

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет	Кафедра
вычислительной техники	МОиПЭВМ
Направление подготовки	09.03.04 «Программная инженерия»
Направленность	Программная инженерия

БАКАЛАВРСКАЯ РАБОТА

на тему

Web-приложение для управления личными
задачами и планирования времени

Студент	_____	Головинов Михаил Юрьевич
	(подпись, дата)	(ФИО полностью)
Руководитель	_____	Такташкин Д.В.
	(подпись, дата)	(фамилия, инициалы)
Нормоконтролёр	_____	Такташкин Д.В.
	(подпись, дата)	(фамилия, инициалы)
Работа допущена к защите (протокол заседания кафедры от _____ № _____)		
Заведующий кафедрой	_____	Козлов А.Ю.
	(подпись)	(фамилия, инициалы)
Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)		
Секретарь ГЭК	_____	Попова Н.А.
	(подпись)	(фамилия, инициалы)

Пенза, 2025

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
Вычислительной техники

Кафедра
МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
А.Ю. Козлов
«20» декабря 2024 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Головинов Михаил
Юрьевич гр. 21ВП1 факультета ФВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Web-приложение для управления личными задачами и планирования времени

Тема утверждена приказом ПГУ № _____ от " _____ " _____ 20 _____

2. Руководитель работы Такташкин Д.В.

3. Задание на работу (назначение разработки, исходные данные и т.п.)

Назначение: разработка web-приложения, предназначенного для управления делами
Функции:

1. Создание, редактирование и удаление личных задач
2. Планирование задач с возможностью выбора дат и времени
3. Организация задач по категориям и приоритетам
4. Ведение списка подзадач
5. Отметка выполнения задач и подзадач
6. Настройка уведомлений о предстоящих событиях
7. Интеграция с Telegram для напоминаний и входа через бота
8. Поддержка смены темы интерфейса

Технологии разработки: клиент-сервер, ООП, дизайн-система

Языки программирования: JavaScript (Node.js, React),
SQL

Среды разработки: Visual Studio Code, pgAdmin, Postman, npm

4. Перечень подлежащих разработке вопросов

1. Анализ предметной области

2. Анализ требований на разработку

3. Проектирование системы

4. Реализация системы

5. Тестирование системы

5. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	08.04.2025 – 14.04.2025	
2	Проектирование программных средств	20%	15.04.2025 – 25.04.2025	
3	Реализация программных средств	30%	26.04.2025 – 12.05.2025	
4	Тестирование программных средств	20%	13.05.2025 – 22.05.2025	
5	Оформление пояснительной записки	20%	23.05.2025 – 03.06.2025	

Дата выдачи задания «20» декабря 2024г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан ФВТ _____ Л.Р. Фионова
(подпись, дата)

Реферат

Пояснительная записка содержит 93 листа, 39 рисунков, 21 использованных источников, 29 таблиц, 2 листинга и 3 приложения

WEB-СЕРВИС, ПЛАНИРОВАНИЕ ВРЕМЕНИ, NODE.JS, POSTGRESQL,
GIT, REACT, EXPRESS.JS, TELEGRAM BOT API, JWT, REST API, CLIENT-
SERVER

Объектом исследования является web-сервис, с помощью которого пользователь может управлять личными задачами и временем.

Предмет исследования – технология разработки web-сервиса для автоматизации управления личными задачами и планирования времени.

Цель работы: реализовать web-сервис для управления задачами и личным планированием.

Технологии разработки: Node.js, PostgreSQL, Git, React, Express.js, Telegram Bot API, JWT, REST API.

Результат работы: спроектирован и разработан web-сервис для планирования времени и управления личными задачами с интеграцией уведомлений через Telegram и поддержкой авторизации, а также возможности работы с задачами в режиме календаря.

					ПГУ 09.03.04 – 8ВП211.05 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата				
Разраб.		Головинов М.Ю.			«Web-приложение для управления личными задачами и планирования времени» Пояснительная записка	Лит.	Лист	Листов
Провер.		Такташкин Д.В.					4	93
Реценз.						Группа 21ВП1		
Н. Контр.		Такташкин Д.В.						
Утверд.		Козлов А.Ю.						

Оглавление

Введение	7
1 Анализ предметной области и постановка задачи на разработку.....	8
1.1 Анализ предметной области	8
1.2 Анализ существующих решений.....	9
1.2.1 Сервис Todolist.....	9
1.2.2 Сервис TickTick	12
1.2.3 Сервис YouGile	15
1.2.4 Сервис Google Календарь	17
1.3 Сравнительный анализ сервисов конкурентов	21
1.4 Постановка цели и задач	27
1.5 Анализ требований к программному обеспечению.....	29
1.5.1 Постановка функциональных требований	29
1.5.2 Нефункциональные требования к клиентскому обеспечению	32
1.5.3 Нефункциональные требования к серверной части	34
1.6 Организация разработки и определение финансовых затрат.	35
2 Проектирование сервиса	39
2.1 Выбор и обоснование средств проектирования.....	39
2.2 Проектирование базы данных	40
2.3 Проектирование архитектуры	41
2.4 Проектирование интерфейса	43
3 Реализация сервиса.....	45
3.1 Обоснование средств разработки	45
3.2 Реализация архитектуры приложения	46
3.3 Реализация базы данных	48

3.4	Реализация интерфейса	49
3.4.1	Интерфейс для авторизованного пользователя	51
3.4.2	Интерфейс администратора.....	54
4	Тестирование сервиса.....	57
4.1	Функциональное тестирование	57
4.2	Модульное тестирование	59
4.3	Анализ метрик качества кода	60
	Заключение.....	64
	Список используемых источников	66
	Приложение А (справочное) Глоссарий.....	68
	Приложение Б (обязательное) Исходной код серверной части	71
	Приложение В (рекомендуемое) Тестирование приложения	88

Введение

В условиях стремительного роста информационной нагрузки и постоянного дефицита времени эффективное планирование стало ключевым фактором успешной профессиональной и личной жизни. Особенно актуальной эта задача становится для студентов, молодых специалистов и людей, работающих в условиях гибкого графика. Цифровые технологии открывают широкие возможности для организации повседневных дел, однако на практике многие пользователи сталкиваются с отсутствием удобных и наглядных инструментов для систематизации задач.

Существующие решения в области тайм-менеджмента либо перегружены функционалом, не нужным большинству пользователей, либо наоборот, предоставляют лишь базовые возможности, не учитывая индивидуальные особенности ведения дел. В частности, немногие приложения поддерживают визуальное планирование по дням недели, интеграцию с Telegram для напоминаний и уведомлений, а также возможность работать с подзадачами и персонализированными настройками.

В связи с этим возникает необходимость разработки специализированного веб-приложения, ориентированного на управление личными задачами и планирование времени с возможностью гибкой настройки, визуальной организации и получения напоминаний в удобной форме. Такое приложение позволит пользователям лучше структурировать повседневные дела, повысить продуктивность и снизить уровень стресса, связанного с забытыми или неорганизованными задачами.

Целью данной выпускной квалификационной работы является разработка удобного и функционального веб-приложения для управления личными задачами и планирования времени, включающего поддержку подзадач, диапазонов дат, визуального представления задач по дням недели, авторизацию и напоминания через Telegram.

1 Анализ предметной области и постановка задачи на разработку

1.1 Анализ предметной области

В условиях высокой нагрузки и большого количества повседневных задач, перед людьми остро встаёт необходимость в систематизации дел и грамотном распределении времени. С ростом популярности гибкой занятости, онлайн-обучения и проектной работы всё больше людей обращаются к цифровым решениям для управления личными задачами и планированием.

В процессе анализа предметной области удалось выделить две ключевые группы пользователей:

- студенты и учащиеся, которым важно отслеживать дедлайны, экзамены, расписание и личные дела;
- работающие специалисты и фрилансеры, нуждающиеся в удобной визуализации проектов, напоминаниях, управлении длительными задачами и синхронизации с мессенджерами.

Также существуют разные подходы к планированию, которые пользователь может применять:

- дневной – организация задач в пределах одного дня;
- недельный – распределение задач на ближайшую неделю с учётом приоритетов;
- диапазонный – задания с начальной и конечной датой, характерные для проектов;
- микроуправление – использование подзадач и промежуточных этапов.

Исходя из этого, требуется веб-сервис, который:

- даст возможность создавать и управлять задачами, имеющими как фиксированную дату, так и диапазон;
- поддерживает подзадачи и статусы выполнения;
- позволяет визуализировать задачи по неделям;
- отправляет уведомления через Telegram;

- сохраняет настройки пользователей;
- основной функционал продукта;
- управление задачами и подзадачами;
- визуализация задач по неделям;
- настройка начала и окончания задачи (в том числе времени);
- telegram-уведомления (сводки и напоминания);
- тёмная/светлая тема;
- аутентификация и хранение настроек.

1.2 Анализ существующих решений

Анализ конкурентных решений позволяет определить потребности целевой аудитории, оценить сильные и слабые стороны текущих реализаций, и на основе этого выстроить приоритеты собственного продукта.

Наиболее известные конкуренты:

- Todolist;
- YouGile;
- TickTick;
- Google Календарь.

1.2.1 Сервис Todolist

Todolist [1] – это облачное приложение для управления личными и рабочими задачами. Оно ориентировано на пользователей, которым необходим простой способ ведения списка дел с возможностью гибкой классификации и фильтрации задач. Данное веб приложение позволяет быстро добавлять новые задачи, классифицировать их по различным проектам, назначать приоритеты и устанавливать сроки выполнения всех дел. Основной интерфейс построен вокруг списков задач с возможностью добавления проектов, приоритетов и сроков (рисунок 1).

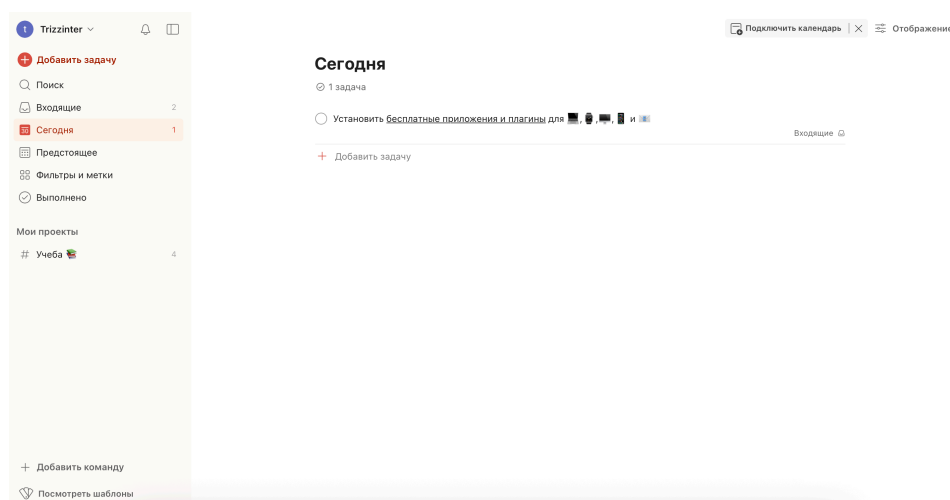


Рисунок 1 – Создание задачи в Todolist

Для создания задач в сервисе предусмотрена удобная форма, в которой можно указать название задачи, выбрать срок исполнения, приоритет, добавить метки, а также вложенные подзадачи. Задачи могут быть объединены в проекты, что позволяет структурировать рабочие процессы (рисунок 2).

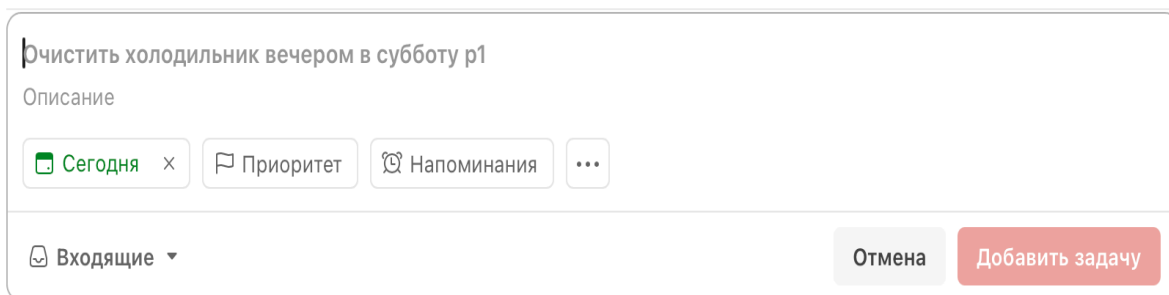


Рисунок 2 – Пример структуры задач и подзадач в Todoist

Также в сервисе реализована поддержка подзадач, что удобно при планировании задач, состоящих из нескольких этапов.

Однако взаимодействие с подзадачами ограничено: они не имеют собственных сроков или приоритетов.

Для отображения задач доступен список задач по дням, а также пользовательские фильтры.

Недельного календаря с визуализацией задач не предусмотрено. Задачи отображаются в виде ленты с указанием даты и дня недели, подсвечивая текущий день специальным красным маркером для наглядности (рисунок 3).

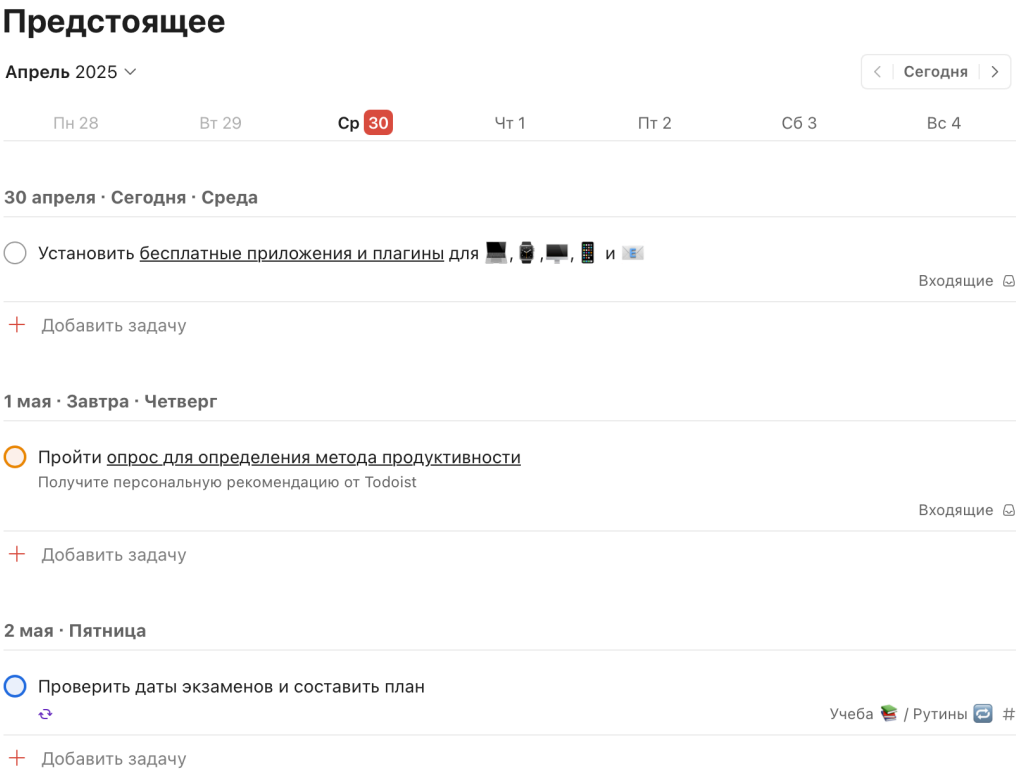


Рисунок 3 – Отображение задач по дням в Todoist

Функционал напоминаний присутствует, но работает только при подключении платного тарифа. Напоминания можно получать по push-уведомлению или email. Интеграции с мессенджерами, в том числе Telegram, отсутствуют (рисунок 4).

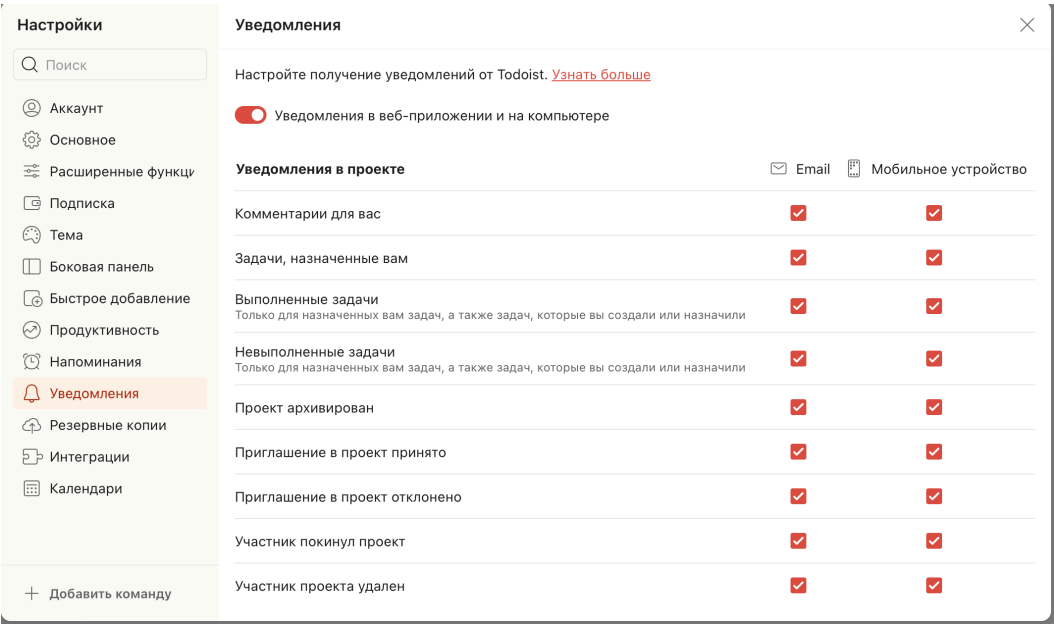


Рисунок 4 – Настройки напоминаний в Todoist

Организация задач реализована через систему проектов, меток и фильтров. Фильтры позволяют создавать пользовательские представления, например, «все задачи на сегодня» или «важные задачи без срока» (рисунок 5).

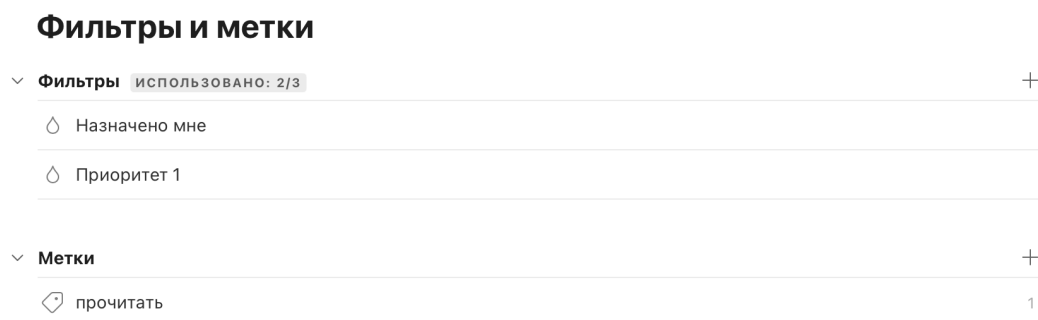


Рисунок 5 – Организация задач по проектам и фильтрам в Todoist

Также стоит отметить, что Todoist не поддерживает задачи с диапазоном дат: можно задать только одну дату исполнения. Это затрудняет использование сервиса для задач длительного характера, требующих планирования на несколько дней вперёд.

Вывод: Todoist – удобный сервис для базового ведения задач и контроля сроков. Он обладает широкой системой фильтрации и организации, но не поддерживает задачи с интервалами, не предоставляет недельного визуального представления и не интегрирован с Telegram. Эти ограничения делают его недостаточным для сценариев с акцентом на недельное планирование и гибкие напоминания.

1.2.2 Сервис TickTick

TickTick [2] – это веб-приложение для ведения задач и планирования времени, которое сочетает в себе функции календаря, трекера привычек и планировщика задач. Сервис ориентирован как на индивидуальных пользователей, так и на небольшие команды. В отличие от многих аналогов, TickTick предоставляет полноценное недельное представление задач и визуальный календарь (рисунок 6).

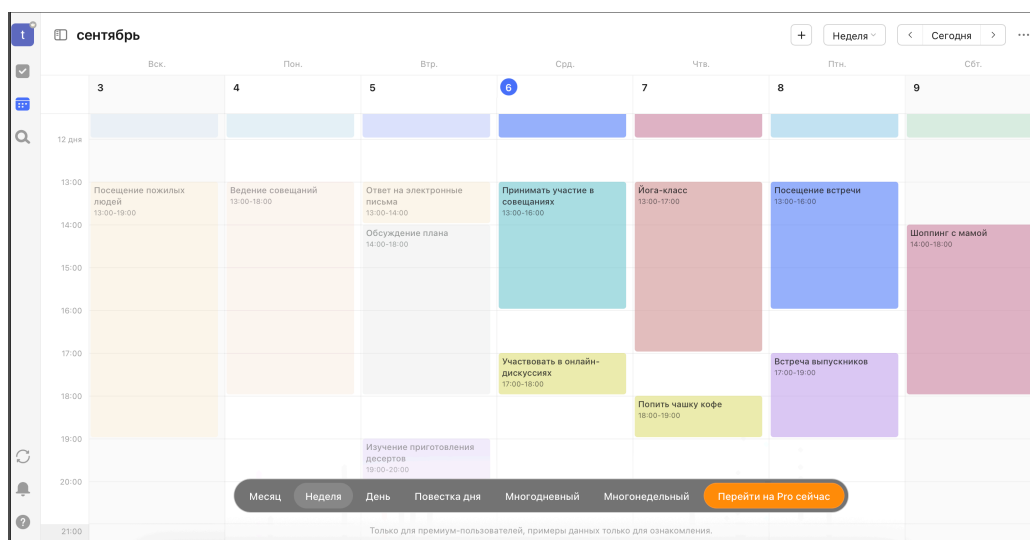


Рисунок 6 – Недельное представление задач в TickTick

Создание задачи реализовано через компактную форму, где можно указать название, дату, время, приоритет, повторяемость и добавить подзадачи (рисунок 7).

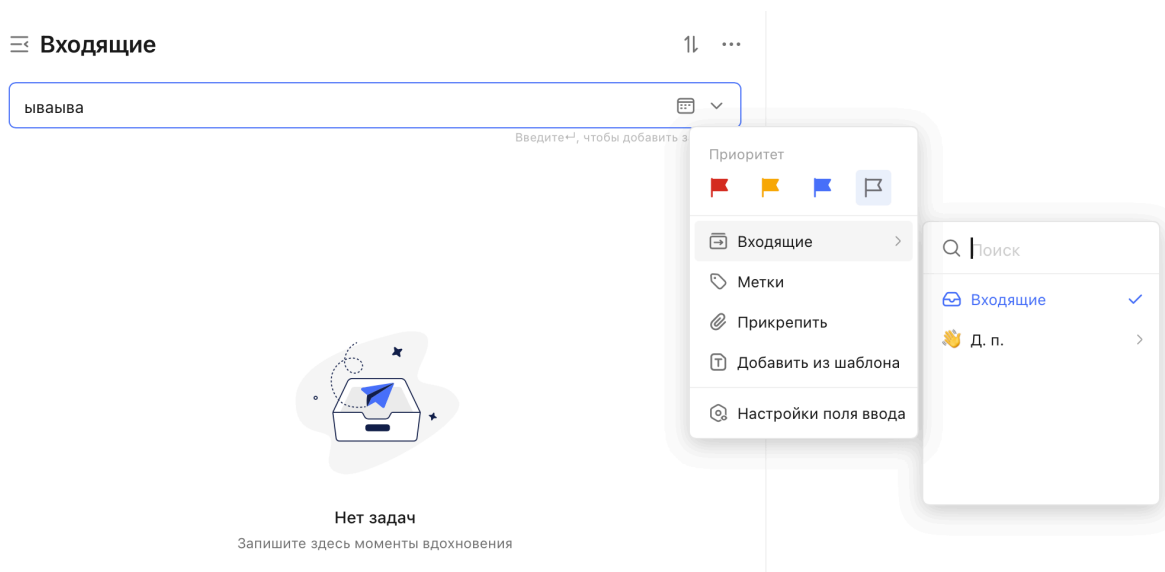


Рисунок 7 – Создание задачи в TickTick

Внутри каждой задачи пользователь может создать подробный список подзадач, что позволяет разбивать крупные дела на более мелкие и контролировать их выполнение поэтапно.

Подзадачи поддерживают флажки выполнения, но не имеют собственных сроков (рисунок 8).



Рисунок 8 – Структура задачи с подзадачами в TickTick

Одним из ключевых преимуществ сервиса является возможность отображения задач в разных режимах: список, день, неделя, месяц. Это делает планирование визуально наглядным (рисунок 9).

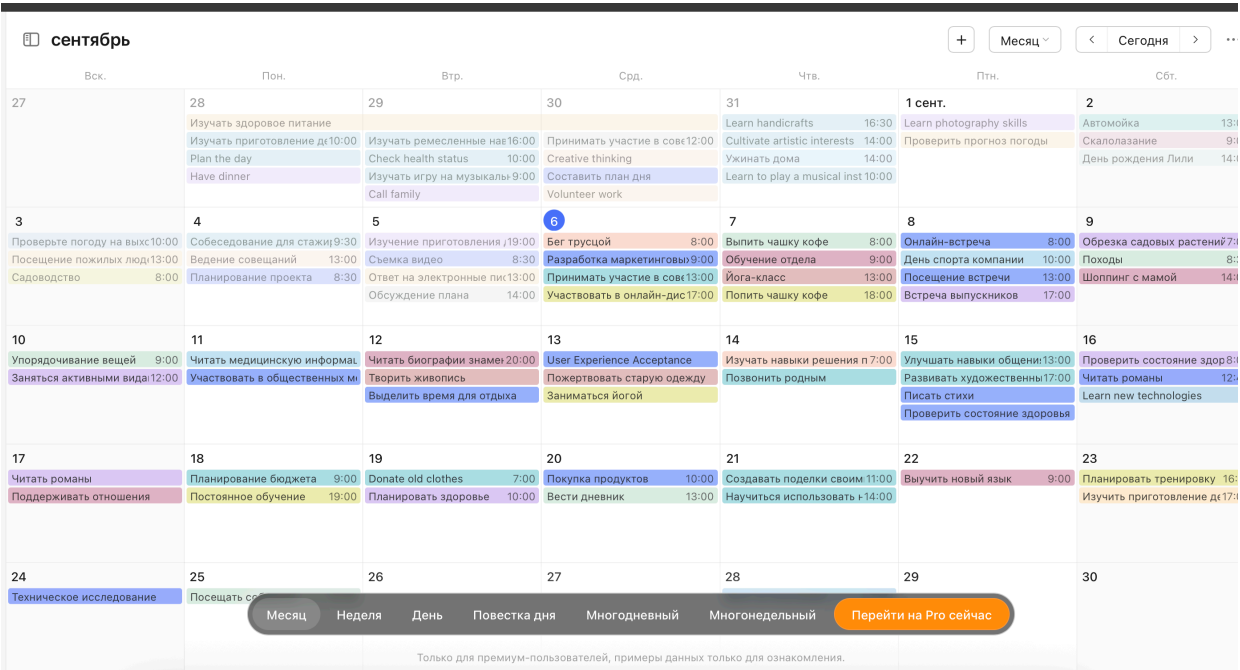


Рисунок 9 – Варианты представления задач в TickTick

TickTick поддерживает уведомления и напоминания, включая напоминания по времени и месту.

Также доступна интеграция со сторонними календарями (например, Google Calendar и прочими популярными решениями), но интеграция с мессенджером Telegram по части уведомлений отсутствует, хотя сервис и имеет другие типы оповещений пользователей, например почтовые письма и браузерные уведомления (рисунок 10).

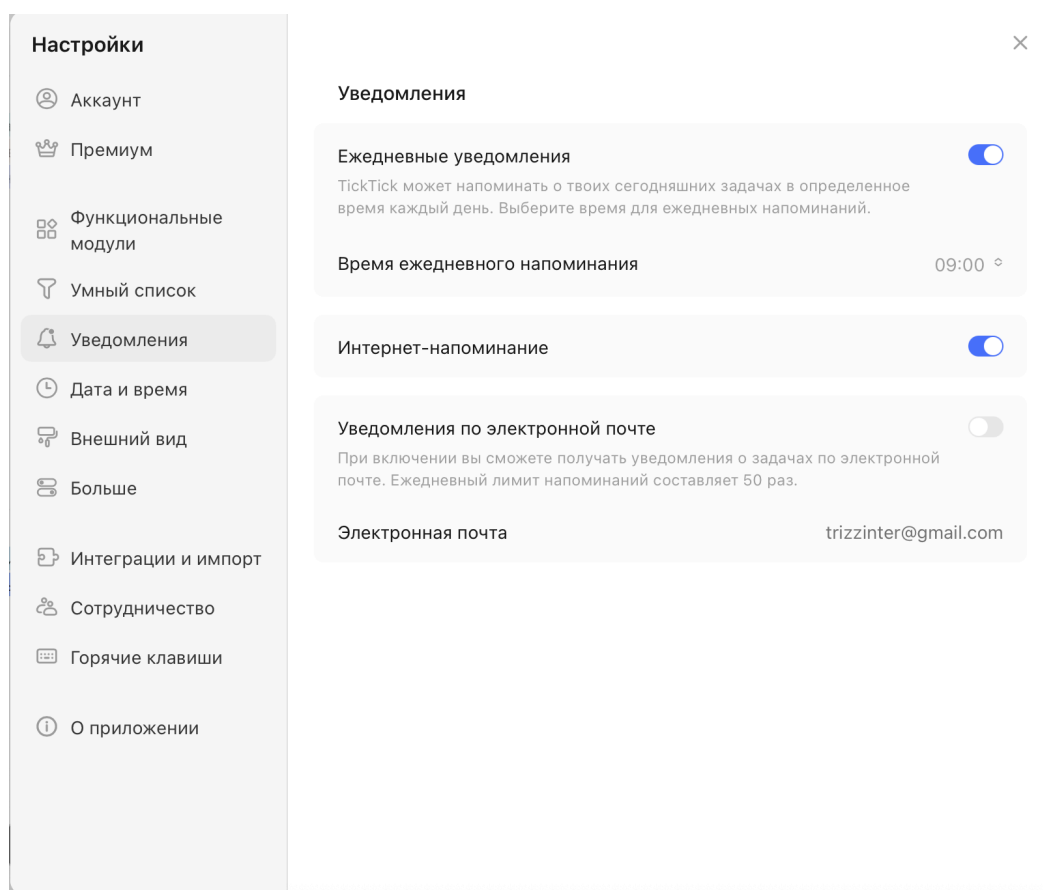


Рисунок 10 – Настройки уведомлений в TickTick

Вывод: TickTick – один из наиболее функциональных планировщиков задач, сочетающий в себе календарь, список дел и трекер привычек. Он поддерживает недельное и месячное представление, задачи с интервалами и гибкую систему напоминаний. Отсутствие интеграции с мессенджерами – один из немногих его недостатков.

1.2.3 Сервис YouGile

YouGile [3] – российская система управления задачами и проектами, разработанная для командной работы. Сервис предоставляет канбан-доски, встроенные чаты и функции для планирования и отслеживания задач. Он полностью адаптирован под российские реалии и доступен без ограничений. Основной интерфейс построен на досках с задачами, разбитыми по этапам выполнения (рисунок 11).

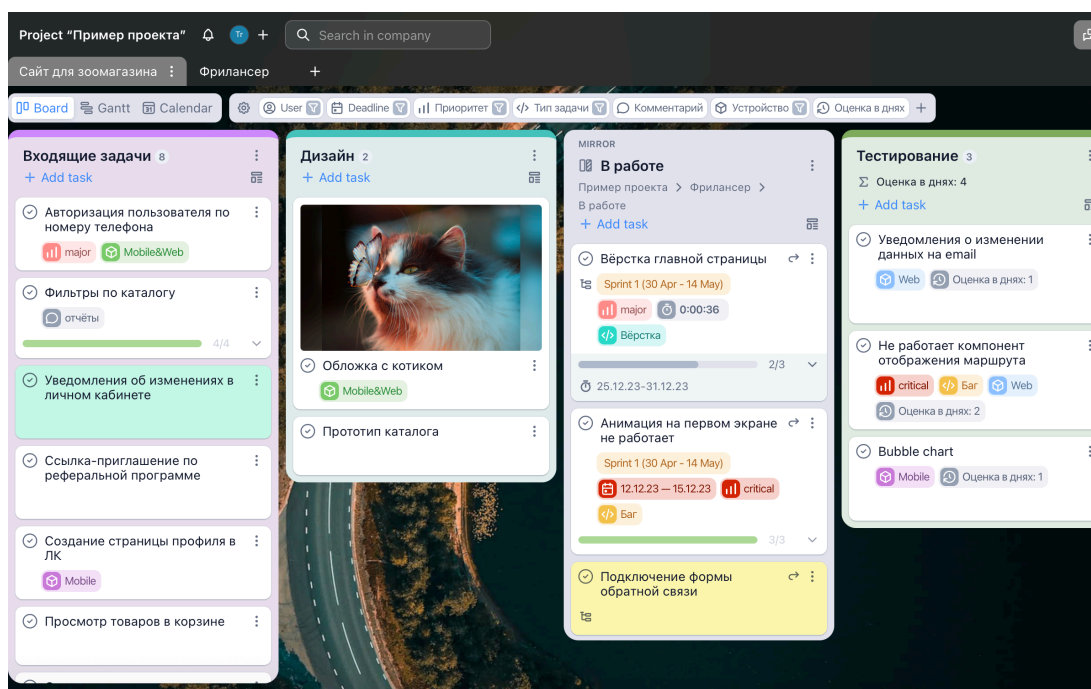


Рисунок 11 – Канбан-доска в YouGile

Создание задачи происходит через удобную карточку, где можно указать название, срок выполнения, приоритет, описание, а также прикрепить чек-листы и подзадачи (рисунок 12).

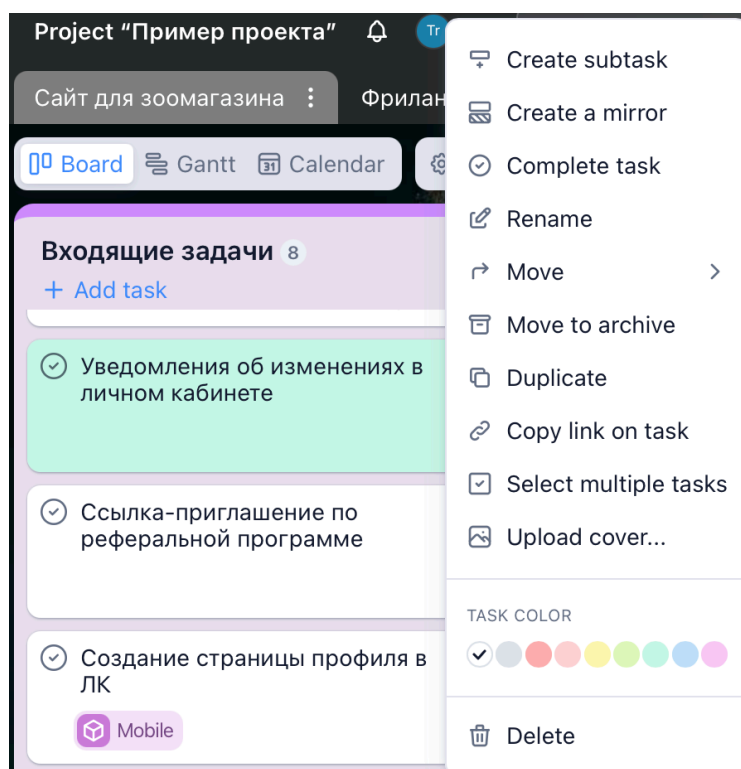


Рисунок 12 – Создание задачи в YouGile

Для визуального планирования используется календарный режим, в котором можно наглядно просматривать задачи по дням и неделям. Это удобно при планировании загрузки команды и контроля сроков (рисунок 13).

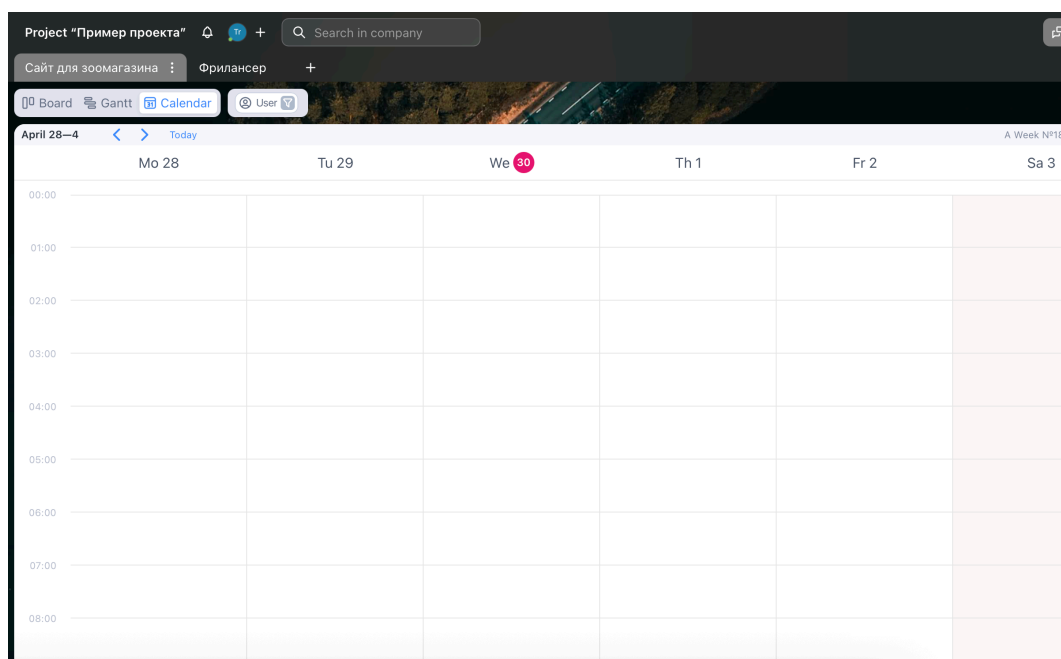


Рисунок 13 – Календарь задач в YouGile

Вывод: YouGile – функциональный и удобный инструмент для управления задачами, поддерживающий визуальное планирование, работу с подзадачами и канбан-доски. Сервис полностью работает в России и подходит как для индивидуального, так и для командного использования.

1.2.4 Сервис Google Календарь

Google Календарь [4] – это онлайн-сервис для планирования задач, встреч и мероприятий, входящий в экосистему Google. Помимо добавления новых событий, сервис поддерживает функцию напоминаний, повторяющихся задач, а также приглашение других пользователей на встречи с помощью интеграции с электронной почтой. Он широко используется как в личных, так и в профессиональных целях. Основной интерфейс построен в виде календаря с возможностью просмотра по дням, неделям и месяцам (рисунок 14).

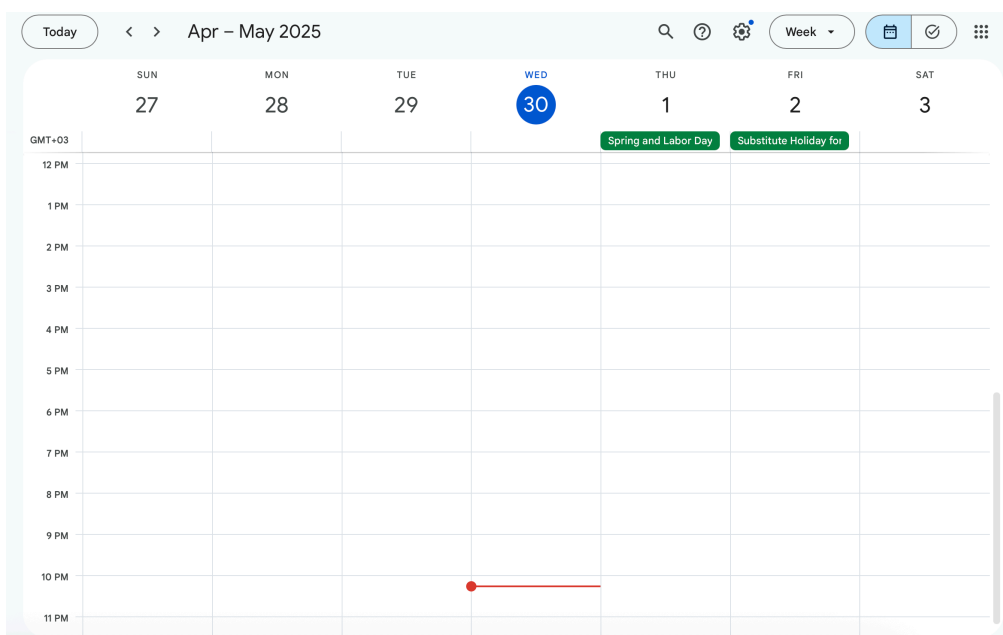


Рисунок 14 – Недельное представление в Google Календаре

Создание задачи (события) происходит через всплывающее окно, где можно указать название, дату, время, повторяемость, местоположение, прикрепить описание и гостей (рисунок 15).

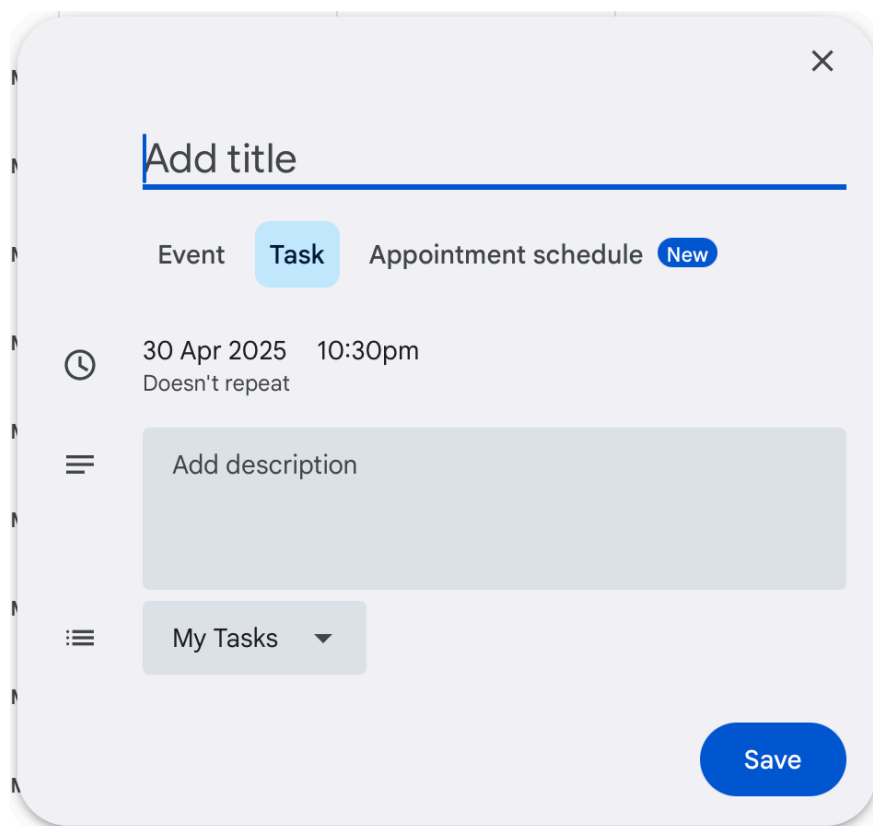


Рисунок 15 – Создание события в Google Календаре

Планировщик позволяет создавать как однодневные задачи, так и задачи с диапазоном дат. Это особенно удобно для мероприятий, растянутых по времени (рисунок 16).

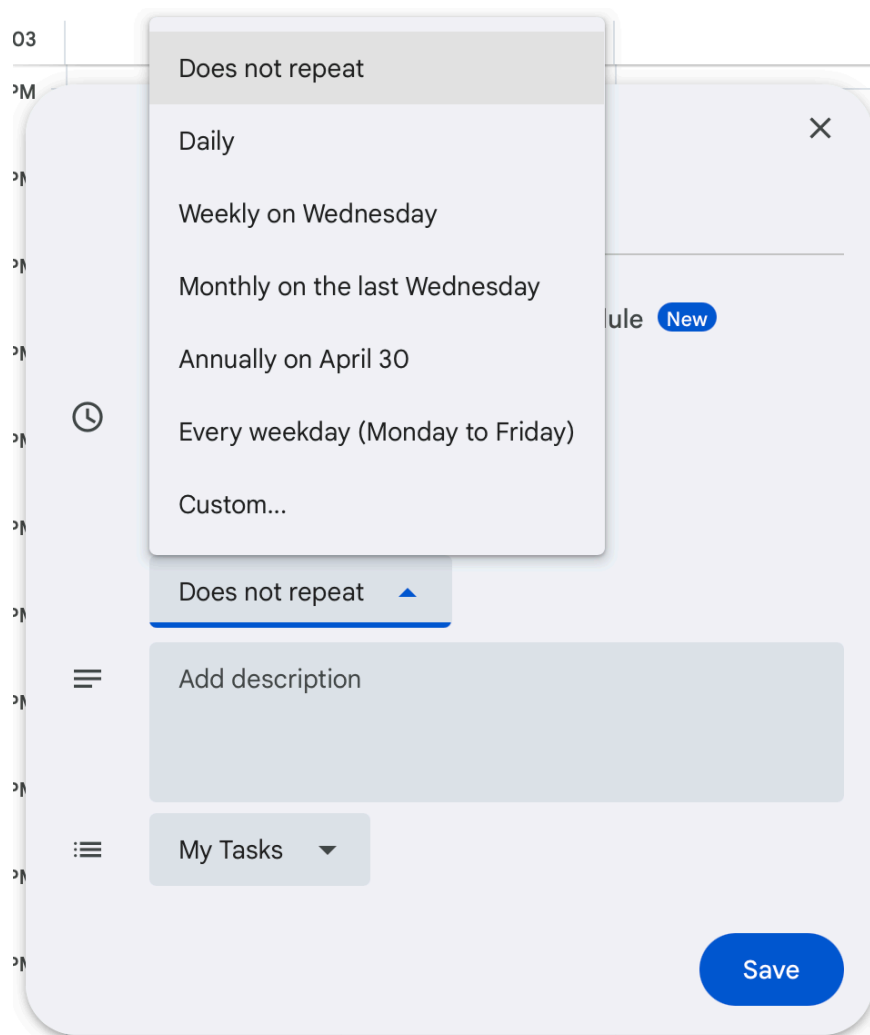


Рисунок 16 – Создание задачи с диапазоном дат в Google Календаре

Механизм напоминаний в системе реализован с учётом обеспечения максимальной информированности пользователя о предстоящих событиях. Для этого предусмотрена поддержка различных каналов оповещения, включая электронную почту, push-уведомления, а также интеграцию с мобильными устройствами, что делает процесс получения напоминаний удобным и гибким в любых условиях (рисунок 17).

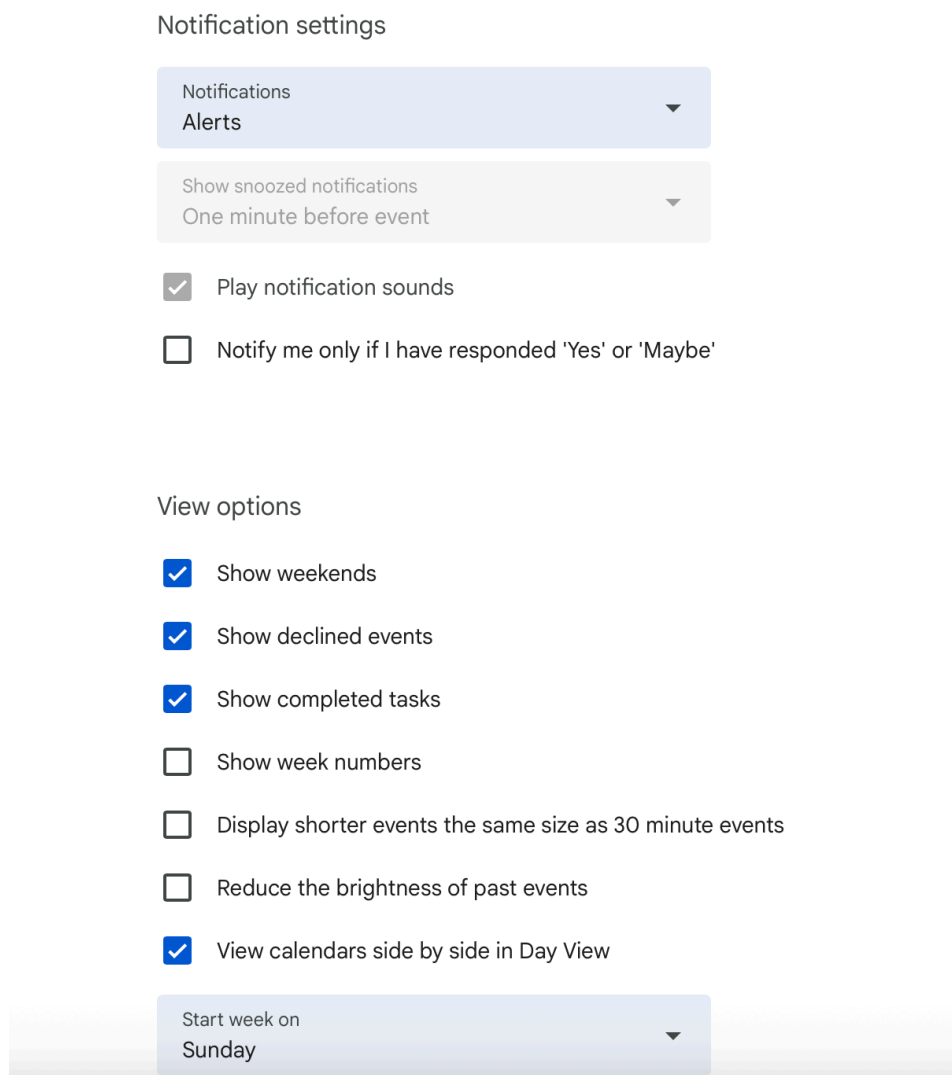


Рисунок 17 – Настройки уведомлений в Google Календаре

Задачи можно помечать цветами и распределять по различным календарям (например, «Учёба», «Работа»). Также поддерживаются совместные календари, где несколько пользователей могут редактировать и просматривать общие события.

Вывод: Google Календарь – мощный инструмент для планирования времени. Он поддерживает задачи с интервалом, напоминания, работу с совместными календарями и синхронизацию между устройствами. Однако он не предоставляет встроенной поддержки подзадач и не предназначен для создания сложных иерархий задач, как в планировщиках.

1.3 Сравнительный анализ сервисов конкурентов

Необходимо провести сравнение различных платформ с аналогичным назначением с использованием метода анализа иерархий. Метод иерархий (МАИ) – это инструмент многокритериального анализа, который помогает принимать решения в ситуациях, где есть неопределенность и множество факторов для учета [6]. Для этого сначала определяют критерии, которые важны при выборе платформы. Затем сравнивают эти платформы на основе каждого критерия и присваивают им относительные приоритеты. После этого анализируются приоритеты, чтобы сделать более информированный выбор.

Выбран арифметический метод анализа критериев вместо геометрического, так как он менее сложен в применении и не требует многочисленных математических вычислений. Кроме того, в анализе отсутствуют экстремальные значения, для обработки которых требуется геометрическое среднее. Изучив предметную область и проведя обзор сервисов, выделены следующие критерии для сравнения платформ-аналогов:

- визуализация задач по неделям;
- поддержка диапазона дат;
- напоминания и уведомления;
- работа с подзадачами;
- интеграция с Telegram.

Необходимо создать матрицы попарных сравнений по качественной шкале, с последующим преобразованием в баллы. Обозначения баллов:

- безразлично (1 балл);
- немного лучше (3 балла);
- лучше (5 баллов);
- значительно лучше (7 баллов);
- принципиально лучше (9 баллов).

Также возможны промежуточные баллы – 2, 4, 6 и 8. Создание матрицы a_{ij} – отношения критерия i к критерию j (таблица 1).

Таблица 1 – Матрица попарных сравнений критериев

	Визуализация	Диапазон дат	Напоминания	Подзадачи	Telegram
Визуализация	1	3	5	5	7
Диапазон дат	1/3	1	3	3	5
Напоминания	1/5	1/3	1	1	3
Подзадачи	1/5	1/3	1	1	3
Telegram	1/7	1/5	1/3	1/3	1

По качеству реализации критерия «Визуализация задач по неделям» сервисы можно расположить по следующим местам:

- 1) TickTick – лучшая визуализация недельного расписания, удобная настройка;
- 2) Google Календарь – удобное отображение недельных задач, наглядность;
- 3) YouGile – простое представление задач по неделям;
- 4) Todolist – слабая визуализация задач по неделям.

Исходя из успешности реализации критерия, составлена таблица сравнения альтернатив по критерию «Визуализация задач по неделям» (таблица 2).

Таблица 2 – Матрица сравнения альтернатив по критерию «Визуализация задач по неделям»

	Todoist	TickTick	YouGile	Google Календарь
Todoist	1	1/9	1/3	1/5
TickTick	9	1	5	3
YouGile	3	1/5	1	1/3
Google Календарь	5	1/3	3	1

По качеству реализации критерия «Поддержка диапазона дат» сервисы можно расположить по следующим местам:

- 1) TickTick – диапазоны реализованы полноценно;

- 2) Google Календарь – можно указать период;
- 3) YouGile – поддержка диапазонов есть, но ограничена;
- 4) Todolist – диапазонов нет.

Исходя из успешности реализации критерия, составлена таблица сравнения альтернатив критерия «Поддержка диапазона дат» (таблица 3).

Таблица 3 – Матрица сравнения альтернатив по критерию «Поддержка диапазона дат»

	Todoist	TickTick	YouGile	Google Календарь
Todolist	1	1/9	1/5	1/3
TickTick	9	1	5	3
YouGile	5	1/5	1	1/3
Google Календарь	3	1/3	3	1

По качеству реализации критерия «Напоминания и уведомления» сервисы можно расположить по следующим местам:

- 1) TickTick – гибкие напоминания по времени и месту;
- 2) Google Календарь – email и push, но без гео-напоминаний;
- 3) Todolist – только push и email в премиуме;
- 4) YouGile – встроенные чат-уведомления, но без расписания.

Исходя из успешности реализации критерия, составлена таблица сравнения альтернатив критерия «Напоминания и уведомления» (таблица 4).

Таблица 4 – Матрица сравнения альтернатив по критерию «Напоминания и уведомления»

Сервис	TickTick	Google Календарь	Todoist	YouGile
TickTick	1	3	5	7
Google Календарь	1/3	1	3	5

Продолжение таблицы 4

Todoist	1/5	1/3	1	3
YouGile	1/7	1/5	1/3	1

По качеству реализации критерия «Работа с подзадачами» сервисы можно расположить по следующим местам:

- 1) YouGile – отдельные сроки для подзадач;
- 2) TickTick – подзадачи без сроков;
- 3) Todoist – подзадачи без приоритетов;
- 4) Google Календарь – нет подзадач.

Исходя из успешности реализации критерия, составлена таблица сравнения альтернатив критерия «Работа с подзадачами» (таблица 5).

Таблица 5 – Матрица сравнения альтернатив по критерию «Работа с подзадачами»

	YouGile	TickTick	Todoist	Google Календарь
YouGile	1	5	7	9
TickTick	1/5	1	3	5
Todoist	1/7	1/3	1	3
Google Календарь	1/9	1/5	1/3	1

По качеству реализации критерия «Каналы уведомлений» сервисы можно расположить по следующим местам:

1. Google Календарь – email, push;
2. TickTick – email, push, локальные;
3. Todoist – push и email в премиуме;
4. YouGile – только внутренняя система.

Исходя из успешности реализации критерия, составлена таблица сравнения альтернатив критерия «Каналы уведомлений», отражающая наличие и разнообразие каналов уведомлений для удобства планирования времени (таблица 6).

Таблица 6 – Матрица сравнения альтернатив по критерию «Уведомления»

Сервис	Google Календарь	TickTick	Todoist	YouGile
Google Календарь	1	3	5	7
TickTick	1/3	1	3	5
Todoist	1/5	1/3	1	3
YouGile	1/7	1/5	1/3	1

Необходимо пронумеровать матрицу попарных сравнений критериев. Для этого сначала вычисляется сумма значений в каждом столбце и затем каждое значение делится на полученную сумму. После этого вычисляется среднее значение для каждой строки, чтобы определить весовой коэффициент. Полученные результаты представлены в таблице 7.

Таблица 7 – Весовые доли критериев

Критерий	Вес в долях	Вес в процентах
Визуализация задач по неделям	$\approx 0,433$	43,3 %
Поддержка диапазона дат	$\approx 0,261$	26,1 %
Настройка напоминаний	$\approx 0,137$	13,7 %
Работа с подзадачами	$\approx 0,120$	12,0 %
Каналы уведомлений	$\approx 0,049$	4,9 %

По проведённым расчётам становится ясно, что наиболее весомыми являются «Визуализация задач по неделям» и «Поддержка диапазона дат», а наименее – «Каналы уведомлений». Далее, аналогичным образом получены векторы весовых коэффициентов сервисов-аналогов по каждому критерию (таблицы 8–12).

Таблица 8 – Весовые доли по критерию «Визуализация...»

Сервис	Вес в долях	Вес в процентах
TickTick	$\approx 0,550$	55,0 %

Продолжение таблицы 8

Google Календарь	$\approx 0,252$	25,2 %
YouGile	$\approx 0,123$	12,3 %
Todoist	$\approx 0,075$	7,5 %

Таблица 9 – Весовые доли по критерию «Диапазон дат»

Сервис	Вес в долях	Вес в процентах
TickTick	$\approx 0,558$	55,8 %
Google Календарь	$\approx 0,263$	26,3 %
YouGile	$\approx 0,122$	12,2 %
Todoist	$\approx 0,057$	5,7 %

Таблица 10 – Весовые доли по критерию «Напоминания»

Сервис	Вес в долях	Вес в процентах
TickTick	$\approx 0,520$	52,0 %
Google Календарь	$\approx 0,201$	20,1 %
Todoist	$\approx 0,201$	20,1 %
YouGile	$\approx 0,079$	7,9 %

Таблица 11 – Весовые доли по критерию «Подзадачи»

Сервис	Вес в долях	Вес в процентах
YouGile	$\approx 0,329$	32,9 %
TickTick	$\approx 0,319$	31,9 %
Todoist	$\approx 0,176$	17,6 %
Google Календарь	$\approx 0,176$	17,6 %

Таблица 12 – Весовые доли по критерию «Уведомления»

Сервис	Вес в долях	Вес в процентах
Google Календарь	$\approx 0,466$	46,6 %

Продолжение таблицы 12

TickTick	$\approx 0,349$	34,9 %
Todoist	$\approx 0,117$	11,7 %
YouGile	$\approx 0,068$	6,8 %

Необходимо свести полученные результаты в одну таблицу для удобства восприятия (таблица 13).

Таблица 13 – Общая таблица весовых долей

Сервис	Визуализация	Диапазон дат	Напоминания	Подзадачи	Уведомления
TickTick	0,550	0,558	0,520	0,319	0,349
Google Календарь	0,252	0,263	0,201	0,176	0,466
YouGile	0,123	0,122	0,079	0,329	0,068
Todoist	0,075	0,057	0,201	0,176	0,117

Чтобы получить лучший вариант по всем критериям, надо умножить матрицу из таблицы 13 на вектор весовых коэффициентов критериев из таблиц 7. Результат умножения приведён в таблице 14.

Таблица 14 – Веса альтернативных приложений

Сервис	Вес в долях	Вес в процентах
TickTick	$\approx 0,478$	47,8 %
Google Календарь	$\approx 0,262$	26,2 %
YouGile	$\approx 0,123$	12,3 %
Todoist	$\approx 0,137$	13,7 %

Согласно сравнению, лучшим выбором согласно заданным критериям является платформа TickTick, которая имеет весовую долю 47,8 %.

1.4 Постановка цели и задач

Целью данной выпускной квалификационной работы является разработка веб-сервиса, предназначенного для управления личными задачами и

эффективного планирования времени с возможностью получения уведомлений через Telegram.

Для достижения поставленной цели необходимо решить следующие задачи:

- проанализировать предметную область и сопоставимые программные продукты;
- определить архитектуру создаваемого сервиса;
- спроектировать пользовательский интерфейс;
- спроектировать панель администрирования;
- реализовать клиентскую и серверную части системы;
- обеспечить интеграцию с Telegram-ботом для уведомлений и входа;
- протестировать и оценить удобство использования разработанного решения.

Пользователи сервиса делятся на три основные категории:

- аутентифицированные пользователи, которые могут создавать задачи, подзадачи, планировать их по времени и дате, получать напоминания и утренние сводки через Telegram-бота, а также сохранять персональные настройки;
- неавторизованные пользователи, не имеющие почти никаких функций кроме возможности зарегистрироваться и авторизоваться;
- администраторы, которые имеют доступ к административной панели и могут просматривать статистику пользователей и задач, редактировать данные пользователей (email, пароль, роль), банить и удалять аккаунты.

Сервис также включает Telegram-бота, с помощью которого осуществляется:

- привязка аккаунта по коду;
- получение напоминаний за 5 минут до начала задач;
- ежедневная утренняя сводка по задачам;
- возможность входа по Telegram-коду.

Таким образом, веб-приложение предоставляет пользователю удобные средства визуального и функционального управления личной активностью, обеспечивая своевременное информирование и гибкую настройку.

1.5 Анализ требований к программному обеспечению

1.5.1 Постановка функциональных требований

В рамках данного проекта требуется реализовать следующий набор ключевых возможностей, обеспечивающих удобство и полноту работы пользователей с системой:

- регистрация и вход пользователей по email и паролю;
- альтернативный вход через Telegram после привязки аккаунта мессенджера к своему аккаунту в системе;
- создание, редактирование и удаление задач и подзадач;
- отображение задач в виде недельного расписания;
- возможность перетаскивания задач и изменения их порядка интуитивно понятным способом;
- настройка внешнего вида интерфейса (тёмная и светлая темы);
- сохранение индивидуальных пользовательских настроек (тип темы, настройка уведомлений);
- интеграция с Telegram для уведомлений и напоминаний;
- административную панель для возможности корректировки работы системы в случае непредвиденных обстоятельств;
- функционал для просмотра статистики сервиса (кол-во регистраций, созданных задач);
- отправка утренней сводки задач (все задачи на день, разбросанные по времени) и напоминаний за 5 минут до начала в мессенджер Telegram.

Диаграмма вариантов использования, иллюстрирующая основные сценарии взаимодействия пользователя с системой и Telegram-ботом, приведена на рисунке 18.

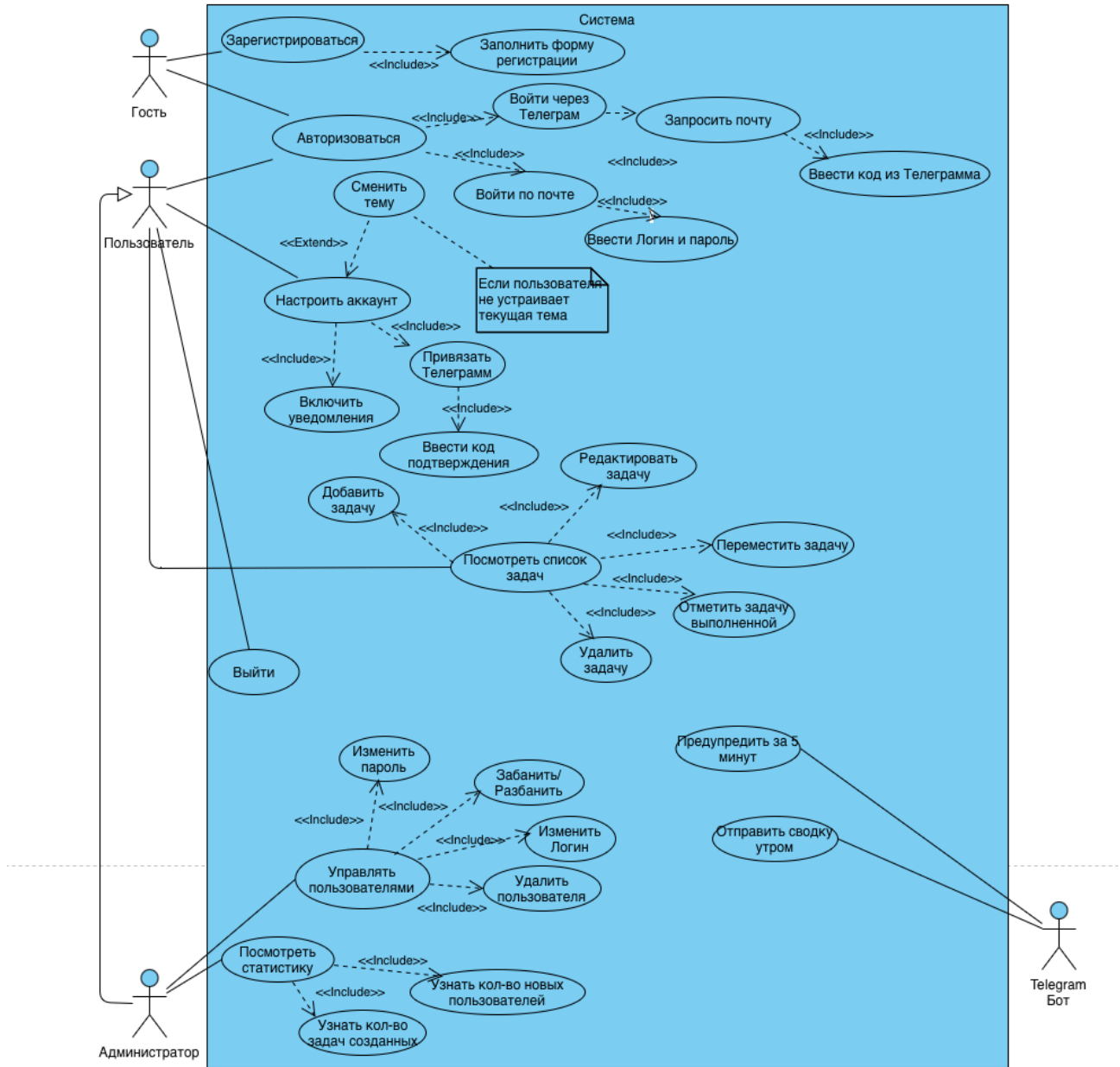


Рисунок 18 – Диаграмма вариантов использования

Процесс обработки данных и спецификация для варианта использования «Создание задачи» приведены в таблице 15.

Таблица 15 – Спецификация варианта использования «Создание задачи»

Прецедент: Создание задачи
ID: 1
Краткое описание: Пользователь создает новую задачу с нужными параметрами
Главные актеры: Пользователь

Продолжение таблицы 15

Предусловия: Пользователь авторизовался в сервисе
Основной поток: <ol style="list-style-type: none"> 1. Пользователь выбирает день и нажимает «Добавить задачу» 2. Вводит название, описание, при необходимости задаёт дату и время 3. Выбирает цвет и при желании добавляет подзадачи 4. Нажимает кнопку «Сохранить»

Процесс обработки данных и спецификация для варианта использования «Редактирование задачи» приведены в таблице 16.

Таблица 16 – Спецификация варианта использования «Редактирование задачи»

Прецедент: Редактирование задачи
ID: 2
Краткое описание: Пользователь вносит изменения в существующую задачу
Главные актеры: Пользователь
Предусловия: <ol style="list-style-type: none"> 1. Пользователь вошёл в систему 2. В системе уже есть созданные задачи
Основной поток: <ol style="list-style-type: none"> 1. Пользователь открывает задачу двойным кликом 2. Вносит нужные изменения (время, цвет, подзадачи) 3. Сохраняет изменения

Процесс обработки данных и спецификация для варианта использования «Привязка Telegram» приведены в таблице 17.

Таблица 17 – Спецификация варианта использования «Привязка Telegram»

Прецедент: Привязка Telegram
ID: 3
Краткое описание: Пользователь подключает Telegram-бота для получения уведомлений
Главные актеры: Пользователь Telegram-бот
Предусловия: <ol style="list-style-type: none"> 1. Пользователь вошёл в систему
Основной поток: <ol style="list-style-type: none"> 1. Пользователь открывает настройки и выбирает «Привязать Telegram» 2. Нажимает кнопку для перехода в Telegram-бот 3. Получает от бота код и вводит его на сайте 4. Telegram-бот подтверждает привязку

Процесс обработки данных и спецификация для варианта использования «Уведомления» приведены в таблице 18.

Таблица 18 – Спецификация варианта использования «Уведомления»

Прецедент: Уведомления
ID: 4
Краткое описание: Система автоматически отправляет напоминания пользователям
Главные актеры: Telegram-бот Пользователь
Предусловия: 1. Telegram привязан и включены соответствующие уведомления в настройках
Основной поток: 1. В 08:00 отправляется утренняя сводка с задачами на день 2. За 5 минут до начала задачи отправляется напоминание

В этом разделе были разработаны и описаны ключевые пользовательские сценарии для работы с задачами и Telegram-ботом.

1.5.2 Нефункциональные требования к клиентскому обеспечению

Клиентская часть веб-приложения должна корректно работать во всех популярных браузерах, таких как Google Chrome, Mozilla Firefox, Microsoft Edge, Safari, Opera и Яндекс.Браузер. Это обеспечивает максимальную доступность системы для широкой аудитории пользователей, вне зависимости от используемого программного обеспечения. Интерфейс обязан адаптироваться под различные размеры экранов, начиная от 360×640 пикселей (стандарт смартфонов), без появления горизонтальной прокрутки. Такая гибкость особенно важна для комфортной работы в мобильных условиях и при смене ориентации экрана. Все управляющие элементы должны располагаться одинаково на всех страницах приложения, что позволяет избежать путаницы и облегчает навигацию.

Интерфейс должен быть интуитивно понятным, единообразным и соответствовать единому стилю, что формирует положительный пользовательский опыт и повышает доверие к системе. Пользователь должен иметь возможность быстро освоить навигацию без необходимости изучать инструкции или проходить дополнительное обучение.

Минимальные технические требования к персональным компьютерам приведены в таблице 19.

Таблица 19 – Минимальные требования при работе на персональном компьютере

Категория	Минимальные требования
Операционная система	Windows 10, macOS 11 или выше
Процессор	Двухъядерный, от 1.5 ГГц
Объем оперативной памяти	Не менее 4 ГБ
Свободное место на диске	Не менее 200 МБ
Интернет-соединение	Скорость от 10 Мбит/с

Минимальные технические требования к мобильным устройствам приведены в таблице 20.

Таблица 20 – Минимальные требования при работе на мобильных и планшетных устройствах

Категория	Минимальные требования
Операционная система	Android 9.0, iOS 13 или выше
Процессор	ARMv8 или Apple A10 и выше
Объем оперативной памяти	Не менее 2 ГБ
Свободное место на устройстве	Не менее 150 МБ
Интернет-соединение	Wi-Fi или мобильная сеть со скоростью от 5 Мбит/с

В этом разделе были установлены минимальные требования для работы с сервисом, как для персональных ПК, так и для мобильных устройств.

1.5.3 Нефункциональные требования к серверной части

В процессе разработки веб-приложения для управления личными задачами и планирования времени были определены используемые технологии и инструменты, необходимые для функционирования серверной части:

- язык программирования JavaScript (среда выполнения Node.js 18);
- система управления базами данных PostgreSQL;
- web-сервер Nginx (в качестве прокси);
- фреймворк Express.js для построения API;
- интеграция с Telegram через бот на базе библиотеки node-telegram-bot-api.

Минимальные и рекомендуемые аппаратные характеристики сервера, на котором может быть размещена серверная часть приложения, представлены в таблице 21.

Таблица 21 – Минимальные требования к серверу

Категория требований	Минимальные требования	Рекомендуемые требования
Операционная система	Linux (например, Ubuntu Server, Debian или CentOS)	Linux (Ubuntu Server LTS, Debian 12)
Процессор	4 ядра, тактовая частота от 2.0 ГГц	8 ядер, частота от 2.5 ГГц и выше
Объем оперативной памяти	от 8 ГБ	от 16 ГБ
Место на жестком диске	от 100 ГБ	от 250 ГБ SSD
Скорость интернет-соединения	100 Мбит/с	Гигабитное подключение

Таким образом, были определены минимальные и рекомендованные параметры серверного оборудования, способные обеспечить стабильную работу backend-части приложения при предполагаемой пользовательской нагрузке.

1.6 Организация разработки и определение финансовых затрат.

Перед началом разработки веб-приложения для управления личными задачами и планирования времени необходимо грамотно организовать процесс и оценить требуемые ресурсы.

Для реализации проекта была выбрана гибкая методология разработки Agile [14, 15], предполагающая разделение работы на небольшие итерации (спринты), совместную работу команды и быструю обратную связь.

Работа над проектом начиналась с формирования бэклога – полного списка задач, необходимых для реализации системы. Все задачи были разбиты по крупным функциональным блокам и распределены по эпику: «Анализ», «Дизайн», «Backend» (рисунок 19).



Рисунок 19 – Часть бэклога

Каждая задача содержала краткое описание, указание исполнителя, приоритет и тип задачи, что позволяло структурировать информацию и обеспечить наглядное представление о её характеристиках.

Пример оформления одной из важных задач планировщика «Создать Telegram-бота» приведён на рисунке 20.



Рисунок 20 – Пример задачи «Регистрация бота»

После формирования бэклога задачи были распределены по двухнедельным спринтам. Это позволило последовательно организовать процесс разработки и отслеживать ход выполнения каждой итерации (рисунок 21).

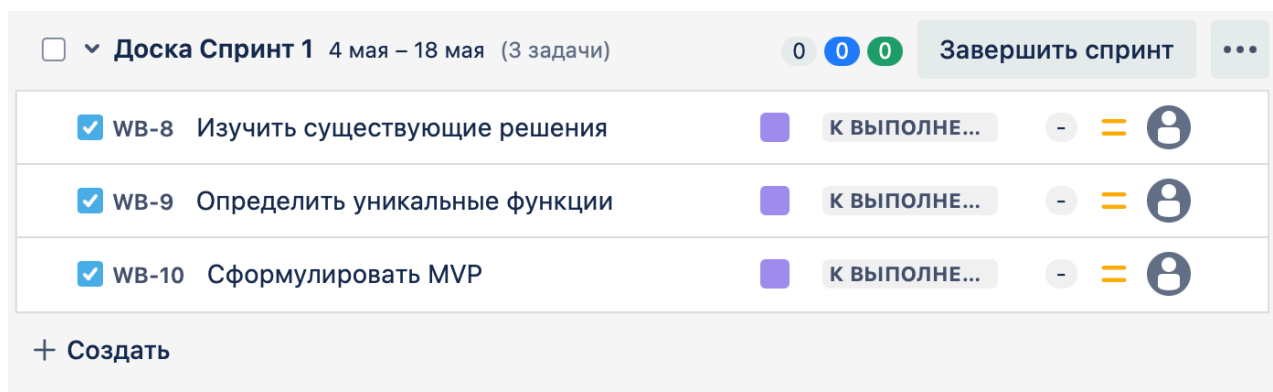


Рисунок 21 – Пример двухнедельного спринта

Jira автоматически сформировала диаграмму Ганта по всем задачам и спринтам, что позволило наглядно представить временные рамки реализации проекта (рисунок 22).

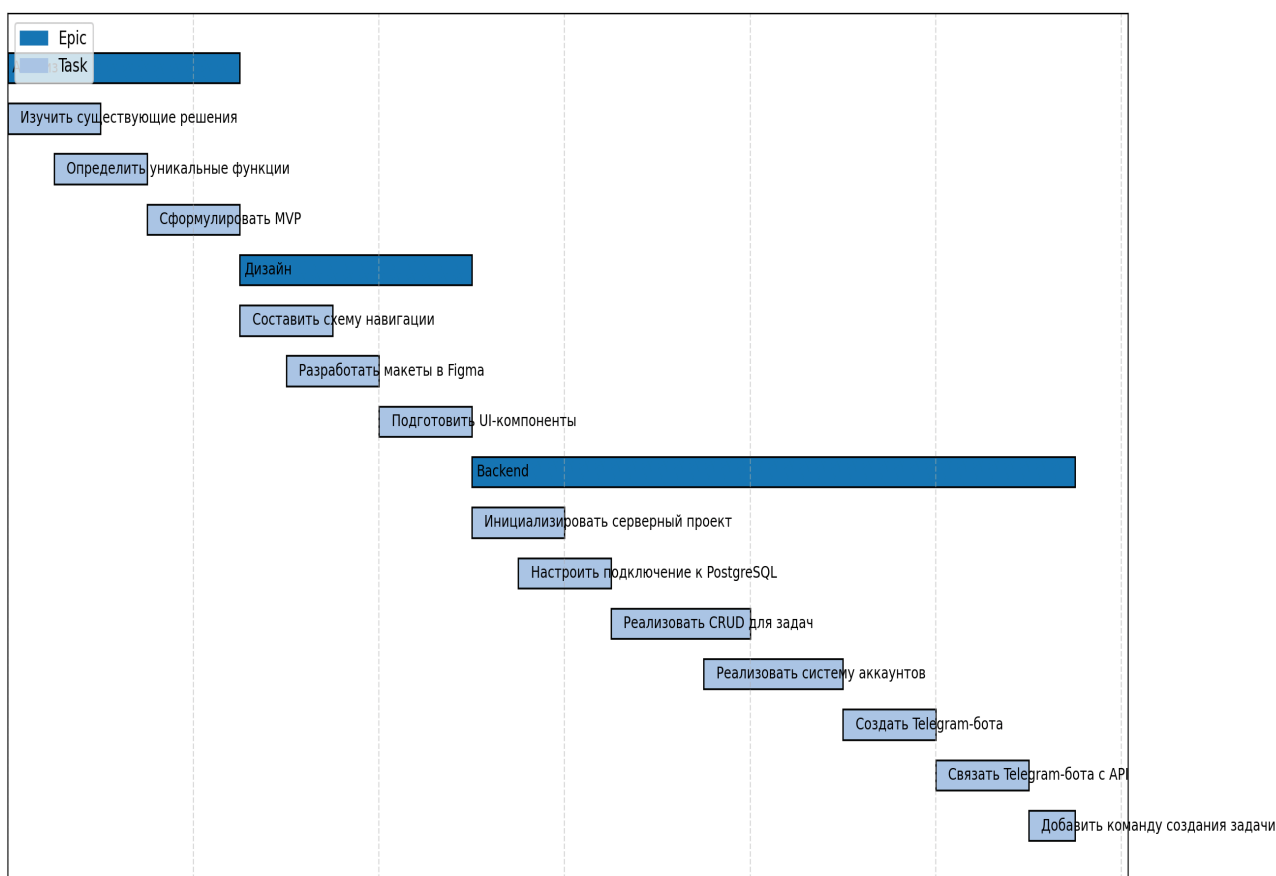


Рисунок 22 – Диаграмма Ганта

Для оценки стоимости разработки были выделены три роли, необходимые для реализации проекта: UI/UX-дизайнер, Frontend-разработчик и Backend-разработчик.

Расчет средней стоимости труда проводился на основе данных с сайта hh.ru [16] (таблица 22).

Таблица 22 – Средняя зарплата специалистов

Должность	Средняя зарплата за 180 часов	Средняя зарплата за час
UI/UX дизайнер	82 031,7 Р	455 Р
Frontend разработчик	84 213,3 Р	467 Р
Backend разработчик	151 465 Р	841 Р

На основе количества часов, затраченных на реализацию каждой задачи, и стоимости часа работы специалиста была рассчитана итоговая стоимость разработки проекта (таблица 23).

Таблица 23 – Итоговая стоимость проекта

Область работы	Стоимость работы за час	Количество часов	Итого
Дизайн и анализ	400 Р	120 ч	48 000 Р
Backend	841 Р	200 ч	168 000 Р
Frontend	420 Р	180 ч	75 600 Р
Итого	291 600 Р		

Таким образом, после планирования бюджета и распределения задач проект готов к этапу технического проектирования.

2 Проектирование сервиса

2.1 Выбор и обоснование средств проектирования

При планировании и визуализации интерфейсов веб-приложения было решено применять современные графические и модельные инструменты, позволяющие объединить дизайн и логику в единую систему. Основу проектной работы составили Figma и язык UML [8, 9].

Для создания интерфейсов выбрана Figma [7] – это облачный инструмент, позволяющий рисовать макеты страниц, настраивать кликабельные прототипы и формировать систему компонентов. Преимущество Figma заключается не только в простом интерфейсе, но и в том, что разработчик может сразу видеть CSS-стили каждого элемента. Это значительно сокращает разрыв между дизайнерской частью и реализацией на фронтенде.

Также в макетах удобно использовать повторяющиеся элементы, например, карточки задач или кнопки. Один раз настроенный компонент можно вставлять на любые экраны – и при изменении шаблона все копии обновляются автоматически. Это снижает риск расхождений между экранами.

Для отображения внутренней архитектуры и логики системы применялись UML-диаграммы [9]. Они позволяют описать, как между собой взаимодействуют компоненты, какие данные хранятся, какие процессы происходят. Это особенно удобно при планировании API, работы с базой данных и Telegram-ботом.

UML был выбран из-за своей универсальности – он используется в большинстве крупных ИТ-проектов и понятен как разработчикам, так и тем, кто не пишет код. С его помощью можно описать и логику аутентификации, и структуру базы, и поведение задач во времени.

Таким образом, сочетание Figma и UML обеспечило удобную базу как для визуального, так и для логического проектирования интерфейсов и функциональности сервиса.

2.2 Проектирование базы данных

Построена логическая модель данных (рисунок 23), отражающая ключевые объекты и их взаимосвязи в системе управления личными задачами. Модель независима от конкретной СУБД и служит основой для последующей физической реализации в PostgreSQL [12].

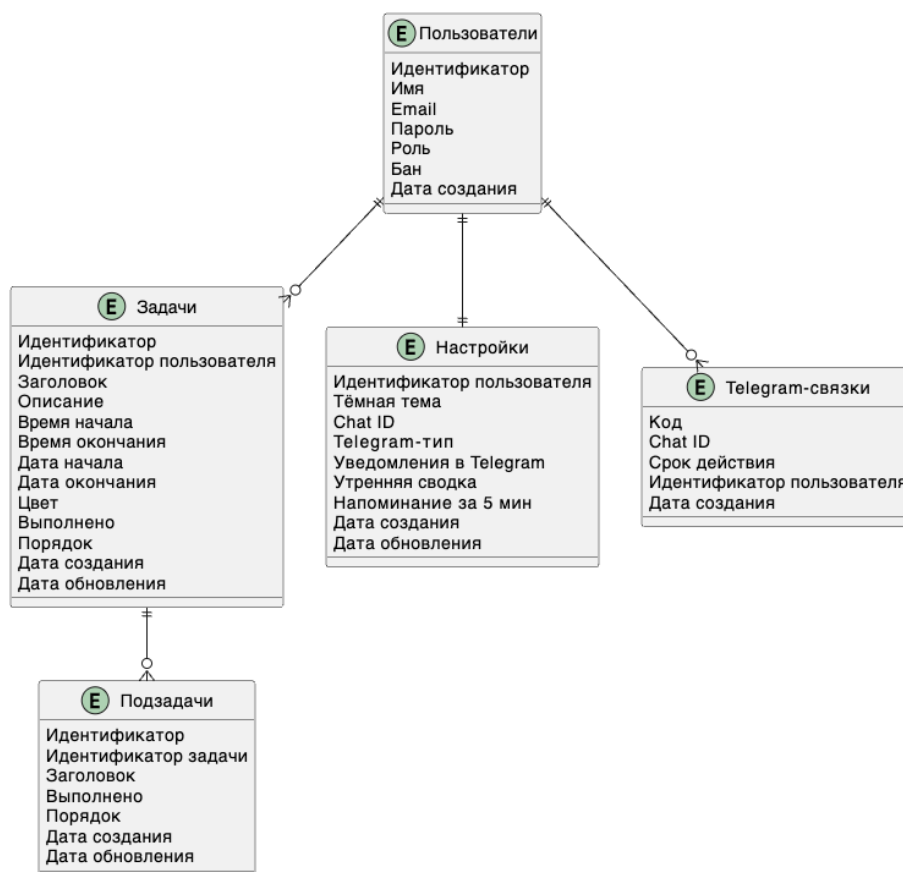


Рисунок 23 – Логическая схема базы данных

Спроектированная схема обеспечивает хранение информации о пользователях, их задачах, настройках и механизме авторизации через Telegram. Ниже приведено краткое описание основных сущностей.

Таблица 24 – Описание сущностей

Название сущности	Описание
Пользователи	Информация о зарегистрированных пользователях: уникальный идентификатор, имя, email, хеш пароля, метка времени создания записи, роль пользователя, заблокирован или нет

Продолжение таблицы 24

Задачи	Основная сущность для хранения задач: заголовок, описание, цвет, дата и время начала, дата и время окончания, статус завершения, порядок
Подзадачи	Элементы, связанные с конкретной задачей: заголовок подзадачи, флаг выполнения, порядок отображения
Настройки	Пользовательские параметры: выбор темы (светлая/тёмная), включение уведомлений в Telegram (утренняя сводка, напоминание), привязка chat_id и username
Telegram-связки	Коды и метаданные для привязки аккаунта к боту: уникальный код, chat_id, время истечения, ссылка на пользователя

Данная структура позволяет хранить все необходимые данные о пользователях, их задачах и предпочтениях, а также организовать механизм уведомлений и входа через Telegram.

2.3 Проектирование архитектуры

В процессе проектирования серверной части приложения используется диаграмма классов, позволяющая наглядно отобразить основные компоненты системы и их взаимодействие. Каждый класс отражает ответственную зону: создание и инициализацию базы данных, работу с таблицами через репозитории, бизнес-логику в сервисах, маршрутизацию HTTP-запросов в обработчиках, интеграцию с Telegram и планирование фоновых задач. Диаграмма демонстрирует, какие атрибуты и методы есть у каждого класса, и каким образом объекты обмениваются данными.

Использование диаграммы классов способствует разделению ответственности, упрощает масштабирование и обеспечивает прозрачность архитектуры. Она позволяет разработчикам и новым участникам команды быстро разобраться в потоке обработки запроса от приёма до отправки уведомления, стандартизировать подходы к работе с данными и снизить вероятность ошибок при доработках.

Для серверной части приложения разработана диаграмма классов, представленная на рисунке 24.

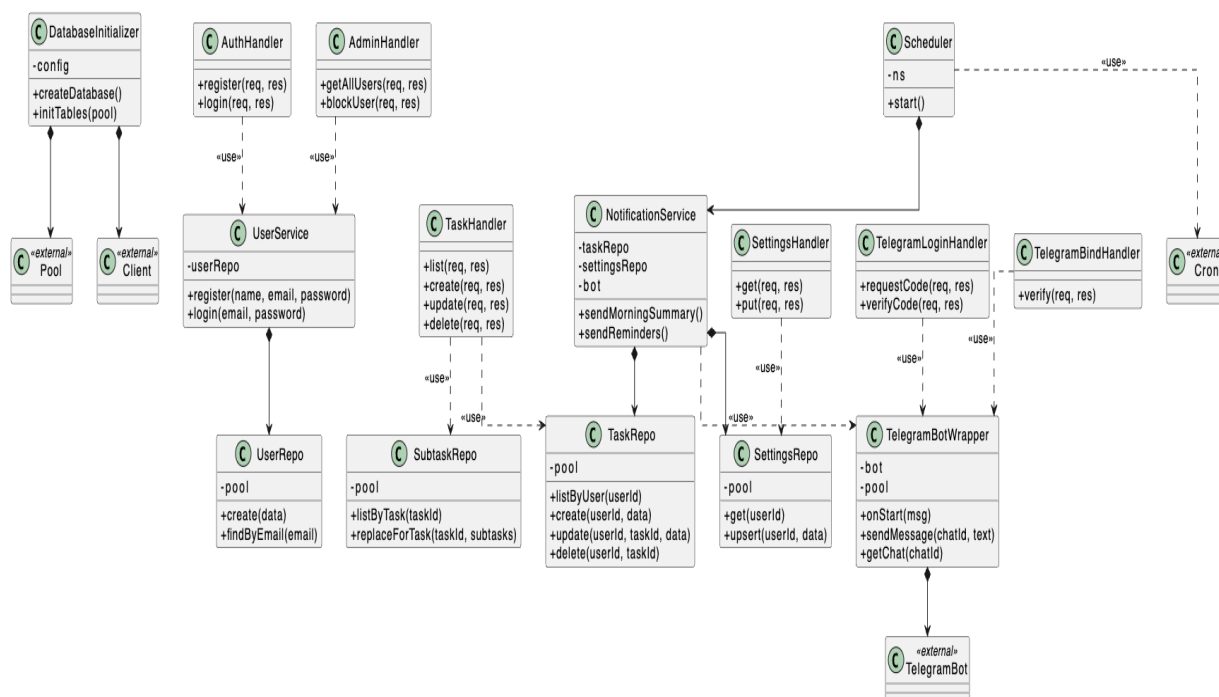


Рисунок 24 – Диаграмма классов

Краткое описание классов приведено в таблице 25.

Таблица 25 – Описание классов

Название класса	Описание
DatabaseInitializer	Создаёт базу данных при первом запуске и инициализирует все необходимые таблицы
UserRepo	Выполняет операции создания и поиска пользователей в таблице users
TaskRepo	Реализует CRUD-методы для работы с задачами в таблице tasks
SubtaskRepo	Управляет подзадачами: получение списка и полная замена набора подзадач для конкретной задачи
SettingsRepo	Получает и сохраняет пользовательские настройки (тема, Telegram-уведомления, утренняя сводка, напоминания)
UserService	Реализует регистрацию и вход пользователя: хеширует пароль, проверяет учётные данные, генерирует JWT-токен
NotificationService	Формирует тексты уведомлений (утренняя сводка и напоминания), выбирает задачи из БД и отправляет их через TelegramBotWrapper
TelegramBotWrapper	Оборачивает работу с node-telegram-bot-api: принимает команду /start, генерирует коды и отправляет сообщения

Продолжение таблицы 25

Scheduler	Настраивает cron-задачи для ежедневной утренней сводки в 08:00 и ежеминутной проверки напоминаний за 5 минут
AuthHandler	Обрабатывает HTTP-запросы /api/register и /api/login: принимает данные, вызывает UserService и возвращает ответ
TelegramLoginHandler	Реализует эндпоинты /api/login/telegram и /api/login/telegram/verify для входа по коду из Telegram
TelegramBindHandler	Обрабатывает /api/telegram/verify-code: верифицирует код, привязывает Telegram-чат к учётной записи и обновляет настройки
TaskHandler	Обеспечивает REST-эндпоинты /api/tasks (GET, POST, PUT, DELETE), проводит аутентификацию по JWT и манипулирует задачами
SettingsHandler	Реализует /api/settings (GET, PUT): возвращает и сохраняет пользовательские настройки через SettingsRepo.
AdminHandler	Обрабатывает административные эндпоинты: список пользователей с подсчётом задач, бан/разбан, удаление, изменение email/пароля/роли и статистика по пользователям и задачам. Доступен только для admin-пользователей по JWT-токену

Таким образом, представленная диаграмма классов охватывает все ключевые аспекты архитектуры серверной части приложения. Чёткое распределение ответственности между классами упрощает сопровождение и развитие системы, а также обеспечивает удобную основу для внедрения новых функций и интеграций.

2.4 Проектирование интерфейса

Перед реализацией визуальной части веб-приложения было выполнено предварительное проектирование интерфейса [19], включающее построение схемы приложения, разработку wireframe-макетов и схемы взаимодействий между экранами.

На первом этапе проектирования была составлена карта приложения (рисунок 25), на которой обозначены основные экраны, их функциональное

наполнение и взаимосвязи. Такая схема позволяет увидеть структуру интерфейса и все возможные переходы между экранами.

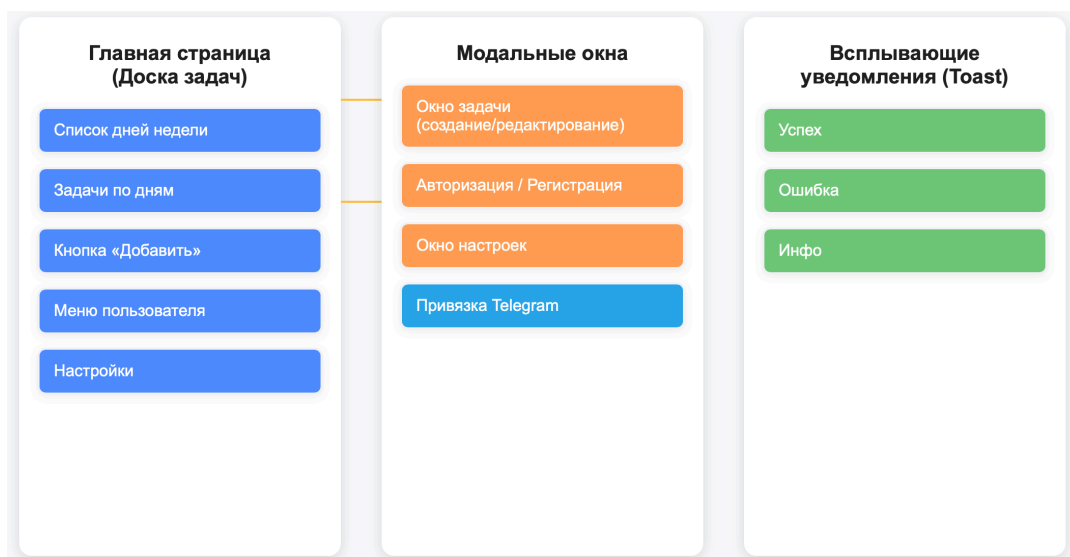


Рисунок 25 – Карта приложения

Далее для каждого основного экрана были созданы wireframe-прототипы. Каркасы позволяют быстро согласовать логику интерфейса до отрисовки финального дизайна. Особое внимание было уделено главной странице (рисунок 26).



Рисунок 26 – Пример wireframe-прототипа Главной страницы

После согласования wireframe был разработан пользовательский сценарий (user flow), который показывает возможные переходы между экранами при выполнении различных действий (рисунок 27).

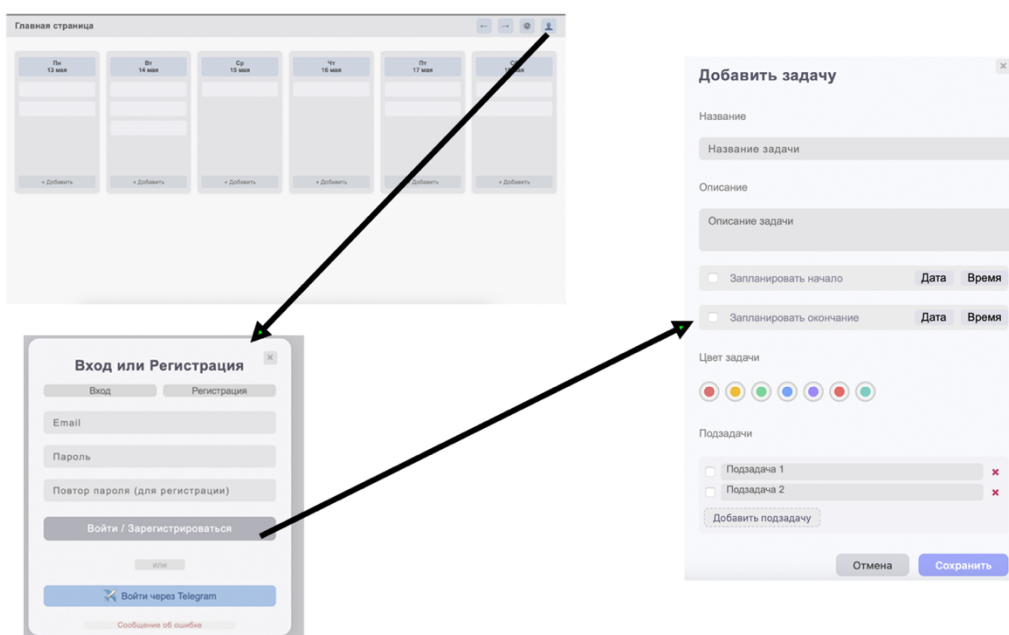


Рисунок 27 – Карта взаимодействия пользователя с интерфейсом приложения

После утверждения структуры и логики интерфейса на основе wireframe-макетов был реализован финальный дизайн интерфейса с учетом современных требований к usability и адаптивности.

Приложение поддерживает как светлую, так и тёмную тему, все элементы имеют интуитивно понятное расположение, обеспечен быстрый доступ к основным функциям.

3 Реализация сервиса

3.1 Обоснование средств разработки

Для создания серверной части приложения использовался стек технологий, основанный на Node.js [10] и Express [11]. Такой выбор обусловлен широкой поддержкой языка JavaScript, высокой производительностью и большим количеством готовых библиотек, упрощающих разработку и сопровождение серверных решений. В качестве среды разработки применялся редактор Visual Studio Code, который обеспечивает удобство работы с кодом, имеет встроенную поддержку работы с системой контроля версий Git и множество расширений для повышения продуктивности.

В качестве системы управления базой данных был выбран PostgreSQL. Это современная реляционная СУБД с открытым исходным кодом, отличающаяся надежностью, гибкостью и широкими возможностями для работы с большими объемами данных. Для взаимодействия Node.js с PostgreSQL использовалась библиотека pg, предоставляющая эффективный и безопасный доступ к данным.

Клиентская часть приложения реализована с помощью React. Данный фреймворк выбран из-за своей популярности, высокой скорости работы, удобства в построении компонентной структуры интерфейса и активного сообщества. Для оформления внешнего вида и быстрой стилизации компонентов применялись библиотеки Tailwind CSS и React Icons, что позволило создавать современный и адаптивный интерфейс.

Выбор инструментов для реализации как серверной, так и клиентской части был сделан с учетом их актуальности, качества документации и совместимости между собой. Такой подход позволил обеспечить стабильную, быструю и удобную в использовании систему для управления задачами и планирования времени.

3.2 Реализация архитектуры приложения

Архитектура приложения для управления личными задачами и планирования времени построена на принципе клиент-серверной модели. В системе выделяются три основных компонента: устройство пользователя (например, компьютер, ноутбук или смартфон), серверное приложение, написанное на Node.js с использованием фреймворка Express, и сервер базы данных PostgreSQL.

Пользователь взаимодействует с системой через современный веб-браузер. Все действия пользователя – создание, редактирование, удаление задач, а также работа с настройками – осуществляются в интерфейсе клиентской части приложения, реализованной на React. Веб-клиент отправляет запросы к серверу по защищённому протоколу HTTP(S).

Серверная часть отвечает за обработку запросов, выполнение бизнес-логики, управление авторизацией, обработку и валидацию данных, а также за организацию связи с базой данных. Все данные о пользователях и их задачах хранятся на сервере базы данных PostgreSQL, доступ к которой осуществляется посредством стандартных SQL-запросов через библиотеку pg.

Взаимодействие между компонентами организовано через сеть с использованием TCP/IP. Сервер и база данных обычно размещаются на одном сервере или в пределах одной локальной сети для повышения безопасности и скорости обмена данными. Исходный код серверной части размещён в Приложении Б, а клиентская часть опубликована в открытом доступе на GitHub [21].

Диаграмма развертывания представлена на рисунке 28.

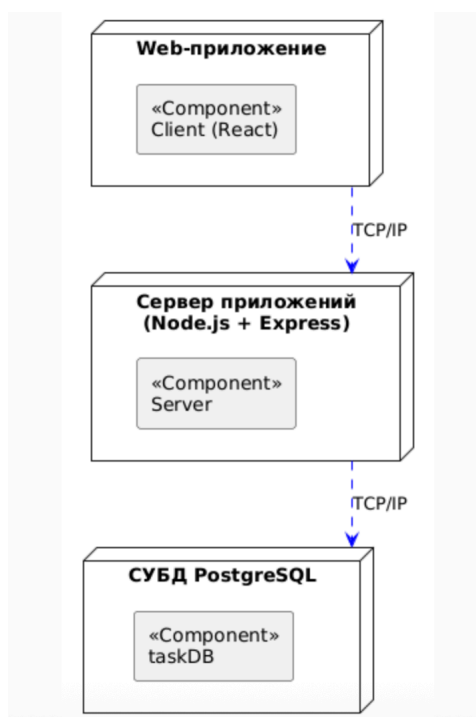


Рисунок 28 – Диаграмма развертывания

Таким образом, архитектура приложения обеспечивает надёжную и масштабируемую работу системы за счёт разделения на клиентскую и серверную части, а также использования современных и проверенных технологий.

3.3 Реализация базы данных

Физическая модель базы данных для веб-приложения управления личными задачами и планирования времени была разработана с использованием системы управления базами данных PostgreSQL. Модель построена на основе предварительно сформированной логической структуры, учитывающей основные сущности приложения. Физическая схема БД представлена на рисунке 29.

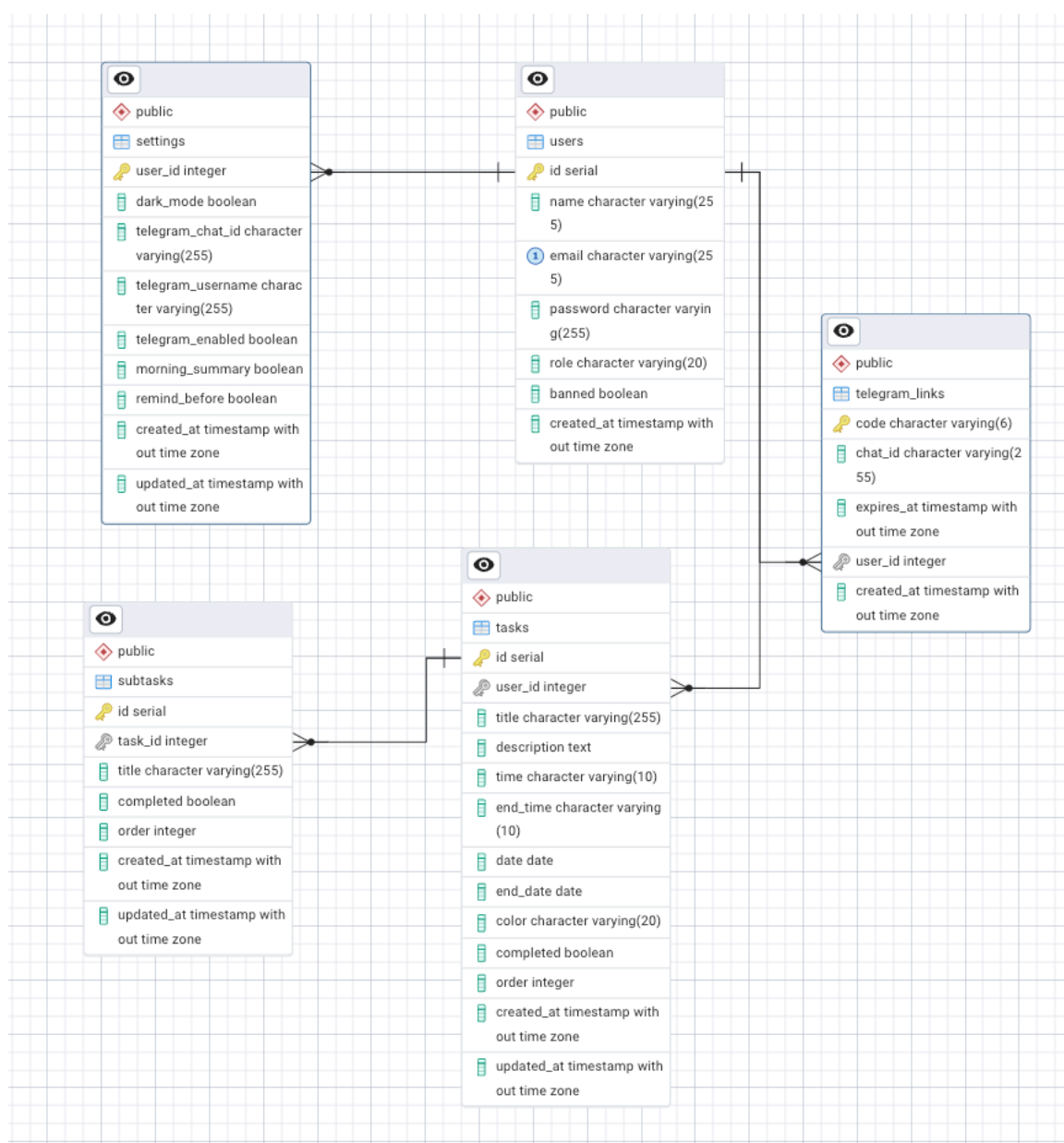


Рисунок 29 – Физическая схема БД

В модели реализованы следующие основные связи между сущностями в таблице 26.

Таблица 26 – Связи между сущностями

Таблица 1	Таблица 2	Связь
users	tasks	Один-ко-многим
tasks	subtasks	Один-ко-многим
users	settings	Один-к-одному
users	telegram_links	Один-ко-многим

База данных спроектирована так, чтобы обеспечить быстрое и безопасное хранение пользовательских данных, задач, подзадач, а также различных настроек. Для поддержания целостности данных использованы внешние ключи между таблицами, а также реализованы индексы на основных полях (например, email пользователя, идентификатор задачи и подзадачи), что положительно сказывается на скорости поиска и обновления информации.

Такая организация базы данных позволяет эффективно управлять задачами и их статусами, реализовать различные пользовательские настройки, а также поддерживать функции напоминаний и интеграцию с внешними сервисами, например, Telegram.

3.4 Реализация интерфейса

Интерфейс веб-приложения разрабатывался с акцентом на простоту, удобство и интуитивную понятность для пользователя. Все основные элементы управления расположены логично и доступны с первых секунд работы. Для реализации внешнего вида приложения использовались современные UI-компоненты, привычные большинству пользователей благодаря их широкому распространению в других сервисах.

Вход в систему (рисунок 30) реализован через универсальную форму авторизации, где пользователь может зарегистрироваться или войти по электронной почте и паролю. Помимо этого, присутствует возможность быстрой авторизации через мессенджер Telegram, что реализовано с помощью отправки

кода в чат с ботом. Такой подход обеспечивает как привычные, так и альтернативные варианты входа, делая процесс максимально гибким и быстрым.

Минималистичный дизайн и современная структура интерфейса позволяют сосредоточиться на основных функциях приложения – планировании и управлении личными задачами. Это снижает время на освоение сервиса и позволяет сразу приступить к использованию основных возможностей. Таким образом, пользователь получает инструмент для эффективной организации своего времени без лишних отвлекающих элементов.

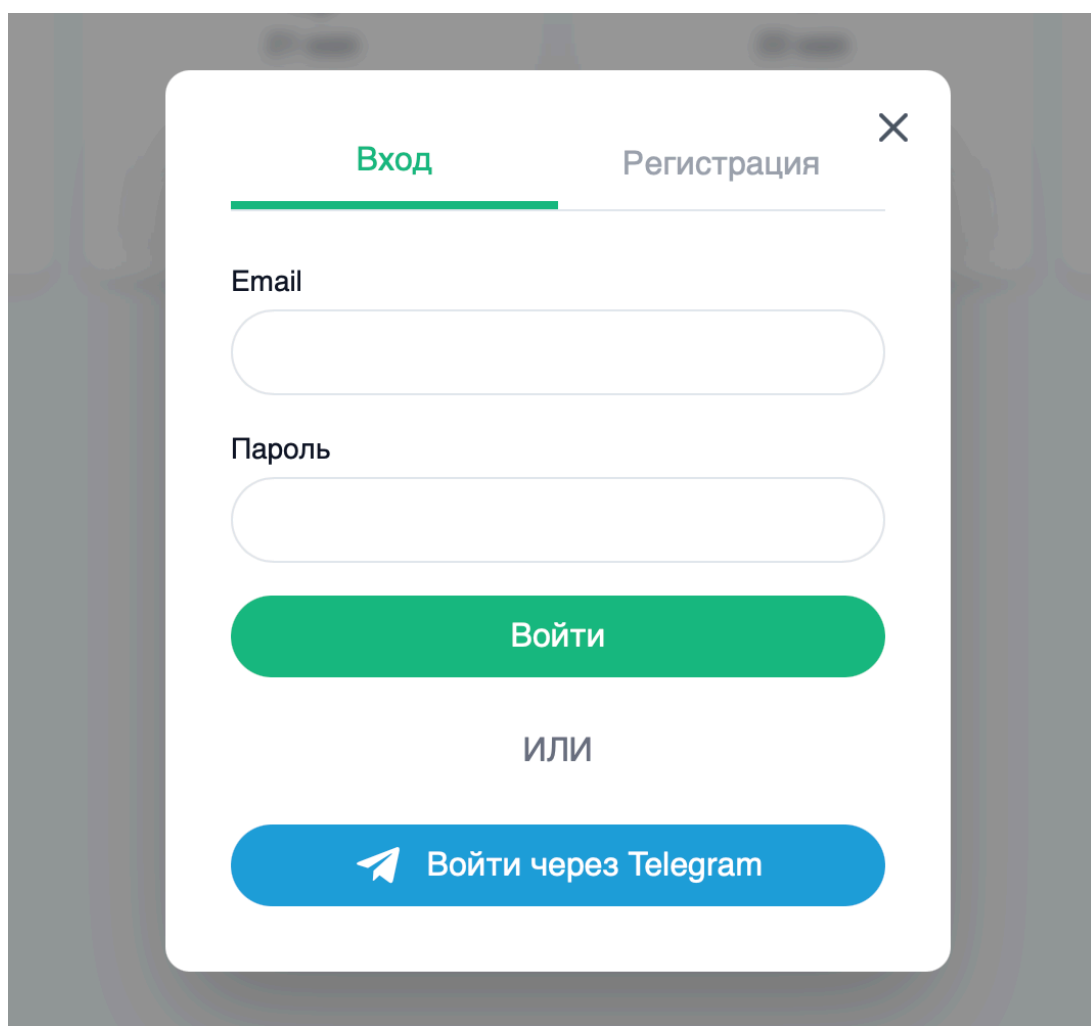


Рисунок 30 – Окно авторизации в системе

Таким образом, реализованный интерфейс сочетает в себе лаконичный дизайн, удобную навигацию и функциональность, необходимые для эффективного взаимодействия пользователя с системой. Он остаётся визуально простым, но при этом не теряет в информативности. Навигационные элементы

расположены логично, что облегчает освоение даже для новых пользователей. Все основные функции доступны без лишних действий, что делает работу с интерфейсом быстрой и понятной.

3.4.1 Интерфейс для авторизованного пользователя

После входа в систему пользователь получает доступ к основному интерфейсу приложения для планирования и управления своими задачами. Главная страница построена так, чтобы все необходимые инструменты были под рукой, а навигация оставалась простой и понятной. В верхней части интерфейса располагается шапка приложения с логотипом и меню для перехода к настройкам профиля. Главная страница и шапка продемонстрированы на рисунке 31.

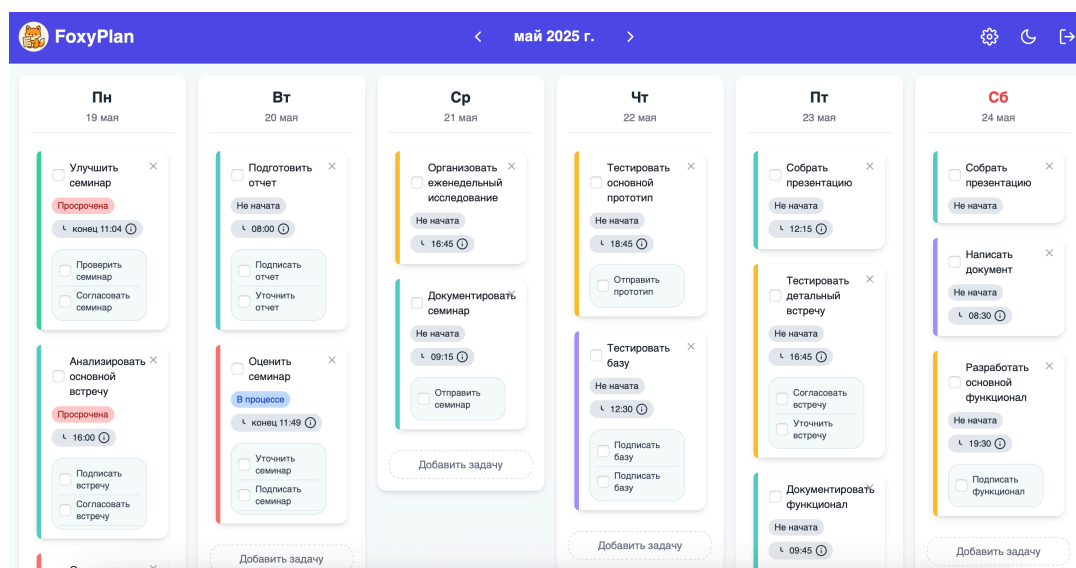


Рисунок 31 – Главная страница и шапка

Основная рабочая область разделена на несколько ключевых блоков. В центре размещается интерактивная доска с задачами, отображающая неделю – по дням. Пользователь может добавлять новые задачи в любой день, редактировать их, перемещать между днями с помощью drag-and-drop, а также отмечать задачи как выполненные. Пример работы drag-and-drop показан на рисунке 32.

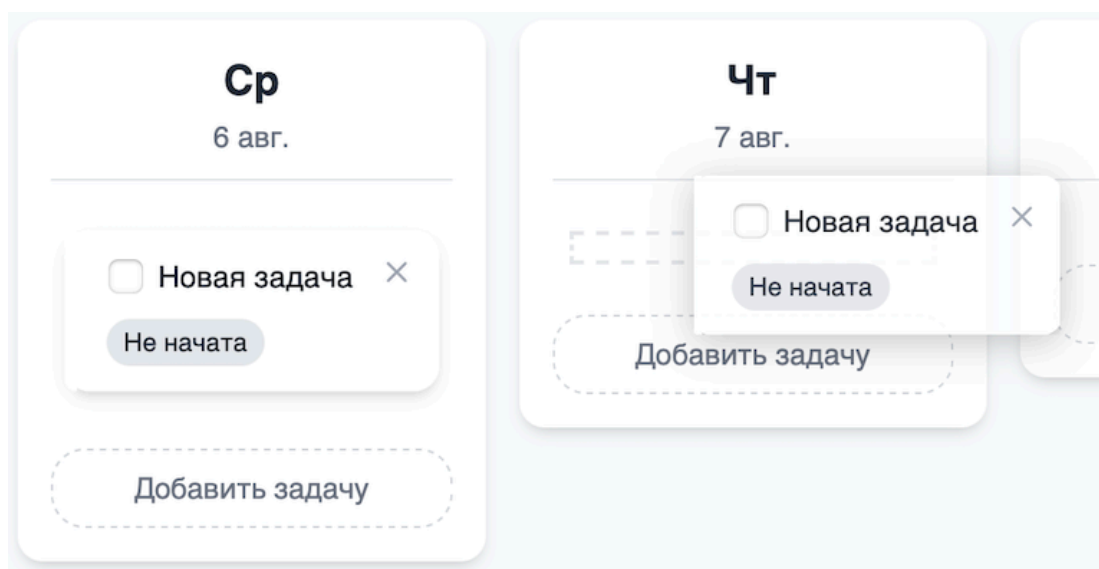


Рисунок 32 – Функция drag-and-drop

Для каждой задачи доступен быстрый просмотр ключевой информации: название, дата и время, цветовая метка, статус выполнения и наличие подзадач (рисунок 33).

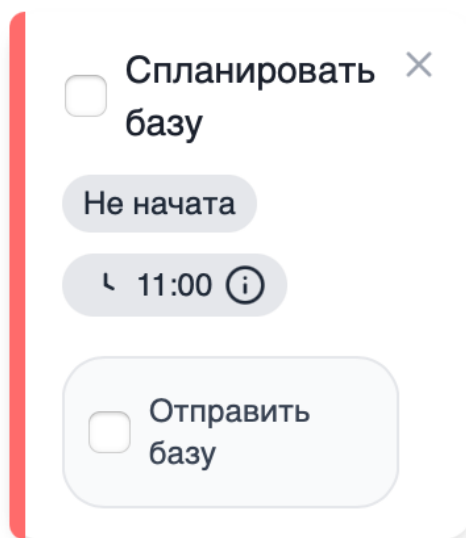


Рисунок 33 – Информация о задаче

При необходимости можно открыть задачу для более детального редактирования: изменить описание, добавить или удалить подзадачи, назначить сроки начала и окончания, а также выбрать цвет для визуальной организации (рисунок 34).

Редактировать задачу

Название
Улучшить предварительный кампанию

Описание
– Составить план
– Согласовать сроки
– Определить метрики

☒ Запланировать начало

Дата: 20.07.2025 Время: 10:00

☒ Запланировать окончание

Дата: 22.07.2025 Время: 11:35

Цвет задачи
● ● ● ● ● ● ●

Подзадачи

- ☐ Уточнить кампанию ✕
- ☐ Проверить кампанию ✕

Рисунок 34 – Редактирование задачи

В настройках профиля доступна интеграция с Telegram-ботом: можно подключить свой аккаунт и получать напоминания о важных задачах, а также утреннюю сводку прямо в мессенджер. Здесь же можно выбрать тему оформления (светлую или тёмную), управлять уведомлениями и другими персональными параметрами.

Все изменения, внесённые пользователем, сохраняются автоматически и применяются при последующих входах в систему (рисунок 35).

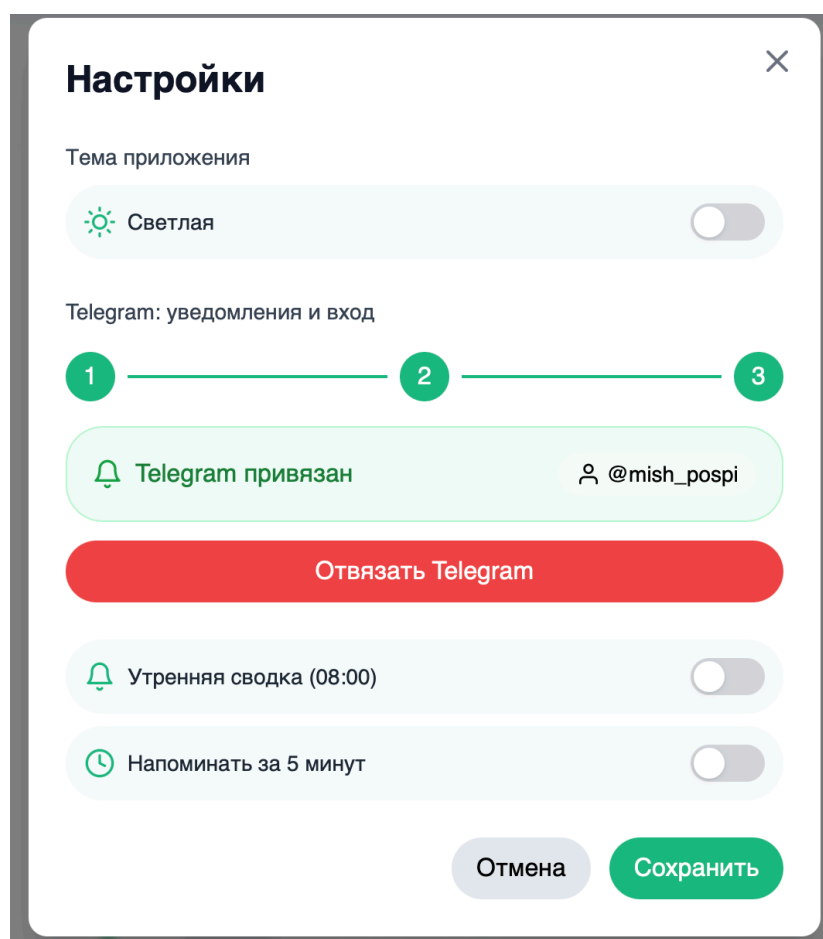


Рисунок 35 – Настройки

Таким образом, интерфейс приложения позволяет удобно планировать свой день и контролировать выполнение дел, используя современные визуальные и навигационные решения. Всё управление осуществляется быстро и просто, что обеспечивает комфортную работу с приложением даже при большом количестве задач. Пользователь получает доступ ко всем основным функциям без лишних переходов, что особенно важно при работе в динамичных условиях. Адаптивность интерфейса под различные устройства делает возможным использование приложения как на компьютере, так и на мобильных платформах.

3.4.2 Интерфейс администратора

После входа в систему с правами администратора пользователь получает доступ к расширенному интерфейсу управления, предназначенному для работы

с пользователями и статистикой. Главная страница административной панели объединяет ключевые функции для модерации и аналитики в одном удобном интерфейсе. Навигация и визуальные элементы выполнены в едином стиле с пользовательской версией, что обеспечивает знакомство и простоту восприятия. Пример интерфейса административной панели приведён на рисунке 36.

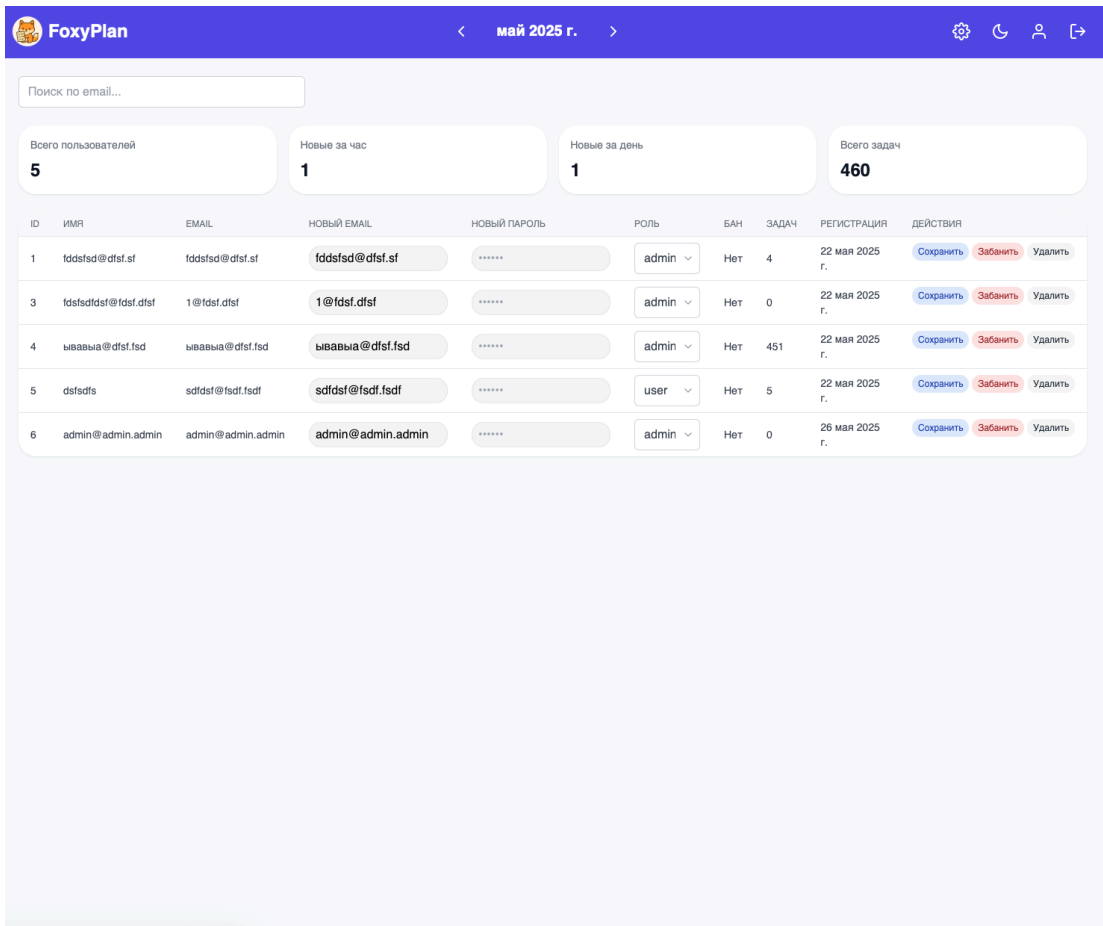


Рисунок 36 – Интерфейс административной панели

В верхней части страницы располагается шапка с текущей датой, логотипом и доступом к настройкам и выходу из системы. Панель также поддерживает переключение между светлой и тёмной темами оформления.

Основная рабочая область административной панели состоит из двух блоков.

Статистика отображает ключевые метрики системы, такие как общее количество пользователей, количество новых регистраций за последний час и

сутки, а также общее число созданных задач. Пример блока статистики показан на рисунке 37.

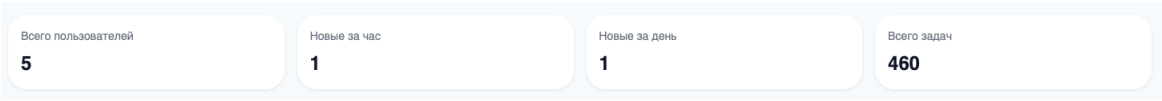


Рисунок 37 – Блок статистики

Управление пользователями – представляет собой таблицу, содержащую информацию обо всех зарегистрированных пользователях.

Для каждого пользователя отображаются ID, имя, email, дата регистрации, количество задач, статус блокировки и роль. Кроме того, доступны поля для редактирования email, пароля и роли. Все действия (сохранение изменений, бан/разбан, удаление) выполняются напрямую из таблицы. Пример интерфейса управления пользователями показан на рисунке 38.

ID	ИМЯ	EMAIL	НОВЫЙ EMAIL	НОВЫЙ ПАРОЛЬ	РОЛЬ	БАН	ЗАДАЧ	РЕГИСТРАЦИЯ	ДЕЙСТВИЯ
1	fddsfdd@dfsf.sf	fddsfdd@dfsf.sf	fddsfdd@dfsf.sf	*****	admin	Нет	4	22 мая 2025 г.	Сохранить Забанить Удалить
3	fdsfddfdsf@fdsf.dfsf	1@fdsf.dfsf	1@fdsf.dfsf	*****	admin	Нет	0	22 мая 2025 г.	Сохранить Забанить Удалить
4	ывавыя@dfsf.fsd	ывавыя@dfsf.fsd	ывавыя@dfsf.fsd	*****	admin	Нет	451	22 мая 2025 г.	Сохранить Забанить Удалить
5	dsfsdfs	sdfdsf@fsdf.fsd	sdfdsf@fsdf.fsd	*****	user	Нет	5	22 мая 2025 г.	Сохранить Забанить Удалить
6	admin@admin.admin	admin@admin.admin	admin@admin.admin	*****	admin	Нет	0	26 мая 2025 г.	Сохранить Забанить Удалить

Рисунок 38 – Управление пользователями

Таблица также содержит поле поиска по email, что упрощает работу с большим количеством записей.

При редактировании данных администратора информируют всплывающие уведомления (toast-сообщения) о результате операции: успешное сохранение, ошибка, отсутствие изменений и др.

Таким образом, административная панель предоставляет полный набор инструментов для контроля состояния системы и управления пользователями. Интерфейс реализован с учётом эргономики и позволяет быстро выполнять повседневные задачи модерации.

4 Тестирование сервиса

4.1 Функциональное тестирование

В процессе проверки работоспособности веб-приложения был использован метод функционального тестирования «черного ящика» [17]. Этот подход заключается в проверке выполнения заявленных функций без анализа внутреннего кода системы. Такой метод позволяет оценить, насколько корректно система реагирует на различные пользовательские действия и соответствует ли заявленным требованиям.

На основании анализа пользовательских сценариев и основных функций сервиса был составлен перечень тестовых случаев, отражающих ключевые действия в системе. Для каждого из них были проведены проверки успешного выполнения операций, что позволило выявить возможные ошибки на стороне интерфейса и серверной логики.

Ниже приведены основные сценарии тестирования приложения для управления задачами и планирования времени (таблица 27).

Таблица 27 – Сценарии тестирования сервиса

№	Название тестового случая	Тест	Ожидаемый результат	Полученный результат
1	Регистрация пользователя	На странице регистрации ввести email, имя и пароль или войти через Telegram	Пользователь зарегистрирован или авторизован	Тест пройден (рисунок В.1)
2	Вход в аккаунт	На странице авторизации ввести email и пароль или войти через Telegram	Пользователь успешно вошёл в систему	Тест пройден (рисунок В.2)
3	Создание задачи	Добавить новую задачу на доске, указав параметры	Задача появляется на доске	Задача создана (рисунок В.3)

Продолжение таблицы 27

4	Добавление подзадач	В редактировании задачи добавить одну или несколько подзадач	Подзадачи отображаются под основной задачей	Подзадачи добавлены (рисунок В.4)
5	Редактирование задачи	Изменить название, дату, цвет, описание или другие параметры	Обновлённые данные сохранены и отображаются	Изменения сохранены (рисунок В.5)
6	Выполнение задачи	Отметить основную задачу или подзадачу как выполненную	Статус задачи/подзадачи изменён на выполнено	Статус изменён (рисунок В.6)
7	Перемещение задачи	Перетащить задачу на другой день или изменить порядок внутри дня	Порядок задач обновлён	Порядок изменён (рисунок В.7)
8	Удаление задачи	Удалить задачу с помощью соответствующей кнопки	Задача исчезает с доски	Задача удалена (рисунок В.8)
9	Привязка Telegram-аккаунта	Привязать Telegram в настройках и активировать уведомления	Telegram успешно привязан	Telegram привязан (рисунок В.9)
10	Получение утренней сводки	Получить уведомление о задачах в Telegram при активированной опции	Сводка приходит в Telegram	Сводка получена (рисунок В.10)
11	Напоминание о задаче	Получить напоминание в Telegram за 5 минут до начала задачи	Напоминание приходит вовремя	Уведомление получено (рисунок В.11)

Проведение указанных тестов показало, что система в полной мере выполняет заявленные функции и обеспечивает корректную работу с задачами, подзадачами, а также взаимодействие с Telegram для напоминаний и уведомлений.

4.2 Модульное тестирование

Для проверки корректности работы отдельных компонентов серверной части приложения было проведено модульное тестирование. В качестве среды тестирования использовался фреймворк Jest, а для тестирования HTTP-запросов – библиотека Supertest.

Основное внимание было уделено тестированию модуля аутентификации. Были реализованы следующие тесты:

- регистрация нового пользователя;
- попытка повторной регистрации с тем же email;
- вход с корректными данными;
- отказ при входе с некорректным паролем.

Пример модульного теста регистрации пользователя приведен в листинге 1.

Листинг 1 – Тест регистрации

```
request = require('supertest');
const app = require('../app'); // ← теперь правильный путь
describe('Auth API', () => {
  test('Регистрация нового пользователя', async () => {
    const res = await request(app)
      .post('/api/register')
      .send({ name: 'Test User', email: 'test@example.com',
password: 'Test1234!' });
    expect(res.statusCode).toBe(200);
    expect(res.body).toHaveProperty('id');
    expect(res.body).toHaveProperty('email', 'test@example.com');
  });
  test('Ошибка при повторной регистрации с тем же email', async ()
=> {
    await request(app)
      .post('/api/register')
      .send({ name: 'Test User', email: 'test@example.com',
password: 'Test1234!' });
    const res = await request(app)
      .post('/api/register')
      .send({ name: 'Test User', email: 'test@example.com',
password: 'Test1234!' });
    expect(res.statusCode).toBe(409);
    expect(res.body).toHaveProperty('error');
  });
  test('Вход с верными данными', async () => {
```

```

    await request(app)
      .post('/api/register')
      .send({ name: 'Test User', email: 'test2@example.com',
password: 'Test1234!' });
    const res = await request(app)
      .post('/api/login')
      .send({ email: 'test2@example.com', password: 'Test1234!'
});
    expect(res.statusCode).toBe(200);
    expect(res.body).toHaveProperty('token');
    expect(res.body).toHaveProperty('user');
  });
  test('Вход с неверным паролем', async () => {
    const res = await request(app)
      .post('/api/login')
      .send({ email: 'test2@example.com', password:
'WrongPassword' });
    expect(res.statusCode).toBe(400);
    expect(res.body).toHaveProperty('error');
  });
});
});

```

При прохождении всех тестов в консоли отображается сообщение об успешном выполнении (рисунок 39). Это подтверждает, что функциональность аутентификации работает корректно при типовых и граничных сценариях.

```

venvmihailgolovinov@MacBook-Air-Mihail: foxyplan-server % npm test
> foxyplan-server@1.0.0 test
> jest

watchman warning: opendir(/Users/mihailgolovinov/Library/Containers/com.apple.corerecents.recents/Data/Library/Recents) -> Operation not permitted. Marking this portion of the tree deleted
To clear this warning, run:
  watchman watch-del '/Users/mihailgolovinov'; watchman watch-project '/Users/mihailgolovinov'

PASS tests/auth.test.js
  Auth API
    ✓ Регистрация нового пользователя (19 ms)
    ✓ Ошибка при повторной регистрации с тем же email (5 ms)
    ✓ Вход с верными данными (3 ms)
    ✓ Вход с неверным паролем (1 ms)

Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        0.332 s, estimated 1 s
Ran all test suites.
venvmihailgolovinov@MacBook-Air-Mihail: foxyplan-server %

```

Рисунок 39 – Успешное выполнение теста

Таким образом, модульное тестирование позволяет убедиться в надёжности критически важных участков серверного кода и своевременно выявлять логические ошибки.

4.3 Анализ метрик качества кода

Для оценки сложности программного кода, реализующего регистрацию пользователей в web-приложении планирования задач, была применена метрика

Холстеда [18]. Этот подход основывается на подсчёте четырёх базовых параметров: количества уникальных операторов, количества уникальных операндов, общего количества операторов и операндов в программной конструкции.

Для анализа была выбрана функция `register` из backend-компонента `UserService`, которая является одной из ключевых в процессе обработки данных пользователя. Она отвечает за создание новой учетной записи, хэширование пароля и сохранение пользователя в базе данных. Эта функция имеет постоянную нагрузку и играет важную роль в логике аутентификации. Программный код приведен в листинге 2.

Листинг 2 – Функция регистрации

```
async register(name,email,password) {
  const hash = await bcrypt.hash(password,10);
  return this.userRepo.create({ name,email,password:hash });
}
```

Анализируемая функция принимает три входных параметра – `name`, `email`, `password`, выполняет хэширование пароля с помощью библиотеки `bcrypt` и передаёт результат в метод `create` репозитория пользователя.

Для метрики Холстеда нам нужно вычислить количество операторов и операндом. Рассчитаем значения по метрике Холстеда. Данные об операторах представлены в таблице 28.

Таблица 28 – Операторы

Оператор	Кол-во
<code>async</code>	1
<code>const</code>	1
<code>=</code>	2
<code>await</code>	1
<code>.</code>	4
<code>()</code>	5
<code>{}</code>	1
<code>return</code>	1

В таблице 29 показаны операнды.

Таблица 29 – Операнды

Операнд	Кол-во
name	2
email	2
password	3
hash	2
bcrypt	1
this.userRepo	1
create	1

Исходя из собранных данных, можно произвести промежуточные расчёты, на основании которых определяется сложность анализируемой функции.

Длина программы определяется как сумма общего количества операторов и операндов:

$$N = N_1 + N_2 = 16 + 12 = 28.$$

Словарь программы рассчитывается как сумма количества уникальных операторов и уникальных операндов:

$$n = n^1 + n^2 = 8 + 7 = 15.$$

Объём программы (Halstead Volume) вычисляется по формуле:

$$HV = N \times \log^2(n) = 28 \times \log^2(15) \approx 28 \times 3,906 = 109,37.$$

Потенциальный объём программы определяется выражением:

$$V * = (2 + n^2) \times \log^2(2 + n^2) = (2 + 7) \times \log^2(9) = 9 \times 3,17 \approx 28,53.$$

Теоретический уровень программы можно найти как отношение потенциального объема к фактическому:

$$L = V * / HV = 28,53 / 109,37 \approx 0,26.$$

Сложность программы рассчитывается по формуле:

$$HD = \left(\frac{n_1}{2}\right) * \left(\frac{N_2}{n_2}\right),$$

$$HD = (n_1 / 2) \times (N_2 / n_2) = (8 / 2) \times (12 / 7) \approx 4 \times 1,714 \approx 6,86.$$

На основании проведённых вычислений можно сделать вывод, что функция register имеет показатель сложности 6,86, что указывает на её низкую

логическую сложность. Это говорит о высокой читаемости, простоте сопровождения и надёжности данной функции в архитектуре приложения.

Заключение

В результате выполнения выпускной квалификационной работы был создан веб-сервис, предназначенный для эффективного управления личными задачами и планирования времени. Разработка охватила все этапы жизненного цикла программного обеспечения: от анализа предметной области и формулирования задачи до проектирования, реализации и тестирования программного продукта.

На этапе анализа были исследованы существующие сервисы для планирования задач и самоорганизации, такие как Todolist, YouGile, TickTick, Google Календарь

Проведён сравнительный анализ этих платформ с использованием метода анализа иерархий, что позволило выявить ключевые функциональные требования и определить приоритеты при проектировании собственного решения.

В ходе проектирования были сформированы требования к функциональности, разработан дизайн пользовательского интерфейса, определена архитектура серверной части приложения, а также спроектирована структура базы данных.

В процессе реализации были созданы клиентское и серверное приложения, а также выполнена интеграция с базой данных. Особое внимание уделялось вопросам безопасности, удобства использования и расширяемости системы.

Для контроля качества разработки было проведено функциональное тестирование веб-сервиса. По итогам тестирования система показала корректную и устойчивую работу, соответствующую заявленным требованиям.

Таким образом, в ходе выполнения работы был успешно разработан современный и надёжный веб-сервис для управления личными задачами, отвечающий актуальным потребностям пользователей. Процесс разработки основывался на анализе существующих решений, тщательном проектировании и

применении современных технологий, что позволило создать функциональный, удобный и масштабируемый программный продукт.

.

Список используемых источников

1. Todoist – Справка и описание функционала // Todoist [Электронный ресурс] URL: <https://todoist.com/ru/help/articles/193874546> (дата обращения: 15.05.2025).
2. TickTick – Официальный сайт // TickTick [Электронный ресурс] URL: <https://ticktick.com/about> (дата обращения: 11.05.2025).
3. YouGile – Описание и возможности // YouGile [Электронный ресурс] URL: <https://yougile.com/> (дата обращения: 19.05.2025).
4. Google Календарь – Описание функций // Google [Электронный ресурс] URL: <https://support.google.com/calendar/answer/2465776?hl=ru> (дата обращения: 21.05.2025).
5. Сравнение лучших приложений для управления задачами // Lifehacker.ru [Электронный ресурс] URL: <https://lifehacker.ru/luchshie-prilozheniya-dlya-zadach/> (дата обращения: 24.05.2025).
6. Сравнение популярных планировщиков задач // Habr [Электронный ресурс] URL: <https://habr.com/ru/articles/572030/> (дата обращения: 27.05.2025).
7. Figma – Руководство по использованию // Figma Help Center [Электронный ресурс] URL: <https://help.figma.com/hc/ru> (дата обращения: 21.05.2025).
8. UML-диаграммы: виды, описание, примеры // Habr [Электронный ресурс] URL: <https://habr.com/ru/articles/508710/> (дата обращения: 03.05.2025).
9. Диаграммы UML для проектирования систем // docs.ensi.tech [Электронный ресурс] URL: <https://docs.ensi.tech/analyst-guides/tools/diagrams/uml/class-diagram> (дата обращения: 04.05.2025).
10. Node.js – Официальная документация // Node.js [Электронный ресурс] URL: <https://nodejs.org/ru/docs/> (дата обращения: 05.05.2025).
11. Express.js – Официальный сайт // Express [Электронный ресурс] URL: <https://expressjs.com/ru/> (дата обращения: 05.05.2025).

12. PostgreSQL – Официальная документация // PostgreSQL [Электронный ресурс] URL: <https://postgrespro.ru/docs/postgresql> (дата обращения: 01.05.2025).
13. React – Официальная документация // React [Электронный ресурс] URL: <https://react.dev/> (дата обращения: 12.05.2025).
14. Agile – Основные принципы // Atlassian [Электронный ресурс] URL: <https://www.atlassian.com/ru/agile> (дата обращения: 19.05.2025).
15. Методология Scrum // Scrum.org [Электронный ресурс] URL: <https://www.scrum.org/resources/what-is-scrum> (дата обращения: 26.05.2025).
16. Распределение зарплат разработчиков в России // hh.ru [Электронный ресурс] URL: <https://zp.hh.ru/> (дата обращения: 01.05.2025).
17. Black-box и White-box тестирование // GeekBrains [Электронный ресурс] URL: <https://gb.ru/posts/black-box-white-box-testing> (дата обращения: 21.05.2025).
18. Основные метрики качества кода // Hexlet [Электронный ресурс] URL: <https://hexlet.io/blog/posts/osnovnye-metriki-kachestva-koda> (дата обращения: 21.05.2025).
19. UI/UX: современные подходы к проектированию // UXPub [Электронный ресурс] URL: <https://ux.pub/> (дата обращения: 21.05.2025).
20. Телеграм-боты: руководство по созданию // Telegram [Электронный ресурс] URL: <https://core.telegram.org/bots> (дата обращения: 21.05.2025).
21. Исходный код клиентской части web-приложения // GitHub [Электронный ресурс]. URL: https://github.com/ntstlk/vkr_client (дата обращения: 29.05.2025).

Приложение А
(справочное)
Глоссарий

API (Application Programming Interface) – интерфейс взаимодействия между программными компонентами, обеспечивающий обмен данными между клиентской и серверной частями веб-приложения.

Back-end – серверная часть веб-приложения, отвечающая за бизнес-логику, обработку запросов и работу с базой данных.

Cron-задача – запланированное фоновое действие, которое выполняется по расписанию, например, отправка утренней сводки задач пользователю.

Drag-and-drop – технология взаимодействия с элементами интерфейса, позволяющая перемещать задачи между днями или изменять их порядок путём перетаскивания мышью.

Express.js – минималистичный и гибкий фреймворк для Node.js, используемый для создания серверной части веб-приложения и построения REST API.

JWT (JSON Web Token) – стандарт безопасной передачи данных между клиентом и сервером, используемый для аутентификации пользователей.

Node.js – серверная платформа, позволяющая запускать JavaScript-код вне браузера и использовать его для написания серверной логики веб-приложения.

PostgreSQL – объектно-реляционная система управления базами данных с открытым исходным кодом, использованная для хранения пользовательских данных, задач и настроек.

React – JavaScript-библиотека для создания пользовательских интерфейсов, основанная на компонентной архитектуре.

REST API – архитектурный стиль взаимодействия между клиентом и сервером, использующий стандартные HTTP-методы (GET, POST, PUT, DELETE).

Scheduler – модуль планирования задач в приложении, позволяющий запускать определённые функции по расписанию (например, отправка напоминаний через Telegram)

Subtask (подзадача) – вспомогательная задача, входящая в состав основной задачи и отражающая этапы её выполнения.

Task (задача) – единица планирования, содержащая название, описание, сроки, статус и при необходимости подзадачи.

Telegram Bot API – интерфейс для создания и управления ботами в Telegram, используемый в приложении для отправки напоминаний и авторизации.

Telegram-бот – программный агент, взаимодействующий с пользователем через Telegram, отправляющий напоминания и принимающий команды для входа в систему.

UI (User Interface) – пользовательский интерфейс приложения, включающий все визуальные элементы: формы, кнопки, списки задач.

UX (User Experience) – пользовательский опыт, оценивающий удобство, понятность и скорость взаимодействия с веб-приложением.

Use Case (вариант использования) – описание сценария взаимодействия пользователя с системой, включающее последовательность действий для достижения конкретной цели.

Visual Schedule – визуальное представление задач по дням недели в виде сетки или календаря.

Wireframe – каркасный макет интерфейса, демонстрирующий расположение элементов и логику взаимодействий без оформления.

Приложение Б
(обязательное)
Исходной код серверной части

Листинг Б.1 – Инициализация базы данных и таблиц

```

const { Pool, Client, types } = require('pg');

types.setTypeParser(1082, v => v);
types.setTypeParser(1182, v => v.slice(1, -
1).split(',').map(s=>s.replace(/"/g, '')));

class DatabaseInitializer {
  constructor(config) {
    this.config = config;
  }
  async createDatabase() {
    const client = new Client({ ...this.config, database:
'postgres' });
    await client.connect();
    const dbName = this.config.database;
    const exists = await client.query(
      `SELECT 1 FROM pg_database WHERE datname=$1`, [dbName]
    );
    if (!exists.rowCount) {
      await client.query(`CREATE DATABASE ${dbName}`);
      console.log(`Database "${dbName}" created.`);
    }
    await client.end();
  }
  async initTables(pool) {
    const sql = `
      CREATE TABLE IF NOT EXISTS users (
        id SERIAL PRIMARY KEY,
        name VARCHAR(255),
        email VARCHAR(255) UNIQUE NOT NULL,
        password VARCHAR(255) NOT NULL,
        role VARCHAR(20) NOT NULL DEFAULT 'user',
        banned BOOLEAN NOT NULL DEFAULT FALSE,
        created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
      );
      CREATE TABLE IF NOT EXISTS tasks (
        id SERIAL PRIMARY KEY,
        user_id INTEGER REFERENCES users(id) ON DELETE CASCADE,
        title VARCHAR(255) NOT NULL,
        description TEXT,
        time VARCHAR(10),
        end_time VARCHAR(10),
        date DATE,
        end_date DATE,
        color VARCHAR(20),
        completed BOOLEAN DEFAULT FALSE,
        "order" INTEGER,
        created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
        updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
      );
      CREATE TABLE IF NOT EXISTS subtasks (

```

```

        id SERIAL PRIMARY KEY,
        task_id INTEGER REFERENCES tasks(id) ON DELETE CASCADE,
        title VARCHAR(255) NOT NULL,
        completed BOOLEAN DEFAULT FALSE,
        "order" INTEGER,
        created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
        updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    );
    CREATE TABLE IF NOT EXISTS settings (
        user_id INTEGER PRIMARY KEY REFERENCES users(id) ON DELETE
CASCADE,
        dark_mode BOOLEAN DEFAULT FALSE,
        telegram_chat_id VARCHAR(255),
        telegram_username VARCHAR(255),
        telegram_enabled BOOLEAN DEFAULT FALSE,
        morning_summary BOOLEAN DEFAULT FALSE,
        remind_before BOOLEAN DEFAULT FALSE,
        created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
        updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    );
    CREATE TABLE IF NOT EXISTS telegram_links (
        code VARCHAR(6) PRIMARY KEY,
        chat_id VARCHAR(255) NOT NULL,
        expires_at TIMESTAMP NOT NULL,
        user_id INTEGER REFERENCES users(id) ON DELETE CASCADE,
        created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    );
`;
const client = await pool.connect();
try {
    await client.query(sql);
    console.log('Tables initialized.');
```

```

    } finally {
        client.release();
    }
}
}

```

Листинг Б.2 – Репозитории для работы с данными

```

const bcrypt = require('bcrypt');

class UserRepo {
    constructor(pool) { this.pool = pool }
    async create({ name, email, password }) {
        const res = await this.pool.query(
            `INSERT INTO users(name,email,password) VALUES($1,$2,$3)
            RETURNING id,name,email,role,created_at`,
            [name,email,password]
        );
        return res.rows[0];
    }
    async findByEmail(email) {

```

```

    const res = await this.pool.query(
      `SELECT * FROM users WHERE email=$1`, [email]
    );
    return res.rows[0] || null;
  }
  async listAll() {
    const res = await this.pool.query(
      `SELECT id, name, email, role, banned, created_at
      FROM users ORDER BY id`
    );
    return res.rows;
  }
  async deleteById(id) {
    await this.pool.query(
      `DELETE FROM users WHERE id=$1`,
      [id]
    );
  }
  async updateCredentials(id, { email, password }) {
    const parts = [];
    const vals = [];
    if (email) {
      parts.push(`email = ${parts.length+1}`);
      vals.push(email);
    }
    if (password) {
      parts.push(`password = ${parts.length+1}`);
      vals.push(await bcrypt.hash(password, 10));
    }
    if (!parts.length) return;
    vals.push(id);
    await this.pool.query(
      `UPDATE users SET ${parts.join(', ')} WHERE id =
      ${vals.length}`,
      vals
    );
  }
  async setBanned(id, banned) {
    await this.pool.query(
      `UPDATE users SET banned = $1 WHERE id = $2`,
      [banned, id]
    );
  }
  async updateRole(id, role) {
    await this.pool.query(
      `UPDATE users SET role = $1 WHERE id = $2`,
      [role, id]
    );
  }
}

class TaskRepo {
  constructor(pool) { this.pool = pool }

```

```

async listByUser(userId) {
  const res = await this.pool.query(
    `SELECT * FROM tasks WHERE user_id=$1 ORDER BY "order",id`,
    [userId]
  );
  return res.rows;
}
async create(userId, data) {
  const res = await this.pool.query(
    `INSERT INTO tasks
      (user_id,title,description,time,end_time,date,end_date,color,completed,"order")
      VALUES ($1,$2,$3,$4,$5,$6,$7,$8,$9,$10)
      RETURNING *`,
    [
      userId,
      data.title,
      data.description || null,
      data.time || null,
      data.end_time || null,
      data.date || null,
      data.end_date || null,
      data.color || null,
      data.completed || false,
      data.order || 0
    ]
  );
  return res.rows[0];
}
async update(userId, taskId, data) {
  const res = await this.pool.query(
    `UPDATE tasks SET
      title=$1,description=$2,time=$3,end_time=$4,
      date=$5,end_date=$6,color=$7,completed=$8,"order"=$9,updated_at=NOW()
      WHERE id=$10 AND user_id=$11
      RETURNING *`,
    [
      data.title,
      data.description || null,
      data.time || null,
      data.end_time || null,
      data.date || null,
      data.end_date || null,
      data.color || null,
      data.completed || false,
      data.order || 0,
      taskId,
      userId
    ]
  );
  return res.rows[0] || null;
}

```

```

    async delete(userId, taskId) {
      await this.pool.query(
        `DELETE FROM tasks WHERE id=$1 AND user_id=$2`,
        [taskId, userId]
      );
    }
  }

class SubtaskRepo {
  constructor(pool) { this.pool = pool }
  async listByTask(taskId) {
    const res = await this.pool.query(
      `SELECT * FROM subtasks WHERE task_id=$1 ORDER BY
"order",id`,
      [taskId]
    );
    return res.rows;
  }
  async replaceForTask(taskId, subtasks) {
    await this.pool.query(
      `DELETE FROM subtasks WHERE task_id=$1`,
      [taskId]
    );
    for (let i=0; i<subtasks.length; i++) {
      const st = subtasks[i];
      await this.pool.query(
        `INSERT INTO subtasks(task_id,title,completed,"order")
VALUES ($1,$2,$3,$4)`,
        [taskId, st.title, st.completed||false, st.order||i]
      );
    }
  }
}

class SettingsRepo {
  constructor(pool) { this.pool = pool }
  async get(userId) {
    const res = await this.pool.query(
      `SELECT
dark_mode,telegram_enabled,telegram_username,morning_summary,remind_before
FROM settings WHERE user_id=$1`,
      [userId]
    );
    return res.rows[0] || null;
  }
  async upsert(userId, data) {
    const exists = await this.pool.query(
      `SELECT 1 FROM settings WHERE user_id=$1`, [userId]
    );
    if (exists.rowCount) {
      await this.pool.query(
        `UPDATE settings SET

```

```

        dark_mode=$1,telegram_enabled=$2,morning_summary=$3,rem
ind_before=$4,updated_at=NOW()
        WHERE user_id=$5`,
        [data.dark_mode, data.telegram_enabled,
data.morning_summary, data.remind_before, userId]
    );
    } else {
        await this.pool.query(
            `INSERT INTO
settings(user_id,dark_mode,telegram_enabled,morning_summary,remind
_before)
            VALUES ($1,$2,$3,$4,$5)`,
            [userId, data.dark_mode, data.telegram_enabled,
data.morning_summary, data.remind_before]
        );
    }
}
}
}

```

Листинг Б.3 – Сервисы аутентификации и уведомлений

```

const jwt = require('jsonwebtoken');

class UserService {
  constructor(userRepo) { this.userRepo = userRepo }
  async register(name,email,password) {
    const hash = await bcrypt.hash(password,10);
    return this.userRepo.create({ name,email,password:hash });
  }
  async login(email,password) {
    const user = await this.userRepo.findByEmail(email);
    if (user.banned) {
      throw new Error('Ваш аккаунт заблокирован');
    }
    if (!user || !await bcrypt.compare(password,user.password)) {
      throw new Error('Неверные данные');
    }
    const token = jwt.sign(
      { id: user.id, email: user.email, role: user.role },
      JWT_SECRET,
      { expiresIn: '10d' }
    );
    return { user: { id: user.id, name: user.name, email:
user.email, role: user.role }, token };
  }
}

class NotificationService {
  constructor(taskRepo, settingsRepo, bot) {
    this.taskRepo = taskRepo;
    this.settingsRepo = settingsRepo;
    this.bot = bot;
  }
}

```

```

async sendMorningSummary() {
  const today = new Date().toISOString().slice(0,10);
  const sres = await this.settingsRepo.pool.query(
    `SELECT user_id,telegram_username,telegram_chat_id
      FROM settings WHERE telegram_enabled AND morning_summary`
  );
  for (let s of sres.rows) {
    const tasks = await this.taskRepo.pool.query(
      `SELECT title,time FROM tasks WHERE date=$1 AND user_id=$2
ORDER BY time`,
      [today, s.user_id]
    );
    if (!tasks.rowCount) continue;
    let txt = 'Утренняя сводка:\n';
    for (let t of tasks.rows) {
      txt += `• ${t.time}||'-'` ${t.title}\n`;
    }
    await this.bot.sendMessage(s.telegram_chat_id, txt);
  }
}

async sendReminders() {
  const now = new Date();
  const in5 = new Date(now.getTime()+5*60*1000);
  const date = now.toISOString().slice(0,10);
  const hhmm = in5.toTimeString().slice(0,5);
  const res = await this.taskRepo.pool.query(
    `SELECT t.title,t.time,s.telegram_chat_id
      FROM tasks t
      JOIN settings s ON s.user_id=t.user_id
      WHERE t.date=$1 AND t.time=$2
      AND s.telegram_enabled AND s.remind_before
      AND s.telegram_chat_id IS NOT NULL
    `, [date, hhmm]);
  for (let r of res.rows) {
    await this.bot.sendMessage(r.telegram_chat_id,
    );
  }
}
}

```

Листинг Б.4 – Обёртка Telegram-бота и расписание задач

```

const TelegramBot = require('node-telegram-bot-api');
const cron = require('node-cron');

class TelegramBotWrapper {
  constructor(token, pool) {
    this.bot = new TelegramBot(token, { polling:true });
    this.pool = pool;
    this.bot.setMyCommands([ { command:'start', description:'Начало
работы' } ]);
    this.bot.onText(/\sstart$/, this.onStart.bind(this));
  }
}

```

```

    async onStart(msg) {
      const chatId = msg.chat.id;
      const code = Math.floor(100000 +
Math.random()*900000).toString();
      const expiresAt = new Date(Date.now()+15*60*1000);
      await this.pool.query(
        `INSERT INTO telegram_links (code,chat_id,expires_at)
        VALUES ($1,$2,$3)
        ON CONFLICT(code) DO UPDATE
        SET chat_id=$2,expires_at=$3,created_at=NOW()`,
        [code,chatId,expiresAt]
      );
      await this.bot.sendMessage(chatId, `Ваш код: ${code}\nВведите
его на сайте.`);
    }
    sendMessage(chatId,text) {
      return this.bot.sendMessage(chatId,text);
    }
    getChat(chatId) {
      return this.bot.getChat(chatId);
    }
  }
}

class Scheduler {
  constructor(ns) { this.ns = ns }
  start() {
    cron.schedule('0 8 * * *', () =>
this.ns.sendMorningSummary());
    cron.schedule('* * * * *', () => this.ns.sendReminders());
    console.log('Scheduler started.');
```

Листинг Б.5 – Обработка регистрации и логина

```

const express = require('express');

class AuthHandler {
  constructor(router, userService) {
    this.us = userService;
    router.post('/register', this.register.bind(this));
    router.post('/login', this.login.bind(this));
  }
  async register(req, res) {
    try {
      const u = await this.us.register(req.body.name,
req.body.email, req.body.password);
      res.json(u);
    } catch (e) {
      console.error(e);
      if (e.code === '23505') {
        return res.status(409).json({ error: 'Пользователь с таким
email уже существует' });
      }
    }
  }
}
```

```

    }
    res.status(500).json({ error: 'Ошибка регистрации' });
  }
}
async login(req,res) {
  try {
    const { user, token } = await
this.us.login(req.body.email,req.body.password);
    res.json({ user, token });
  } catch(e) {
    res.status(400).json({ error:e.message });
  }
}
}
}

```

Листинг Б.6 – Обработка Telegram-логина

```

class TelegramLoginHandler {
  constructor(app, pool, bot) {
    this.pool = pool;
    this.bot = bot;
    app.post('/api/login/telegram', this.requestCode.bind(this));
    app.post('/api/login/telegram/verify',
this.verifyCode.bind(this));
  }
  async requestCode(req,res) {
    const { email } = req.body;
    if (!email) return res.status(400).json({ error:'Email
обязателен' });
    try {
      const client = await this.pool.connect();
      const userR = await client.query(`SELECT id FROM users
WHERE email=$1`, [email]);
      if (!userR.rowCount) { client.release(); return
res.status(400).json({ error:'Пользователь не найден' }); }
      const userId = userR.rows[0].id;
      const setR = await client.query(
`SELECT telegram_chat_id FROM settings WHERE user_id=$1
AND telegram_enabled`,
        [userId]
      );
      if (!setR.rowCount || !setR.rows[0].telegram_chat_id) {
        client.release();
        return res.status(400).json({ error:'Telegram не привязан
к этому аккаунту' });
      }
      const chatId = setR.rows[0].telegram_chat_id;
      const code =
Math.floor(100000+Math.random()*900000).toString();
      const expiresAt = new Date(Date.now()+15*60*1000);
      await client.query(
`INSERT INTO
telegram_links(code,chat_id,expires_at,user_id)

```

```

VALUES ($1,$2,$3,$4)
ON CONFLICT(code) DO UPDATE
SET
chat_id=$2,expires_at=$3,user_id=$4,created_at=NOW()`,
    [code,chatId,expiresAt,userId]
);
client.release();
await this.bot.sendMessage(chatId, `Ваш код для входа:
${code}\nОн действителен 15 минут.`);
res.json({ success:true });
} catch(e) {
console.error(e);
res.status(500).json({ error:'Не удалось отправить код' });
}
}
async verifyCode(req,res) {
const { code } = req.body;
if (!code) return res.status(400).json({ error:'Код
обязателен' });
try {
const client = await this.pool.connect();
const linkR = await client.query(
`SELECT chat_id,user_id,expires_at FROM telegram_links
WHERE code=$1`, [code]
);
if (!linkR.rowCount) { client.release(); return
res.status(400).json({ error:'Неверный код или истёк' }); }
const { chat_id, user_id, expires_at } = linkR.rows[0];
if (new Date().>expires_at) {
await client.query(`DELETE FROM telegram_links WHERE
code=$1`, [code]);
client.release();
return res.status(400).json({ error:'Код истёк' });
}
await client.query(`DELETE FROM telegram_links WHERE
code=$1`, [code]);
const userR = await client.query(`SELECT id,name,email FROM
users WHERE id=$1`, [user_id]);
client.release();
const user = userR.rows[0];
const token = jwt.sign({ id:user.id,email:user.email },
JWT_SECRET, { expiresIn:'10d' });
const loginTime = new Date().toISOString().replace('T','')
.slice(0,19);
await this.bot.sendMessage(chat_id, `Вход выполнен\nВремя:
${loginTime}`);
res.json({ user:{ id:user.id,name:user.name,email:user.email
}, token });
} catch(e) {
console.error(e);
res.status(500).json({ error:'Ошибка авторизации через
Telegram' });
}
}

```

```

    }
}

```

Листинг Б.7 – Привязка Telegram к аккаунту

```

class TelegramBindHandler {
  constructor(app, pool, bot) {
    this.pool = pool;
    this.bot = bot;
    app.post('/api/telegram/verify-code', this.verify.bind(this));
  }
  async verify(req, res) {
    const authHeader = req.headers['authorization'];
    const token = authHeader && authHeader.split(' ')[1];
    if (!token) return res.sendStatus(401);
    let payload;
    try {
      payload = jwt.verify(token, JWT_SECRET);
    } catch {
      return res.sendStatus(403);
    }
    const userId = payload.id;
    const { code, force } = req.body;
    if (!code) return res.status(400).json({ error: 'Код
обязателен' });

    try {
      const client = await this.pool.connect();
      const linkR = await client.query(
        `SELECT chat_id, expires_at FROM telegram_links WHERE
code=$1`,
        [code]
      );
      if (!linkR.rowCount) {
        client.release();
        return res.status(400).json({ error: 'Неверный код или
истёк' });
      }
      const { chat_id, expires_at } = linkR.rows[0];
      if (new Date() > expires_at) {
        await client.query(`DELETE FROM telegram_links WHERE
code=$1`, [code]);
        client.release();
        return res.status(400).json({ error: 'Код истёк. Повторите
привязку' });
      }

      const accounts = await client.query(
        `SELECT users.email FROM settings
        JOIN users ON users.id = settings.user_id
        WHERE telegram_chat_id = $1 AND user_id != $2`,
        [chat_id, userId]
      );
    }
  }
}

```

```

    if (accounts.rowCount > 0 && !force) {
      client.release();
      return res.status(409).json({
        error: 'Этот Telegram уже привязан к другим аккаунтам.',
        accounts: accounts.rows.map(r => r.email)
      });
    }

    const chatInfo = await this.bot.getChat(chat_id);
    const username = chatInfo.username || chatInfo.first_name ||
    '';

    await client.query(
      `INSERT INTO settings(user_id, telegram_chat_id,
telegram_username, telegram_enabled)
      VALUES($1, $2, $3, $4)
      ON CONFLICT(user_id) DO UPDATE
      SET telegram_chat_id=$2, telegram_username=$3,
telegram_enabled=$4, updated_at=NOW()`,
      [userId, chat_id, username, true]
    );
    await client.query(`DELETE FROM telegram_links WHERE
code=$1`, [code]);
    client.release();
    await this.bot.sendMessage(chat_id,
      'Вы успешно привязали Telegram к своему аккаунту.\nТеперь
получите уведомления и сможете входить через него.'
    );
    res.json({ success: true });
  } catch (e) {
    console.error(e);
    res.status(500).json({ error: 'Ошибка подтверждения кода'
  });
  }
}
}
}

```

Листинг Б.8 – Администрирование

```

class AdminHandler {
  constructor(router, userRepo) {
    this.ur = userRepo;
    router.use(this._requireAdmin.bind(this));
    router.get('/users', this.listUsers.bind(this));
    router.delete('/users/:id', this.deleteUser.bind(this));
    router.put('/users/:id', this.updateUser.bind(this));
    router.put('/users/:id/ban', this.banUser.bind(this));
    router.get('/stats', this.stats.bind(this));
  }

  _authenticate(req) {
    const header = req.headers['authorization'] || '';
    const token = header.split(' ')[1];
  }
}

```

```

async banUser(req, res) {
  await this.ur.setBanned(req.params.id, req.body.banned);
  res.json({ success: true });
}

async stats(req, res) {
  const { rows } = await this.ur.pool.query(`
    SELECT
      (SELECT COUNT(*) FROM
users)::INT                                AS "totalUsers",
      (SELECT COUNT(*) FROM users WHERE created_at >= NOW() -
INTERVAL '1 hour')::INT AS "newLastHour",
      (SELECT COUNT(*) FROM users WHERE created_at >= NOW() -
INTERVAL '1 day')::INT  AS "newLastDay",
      (SELECT COUNT(*) FROM users WHERE created_at >= NOW() -
INTERVAL '7 day')::INT  AS "newLastWeek",
      (SELECT COUNT(*) FROM
tasks)::INT                                AS "totalTasks"
  `);
  res.json(rows[0]);
}
}

```

Листинг Б.9 – Обработка задач

```

class TaskHandler {
  constructor(app, taskRepo, subtaskRepo) {
    this.tr = taskRepo;
    this.sr = subtaskRepo;
    app.get('/api/tasks', this.list.bind(this));
    app.post('/api/tasks', this.create.bind(this));
    app.put('/api/tasks/:id', this.update.bind(this));
    app.delete('/api/tasks/:id', this.delete.bind(this));
  }
  _authenticate(req) {
    const authHeader = req.headers['authorization'];
    const token      = authHeader && authHeader.split(' ')[1];
    if (!token) throw { status: 401 };
    try {
      return jwt.verify(token, JWT_SECRET).id;
    } catch {
      throw { status: 403 };
    }
  }
  async list(req, res) {
    try {
      const userId = this._authenticate(req);
      const tasks  = await this.tr.listByUser(userId);
      for (let t of tasks) {
        t.subtasks = await this.sr.listByTask(t.id);
      }
      res.json(tasks);
    } catch (e) {

```

```

        res.sendStatus(e.status||500);
    }
}
async create(req,res) {
    try {
        const userId = this._authenticate(req);
        const task    = await this.tr.create(userId, req.body);
        await this.sr.replaceForTask(task.id,
req.body.subtasks||[]);
        task.subtasks = await this.sr.listByTask(task.id);
        res.json(task);
    } catch(e) {
        res.sendStatus(e.status||500);
    }
}
async update(req,res) {
    try {
        const userId = this._authenticate(req);
        const t      = await this.tr.update(userId, req.params.id,
req.body);
        if (!t) return res.sendStatus(404);
        await this.sr.replaceForTask(t.id, req.body.subtasks||[]);
        t.subtasks = await this.sr.listByTask(t.id);
        res.json(t);
    } catch(e) {
        res.sendStatus(e.status||500);
    }
}
async delete(req,res) {
    try {
        const userId = this._authenticate(req);
        await this.tr.delete(userId, req.params.id);
        res.json({ message:'Задача удалена' });
    } catch(e) {
        res.sendStatus(e.status||500);
    }
}
}
}

```

Листинг Б.10 – Обработка настроек

```

class SettingsHandler {
    constructor(app, settingsRepo) {
        this.sr = settingsRepo;
        app.get('/api/settings', this.get.bind(this));
        app.put('/api/settings', this.put.bind(this));
    }
    _authenticate(req) {
        const authHeader = req.headers['authorization'];
        const token      = authHeader && authHeader.split(' ')[1];
        if (!token) throw { status:401 };
        try {
            return jwt.verify(token,JWT_SECRET).id;

```

```

    } catch {
      throw { status:403 };
    }
  }
  async get(req,res) {
    try {
      const userId = this._authenticate(req);
      let s = await this.sr.get(userId);
      if (!s) {
        s = {
dark_mode:false,telegram_enabled:false,telegram_username:null,morn
ing_summary:false,remind_before:false };
      }
      res.json({
        darkMode: s.dark_mode,
        telegramEnabled: s.telegram_enabled,
        telegramUsername: s.telegram_username,
        morningSummary: s.morning_summary,
        remindBefore: s.remind_before
      });
    } catch(e) {
      res.sendStatus(e.status||500);
    }
  }
  async put(req,res) {
    try {
      const userId = this._authenticate(req);
      const b = req.body;
      await this.sr.upsert(userId, {
        dark_mode: b.darkMode,
        telegram_enabled: b.telegramEnabled,
        morning_summary: b.morningSummary,
        remind_before: b.remindBefore
      });
      res.json(b);
    } catch(e) {
      res.sendStatus(e.status||500);
    }
  }
}

```

Листинг Б.11 – Запуск сервера и подключение всех компонентов

```

const express = require('express');
const cors = require('cors');

(async()=>{
  const app = express();
  const allowedOrigins = [
    'http://localhost:5173',
    'http://127.0.0.1:5173',
    'http://192.168.1.101:5173',
    'http://192.168.1.102:5173',

```

```

];

app.use(cors({
  origin: allowedOrigins,
  credentials: true
}));
app.use(express.json());

const config = {
  host: 'localhost', user: 'postgres', password: 'vvv570486', database: 'fo
xyplan', port: 5432 };
const dbInit = new DatabaseInitializer(config);
await dbInit.createDatabase();

const pool = new Pool(config);
await dbInit.initTables(pool);

const botWrapper = new TelegramBotWrapper(TELEGRAM_TOKEN, pool);

const userRepo      = new UserRepo(pool);
const taskRepo      = new TaskRepo(pool);
const subtaskRepo   = new SubtaskRepo(pool);
const settingsRepo  = new SettingsRepo(pool);

const userService = new UserService(userRepo);
const notifService= new
NotificationService(taskRepo, settingsRepo, botWrapper);

new Scheduler(notifService).start();

const apiRouter = express.Router();
new AuthHandler(apiRouter, userService);
app.use('/api', apiRouter);

new TelegramLoginHandler(app, pool, botWrapper.bot);
new TelegramBindHandler(app, pool, botWrapper.bot);

const adminRouter = express.Router();
new AdminHandler(adminRouter, userRepo);
app.use('/api/admin', adminRouter);

app.listen(PORT, () => {
  });
}) ();

```

Приложение В
(рекомендуемое)
Тестирование приложения

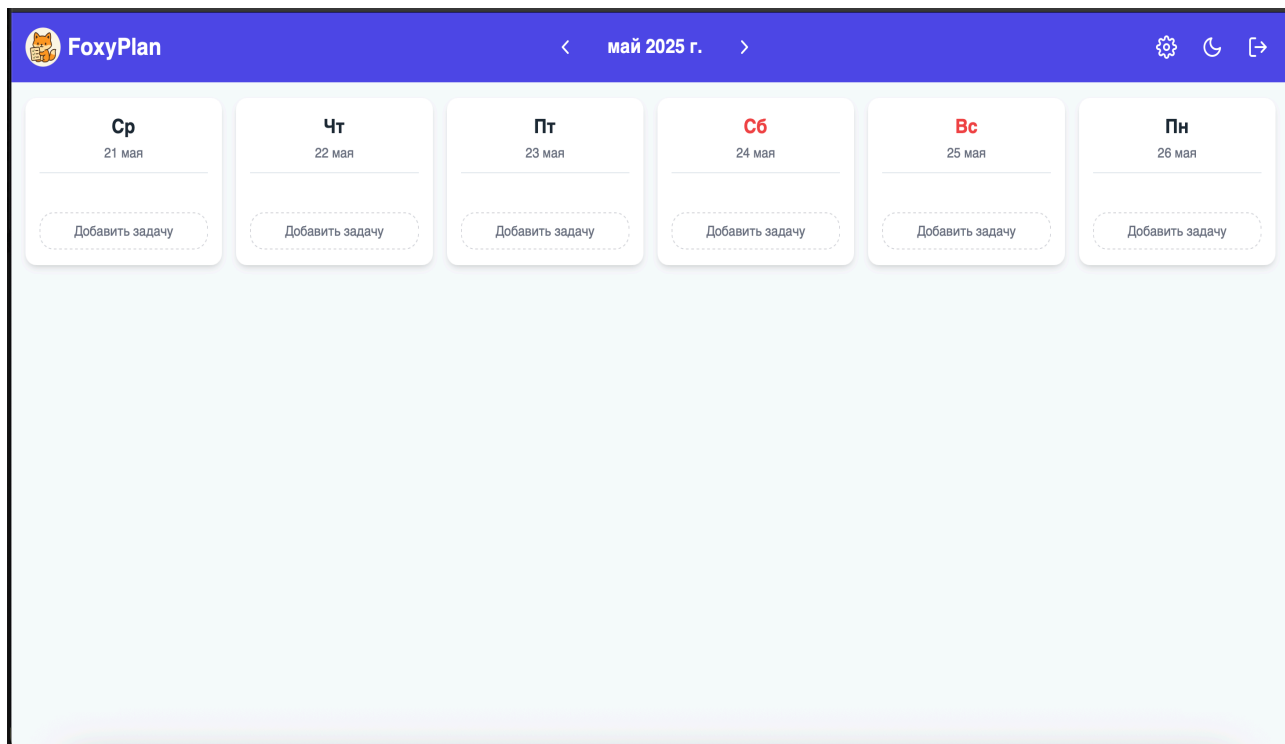


Рисунок В.1 – Аккаунт зарегистрирован

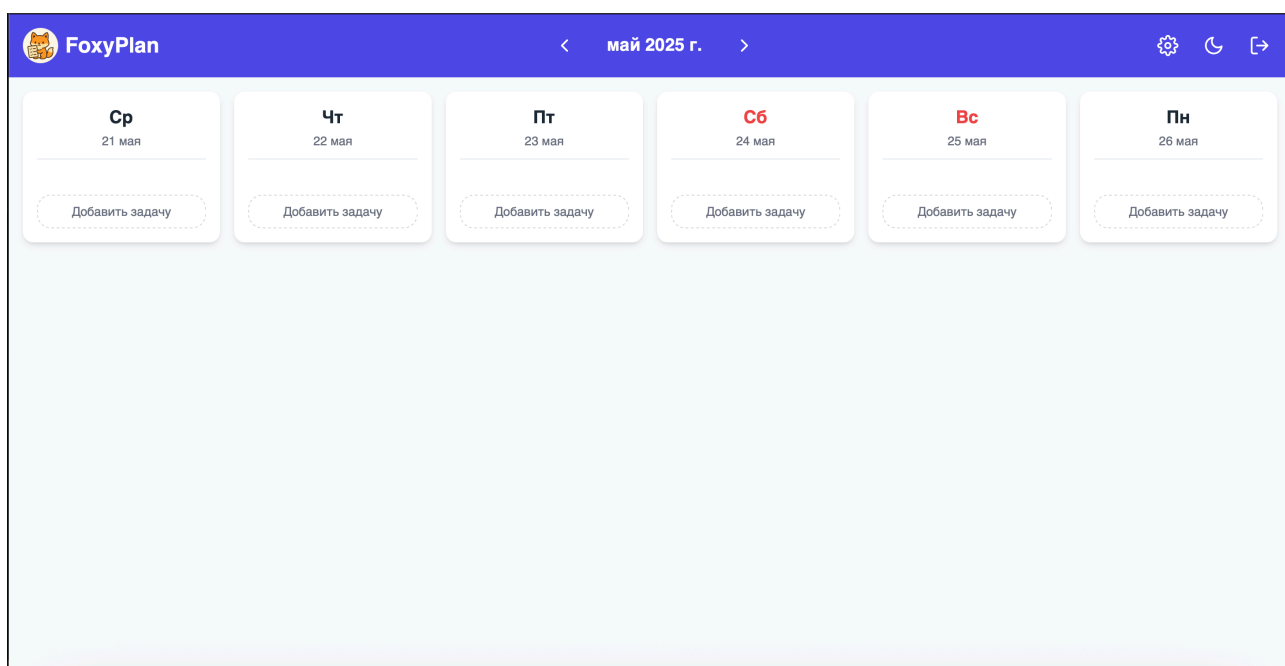


Рисунок В.2 – Вход в аккаунт

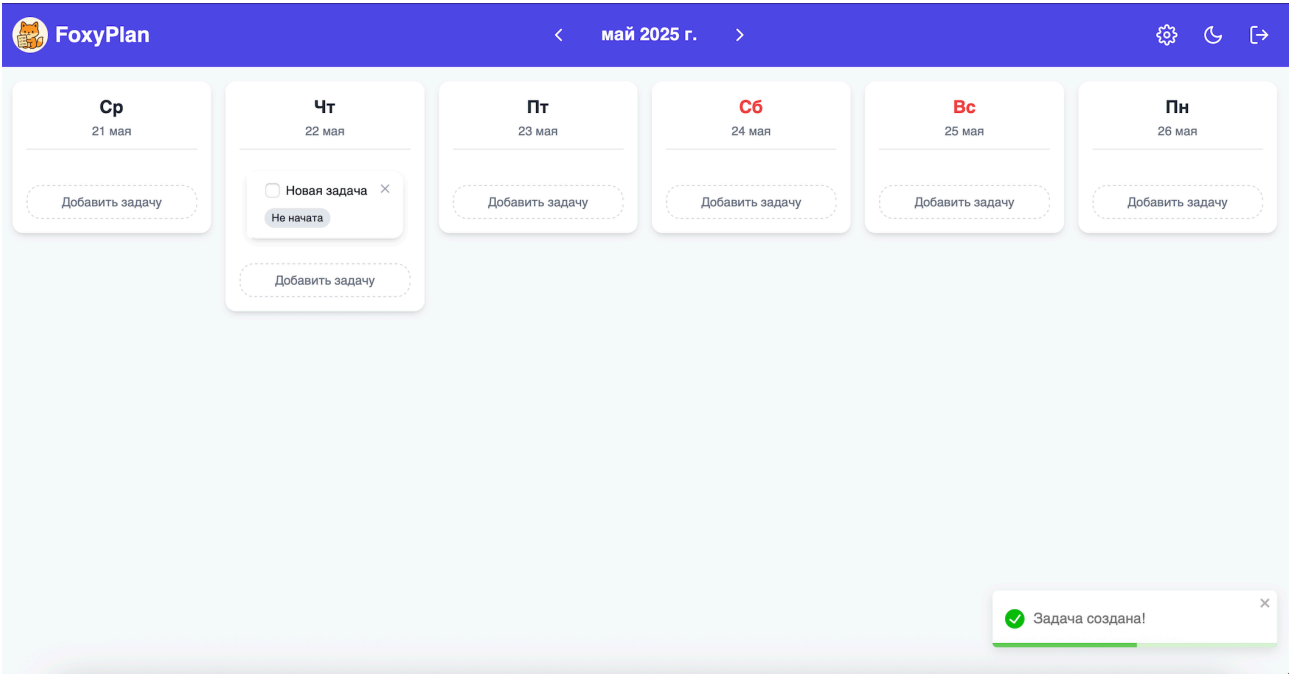


Рисунок В.3 – Задача создана

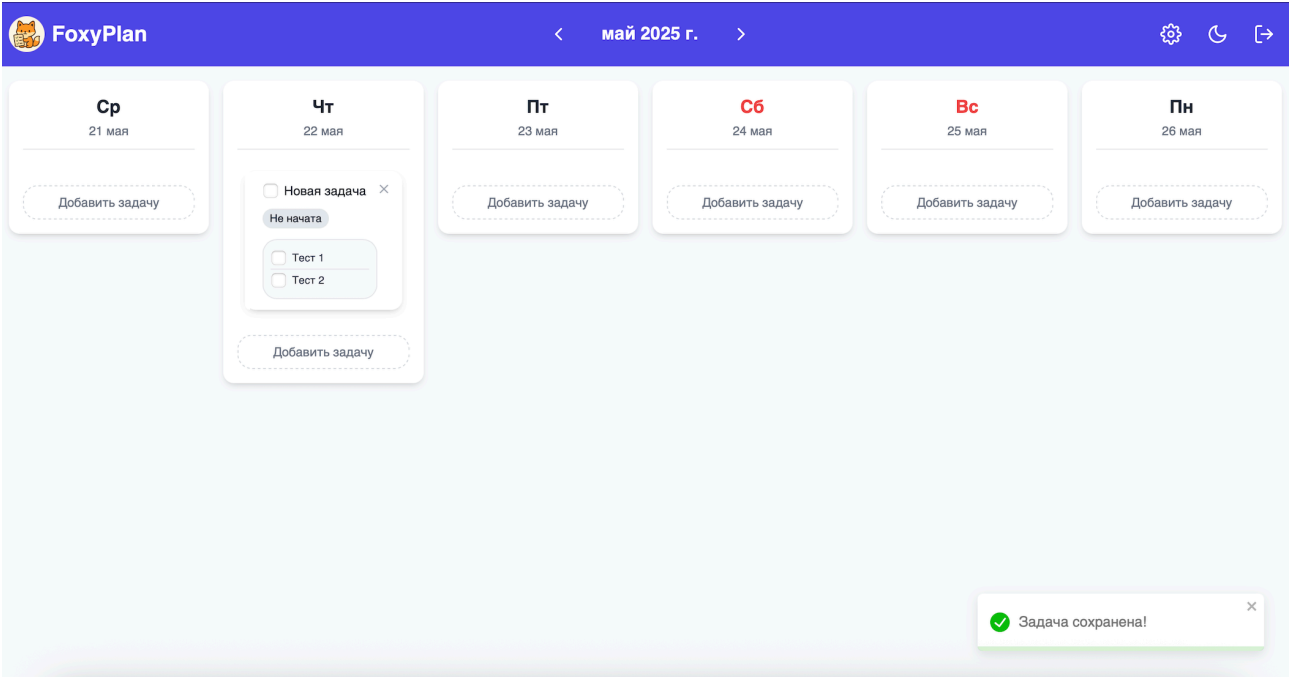


Рисунок В.4 – Подзадачи добавлены

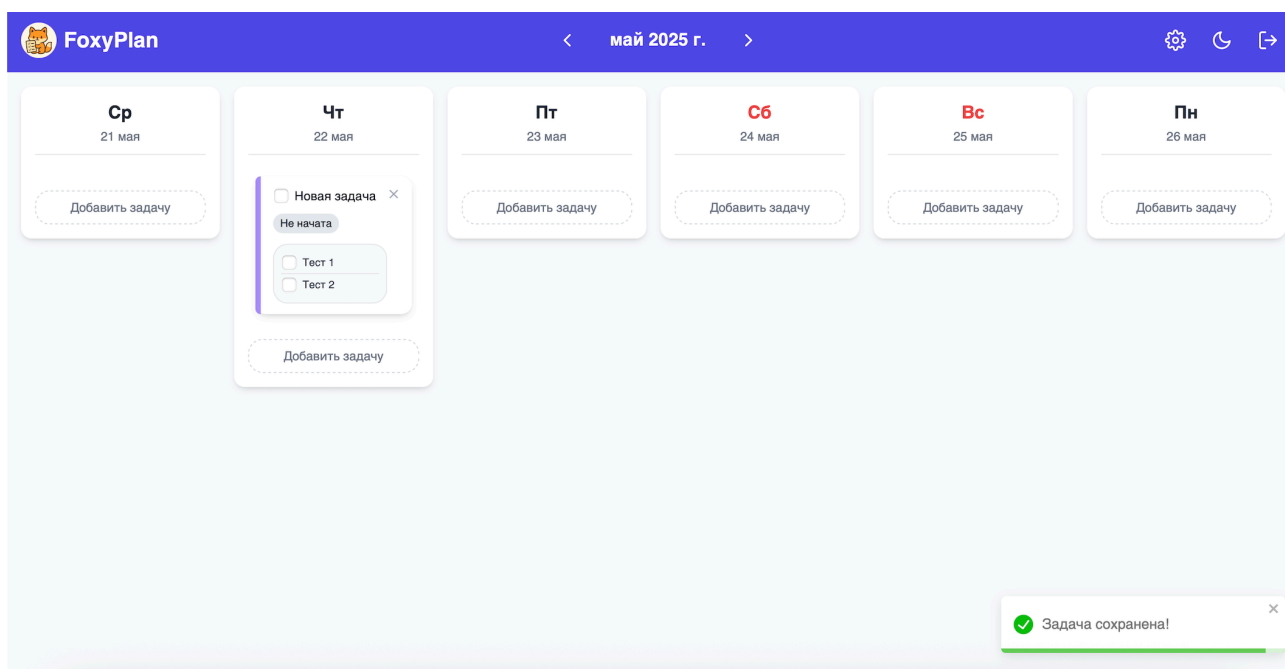


Рисунок В.5 – Изменения сохранены

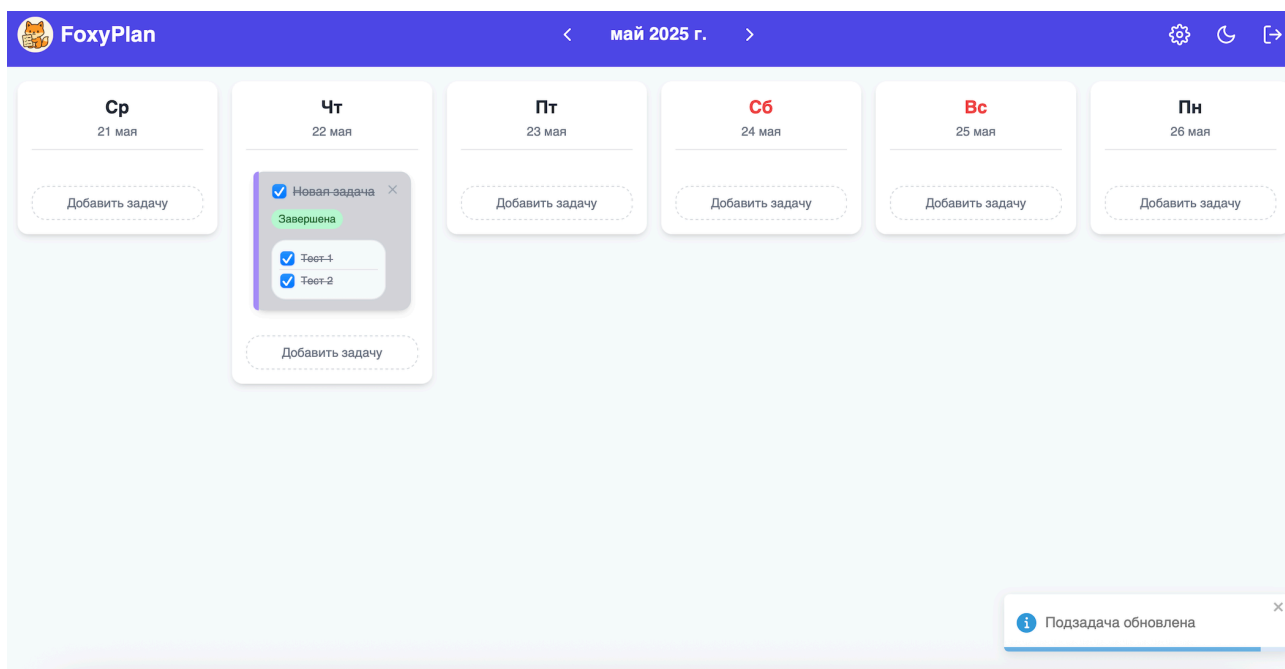


Рисунок В.6 – Статус изменён

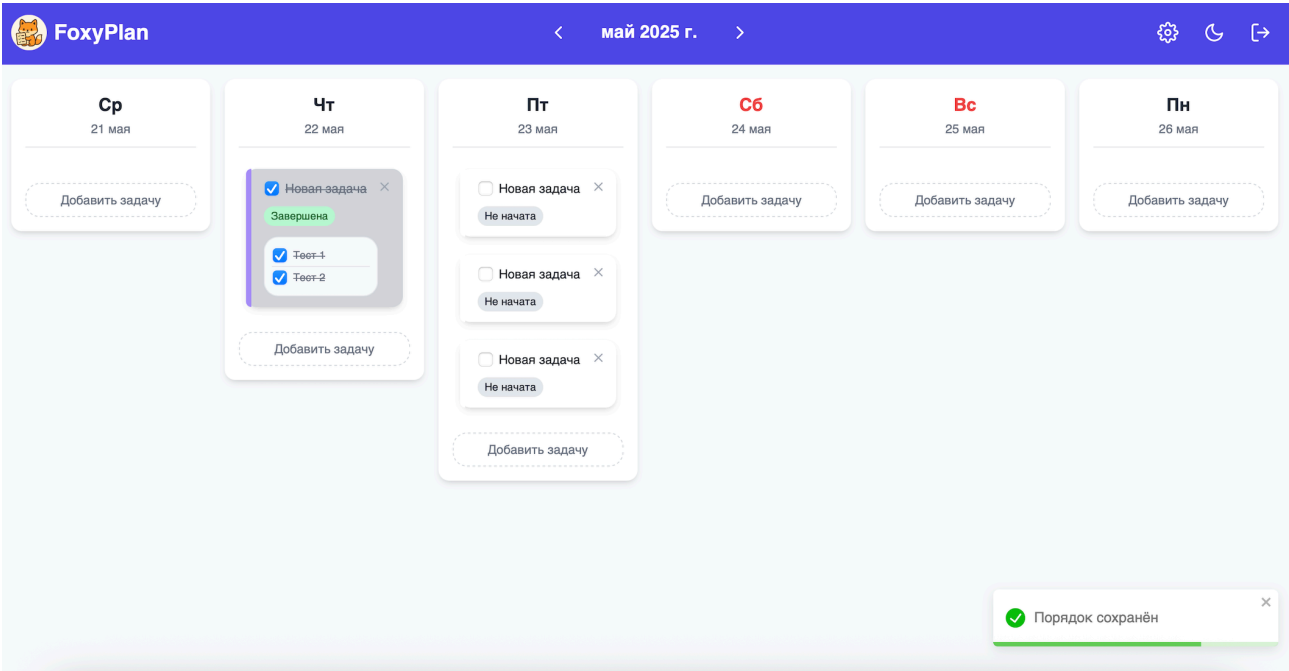


Рисунок В.7 – Порядок изменён

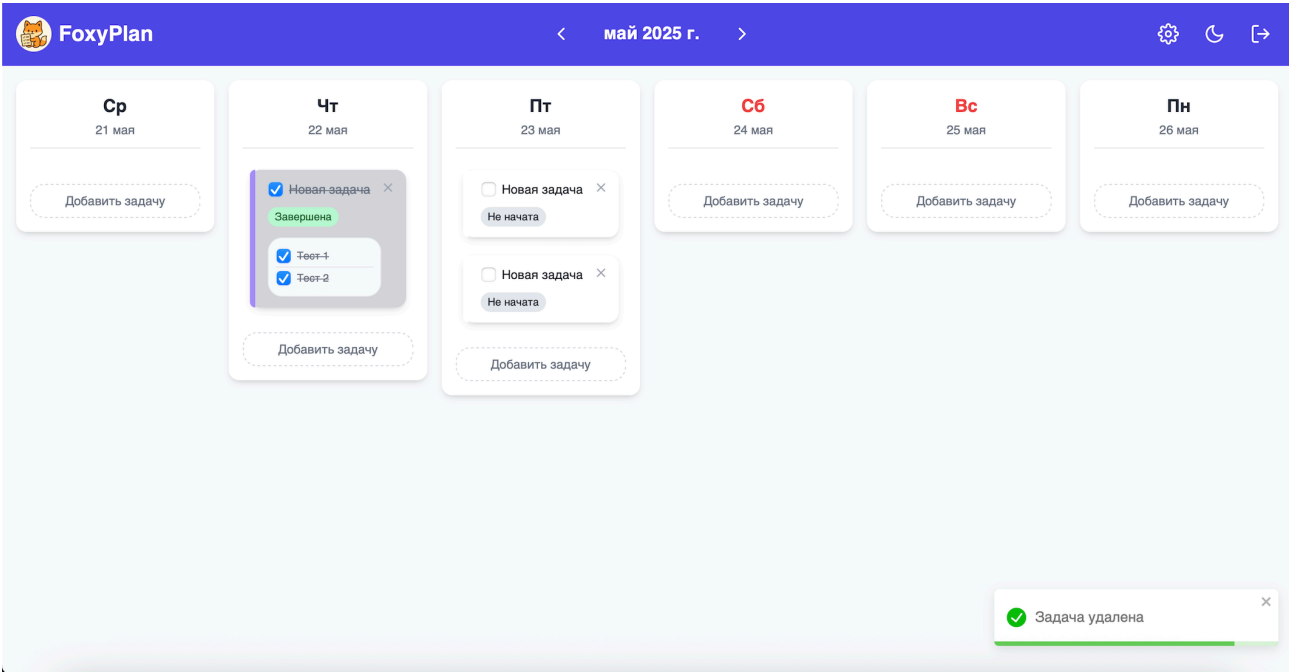


Рисунок В.8 – Задача удалена

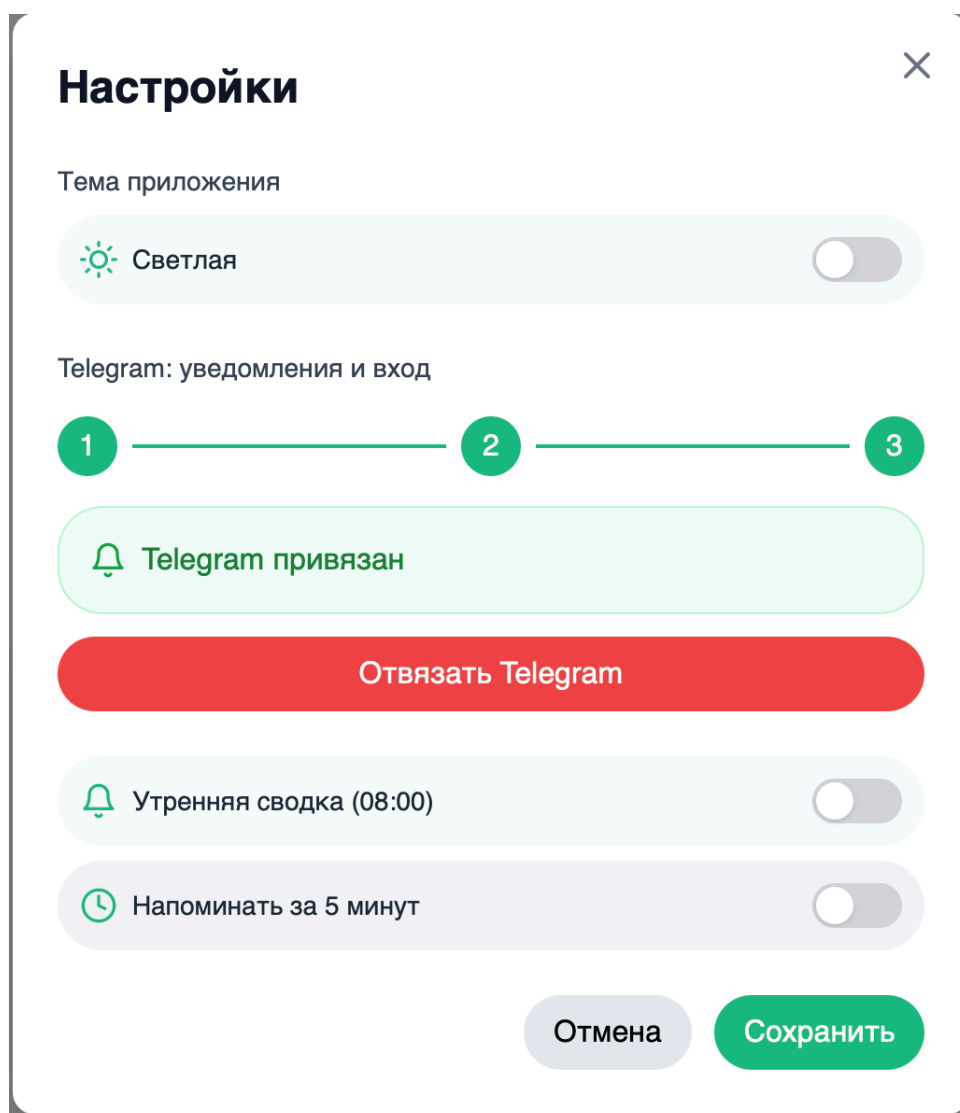


Рисунок В.9 – Telegram привязан

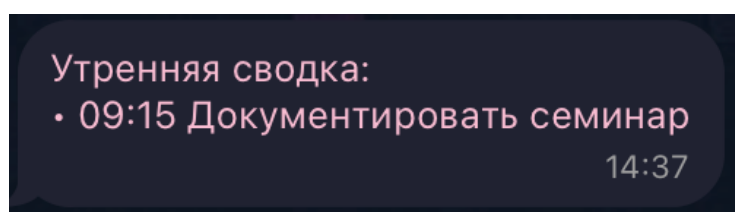


Рисунок В.10 – Сводка получена

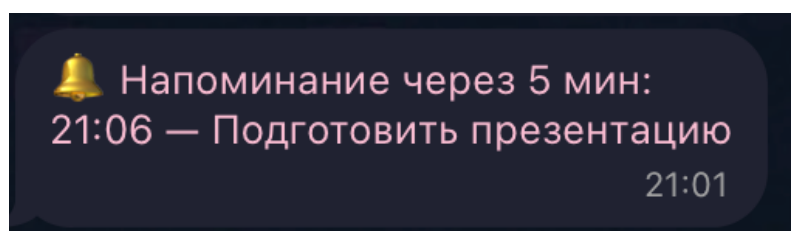


Рисунок В.11 – Уведомление получено