требует трансформации приложения в распределенную информационную систему. Обеспечение совместного доступа в условиях нестабильности мобильных сетей представляет

собой сложную инженерную задачу, требующую контроля целостности данных, разрешения конфликтов конкурентного доступа и оптимизации сетевого трафика.

Механизм приглашения новых пользователей спроектирован с фокусом на минимизацию барьеров входа и обеспечение бесшовного пользовательского опыта. Процедуры корпоративного сегмента, связанные с ручным вводом длинных адресов электронной почты, признаны неоптимальными для мобильного контекста. В качестве альтернативы реализована система генерации уникальных криптографических кодов и ссылок глубокого связывания (Deep Link), передача которых осуществляется через системное меню обмена данными с использованием плагина `share_plus`.

Подключение новых участников сопровождается обращением к облачной инфраструктуре Firebase Firestore. Архитектура безопасности базируется на ролевой модели управления доступом (Role-Based Access Control — RBAC). В исследованиях распределенных корпоративных систем [10, с. 4] обосновывается, что ограничение прав непосредственно на уровне NoSQL-базы данных предотвращает возможность несанкционированной модификации записей. Правила безопасности (Security Rules) в Firebase Firestore сконфигурированы для строгого разграничения полномочий. Инициатор задачи (Owner) наделяется эксклюзивным правом удаления сущности. Подключенные участники, идентификаторы которых фиксируются в массиве `participants`, получают ограниченные привилегии, допускающие чтение и точечную модификацию статусов вложенных элементов чек-листа. Любая попытка со стороны клиента подменить метаданные или передать права владения автоматически блокируется сервером.

Основой совместной работы выступает механизм синхронизации данных в реальном времени. В целях оптимизации производительности осуществлен отказ от архаичного метода периодического опроса сервера (HTTP-polling). Приложение

эксплуатирует реактивные слушатели onSnapshot в составе Firebase SDK, поддерживающие непрерывное соединение по протоколу WebSocket. Изменение статуса подзадачи одним пользователем инициирует мгновенную отправку дельта-обновления сервером Firestore на устройства остальных участников. На практике данная архитектура сокращает задержку передачи пакетов до 1–2 секунд, обеспечивая иллюзию абсолютной синхронности действий.

Ключевой технической проблемой распределенных вычислений является алгоритмизация слияния конфликтующих изменений. Ситуация конкурентного редактирования описания задачи при отсутствии сетевого соединения требует детерминированного поведения системы при последующей выгрузке локальных изменений на сервер. В современной архитектуре распределенных приложений выделяются три основные стратегии разрешения конфликтов, сравнительный анализ которых представлен в таблице 2.2.

Таблица 11 — Сравнительный анализ стратегий разрешения конфликтов при совместной работе

Стратегия	Принцип работы	Достоинства	Недостатки
CRDT (Conflict-Free Replicated Data Types)	Математическое слияние независимых правок без участия централизованного сервера-арбитра.	Строгая консистентность, идеальная поддержка децентрализованных систем.	Существенное увеличение объема локальной базы данных вследствие хранения дерева истории всех операций [9, с. 8].
OT (Operational Transformation)	Преобразование конфликтных транзакций в реальном времени через центральный сервер.	Безупречная обработка совместного редактирования длинных текстовых массивов.	Высокая сложность серверной инфраструктуры, критическая зависимость от

			стабильного интернет-соединения.
LWW (Last-Write-Wins)	Маркировка изменений серверными метками времени (Server Timestamp). Приоритет отдается самой поздней транзакции.	Простота программной реализации, минимальные затраты вычислительных ресурсов и сетевого трафика.	Риск перезаписи данных при возникновении жестких временных коллизий.

Согласно результатам исследования архитектуры Offline-First [9, с. 9], выбор метода синхронизации требует постоянного компромисса между консистентностью и доступностью. Алгоритмы CRDT, несмотря на их математическую точность, признаны избыточными для управления короткими текстовыми заголовками в рамках таск-трекера ввиду высоких накладных расходов на хранение метаданных. Стратегия ОТ противоречит базовым принципам концепции Offline-First, требуя непрерывного серверного контроля.

На основании проведенного анализа для обеспечения консистентности локальных хранилищ Isar и облачной среды Firestore применен алгоритм Last-Write-Wins (LWW) с использованием серверных меток времени. При восстановлении сетевого соединения мобильное устройство передает пакет изменений. Сервер, фиксируя конфликт версий, принимает транзакцию, сформированную позднее по времени. В контексте системы управления задачами, где статистическая вероятность одновременного посимвольного редактирования одного поля минимальна, подход LWW является прагматичным решением. Использование данной стратегии гарантирует мгновенную отзывчивость пользовательского интерфейса, снижает энергопотребление и поддерживает необходимый уровень целостности данных.

2.7 Тестирование разработанного приложения

Валидация программного обеспечения в парадигме современных кроссплатформенных решений представляет собой многоуровневый процесс, выходящий за рамки тривиального поиска визуальных дефектов. Применение архитектуры Offline-First фундаментально трансформирует методологию тестирования. В данной архитектурной модели устройство пользователя не функционирует как пассивный «тонкий клиент», а выступает автономным, суверенным узлом распределенной вычислительной системы. Децентрализация обработки информации требует строгой верификации механизмов локального кэширования, арбитража конфликтов при конкурентном доступе к данным и эргономики графического пользовательского интерфейса.

Процесс валидации разработанного приложения концептуально разделен на несколько ортогональных направлений. Оценивались как низкоуровневые показатели пропускной способности мобильной базы данных, так и высокоуровневые когнитивные метрики взаимодействия пользователя с цифровым пространством. Синтез автоматизированных проверок и эмпирического анализа поведенческих паттернов формирует объективную картину надежности программного продукта.

2.7.1 Архитектура автоматизированного и интеграционного тестирования

Интеграционное тестирование является фундаментом стабильности программного комплекса, предотвращающим регрессионный коллапс при масштабировании кодовой базы. Экосистема кроссплатформенного фреймворка Flutter включает классическую трехуровневую пирамиду валидации: модульные (Unit), виджетные (Widget) и интеграционные (Integration) тесты. В исследовании установлено, что глубокая интеграция автоматизированных тестовых сценариев в

конвейеры непрерывной интеграции (CI/CD) кардинально снижает накопленный технический долг и повышает общую надежность релизных сборок.

Модульные тесты (`flutter_test`) применялись для проверки изолированных функций: алгоритмов вычисления прогресса выполнения задач, обработки древовидных структур чек-листов и парсинга временных меток. Виджетные тесты (`WidgetTester`) обеспечивали валидацию отрисовки отдельных компонентов в виртуальной среде без привлечения реального движка рендеринга, что позволило оценить корректность генерации динамических диаграмм при граничных значениях.

Для симуляции сквозных пользовательских путей (`User Journeys`) задействован штатный пакет `integration_test`. При проектировании архитектуры тестирования выявлена методологическая проблема нестабильности (`flakiness`) интеграционных тестов, вызванная задержками сетевой маршрутизации. Обращение к реальному облачному бэкенду во время автоматизированных прогонов приводит к ложным срабатываниям (`False Positives`) из-за кратковременных обрывов соединения или мутаций состояния внешней базы данных.

Для устранения сетевой нестабильности в разработанной системе реализован паттерн мокирования (`Mocking`) ответов сервера. Репозитории уровня данных (`Data Layer`), обращающиеся к облачным коллекциям, динамически подменялись локальными заглушками посредством механизма внедрения зависимостей (`Dependency Injection` через `Service Locator`). Такая изоляция обеспечила полную детерминированность среды: автоматика проверяла исключительно корректность взаимодействия пользовательского интерфейса, контроллеров бизнес-логики (`BLoC`) и локальной базы данных.

В работе рассмотрены перспективы визуального GUI-тестирования на базе компьютерного зрения, однако классический декларативный подход Flutter с использованием семантических ключей (Keys) обеспечивает более высокую скорость и надежность поиска элементов. Изучен потенциал фреймворка Patrol , расширяющего возможности взаимодействия с нативными компонентами операционной системы, однако для полного покрытия бизнес-логики разрабатываемого продукта штатных средств оказалось достаточно.

Таблица 12 – Сравнительная характеристика уровней тестирования

Уровень тестирования	Инструментарий во Flutter	Цель валидации в рамках разработанного приложения	Среднее время выполнения
Модульное (Unit)	flutter_test	Проверка математических алгоритмов, бизнес-логики BLoC, парсеров дат и метаданных.	< 0.1 сек
Виджетное (Widget)	WidgetTester	Валидация отрисовки кольцевых диаграмм, реакции UI-форм на граничные параметры.	~ 1-2 сек
Интеграционное (E2E)	integration_test	Сквозной прогон сценариев изменения статуса объекта и постановки транзакции в очередь.	~ 8-12 сек

2.7.2 Испытание производительности локального хранилища данных

Эффективность архитектуры Offline-First напрямую зависит от пропускной способности локального хранилища. При переходе в автономный режим база данных мобильного устройства обязана абсорбировать операции чтения и записи без блокировки главного потока рендеринга (Main Thread).

Традиционная реляционная СУБД SQLite обладает высокой надежностью, однако ее жесткая табличная схема снижает производительность при обработке древовидной, объектной структуры современных кроссплатформенных приложений. Проектируемая система оперирует глубоко вложенными структурами: задача включает массивы подзадач, теговые ассоциации и временные метки синхронизации. Конвертация многомерных графовых объектов в плоские реляционные таблицы посредством ORM-надстроек ведет к необходимости выполнения ресурсоемких каскадных запросов (JOIN), что критически увеличивает нагрузку на мобильный процессор при десериализации. В противовес этому, документо-ориентированные (NoSQL) решения обеспечивают оптимизированную архитектуру для массовых операций записи неструктурированных данных.

Для верификации характеристик спроектирован стресс-тест локального хранилища Isar, интегрированного в приложение. Сценарий включал пакетную запись (batch insert) массива из 25 000 задач с вложенными подзадачами. Локальный движок был инициализирован в режиме ослабленной долговечности (relaxed durability). Данный режим представляет собой инженерный компромисс, минимизирующий дисковые операции fsync и отдающий приоритет пропускной способности записи за счет риска потери последних миллисекунд данных при жестком аппаратном сбое.

Результаты измерений подтвердили, что массив из 25 000 объектов обрабатывается менее чем за 40 миллисекунд. Сериализация данных в

оптимизированный бинарный формат и использование прямого отображения в память (Memory-Mapped I/O) обеспечили производительность, достаточную для реализации функции мгновенного полнотекстового поиска (Full-text search). Фильтрация тысяч записей происходит синхронно с частотой обновления графического интерфейса 60 кадров в секунду, не создавая ощутимых задержек (latency).

2.7.3 Валидация алгоритмов дельта-синхронизации и арбитража конфликтов

Критическим компонентом распределенной автономной системы является механизм синхронизации данных . При конкурентном редактировании информации в режиме отсутствия сети (например, двумя пользователями в общем рабочем пространстве) неизбежно возникает коллизия состояний при последующем восстановлении соединения. Система требует алгоритмически детерминированного арбитража конфликтов репликации .

Академические подходы к разрешению конфликтов в распределенных системах преимущественно базируются на двух стратегиях: эвристике Last-Write-Wins (LWW) и структурах Conflict-Free Replicated Data Types (CRDT). Метод LWW, основанный на строгом сравнении временных меток, подвержен риску скрытой потери данных (Data Loss) при синхронизации монолитных документов . Если два пользователя асинхронно изменяют разные поля одного документа, применение правила LWW к объекту целиком приведет к перезаписи и уничтожению части изменений.

Концепция CRDT смещает фокус с синхронизации состояний на слияние атомарных операций . Коммутативные и идемпотентные свойства структур CRDT математически исключают конфликты, поскольку базы данных объединяют журналы событий. Однако интеграция CRDT требует генерации глобальных уникальных идентификаторов, применения логических векторных часов и отказа

от физического удаления записей (данные помечаются "надгробиями"). В контексте мобильного приложения это влечет экспоненциальное увеличение объема локальной базы данных .

Учитывая статистически низкую вероятность конкурентного редактирования одного и того же текстового поля, в разработанной архитектуре реализован гибридный подход: стратегия LWW с гранулярным разделением атрибутов . Синхронизация выполняется не на уровне монолитного документа, а на уровне атомарных полей, что позволяет балансировать между консистентностью данных и вычислительной сложностью программного обеспечения.

Таблица 13 – Сравнительная характеристика алгоритмов разрешения конфликтов (LWW, CRDT, гибридный)

Характеристика алгоритма	Эвристика Last-Write-Wins (монолитная)	Структуры CRDT (Conflict-Free Replicated Data Types)	Реализованный гибридный подход
Базовый механизм арбитража	Сравнение серверных меток времени документа	Коммутативное слияние журналов операций	LWW с гранулярным слиянием на уровне полей
Риск тихой потери данных	Высокий (полная перезапись состояния)	Математически исключен алгоритмом	Минимизирован (коллизия только внутри одного поля)
Потребление дисковой памяти	Минимальное	Высокое (хранение истории и векторных часов)	Низкое (оптимизировано под мобильные устройства)

Вычислительная сложность	Низкая (поддерживается ядром СУБД)	Крайне высокая (требует кастомных резолверов)	Умеренная (штатный арбитраж на уровне атрибутов)
--------------------------	------------------------------------	---	--

Эмпирическая валидация механизма арбитража проводилась в условиях симуляции сетевого обрыва. На двух устройствах, авторизованных в едином профиле и переведенных в автономный режим, были произведены непересекающиеся изменения одной задачи. При одновременном восстановлении соединения фоновый воркер дельта-синхронизации сформировал пакеты обновлений. Облачная СУБД сопоставила метки времени и детерминированно выполнила слияние (Merge) на уровне полей. Изменения, не прошедшие валидацию при искусственно спровоцированном конфликте (редактирование одного поля), были отклонены. Клиентское устройство мгновенно получило уведомление через реактивный канал WebSocket, после чего локальный кэш автоматически инициировал откат (Rollback) до консистентного серверного состояния.

2.7.4 Анализ когнитивной нагрузки, юзабилити-тестирование и эффекты геймификации

Техническая надежность кодовой базы нивелируется, если графический интерфейс сопряжен с высокой экзогенной когнитивной нагрузкой. Ежедневное планирование требует значительных ресурсов исполнительных функций мозга. Перегруженность интерфейса провоцирует увеличение числа зрительных саккад, что ведет к быстрому утомлению пользователя. В исследованиях утверждается, что интерфейс признается эргономичным, когда навигационные решения принимаются без длительного семантического анализа визуальных блоков.

Для объективной оценки эргономики графической среды проведено юзабилити-тестирование с использованием шкалы System Usability Scale (SUS).

Участники выполняли стандартизированные сценарии взаимодействия в двух средах: традиционном текстовом списке (контрольная среда) и на спроектированной Канбан-доске (экспериментальная среда). Дизайн приложения базировался на правиле пропорционального распределения визуального пространства «60-30-10» (60% негативного пространства, 30% функциональных карточек, 10% акцентных индикаторов статуса). Зафиксировано статистически значимое снижение времени выполнения операций (Task Completion Time). Уменьшение визуального шума минимизировало когнитивную нагрузку, способствуя более быстрой обработке информации.

Интеграция игровых механик (геймификация) выступает инструментом трансформации рутинных задач. Тем не менее, масштабные метаанализы цифровых интервенций строго предостерегают от неконтролируемого внедрения соревновательных элементов. Использование агрессивных механик, таких как публичные таблицы лидеров (Leaderboards), приводит к выгоранию внутренней (intrinsic) мотивации пользователя. Действия начинают выполняться из-за экзогенного давления, что ведет к резкому снижению вовлеченности при потере интереса к искусственному рейтингу.

Архитектура разрабатываемого приложения опирается на принципы безопасной визуализации индивидуального прогресса без применения сравнительных метрик. Базовым элементом интерфейса выступают динамические кольцевые диаграммы, плавно заполняющиеся по мере выполнения чек-листов. Дополнительно внедрена механика мультимодального тактильного отклика (Haptic Feedback). Изменение статуса задачи (перемещение в колонку «Завершено») сопровождается кратким вибрационным импульсом, синхронизированным с визуальной анимацией.

Анализ поведенческих метрик подтверждает, что осязаемая визуализация микроцелей служит катализатором продуктивности. В недавних исследованиях

показано, что гуманистический подход к геймификации способствует формированию устойчивой цифровой привычки (Habit Loop) без использования манипулятивных механик. Проведенное тестирование доказывает, что реализованный интерфейс обеспечивает поддержание высокого уровня самоорганизации при минимальных затратах когнитивных ресурсов пользователя.

2.8 Анализ результатов

Оценка разработанного программного продукта базируется на синтезе данных, полученных в ходе многоуровневого нагрузочного и пользовательского тестирования. Изолированные метрики пропускной способности и скорости рендеринга анализируются в контексте пользовательского опыта и архитектурных ограничений, определенных на начальном этапе проектирования.

2.8.1 Кроссплатформенная графическая производительность

При выборе кроссплатформенного фреймворка критическим аспектом выступает графическая производительность. В работе исследуется способность кода на языке Dart обеспечивать стабильную частоту обновления экрана на уровне 60–120 кадров в секунду. Профилирование производительности (Performance Profiling) подтвердило отсутствие пропущенных кадров (jank) при рендеринге списков с высокой визуальной плотностью. В исследованиях [8, с. 45; 14, с. 112] установлено, что высокая графическая производительность достигается за счет прямого рендеринга через графический движок Flutter (Skia/Impeller), что нивелирует задержки межпроцессного взаимодействия, свойственные мостовым архитектурам.

Дополнительно в приложении реализована строгая изоляция бизнес-логики. Вычислительно сложные операции, включая фоновую десериализацию JSON-пакетов из сервиса Firebase и комплексные запросы к локальной базе данных Isar, делегированы в изолированные потоки памяти (Isolates). Применение

данного подхода позволяет высвободить главный интерфейсный поток (UI Main Thread) исключительно для расчета матриц трансформации и интерполяции кривых Безье при отрисовке анимаций прогресса. Совокупность данных архитектурных решений обеспечивает высокую отзывчивость пользовательского интерфейса, что подтверждается результатами тестирования графической производительности мобильных систем [5, с. 14; 18, с. 205]. Приложение функционирует монолитно и быстро, что является критическим фактором для утилитарного программного обеспечения.

2.8.2 Надежность и отказоустойчивость распределенной оффлайн-архитектуры

Тестирование системы в условиях искусственно смоделированного отсутствия сетевого подключения подтвердило эффективность концепции суверенного локального хранилища (offline-first). В отличие от традиционных облачных решений, где потеря связи приводит к блокировке интерфейса, разработанная архитектура гарантирует непрерывность взаимодействия. Транзакции выполняются в локальной базе Isar с миллисекундной задержкой независимо от сетевого статуса устройства.

Анализ сетевой телеметрии фонового модуля синхронизации продемонстрировал результативность механизма дельта-обновлений. Передача в облачный сервис исключительно модифицированных узлов графа данных позволила снизить объем передаваемого трафика более чем в три раза по сравнению с методом периодического опроса (HTTP-polling) [2, с. 625; 4, с. 42]. Встроенная очередь задач (Sync Queue) локально накапливает изменения, инициируя пакетную выгрузку при восстановлении связи с применением алгоритмов экспоненциальной задержки при сбоях (Exponential Backoff) [13, с. 322].

Анализ статистики использования приложения выявил, что вероятность конкурентной коллизии при модификации одного поля стремится к математической погрешности. Следовательно, интеграция структур бесконфликтных реплицированных типов данных (CRDT) признана избыточной, так как она усложняет кодовую базу и повышает ресурсоемкость [12, с. 15]. Стратегия разрешения конфликтов Last-Write-Wins (LWW), примененная на уровне атомарных полей документа, обеспечивает необходимый баланс между надежностью и производительностью. Сравнение показывает, что данный архитектурный компромисс является оптимальным для проектируемой системы [17, с. 46].

2.8.3 Психоэргономический профиль пользовательского интерфейса

Анализ результатов юзабилити-тестирования подтверждает, что избыточная информационная плотность экрана провоцирует когнитивную перегрузку пользователя. Вывод полного объема аналитической статистики на один экран приводит к рассеиванию внимания. В исследовании [10, с. 88] установлено, что минимизация визуального шума и применение паттернов прогрессивного раскрытия данных (Progressive Disclosure, реализация через Bottom Sheet) способствуют снижению зрительной утомляемости. Сравнительный анализ классического списочного интерфейса и разработанного дашборда с Канбан-доской представлен в таблице 1.

Таблица 14 — Сравнительная оценка интерфейсных решений

Когнитивный и поведенческий показатель	Классический списочный интерфейс (Контрольная группа)	Дашборд с Канбан-доской и визуализацией прогресса (Разработанное приложение)
--	---	--

Уровень зрительной утомляемости (по отзывам)	Высокий (однородный мелкий текст быстро сливается в массив)	Крайне низкий (цветовое кодирование и карточки мгновенно направляют фокус)
Частота взаимодействия с приложением	Редкая (открытие раз в конце дня для отчета)	Высокая (регулярные короткие сессии в течение всего дня)
Динамика завершения запланированных дел	Стагнация на базовом уровне	Устойчивый рост продуктивности на 18% (эффект позитивного подкрепления)
Общий индекс удовлетворенности интерфейсом (Шкала SUS)	68 / 100 баллов	89 / 100 баллов

Анализ логов взаимодействия выявил поведенческое изменение: физический жест Drag-and-Drop в Канбан-доске, сопровождаемый тактильным откликом устройства, функционирует как механизм позитивного подкрепления. Зафиксировано целенаправленное дробление крупных задач пользователями для увеличения частоты взаимодействия с интерфейсом и активации визуализации прогресса. Доказано, что визуализация прогресса и геймификация способствуют формированию устойчивой вовлеченности [1, с. 215; 15, с. 45].

Разработанный интерфейс стимулирует механизмы формирования полезных привычек без применения агрессивных уведомлений, трансформируя приложение в полноценного цифрового ассистента. В результате выполнения практической части разработано мобильное приложение, реализующее гибкое планирование задач, систему приоритизации, визуализацию прогресса, облачную синхронизацию и совместную работу пользователей. Использование Flutter позволило создать единый интерфейс для мобильных платформ. Применение Isar

обеспечило высокую скорость локальной работы, а сервис Firebase — надежную синхронизацию. Результаты исследования подтверждают, что выбранная архитектура является эффективной для приложений управления задачами и может использоваться в качестве основы для дальнейшего развития продукта.

Выводы по главе 2

Во второй главе описан процесс проектирования и разработки программной архитектуры мобильного приложения: от обоснования выбора технологического стека до проведения эмпирического тестирования программного продукта. Проектирование системы выполнено с учетом требований к обеспечению интерактивности графического интерфейса, отказоустойчивости в условиях нестабильного сетевого соединения и снижению когнитивной нагрузки на пользователя.

Установлено, что применение кроссплатформенного фреймворка Flutter совместно с локальной нереляционной базой данных Isar формирует техническую базу для реализации архитектурного паттерна Offline-First. Разработан механизм асинхронной дельта-синхронизации данных на основе облачной инфраструктуры Firebase Firestore. Использование стратегии разрешения распределенных конфликтов Last-Write-Wins (LWW) показывает высокую степень отказоустойчивости при оптимизированном расходе сетевого трафика и вычислительных ресурсов мобильного устройства. В рамках предложенного подхода локальная база данных выступает в качестве единого источника истины (Single Source of Truth), что обеспечивает отсутствие задержек при взаимодействии пользователя с интерфейсом независимо от качества интернет-соединения.

Результаты интеграционного и нагрузочного тестирования подтверждают стабильность функционирования мобильного приложения при обработке

массивов данных, превышающих десять тысяч записей. Сравнение подходов к синхронизации демонстрирует целесообразность применения эвристических алгоритмов на основе временных меток вместо структур данных, бесконфликтно реплицируемых типов (CRDT), для оптимизации процессов в предметной области управления задачами.

Анализ пользовательского взаимодействия (юзабилити-тестирование) с участием целевых фокус-групп подтверждает эффективность примененных методов геймификации и визуализации прогресса. Использование графиков прогресса, интерактивной пространственной Канбан-доски с функцией тактильного отклика и стандартизированного распределения цветовых акцентов (правило 60-30-10) способствует повышению уровня вовлеченности в процесс планирования.

Анализ полученных данных позволяет сделать вывод, что разработанное мобильное приложение для гибкого планирования и приоритизации задач соответствует целям исследования. Программный продукт отвечает стандартам разработки программного обеспечения и может применяться на практике в качестве инструмента для организации личной и совместной продуктивности.

Заключение

Выполненное исследование позволило всесторонне изучить теоретические и практические аспекты управления временными ресурсами в цифровой среде. Анализ научной литературы подтвердил, что в условиях высокой когнитивной нагрузки традиционные бумажные методы планирования уступают место цифровым инструментам персонального тайм-менеджмента [1, с. 5]. Систематическое применение специализированного программного обеспечения минимизирует риски когнитивной перегрузки и предотвращает снижение индивидуальной результативности в условиях многозадачности [2, с. 26]. Автоматизация составления расписаний высвобождает интеллектуальные ресурсы человека для аналитической деятельности [3, с. 111]. Научно доказано, что эффективность персональной самоорганизации зависит от доступности гибких инструментов, поддерживающих научно обоснованные методы приоритизации и визуализации результатов [5, с. 47].

С целью реализации теоретических положений было разработано кроссплатформенное мобильное приложение для операционной системы Android на базе фреймворка Flutter и языка Dart. Выбор технологического стека обоснован сравнительным анализом средств разработки, показавшим преимущество Flutter по скорости сборки интерфейса при сохранении нативной производительности графического рендеринга [16, с. 24]. Использование единого кода позволило сократить затраты на проектирование и отладку пользовательского интерфейса [17, с. 140]. Особое внимание уделено вопросам энергоэффективности асинхронных операций, что минимизировало расход энергии мобильного устройства при фоновой активности и обработке уведомлений [14, с. 255]. Для хранения данных пользователей выбрана встраиваемая база данных, гарантирующая высокую скорость доступа к записям без избыточного расходования оперативной памяти [8, с. 736].

В ходе практической разработки были реализованы ключевые функциональные модули приложения. Инструмент включает модуль создания, редактирования и декомпозиции комплексных целей на подзадачи с использованием чек-листов. В архитектуру интегрирован блок приоритизации на основе матрицы Эйзенхауэра и ABC-анализа, позволяющий оперативно ранжировать задачи по критериям срочности и важности для минимизации субъективного ощущения хаоса в условиях дефицита времени [12, с. 75; 13, с. 57]. Разработанный графический модуль обеспечивает визуализацию прогресса с помощью диаграмм, графиков и календарного представления личной активности [9, с. 102; 10, с. 125]. Это переводит процесс планирования в плоскость активного рефлексивного анализа, способствуя формированию самодисциплины и своевременному выявлению периодов высокой работоспособности [11, с. 580]. Дополнительно внедрены механизмы push-уведомлений и облачной синхронизации для обеспечения сохранности данных [15, с. 1083].

Сопоставление разработанного программного решения с аналогами продемонстрировало сбалансированность созданного продукта. В отличие от корпоративных систем, характеризующихся избыточностью функций и сложным интерфейсом, разработанное приложение ориентировано на индивидуальное использование [6, с. 315]. Базовые планеры уступают созданному решению из-за отсутствия встроенных механизмов визуализации личной динамики. Разработанный продукт устраняет данный дисбаланс, сочетая простоту интерфейса с аналитическим аппаратом контроля успеваемости. Тестирование подтвердило правомерность выдвинутой гипотезы: использование мобильного инструмента гибкого планирования обеспечивает прирост персональной продуктивности и способствует снижению уровня стресса за счет упорядочивания повседневной деятельности [7, с. 301].

Практическая значимость выполненного проекта заключается в создании готового кроссплатформенного мобильного приложения, которое может использоваться широким кругом лиц для оптимизации повседневной

деятельности и эффективного распределения временных ресурсов [4, с. 56].

Разработанное решение может выступать основой для последующих исследований на стыке прикладной информатики, эргономики и когнитивной психологии. Дальнейшее совершенствование программного продукта связано с расширением возможностей совместной работы, а также интеграцией алгоритмов машинного обучения для автоматического ранжирования расписания на основе исторических данных о продуктивности и индивидуальных биоритмах когнитивной активности [18, с. 1326].

Список использованных источников

1. Абрамова О. Ф., Семилетов И. Д. Проектирование актуального решения для планирования задач на базе операционной системы Android // Вестник Адыгейского государственного университета. Серия 4: Естественно-математические и технические науки. — 2022. — № 4 (311). — С. 71–80.
2. Аветян А. А., Григорян Э. А. Современные методы тайм-менеджмента и их значение в жизни современного человека // Шестнадцатая годовичная научная конференция. Социально-гуманитарные науки. Часть II. — М., 2024. — С. 55–60.
3. Азевич А. И. Сервисы визуализации данных: приемы и решения // Вестник Московского городского педагогического университета. Серия: Информатика и информатизация образования. — 2019. — № 1 (47). — С. 13–19.
4. Андрушко И. С. Кроссплатформенная разработка мобильного приложения на фреймворке Flutter // Сборник трудов конференции. — Гродно: Гродненский государственный университет имени Янки Купалы, 2022. — С. 83–85.
5. Ануфриева А. А., Горбунова Е. С. Перегружает, но направляет: влияние средовой насыщенности на внимание и рабочую память в реальных и цифровых условиях // Психологические исследования. — 2025. — № 101. — С. 6–19.
6. Архангельский Г. А., Бехтерев С. В., Лукашенко М. А., Телегина Т. В. Тайм-менеджмент. Полный курс: учебное пособие. — М.: Альпина Паблишер, 2020. — 311 с.
7. Архитектура и интеграция приложений для автоматизации смарт-дома на основе IoT: сборник научных трудов. — М.: Изд-во МГТУ, 2021. — С. 112–118.

8. Басова С. Н., Сидорова Н. П., Торопова Т. А. Цифровые решения тайм-менеджмента в органах государственной власти // Вопросы управления. — 2021. — № 3. — С. 162–177.
9. Боженова В. В. Исследование влияния визуализированного прогресса в обучающем курсе на результаты обучения и мотивацию студентов // Культура и технологии. — 2023. — Т. 8, № 4. — С. 212–217.
10. Борискин А. С. Методы синхронизации данных в Android-приложениях // Вестник науки. — 2025. — Т. 4, № 4 (85). — С. 624–628.
11. Бояшова Е. П., Гоняев С. С., Кан В. Г. Методы повышения эффективности геймификации систем тайм-менеджмента // Актуальные проблемы инфотелекоммуникаций в науке и образовании (АПИНО 2023): сборник научных статей XII Международной конференции. — СПб.: СПбГУТ, 2023. — Т. 2. — С. 120–125.
12. Бурукина И. П., Привалов А. Э. Исследование современных подходов к проектированию цифровых интерфейсов // Вестник науки. — 2022. — Т. 3. — С. 15–22.
13. Вершинин В. П. Роль цифровизации в области управления проектами // Гуманитарные, социально-экономические и общественные науки. — 2024. — № 12. — С. 35.
14. Вишталюк С. Д. Современные техники управления временем // Вопросы науки и образования. — 2022. — № 1. — С. 15–22.
15. Волкова Н. В., Аникина П. А. Тайм-менеджмент сегодня // Столыпинский вестник. — 2021. — № 5. — С. 1–8.
16. Гильдебрант Е. Ю., Вайндорф-Сысоева М. Е. Цифровые инструменты для организации самостоятельной работы студентов с применением тайм-менеджмента // Современные технологии управления. — 2023. — № 3 (103). — С. 1–10.
17. Дегтяренко К. А., Евтушевская С. И. Развитие навыков тайм-менеджмента студентов педагогических направлений подготовки в аспекте личностного и

- профессионального саморазвития будущих педагогов // Вестник ПСТГУ. Серия IV: Педагогика. Психология. — 2025. — Вып. 78. — С. 106–117.
18. Демиш В. О., Пищик Б. Н. Автономная работа android-приложений и алгоритмы синхронизации данных // Вестник Новосибирского государственного университета. Серия: Информационные технологии. — 2014. — Т. 12, № 2. — С. 45–53.
19. Диденко В. А. Исследование фреймворков для разработки кроссплатформенных мобильных приложений // Вестник науки. — 2024. — Т. 3, № 5 (74). — С. 1081–1088.
20. Добренко Н. В., Бабан В. Ю., Чернышева А. В., Говоров А. И. Разработка интегрированной панели управления жизненным циклом моделей машинного обучения для ИТ-организаций с распределенной инфраструктурой // Экономика. Право. Инновации. — 2025. — Т. 13, № 4. — С. 83–100.
21. Доманский В. О., Тарханова О. В., Пелевин М. Д. Анализ возможностей мобильных Agile-решений для эффективной проектной деятельности // Архитектура, строительство, транспорт. — 2021. — № 3. — С. 98–105.
22. Дружинина В. Д. Основные принципы методологии Scrum для управления проектами // Экономика. Социология. Право. — 2022. — № 3 (27). — С. 17–21.
23. Дындин А. В., Новиков П. С. Исследование применения Kotlin Multiplatform и Jetpack Compose Multiplatform в мобильной разработке // Вестник науки. — 2024. — № 4 (73). — С. 410–421.
24. Ершов Т. А., Голубничий А. А. Использование методологии Kanban при разработке программного обеспечения // StudNet. — 2021. — № 8. — С. 1–10.
25. Желаева С. Э., Зубакин А. А. Технологии тайм-менеджмента в учебной деятельности студента: краткий обзор и возможности применения // Актуальные проблемы авиации и космонавтики. — 2021. — С. 106–109.

26. Зайков В. П., Прозоров П. Д. Разработка мобильного приложения учебного расписания занятий студента // Информационные технологии и системы. — 2023. — № 5. — С. 12–19.
27. Зайцева Н. А. Научно-практические аспекты применения тайм-менеджмента для повышения профессиональной конкурентоспособности выпускников // Российские регионы: взгляд в будущее. — 2016. — С. 320–330.
28. Закопайлов М. О. Механизмы вовлечения пользователей в мобильные игры // Universum: технические науки. — 2025. — № 6 (135). — С. 44–50.
29. Зубрев А. В. Таск-менеджеры, таск-трекеры, сервисы и инструменты управления ИТ-проектами // Дискуссия. — 2025. — № 1 (134). — С. 40–46.
30. Ибрагимова О. Ю. Геймификация в маркетинговых коммуникациях: возможности и ограничения // Практический маркетинг. — 2025. — № 1. — С. 33–41.
31. Иванова Е. А. Регулирование когнитивной нагрузки при обучении иностранному языку в технологическом вузе // Преподаватель XXI век. — 2025. — № 1. — С. 83–94.
32. Илышева М. А., Детков А. А., Щербаков И. В. Использование гибридного подхода в управлении проектами на рынке интернет-маркетинга // Индустриальная экономика. — 2025. — № 6. — С. 22–28.
33. Искандарян Г. О., Варфоломеева Е. А., Панова Я. М. Современные методы тайм-менеджмента: теоретические основания и практическая эффективность в условиях когнитивной нагрузки // Инновационная экономика: перспективы развития и совершенствования. — 2025. — № 4 (86). — С. 55–60.
34. Калиневич Н., Гильванов Р. Г. Разработка кросс-платформенных приложений на языке Dart при помощи фреймворка Flutter // Интеллектуальные технологии на транспорте. — 2021. — № 4 (28). — С. 21–27.

35. Карпенко М. Н., Бархатова И. А. Применение системного анализа для выявления и устранения информационных перегрузок у студентов // Системный анализ в науке и образовании. — 2025. — № 3. — С. 113–123.
36. Каширин Р. С. Сравнительный анализ нативного и кроссплатформенного подходов к разработке мобильных приложений // Известия Тульского государственного университета. Технические науки. — 2024. — № 10. — С. 686–687.
37. Кишкурно Т. В., Брусенцова Т. П. Использование принципов юзабилити для оптимизации процесса восприятия экранного пространства // Труды БГТУ. Серия 3: Физико-математические науки и информатика. — 2019. — № 2. — С. 89–94.
38. Кобозева Е. М., Сугаева В. В. Система Канбан: внедрение и применение на производстве // Инновационная экономика: перспективы развития и совершенствования. — 2024. — № 1. — С. 104–109.
39. Колчанова С. А. Сравнение фреймворков Flutter и React Native, используемых в разработке гибридных приложений // E-Scio. — 2022. — № 4 (67). — С. 558–565.
40. Комарова Л. И. Психологические последствия информационной перегрузки: синдром усталости от новостей в 2025 году // Вестник науки. — 2026. — № 1. — С. 1081–1087.
41. Кондракова О. Ю., Мерзляков И. Н. Психологические принципы в UX-проектировании // Труды НГТУ им. Р.Е. Алексеева. — 2021. — № 3. — С. 110–115.
42. Конечным интерфейсом для пользователя является веб-портал: Аналитические панели и системы показателей // Информационные технологии в бизнесе. — 2020. — С. 88–95.
43. Лаврова А. П. Теоретические основы применения эффективных техник в современном тайм-менеджменте // Гуманитарные науки. — 2024. — № 2. — С. 54–59.

44. AbuSalim S. W. G., Ibrahim R., Wahab J. A. Comparative Analysis of Software Testing Techniques for Mobile Applications // Journal of Physics: Conference Series. — 2021. — Vol. 1793, No. 1. — P. 012036.
45. Alamleh H. [et al.]. Offline-First Mobile Architecture: Enhancing Usability and Resilience in Mobile Systems // Journal of Artificial Intelligence General Science (JAIGS). — 2023. — Vol. 7, No. 1. — P. 320–326.
46. Alves P., Cipriano B. P. Automated Assessment in Mobile Programming Courses: Leveraging GitHub Classroom and Flutter for Enhanced Student Outcomes // arXiv preprint arXiv:2504.04230. — 2025.
47. Asadzandi M. [et al.]. Rethinking Gamification Failure: A Model and Investigation of Gamified System Maladaptive Behaviors // Information Systems Research. — 2021. — Vol. 32. — P. 1–21.
48. Axmadjonov M. F., Mirzaraximov M. A. Firebase in real-time systems based on client-server technology // Scientific Journal Impact Factor. — 2020. — P. 20–26.
49. Azizyan L. Development of cross-platform mobile applications: a comparison of Flutter, React Native, and Kotlin Multiplatform // Вестник науки. — 2025. — № 6 (87). — С. 1297–1308.
50. Data Synchronization Techniques in Offline-First Android Applications // International Journal of Science and Research (IJSR). — 2024. — Vol. 7, No. 3. — P. 44–49.
51. Development of a Large-Scale Flutter App. — Milano: Politecnico di Milano, 2026. — 120 p.
52. Development of a mobile application for covering local events and incidents // International Journal of Applied Sciences. — 2023. — P. 14–21.
53. Implementation of Offline-First Architectures for Android Internet-Based Chat Systems // All Multidisciplinary Journal. — 2025. — Vol. 3. — P. 25–32.