

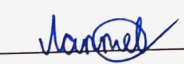

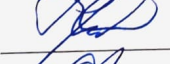






МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «ВГУ»)

Факультет Компьютерных наук
Кафедра программирования и информационных технологий

Курсовой проект
Разработка мобильного приложения для самоорганизации
«Taskbench»

Зав. кафедрой		д.ф.-м.н., профессор С. Д. Махортов
Обучающийся		ст. 3 курса оч. отд. А. М. Косенко
Обучающийся		ст. 3 курса оч. отд. А. М. Лаптев
Обучающийся		ст. 3 курса оч. отд. Е. Е. Мальныхина
Обучающийся		ст. 3 курса оч. отд. Е. В. Саломатова
Обучающийся		ст. 3 курса оч. отд. Д. Д. Шульга
Обучающийся		ст. 3 курса оч. отд. Н. О. Шульга
Руководитель		Ю. В. Шишко, преподаватель
Руководитель		В.С. Тарасов, ст. преподаватель

Воронеж 2025

Содержание

Определения, обозначения и сокращения	4
Введение	6
1 Постановка задачи.....	7
1.1 Назначение приложения	7
1.2 Цели и задачи работы.....	7
2 Анализ предметной области	9
2.1 Анализ конкурентов	9
2.2 Выявление сильных сторон своего продукта	9
2.3 Анализ целевой аудитории	10
2.3.1 Ключевые группы пользователей.....	10
2.4 Проблемы пользователей и их решения	11
2.5 Финансовые прогнозы	12
3 Архитектура системы.....	14
3.1 Обзор общих функциональных характеристик приложения.....	14
3.2 Сценарии действий и возможности пользователей.....	14
3.3 Диаграмма классов.....	16
3.4 Диаграмма компонентов	17
3.5 Диаграмма коммуникаций	17
4 Реализация системы	19
4.1 Средства реализации	19
4.1.1 Сервер и логика	19
4.1.2 Мобильный клиент.....	20
4.1.3 База данных	21
4.1.4 Развертывание серверной части	21
4.1.5 Искусственный интеллект	21
4.1.6 Система контроля версий и средства тестирования.....	22
4.2 Обеспечение безопасности	22
4.3 Реализация Backend-разработки.....	23
4.4 Реализация админ-панели	24
4.5 Работа с GigaChat	25
4.5.1 Разбиение на подзадачи	25
4.5.2 Определение категории.....	26
4.5.3 Выбор срока выполнения.....	26

4.6 Реализация системы оплаты	26
4.7 Реализация клиентской части	27
4.7.1 Уровень UI.....	27
4.7.2 Уровень Domain	29
4.7.3 Уровень Data.....	29
4.8 Реализация аналитики и сбор данных о поведении пользователей.....	30
4.9 Доработка по результатам тестирования и обратной связи.....	33
5 Анализ соответствия поставленным требованиям	34
5.1 Базовая функциональность	34
5.2 Пользовательский опыт	34
5.3 Безопасность данных	34
6 Перспективы развития приложения	36
6.1 Исследование расширения возможностей приложения.....	36
6.2 Оценка монетизации приложения	36
Заключение	38
Список использованных источников	39
ПРИЛОЖЕНИЕ А	40
ПРИЛОЖЕНИЕ Б.....	41
ПРИЛОЖЕНИЕ В	43
ПРИЛОЖЕНИЕ Г	45
ПРИЛОЖЕНИЕ Д	46

Определения, обозначения и сокращения

В настоящем отчете о проекте применяют следующие термины с соответствующими определениями.

- **Python** — высокоуровневый язык программирования общего назначения с динамической строгой типизацией и автоматическим управлением памятью.
- **Django** — фреймворк на языке программирования Python, который позволяет быстро создавать безопасные веб-приложения.
- **Compose UI** — декларативный фреймворк на языке программирования Kotlin для графического интерфейса под Android-устройства с возможностью расширения под разные платформы.
- **GitHub** - крупнейший веб-сервис для хостинга IT-проектов и их совместной разработки.
- **REST API** — стиль архитектуры программного обеспечения для построения распределённых масштабируемых веб-сервисов.
- **PostgreSQL** — реляционная база данных с открытым исходным кодом с большими возможностями для масштабирования.
- **Клиентская сторона** — программно-аппаратная часть веб-приложения, работающая на устройстве пользователя. Отвечает за отображение интерфейса, обработку ввода и взаимодействие с сервером.
- **Сервер, серверная сторона** — компьютер, обслуживающий другие компьютеры (клиентов) и предоставляющий им свои ресурсы для выполнения определенных задач.
- **Фреймворк** — программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта.

- **Getting Things Done** — методика повышения личной эффективности посредством перенесения списка текущих задач на внешний носитель, созданная Дэвидом Алленом.
- **JSON Web Token** — зашифрованный контейнер для хранения пользовательских данных на клиентской стороне. Используется для аутентификации.
- **HTTPS** — защищённый протокол передачи гипертекста, использующий TLS/SSL для шифрования передаваемых данных между клиентом и сервером. Обеспечивает конфиденциальность и целостность данных.
- **TLS** — криптографический протокол, обеспечивающий защищённую передачу данных по сети. Используется совместно с протоколами, такими как HTTPS, для шифрования трафика и проверки подлинности сторон соединения.

Введение

Современные цифровые технологии всё активнее внедряются во все сферы жизни человека, включая организацию личного времени, управление задачами и повышение личной продуктивности. В условиях растущего объема информации, множества параллельно выполняемых дел и необходимости четкого планирования возрастает потребность в эффективных инструментах самоорганизации. Одним из наиболее удобных и доступных решений в данной области являются мобильные приложения.

Мобильные приложения стали неотъемлемой частью повседневной жизни миллионов пользователей. Благодаря удобству, доступности и широкому функционалу они позволяют решать различные задачи — от банальных напоминаний до комплексного планирования рабочих процессов. В условиях высокой конкуренции на рынке приложений важно создавать не только функциональные, но и интуитивно понятные и адаптивные решения, учитывающие реальные потребности пользователей.

Целью данной курсовой работы является разработка мобильного приложения «Taskbench», основная задача которого — помочь пользователю в эффективной самоорганизации, планировании задач и отслеживании продуктивности. Приложение разработано для упрощения ввода задач, их автоматическую классификацию, а также использование технологий искусственного интеллекта для улучшения пользовательского опыта.

Актуальность разработки обусловлена потребностью в персонализированных и интеллектуальных решениях в сфере самоорганизации. Результатом выполнения работы станет полностью функционирующее мобильное приложение, сопровождаемое соответствующей технической документацией. Курсовая работа охватывает полный цикл проектирования, разработки и тестирования программного продукта.

1 Постановка задачи

1.1 Назначение приложения

Приложение предназначено для автоматизации и упрощения процесса самоорганизации пользователя. Чтобы улучшить пользовательский опыт предлагается провести следующие действия:

- Ускорить и упростить ввод и создание новой задачи.
- Автоматически классифицировать создаваемые задачи в общем списке.

1.2 Цели и задачи работы

Цели создания приложения:

- Внедрение технологии искусственного интеллекта в систему обработки ввода пользователя.
- Создание базы данных для хранения пользовательских записей для доступа к ним с различных устройств.
- Предоставление возможности анализа пользовательской продуктивности с выводом статистических данных пользователю.

Для достижения этой цели были поставлены следующие задачи:

- Исследование существующего рынка мобильных приложений для самоорганизации и анализ конкурентов.
- Определение функциональных требований и технических спецификаций на основе потребностей целевой аудитории.
- Проектирование архитектуры и интерфейса пользователя, обеспечивающих удобство использования и эстетическую привлекательность.
- Реализация основных функций приложения, таких как регистрация, добавление и редактирование задач, их просмотр и возможность разбиения на категории.

- Обеспечение безопасности пользовательских данных и защита от несанкционированного доступа.

2 Анализ предметной области

2.1 Анализ конкурентов

Для выяснения преимуществ разрабатываемого приложения перед конкурентами и выделения его ключевых особенностей были проведены сравнение и анализ конкурентов.

В сравнительном анализе участвовали следующие конкуренты разрабатываемого программного продукта:

- Things («Cultured Code GmbH & Co. KG»).
- Notion («Notion Labs, Inc.»).
- UseMotion («Nexusbird, Inc.»).
- Routine («Routine SAS»).

Ниже в таблице 1 приведены результаты сравнительного анализа этих конкурентов.

	Things	Notion	UseMotion	Routine	Taskbench
Наличие категорий задач	✓	✓	✗	✓	✓
Разбиение на подзадачи	✗	✓	✗	✗	✓
ИИ-подсказки	✗	✗	✓	✗	✓
Кроссплатформенность	✗	✓	✓	✓	✗
Доступность в РФ	✗	✗	✗	✗	✓

Таблица 1 - Сравнение конкурентов.

Таким образом, можно заметить, что ни один конкурентный продукт не удовлетворяет всем необходимым требованиям.

2.2 Выявление сильных сторон своего продукта

В рамках анализа у разрабатываемого продукта были выделены следующие сильные стороны:

- Использование большой языковой модели (нейросети) для автоматического выделения у задачи подзадач, определения её сроков и категории.
- Облачная синхронизация задач пользователя между различными устройствами с использованием Интернета.
- Чрезвычайное упрощение процесса создания задачи, а также добавления к ней подзадач.

2.3 Анализ целевой аудитории

Для определения круга пользователей разрабатываемого программного продукта был проведён анализ целевой аудитории.

Разрабатываемое приложение ориентировано на пользователей, стремящихся эффективно организовывать свои задачи, автоматизировать повседневные процессы и получать персонализированные рекомендации с помощью технологий искусственного интеллекта.

2.3.1 Ключевые группы пользователей

Было выделено две ключевые группы пользователей:

- *Студенты и учащиеся до 25 лет.* Данной категории пользователей необходимо каким-то образом совмещать учёбу, проекты и личные дела, что представляет определённые трудности. Таким людям было бы удобно автоматически разбивать крупные задачи (например, курсовые работы) на более мелкие и соответственно менее трудновыполнимые задачи, а также автоматически определять дедлайны таких задач.
- *Работающие люди 26–45 лет.* К данной категории пользователей относятся офисные сотрудники, менеджеры, предприниматели и фрилансеры, ежедневно сталкивающиеся с большим объёмом задач. Данной категории пользователей важна возможность структурирования рабочего процесса, а также помощь в избегании прокрастинации. Пользователи этой категории более платёжеспособны и в среднем лучше

готовы платить за дополнительную функциональные возможности приложения.

Так, на момент анализ аудитории было выделено две основные категории пользователей, однако уже после окончания работы над реализацией проекта оказалось, что приложение может быть востребовано и среди пользователей старше 45 лет. Благодаря интуитивно понятному интерфейсу и простой логике работы, оно подходит даже для тех, кто не имеет большого опыта в использовании цифровых технологий.

Например, пожилые пользователи, такие как бабушки, могут использовать приложение для планирования домашних дел или помощи с рецептами: удобно записывать рецепты или находить новые, структурировать их по категориям (завтраки, обеды, праздники), добавлять напоминания о покупке ингредиентов.

2.4 Проблемы пользователей и их решения

В результате анализа было выделено две основные проблемы, испытываемые ключевыми категориями пользователей:

- *Перегруженность задачами.* В связи с высокой загруженностью, значительно возрастают затраты времени на планирование задач. Также возрастает роль человеческого фактора, приводящего к забыванию важных дел.
- *Плохая видимость общего прогресса выполнения задачи.* Отсутствие чувства прогресса над задачами приводит к снижению мотивации пользователей, даже если в действительности они делают много работы.

Для решения этих проблем были предложены следующие решения:

- Использовать функции искусственного интеллекта в приложении для быстрого создания более мелких подзадач, которые проще выполнять.
- Визуально отображать подзадачи на карточке задачи в виде галочек (чекбоксов). Таким образом, пользователи смогут быстро визуальное оценить прогресс по конкретной задаче.

- Добавить экран с мотивационной статистикой с отображением выполненных за сегодня задач, рекорда по количеству выполненных за день задач, а также различных графиков и диаграмм.

2.5 Финансовые прогнозы

К основным статьям расходов относятся:

- Разработка (backend, мобильное приложение, тестирование) — 900 000 рублей.
- Серверные затраты — 40 000 рублей в месяц.
- Маркетинг (привлечение пользователей) — 30 000 рублей в месяц.
- Операционные затраты (поддержка, налоги, комиссии) — 20 000 рублей в месяц.

Ориентировочный ежемесячный доход через год:

- Ожидаемое количество активных пользователей через год: 10 000 человек.
- Конверсия в подписку — 10% пользователей.
- Средний доход с подписчика — 149 руб./мес.
- Итоговый ежемесячный доход через год: $149 \times 1000 = 149\,000$ рублей.

Расчёт RoI (Return on Investment) на три года:

- Инвестиции в разработку и маркетинг: $(20\,000 + 30\,000 + 40\,000) \times 12 \times 3 + 900\,000 = 4\,140\,000$ рублей.
- Прогнозируемая выручка за 3 года: $149 \times 2000 \times 12 \times 3 = 10\,728\,000$ рублей (при увеличении базы пользователей до 20 000 подписчиков).
- Прибыль: $10\,728\,000 - 4\,140\,000 = 6\,588\,000$ рублей.
- $RoI = 6\,588\,000 \text{ руб.} / 4\,140\,000 \text{ руб.} = 1.59$ (выгодная модель выше необходимого уровня 1,05).

Таким образом, разрабатываемое приложение сможет выйти в прибыльность в течение второго года работы. Для улучшения показателей потребуется повышать конверсию пользователей в подписчиков, а также оптимизировать маркетинговые расходы.

3 Архитектура системы

Разработка мобильного приложения требует не только реализации пользовательских функций, но и грамотного проектирования архитектуры. Именно архитектура определяет, насколько удобно будет развивать, масштабировать и поддерживать продукт в будущем. На этом этапе важно определить ключевые компоненты системы, способы взаимодействия между ними, а также выбрать подходящие технологии и шаблоны проектирования. Хорошо продуманная архитектура обеспечивает надёжность, гибкость и устойчивость приложения к изменяющимся требованиям.

3.1 Обзор общих функциональных характеристик приложения

Основные возможности, предоставляемые приложением:

- **Регистрация и аутентификация:** позволяет пользователям создавать учетные записи и входить в систему для доступа к функциям.
- **Добавление и управление задачами:** пользователи могут добавлять свои задачи, включая текстовые описания и разбиение на подзадачи, а также редактировать и удалять их.
- **Поиск и фильтрация:** возможность поиска задач по различным параметрам, таким как дата или категории.
- **Интерактивные функции:** возможность отмечать задачи выполненными, просматривать свою статистику по их выполнению.
- **Личный профиль:** просмотр и редактирование личной информации, управление своей подпиской.

3.2 Сценарии действий и возможности пользователей

Для более полного понимания взаимодействия пользователя с системой целесообразно рассмотреть основные варианты использования (UseCase). Они описывают, как пользователь будет взаимодействовать с приложением в типичных сценариях, включая действия, цели и ожидаемый результат.

Приложение подразумевает наличие следующих видов пользователей: неавторизованный пользователь, авторизованный пользователь, премиум пользователь и администратор.

На рисунке 1 представлены сценарии использования для неавторизованного, авторизованного и премиум пользователей.

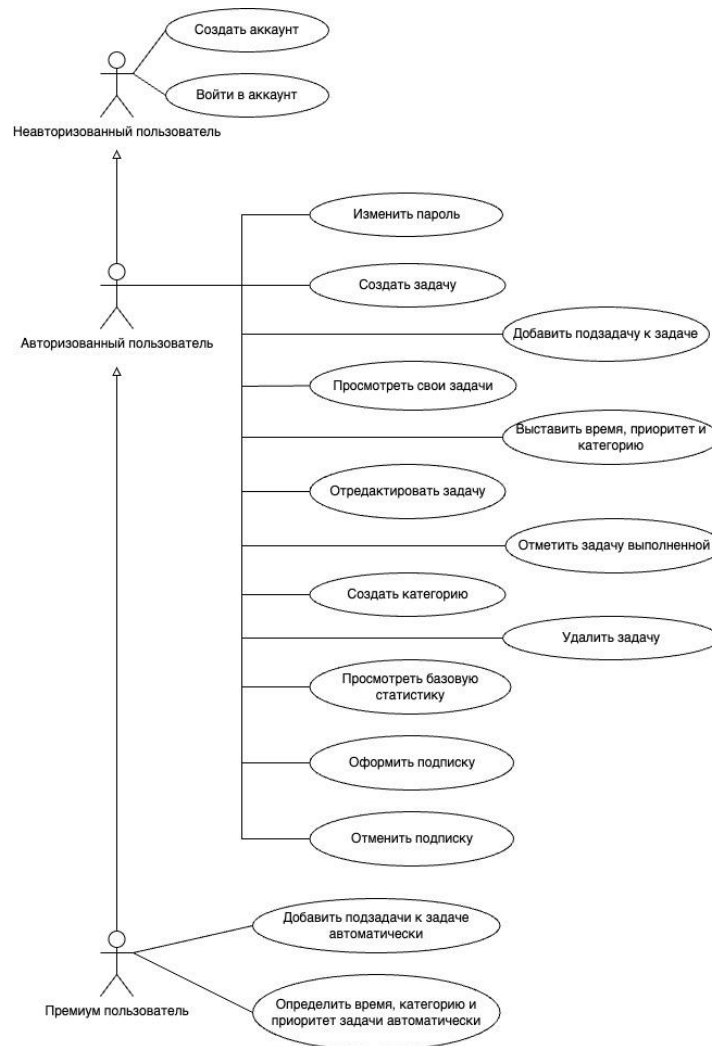


Рисунок 1 - Диаграмма прецедентов. Неавторизованный, авторизованный и премиум пользователи.

На рисунке 2 отображены возможности администратора.

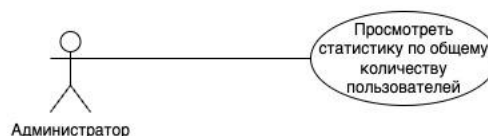


Рисунок 2 - Диаграмма прецедентов. Администратор.

3.3 Диаграмма классов

Для проектирования структуры базы данных приложения была разработана ER-диаграмма (диаграмма «сущность-связь»), отражающая основные таблицы, их атрибуты, типы данных и логические связи между ними. В приложении реализована реляционная база данных, в которой выделяются следующие ключевые сущности: пользователь, задача, категория, подзадача и статистика.

Связи между сущностями, их атрибуты и зависимости представлены на рисунке 3.

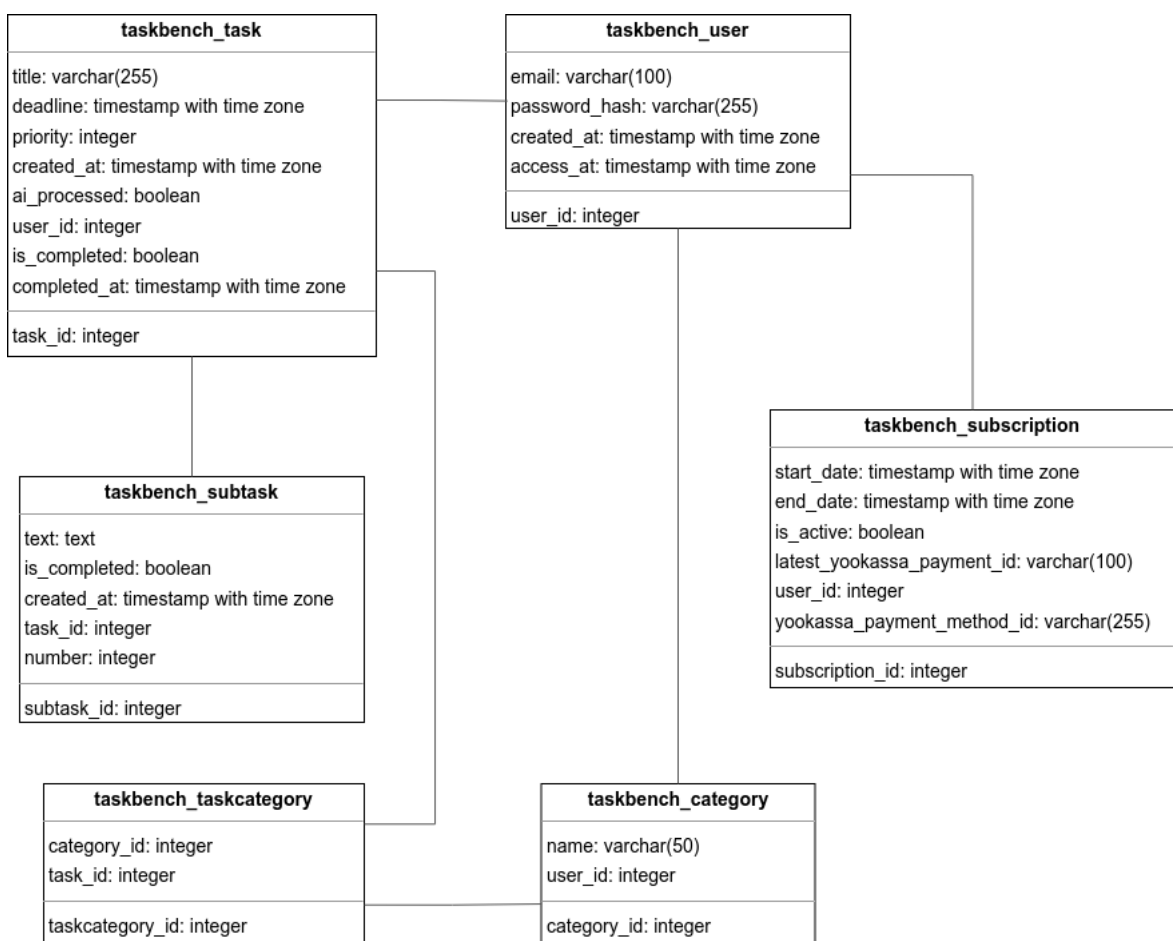


Рисунок 3 - ER-диаграмма для базового функционала.

3.4 Диаграмма компонентов

Для визуализации структуры приложения была разработана диаграмма компонентов, отражающая основные программные и аппаратные модули системы, а также связи между ними. Диаграмма позволяет наглядно представить, из каких частей состоит приложение, как оно взаимодействует с сервером, базой данных, внешними API и устройствами пользователя. Она представлена на рисунке 4.

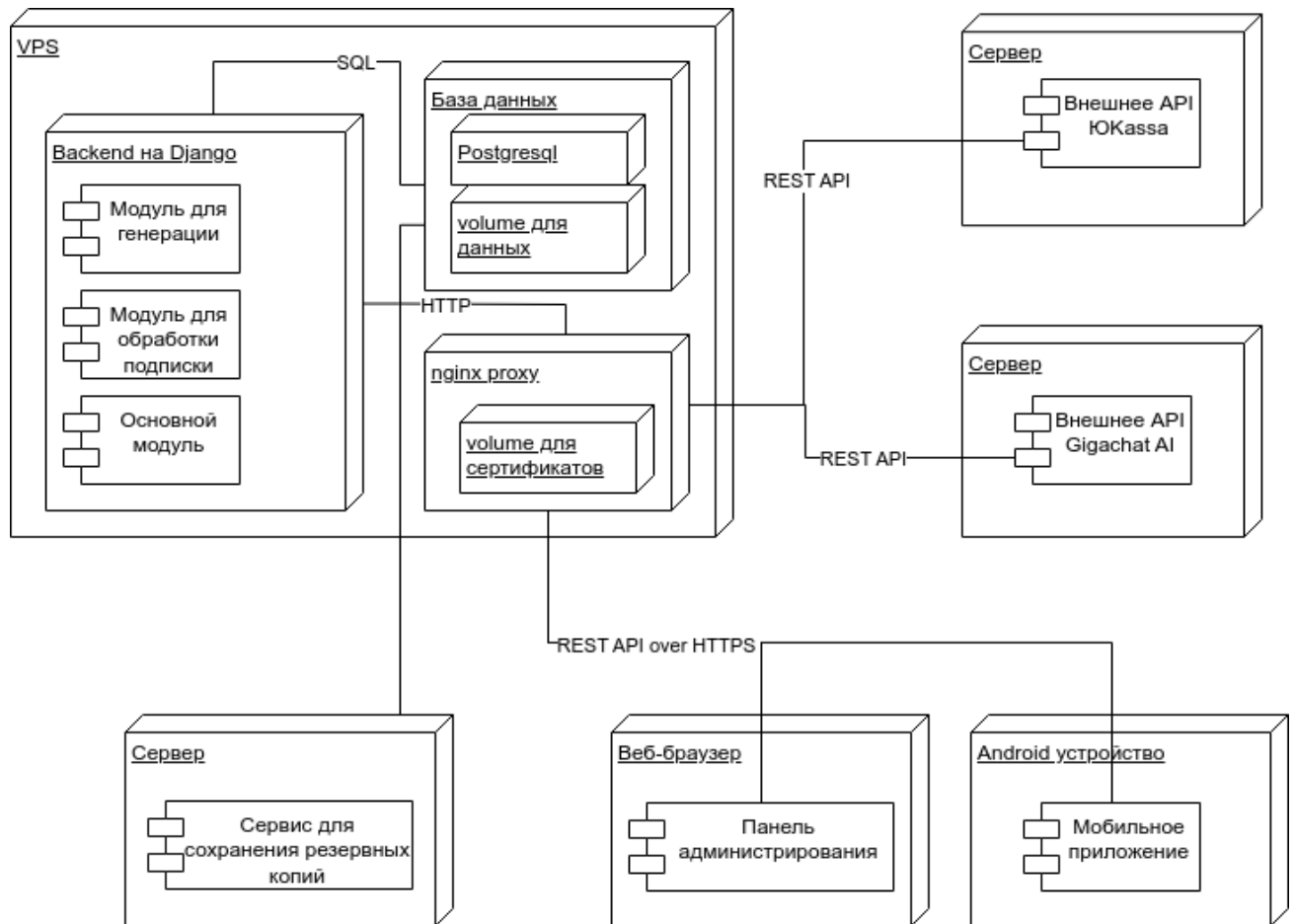


Рисунок 4 - Диаграмма разворачивания.

3.5 Диаграмма коммуникаций

Для наглядного представления взаимодействия между компонентами системы разработана диаграмма коммуникаций. Она демонстрирует, как происходит обмен данными между мобильным приложением, серверной частью, базой данных и внешними сервисами. Диаграмма позволяет проследить логическую последовательность передачи данных, включая их обработку, взаимодействие с внешними сервисами и отправку пользователю.

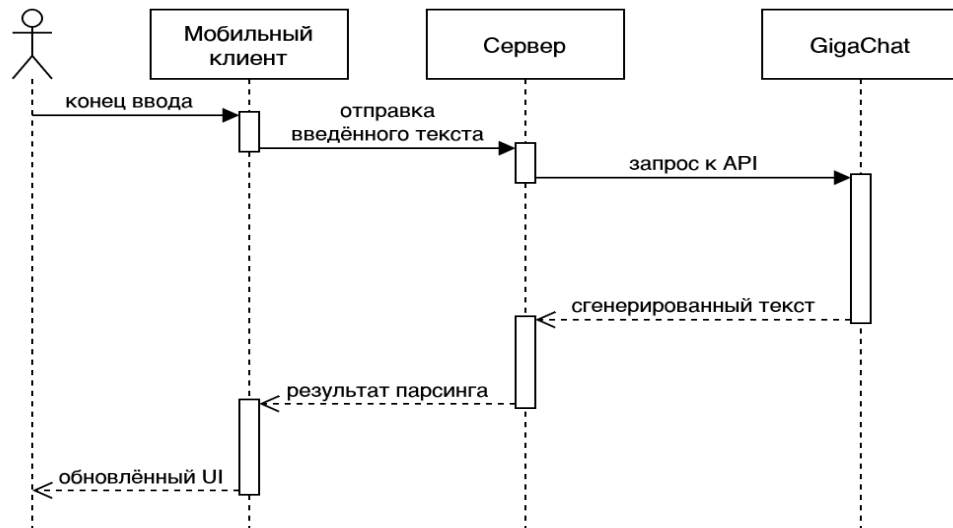


Рисунок 5 - Диаграмма последовательности для реализации ввода задачи.

4 Реализация системы

В системе можно выделить несколько подсистем:

- Мобильное приложение.
- Панель администрирования.
- Серверная часть.
- База данных.
- Сервис искусственного интеллекта.

Пользователь взаимодействует с системой через мобильное приложение, которое связано с серверной частью посредством REST API. Сервер обрабатывает входящие запросы мобильного приложения, используя фреймворк Django. В качестве базы данных используется PostgreSQL. Сервер связывается с внешним сервисом искусственного интеллекта и внешним сервисом оплаты посредством публичного REST API и при необходимости запрашивает обработку данных.

4.1 Средства реализации

4.1.1 Сервер и логика

Для реализации серверной части проекта был выбран фреймворк Django, написанный на языке программирования Python. Django предоставляет удобную архитектуру для построения работы приложения, обладает встроенной системой маршрутизации, ORM для работы с базой данных и средствами аутентификации, что делает его отличным выбором для разработки надёжного и масштабируемого backend-приложения.

Причины, почему был выбран именно Django:

- Модульность и широкий набор готовых решений.
- В фреймворке Django уже реализовано много нюансов для работы с базой данных. Например, защита от SQL-инъекций.
- Обширная и хорошая документация: Django хорошо документирован, поддерживается сообществом.

- Имеет гибкую структуру, что упрощает расширение функциональности.
- Поддерживает собственную панель администрирования: в Django можно развернуть его встроенную админ-панель в web-браузере и настроить её под себя и свои цели.
- Язык Python, на котором работает Django, читаем и лаконичен, что облегчает поддержку проекта и ускоряет написание серверной логики. Синтаксис Python делает код понятным даже для начинающих разработчиков. Это ускоряет разработку и снижает количество ошибок.

Также часть прикладной логики, связанной с обработкой и анализом данных, будет реализована на Python внутри проекта Django. Такой подход обеспечивает единый стек технологий на серверной стороне и удобную интеграцию всех компонентов приложения.

4.1.2 Мобильный клиент

Для создания мобильного клиента приложения «Taskbench» использовался язык Kotlin, фреймворки Jetpack Compose, Koin и библиотека Retrofit.

Kotlin — это современный язык программирования, официально поддерживаемый Google для разработки Android-приложений. Он прост в использовании, безопасен (например, защищает от null-ошибок) и позволяет писать меньше кода.

Jetpack Compose — это инструмент для создания интерфейсов без XML. Всё рисуется прямо в коде с помощью функций. Это позволило:

- быстро создавать экраны приложения;
- легко обновлять интерфейс при изменении данных (например, списка задач);
- переиспользовать компоненты (например, карточки задач и категории);
- делать адаптивный дизайн под разные размеры экранов.

Koin - фреймворк для инъекции зависимостей. Использование такого подхода позволяет делать код слабосвязным и сделать архитектуру более

расширяемой.

Retrofit - библиотека для HTTP клиента. С помощью Retrofit осуществляется взаимодействие с серверной частью.

4.1.3 База данных

В качестве системы управления базами данных в проекте используется PostgreSQL.

Выбор PostgreSQL обусловлен её популярностью, удобством работы и наличием достаточного большого личного опыта работы с этим инструментом, что позволяет быстро и уверенно проектировать структуру данных, настраивать связи и оптимизировать запросы.

Система будет использовать PostgreSQL совместно с Django ORM, что обеспечит надёжное и удобное взаимодействие между серверной логикой и хранилищем данных.

4.1.4 Развертывание серверной части

Для упрощения развертывания серверной части приложения было решено использовать контейнеризацию. Приложение и все необходимое окружение развернуто как Docker-контейнер. Инструмент Docker был выбран для проекта как один из самых популярных и широко используемых решений для контейнеризации.

Для обеспечения шифрования трафика используется HTTP сервер nginx в качестве прокси. Такой выбор инструмента обусловлен высокой производительностью, хорошей документацией и легкой настройкой через конфигурационные файлы.

4.1.5 Искусственный интеллект

Для реализации функции разбиения задач на подзадачи в приложении «Taskbench» используется искусственный интеллект GigaChat, разработанный ПАО «Сбербанк».

Выбор GigaChat обусловлен следующими преимуществами:

- Поддержка русского языка на высоком уровне, включая понимание контекста, команд и формулировок, характерных для повседневной речи;

- Удобное REST API, позволяющее легко интегрировать ИИ-сервис в серверную часть приложения;
- Стабильная работа;
- Возможность масштабирования модели.

4.1.6 Система контроля версий и средства тестирования

Для совместной работы над проектом будет использоваться система контроля версий Git, чтобы все разработчики могли вносить свои изменения и добавлять коммиты в общий репозиторий проекта.

В рамках проекта «Taskbench» мы будем использовать подход CI/CD (Continuous Integration / Continuous Delivery) для автоматизации процесса сборки и тестирования мобильного приложения. Это позволяет ускорить разработку, сократить количество ошибок и обеспечить стабильность продукта на всех этапах.

На этапе CI при создании pull request запускаются проверки корректности кода: автоматически прогоняются тесты и осуществляется валидация изменений. Это позволяет заранее выявлять ошибки и конфликты до объединения кода в основную ветку.

На CD происходит автоматическая сборка Docker-образа backend-приложения и его доставка на удалённый сервер. Развёртывание происходит без участия пользователя, что позволяет быстро доставлять обновления и замечать баги. Такой подход сокращает время между внесением изменений и их появлением в рабочем окружении.

4.2 Обеспечение безопасности

Система должна обеспечивать безопасность данных. Для этого реализуются следующие меры:

- Защита от SQL-инъекций посредством встроенных во фреймворк Django инструментов для безопасного взаимодействия с базой данных.

- Передача данных между клиентской и серверной частью осуществляется через защищённый протокол HTTPS с обязательным использованием TLS (версии не ниже 1.2).
- Все пароли пользователей при хранении в базе данных хешируются.
- Аутентификация пользователей происходит с помощью JSON Web Token. Время жизни access-токена — 2 часа, время жизни refresh-токена — 14 дней.
- В систему должно быть введено регулярное автоматическое резервное копирование базы данных.

4.3 Реализация Backend-разработки

Проект приложения разделен на несколько модулей, отвечающих за свои задачи:

- Основной модуль с логикой пользователей, задач и статистики.
- Модуль для работы с искусственным интеллектом и генерацией умных предложений.
- Модуль для взаимодействия с системой оплаты и обработкой пользовательских подписок.
- Модуль для панели администрирования.
- Архитектура с разделением логики по модулям обеспечивает легкий перевод на микросервисную архитектуру при необходимости в будущем.

В каждом модуле есть несколько слоев:

- контроллеры для обработки HTTP запросов;
- сериализаторы для получения данных из запросов и обработку данных перед отправкой ответа;
- сервисы, в которых выполняется бизнес-логика приложения.

Взаимодействие с базой данных осуществляется за счет моделей:

- User.
- Task.
- Subtask.
- Category.
- TaskCategory.
- Subscription.

Пример кода, описывающего модель профиля задачи представлен в приложении А.

Схема базы данных представлена в виде ER-диаграммы (см. Рисунок 2). Связи между сущностями организованы по принципу «один ко многим»: один пользователь — много задач, одна задача — много подзадач, одна категория — много задач.

4.4 Реализация админ-панели

Для администратора реализована веб-панель, разработанная на Django. Доступ к панели предоставляется только пользователям с правами администратора. После авторизации администратор может просматривать общую статистику использования приложения, а также список всех активных подписок пользователей. Интерфейс состоит из нескольких страниц: панель статистики и страница подписок. Используется встроенная система аутентификации Django для защиты маршрутов.

Ниже на рисунке 6 представлен интерфейс страницы со статистикой.

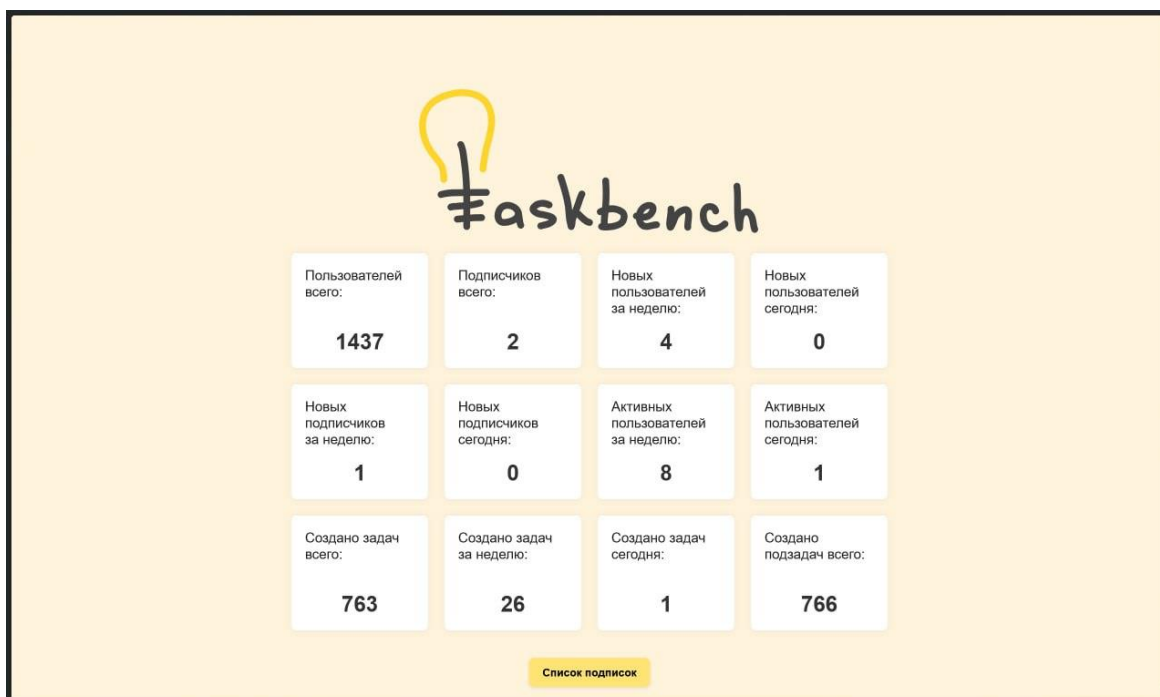


Рисунок 6 - Экран статистики в админ-панели.

4.5 Работа с GigaChat

В приложении «Taskbench» реализована функция интеллектуальной помощи при создании задач, доступная для пользователей с активной подпиской. Для этого используется языковая модель GigaChat, Взаимодействие с моделью осуществляется через бесплатный API. Наша работа с этим инструментом реализована так, что модель всегда выдаёт только информацию, отвечающую на поставленные вопросы.

Модель GigaChat применяется в трёх основных сценариях:

- Автоматическое разбиение задачи на подзадачи.
- Автоматический выбор категории.
- Предложение подходящего срока выполнения.

4.5.1 Разбиение на подзадачи

При вводе основной задачи приложение отправляет текст в GigaChat и получает список подзадач в текстовом виде, где каждая подзадача — на новой строке. Ответ модели дополнительно очищается с помощью регулярных выражений для удаления лишних символов, обозначений списков и других

ненужных элементов. Это обеспечивает корректную интеграцию результата в пользовательский интерфейс.

4.5.2 Определение категории

Для определения категории пользователем вводится текст задачи, а также передаётся список его существующих категорий. GigaChat выбирает наиболее подходящую из них и возвращает только название. Далее результат сверяется с базой данных пользователя и назначается соответствующая категория.

4.5.3 Выбор срока выполнения

Срок выполнения — наиболее сложный элемент обработки. В модель передаётся текущая дата и время пользователя, а также текст задачи. GigaChat должен определить дату и время исполнения, ориентируясь на будущее, и выдать их в формате ISO 8601 с точностью до минут. Если модель не справляется, подключается резервный механизм: библиотека `dateparser` на сервере, которая алгоритмически анализирует текст и извлекает предполагаемую дату.

Пример Python-функции, реализующей отправку запроса к GigaChat с системным промптом и пользовательским текстом представлен в [приложении Б](#). Также в [приложении В](#) можно увидеть эти используемые промты.

4.6 Реализация системы оплаты

В приложении «Taskbench» реализована интеграция с платёжной системой ЮKassa для оформления подписки на премиум-функции. На текущем этапе используется тестовая среда (песочница) ЮKassa, что позволяет безопасно проверять работу платёжной логики без реальных транзакций. Интеграция построена по REST-протоколу и включает генерацию платёжной ссылки, перенаправление пользователя на платёжную страницу, а также обработку ответа о статусе платежа. В дальнейшем, при переходе к реальному использованию, потребуется подключение официального кабинета и проведение сертификации, предусмотренной политикой ЮKassa.

4.7 Реализация клиентской части

В качестве клиентской части выступает мобильное приложение, написанное для версии Android 8.0 и выше.

Архитектура приложения состоит из трех слоев:

- Уровень "UI". Этот слой отвечает за отрисовку пользовательских данных с помощью фреймворка Jetpack Compose.
- Уровень "Domain". Этот слой отвечает за представление данных в приложении в виде моделей и за некоторые сценарии.
- Уровень "Data" отвечает за хранение данных и за сообщение с сервером.

Обмен данными между мобильным клиентом и сервером осуществляется по принципам архитектуры REST, в формате JSON. Аутентификация осуществляется через JWT, данные шифруются, пароли хэшируются средствами Django.

4.7.1 Уровень UI

Графический интерфейс создан в фирменном стиле. Для этого был разработан набор переиспользуемых компонентов пользовательского интерфейса, что добавило приложению единообразия, а также облегчило разработку и процесс приучения пользователя к системе.

Примеры экранов интерфейса можно увидеть на следующих рисунках:

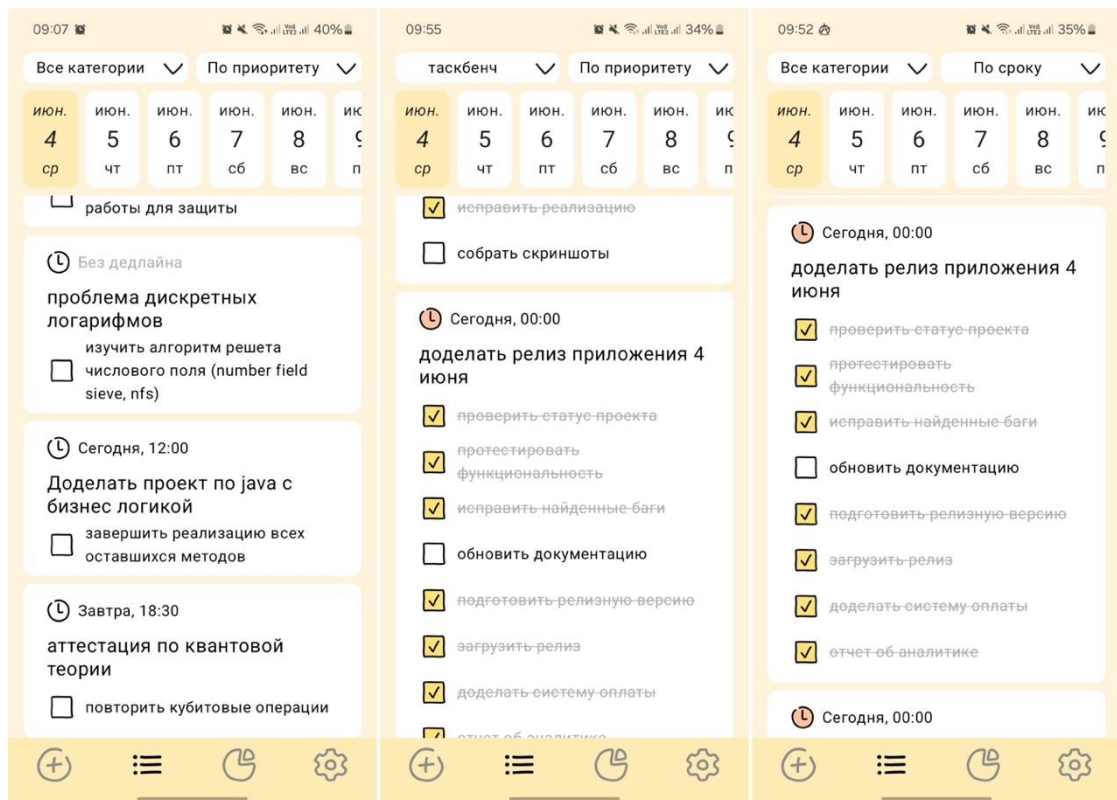


Рисунок 7 - Экран со списком добавленных задач и демонстрация сортировки и фильтрации по категории, приоритету или сроку.

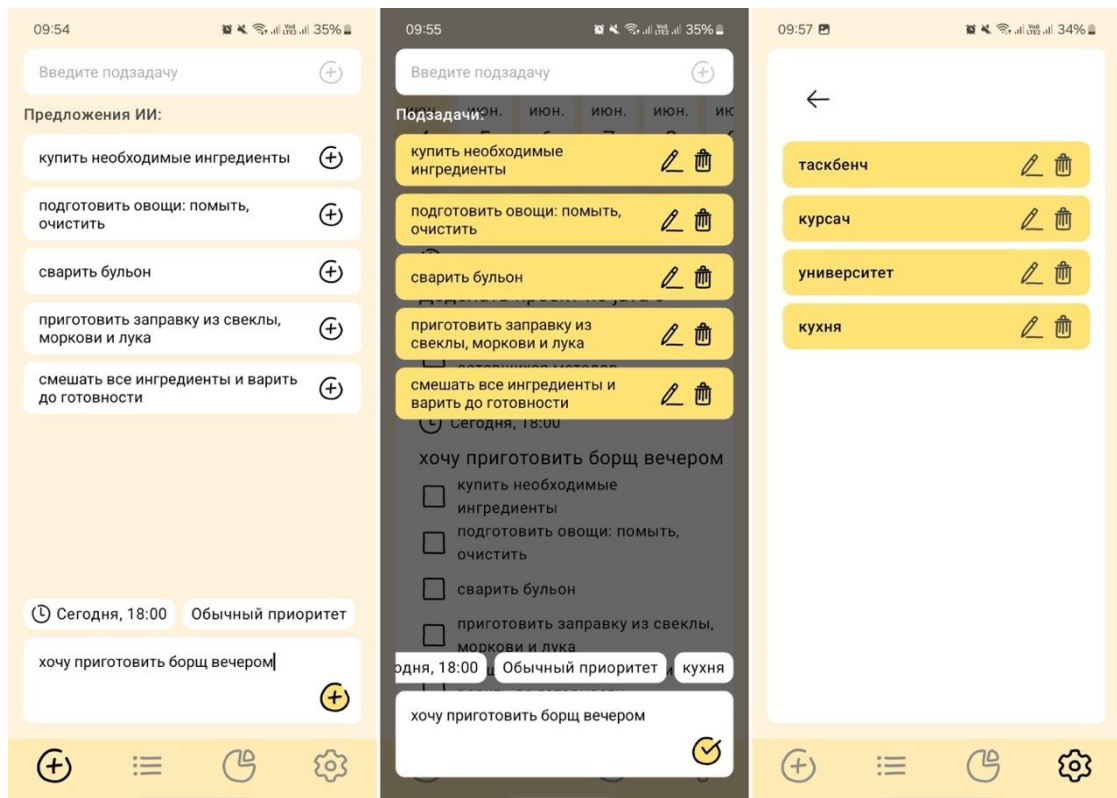


Рисунок 8 - Экран добавления задачи, возможность редактировать и добавлять подзадачи, а также экран категорий.

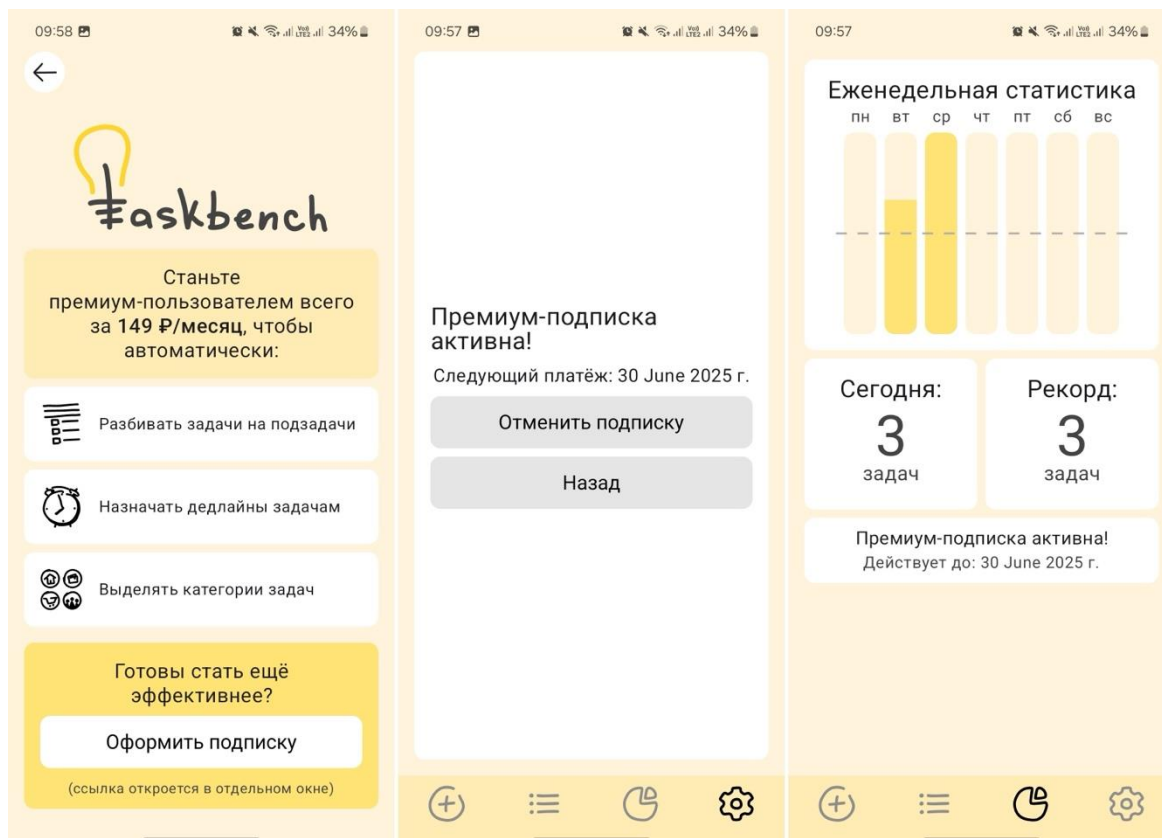


Рисунок 9 - Экран подписки и персональной статистики.

4.7.2 Уровень Domain

В приложении используются следующие модели:

- AiSuggestions.
- Category.
- Statistics.
- Task.
- UserStatus.

Пример кода, описывающего модель можно увидеть в [приложении Г](#).

4.7.3 Уровень Data

Для хранения, тестирования и передачи данных с сервером используются репозитории, выступающие в роли хранилища. Рассматривая на примере модели задач, за взаимодействие с backend-сервером отвечает класс `NetworkTaskRepository`, реализующий интерфейс `TaskRepository`. Он выполняет авторизованные HTTP-запросы к REST API сервера (например,

получение задач, добавление новых, редактирование и удаление), обрабатывает ответы и преобразует их в модели приложения.

Пример функции получения задач, в которой с помощью авторизации выполняется HTTP-запрос на сервер. Ответ преобразуется в локальные модели задач для отображения в UI. Пример кода, демонстрирующий выполнение описанных действий, представлен в [приложении Д](#).

4.8 Реализация аналитики и сбор данных о поведении пользователей

В нашем приложении реализована базовая система аналитики, позволяющая отслеживать действия пользователей и оценивать поведение в процессе взаимодействия с системой.

Анализ осуществляется по следующим ключевым событиям:

- Авторизация и регистрация пользователей.
- Просмотр и редактирование задач.
- Добавление задач и подзадач.
- Обновление приоритета задачи.
- Просмотр статистики.
- Использование интеллектуального помощника (GigaChat) для генерации подзадач.
- Использование фильтров и сортировок.
- Оформление и отмена подписки.

Целью внедрения аналитических событий является не только сбор статистики, но и повышение удобства приложения путём понимания того, какие функции используются чаще всего, где возникают ошибки и в каких местах пользователи покидают интерфейс. Это позволяет своевременно выявлять узкие места и принимать решения по улучшению UX.

В качестве системы аналитики использовался встроенный механизм логирования событий на клиентской стороне, с возможностью расширения для интеграции с внешними сервисами: в нашем случае это Yandex AppMetrica.

Все действия фиксируются через централизованные методы отслеживания, например:

```
Analytics.logEvent("task_created", mapOf("with_sub-  
tasks" to true))
```

Каждое событие сопровождается контекстной информацией — датой, временем, пользовательским идентификатором (в анонимизированной форме), типом действия и деталями (например, была ли задача создана с подзадачами или нет).

Таким образом, система аналитики в «Taskbench» играет важную роль в процессе поддержки и развития продукта, помогая сделать приложение более удобным и ориентированным на пользователя.

Ниже приведены примеры экранов с получением аналитики:

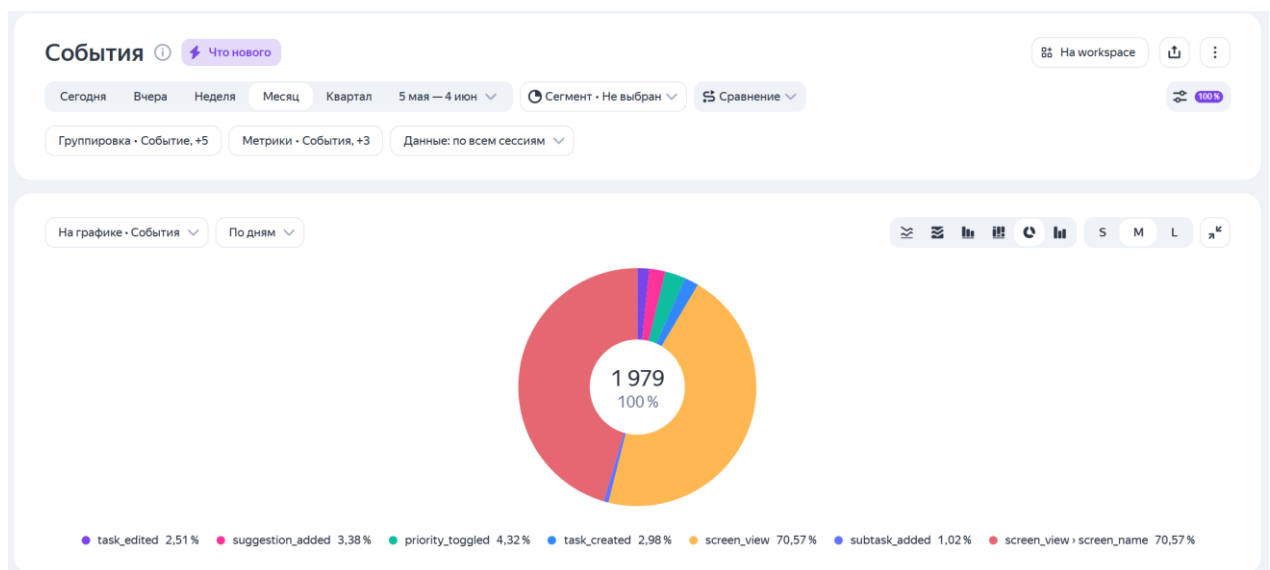


Рисунок 10 - Диаграмма, демонстрирующая срабатывание событий при использовании приложения.

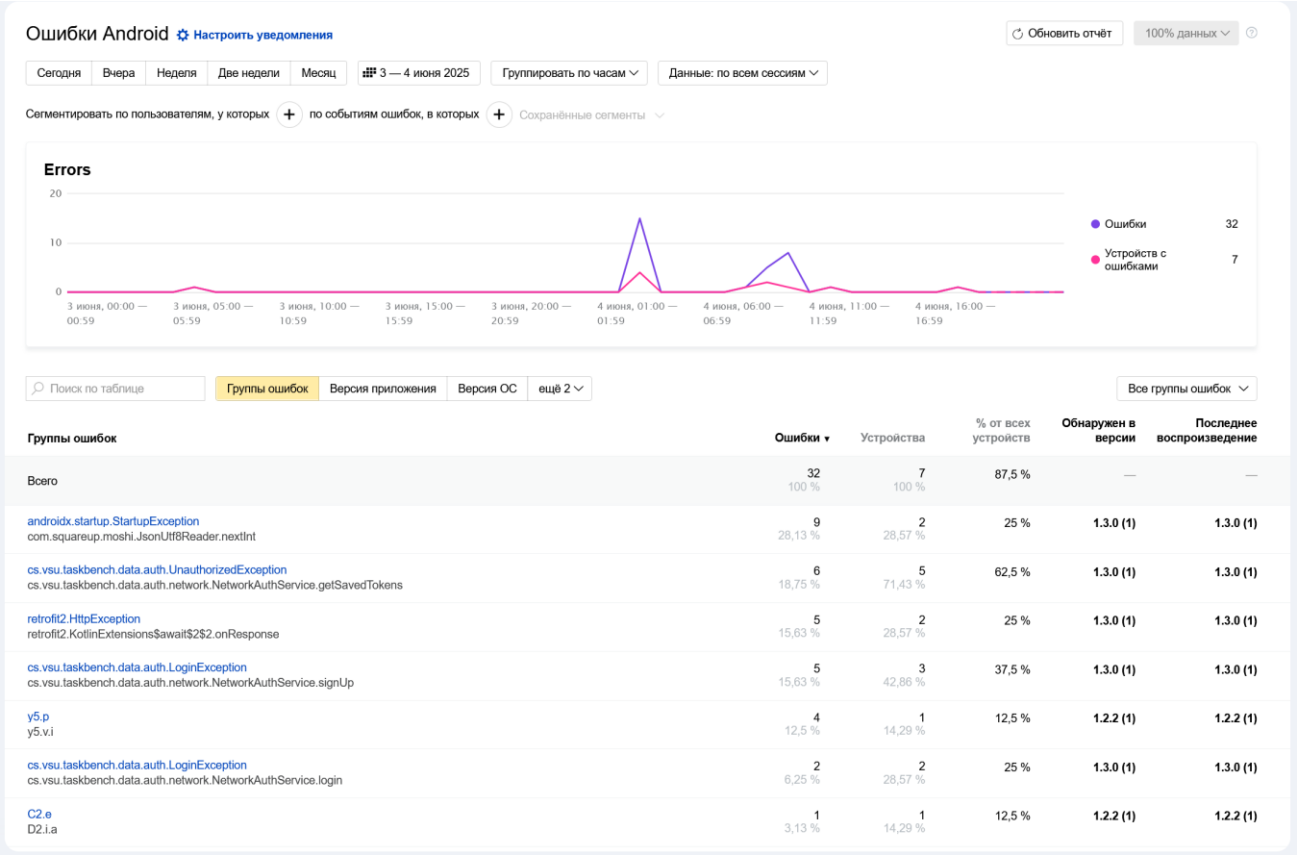


Рисунок 11 - График ошибок, возникающих у пользователей при использовании приложения.

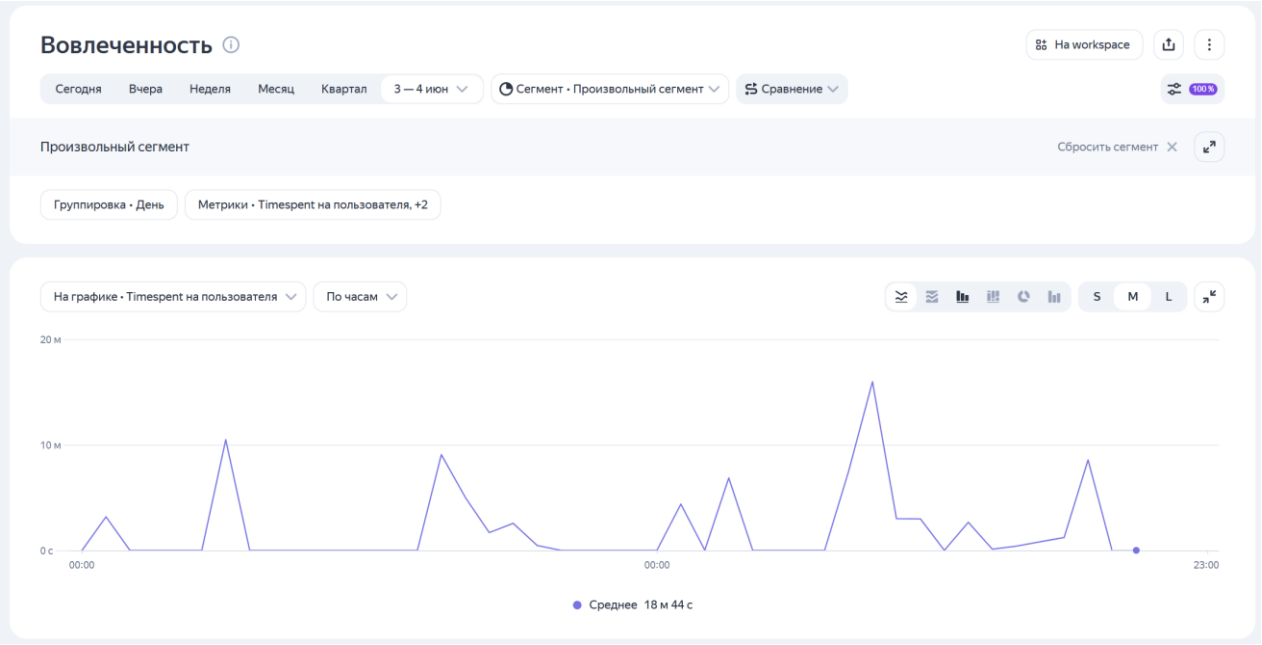


Рисунок 12 - График среднего времени, проведённого пользователем в приложении

4.9 Доработка по результатам тестирования и обратной связи

В процессе разработки и особенно во время тестирования приложения «Taskbench» (включая взаимную проверку между участниками команд) были выявлены и реализованы улучшения, не предусмотренные первоначальным техническим заданием. Многие из них стали результатом обратной связи и анализа пользовательского опыта.

В частности, изначально рассматривалась возможность автоматического назначения приоритета задач, однако в процессе тестирования идеи было решено отказаться от этой функции. Причиной стало то, что автоматическое указание приоритета воспринимается как навязывание, нарушающее ощущение контроля пользователя. Вместо этого реализован простой переключатель приоритета (низкий / высокий) вручную, что соответствует философии проекта — удобный и деликатный ввод без давления со стороны системы.

Кроме того, была улучшен внешний вид некоторых экранов, добавлены возможности управления категориями и добавлена функция отмены случайной отметки задачи выполненной.

5 Анализ соответствия поставленным требованиям

5.1 Базовая функциональность

Поставленные требования к программному продукту:

- Приложение должно соответствовать перечисленным в техническом задании требованиям и включать все перечисленные возможности для соответствующих ролей пользователей;
- Архитектура системы должна позволять легко расширять функциональность и масштабировать приложение без значительных доработок

Можно утверждать, что требования к функциональности, безопасности и надежности системы соблюдены более чем на 90 процентов.

5.2 Пользовательский опыт

Заявленные требования:

- У пользователя не должно уходить много времени, чтобы привыкнуть к структуре приложения и его функциям;
- Все действия пользователя должны происходить быстро, с минимальным ожиданием.

По результатам тестирования самым затратным действием с длинным откликом является умная генерация предложений. Большая часть времени уходит на обмен данными из-за специфичного расположения сервера. Во время ожидания ответа пользователю показывается анимированный индикатор, из-за чего процесс ожидания ощущается менее болезненно.

Ключевые пользовательские сценарии о создании и просмотре задач оказались понятны в работе пользователям, которые тестировали приложение, поэтому можно считать этот критерий выполненным.

5.3 Безопасность данных

Заявленные требования к безопасности:

- Должны быть реализованы надежные механизмы предотвращения утечек данных, включая шифрование и защиту от угроз (SQL-инъекции, XSS, CSRF).

Безопасность системы реализуется за счет использования инструментов Django и шифрования трафика, поэтому считать критерий выполненным.

6 Перспективы развития приложения

6.1 Исследование расширения возможностей приложения

Развитие технологий и растущие ожидания пользователей создают предпосылки для дальнейшего расширения функциональности приложения «Taskbench». В перспективе возможно внедрение интеграции с внешними сервисами — такими как календари, заметки и системы напоминаний, что позволит пользователю более полно управлять своими задачами в рамках единой системы.

Также рассматривается добавление ночной темы оформления и расширенных параметров отображения, с акцентом на сохранение визуальной чистоты и минимализма, чтобы избежать перегрузки интерфейса. Ещё одним из направлений развития является реализация совместного использования задач — возможность делиться отдельными задачами или списками с другими пользователями приложения.

Дополнительно рассматривается возможность использования голосового ввода задач.

Кроме того, возможно усиление интерактивности пользовательского интерфейса, чтобы сделать процесс взаимодействия с задачами более живым, интуитивным и увлекательным.

6.2 Оценка монетизации приложения

С учётом современных тенденций в мобильной разработке, вопрос монетизации является важным этапом жизненного цикла приложения. Для проекта «Taskbench» выбран подход, при котором основная функциональность доступна бесплатно, а расширенные возможности предлагаются в рамках платной подписки.

Для реализации механизма оплаты в проекте уже используется платёжная система ЮKassa, на данный момент интегрированная через тестовую среду (песочницу). Однако в будущем при реализации проекта возможно быстрое подключение ИП аккаунта и запуск продаж.

Модель монетизации через подписку является гибкой и масштабируемой. В перспективе возможно:

- внедрение нескольких тарифных планов (например, базовый, расширенный, командный);
- подключение других платёжных систем (например, Google Pay, Apple Pay, банковские карты);
- введение системы пробного периода или разовых покупок (например, разовая генерация задач ИИ).

Такой подход обеспечивает устойчивую финансовую модель проекта и позволяет сосредоточиться на улучшении качества приложения и пользовательского опыта.

Заключение

Таким образом, в рамках курсового проекта была спроектирована и реализована мобильная система, предназначенная для повышения личной продуктивности пользователей путём организации и управления задачами. Основное внимание при разработке уделялось удобству пользовательского интерфейса, устойчивости архитектуры и возможности расширения функционала в будущем.

В процессе работы была разработана полноценная Android-клиентская часть на языке Kotlin с использованием Jetpack Compose, а также серверная часть на базе Django и PostgreSQL. Для авторизации применён механизм JSON Web Token (JWT), обеспечивающий безопасную и удобную систему входа. Отдельным направлением стала реализация интеграции с ИИ-моделью GigaChat, благодаря которой приложение получило возможность автоматически разбивать задачи на подзадачи, определять категории и предлагать сроки выполнения.

Также подключена тестовая система оплаты через ЮKassa, что позволило создать основу для дальнейшей коммерциализации проекта.

В ходе разработки активно применялись современные практики: архитектура MVVM, использование CI/CD для автоматического тестирования и сборки, а также ведение общего Git-репозитория для командной работы.

Проект демонстрирует устойчивую архитектуру, потенциальную масштабируемость и высокую степень готовности к публикации.

Приложение имеет потенциал для дальнейшего развития — в том числе в направлениях интеграции со множеством сервисов. Таким образом, проект является не только учебной работой, но и основой для практического продукта, способного принести реальную пользу пользователям.

Список использованных источников

1. Django documentations [электронный ресурс]. – Режим доступа: <https://www.djangoproject.com>.
2. Kotlin documentations [электронный ресурс]. – Режим доступа: <https://kotlinlang.org/docs>.
3. GigaChat API [электронный ресурс]. – Режим доступа: <https://developers.sber.ru/docs/ru/gigachat/api/reference/rest/gigachat-api>.
4. ЮKassa API [электронный ресурс]. – Режим доступа: <https://yookassa.ru/developers>.
5. The MVVM Pattern [электронный ресурс]. – Режим доступа: [https://learn.microsoft.com/en-us/previous-versions/msp-n-p/hh848246\(v=pandp.10\)](https://learn.microsoft.com/en-us/previous-versions/msp-n-p/hh848246(v=pandp.10)).

ПРИЛОЖЕНИЕ А

Пример кода, описывающего модель профиля задачи на Python:

```
class Task(models.Model):

    task_id = models.AutoField(primary_key=True)

    title = models.CharField(max_length=255,
null=False)

    deadline = models.DateTimeField(null=True)

    priority = models.IntegerField(null=False, de-
fault=0)

    is_completed = models.BooleanField(default=False)

    created_at = models.DateTimeField(default=time-
zone.now)

    completed_at = models.DateTimeField(null=True)

    ai_processed = models.BooleanField(default=False)

    user = models.ForeignKey(User, on_delete=mod-
els.CASCADE, null=False, blank=False, re-
lated_name='tasks') #многие к одному к юзеру

    def __str__(self):

        return self.title
```


ПРИЛОЖЕНИЕ Б

Пример Python-функции, реализующей отправку запроса к GigaChat с системным промптом и пользовательским текстом:

```
def send_message_with_system_prompt(self, system_prompt: str, user_text: str):

    if self.debug: return None

    self.update_token()

    result = self.giga.chat(

        Chat(

            messages=[

                Messages(

                    role=MessagesRole.SYSTEM,

                    content=system_prompt

                ), Messages(

                    role=MessagesRole.USER,

                    content=user_text

                )

            ]

        )

    )
```

```
print(result.choices[0].message.content)

return result
```

ПРИЛОЖЕНИЕ В

Используемые промты:

```
SUBTASK_SYSTEM_PROMPT = ""
```

Разбей введенную пользователем задачу на несколько более мелких подзадач, состоящие не более чем из четырех слов.

Каждая подзадача должна быть короткой и представлена на отдельной строке.

Не используй знаки препинания или обозначения списка, просто пиши только подзадачи с новой строки.

```
""
```

```
SUBTASK_SYSTEM_PROMPT_V2 = ""
```

Предложи несколько мелких подзадач к введенной пользователем задаче, состоящих не более чем из четырех слов.

Каждая подзадача должна быть короткой и представлена на отдельной строке.

Не пиши заголовок.

Не пиши нумерацию подзадачи, пиши ТОЛЬКО текст подзадач с новой строки.

```
""
```

TIME_SYSTEM_PROMPT = ""

Предложи предположительную дату и время,
соответствующие сроку введенной пользователем задачи.

Если время указано относительно, например словами
'завтра' или 'в следующую среду', в качестве точки
отсчета используй текущее время.

В приоритете всегда предполагай время из будущего.

Если и время и дата не указаны, отправь ТОЛЬКО ОДИН
СИМВОЛ: "-".

Если известно только время, считай датой сегодня.

Если известна только дата, считай время таким же как
сейчас.

Отправь ТОЛЬКО дату и время в формате YYYY:MM:DD hh:mm.
Не добавляй никакого другого текста или пояснений.

Например: 2024:03:15 10:30

""

CATEGORY_SYSTEM_PROMPT = ""

Соотнеси пользовательский текст с одной из категорий,
соответствующих следующему списку. Напиши только одно
слово - название категории. Список категорий:\n

""

ПРИЛОЖЕНИЕ Г

Пример кода, описывающего модель Domain:

```
package cs.vsu.taskbench.domain.model

import androidx.compose.runtime.Immutable

import java.time.LocalDateTime

@Immutable

data class AiSuggestions(

    val subtasks: List<String> = listOf(),

    val deadline: LocalDateTime? = null,

    val isHighPriority: Boolean? = null,

    val category: Category? = null,

)
```

ПРИЛОЖЕНИЕ Д

Пример функции получения задач, в которой с помощью авторизации выполняется HTTP-запрос на сервер. Ответ преобразуется в локальные модели задач для отображения в UI:

```
override suspend fun getTasks(

    categoryFilter: CategoryFilterState,

    sortByMode: TaskRepository.SortByMode,

    deadline: LocalDate?

): List<Task> {

    authService.withAuth { access ->

        val response = dataSource.getAllTasks(

            access = access,

            offset = 0,

            limit = 1000,

            date = deadline?.format(DateTimeFormatter.ISO_LOCAL_DATE),

            categoryId = when (categoryFilter) {

                CategoryFilterState.Disabled -> null

                is CategoryFilterState.Enabled -> categoryFilter.category?.id

            }
        )
    }
}
```

```

        },

        sortBy = when (sortByMode) {

            TaskRepository.SortByMode.Priority ->
            "priority"

            TaskRepository.SortByMode.Deadline ->
            "deadline"

        }

    )

    return response.map { it.toModel() }

}

error("")

}

```