

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»
Профиль подготовки: «Программное обеспечение средств вычислительной
техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

БИЗНЕС-ПРОЕКТ «РАЗРАБОТКА СИСТЕМЫ НА ОСНОВЕ TELEGRAM-БОТА ДЛЯ ЗДОРОВЬЯ И ФИТНЕСА»

Студент

 03.06.25
(подпись, дата)

Морозов Константин Вадимович
(ФИО полностью)

Руководитель

 03.06.25
(подпись, дата)

Карамышева Н.С.
(фамилия, инициалы)

Нормоконтролёр

 03.06.25
(подпись, дата)

Бычков А.С.
(фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 28.05.2025 г. № 16)

Заведующий кафедрой


(подпись)

М.А. Митрохин
(фамилия, инициалы)

Работа защищена с отметкой отлично (протокол заседания ГЭК от

11 июня 2025 г. № 29)

Секретарь ГЭК


(подпись)

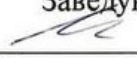
Е.И. Гурин
(фамилия, инициалы)

Пенза 2025

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
 М.А. Митрохин
«06» __ 05 __ 2025 __ г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Морозов Константин Вадимович гр. 21ВВП2 факультета ФВТ

2. Тема работы Бизнес-проект «Разработка системы на основе telegram-бота для здоровья и фитнеса»

Тема утверждена приказом ПГУ № 601/о от "06" мая 2025 г.

3. Руководитель бакалаврской работы Карамышева Н.С.

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

Разработать информационную систему для здоровья и фитнеса, использующую телеграм-бота. Составить бизнес-модель и финансовый план. Язык программирования python, среда разработки PyCharm. Программа должна обладать интуитивно понятным интерфейсом в мессенджере telegram и использовать следующие технологии:

1. Telegram

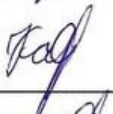
2. Aiogram

3. Асинхронное
программирование

5. Перечень подлежащих разработке вопросов

Требуется произвести анализ предметных областей: изучить существующие информационные системы, выделить их достоинства и недостатки. Выявить потребности пользователей. Выбрать язык программирования, среду разработки, СУБД, хостинг которые подойдут для создания телеграм-бота. Определить архитектуру приложения и вспомогательные средства разработки. Изучить функционал библиотек для разработки информационной системы на основе телеграм-бота.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	64	06.05.2025 – 28.05.2025	
2	Анализ требований	32	29.04.2025 – 28.05.2025	
3	Создание макета графического интерфейса	30	31.04.2025 – 3.05.2025	
4	Выбор технологий, применяемых при проектировании и разработке	40	4.05.2025 – 8.05.2025	
5	Реализация графического интерфейса	66	9.05.2025 – 15.05.2025	
6	Разработка основной части программы	70	1.05.2025 – 27.05.2025	
7	Оформление пояснительной записки	54	24.05.2025 – 30.05.2025	

Дата выдачи задания "06" ____ мая ____ 2025 ____ г.

Руководитель бакалаврской работы  03.06.25г. Карамышева Н.С.
(подпись, дата)

Задание к исполнению принял студент  03.06.25г. Морозов К.В.
(подпись, дата)

Работу к защите допустить
Декан факультета  05.06.2025 Фионова Л.Р.
(подпись, дата)

Реферат

Бакалаврская работа на тему «Разработка системы на основе Telegram-бота для здоровья и фитнеса» содержит 215 листа, включает 73 рисунка, 1 формулу, 8 таблиц и 8 источников. Работа дополнена 3 приложениями, содержащими UML-диаграммы, листинг программы и презентацию.

TELEGRAM-БОТ, ОБРАБОТКА СОСТОЯНИЙ, УПРАВЛЕНИЕ ДАННЫМИ, ТРЕКИНГ АКТИВНОСТИ, ТРЕНИРОВКИ, РЕЦЕПТЫ, СИСТЕМА УВЕДОМЛЕНИЙ, ПРЕМИУМ-ФУНКЦИИ, БАЗА ДАННЫХ, AIOGRAM,, PYTHON.

Цель работы - разработка системы на основе Telegram-бота, направленной на автоматизацию процессов в сфере здоровья и фитнеса.

Одним из ключевых решений для реализации бизнес-проекта является информационная система на основе Telegram-бота, обеспечивающая выполнение следующих функций: авторизация пользователей, просмотр и редактирование личных данных, управление расписанием тренировок, отслеживание прогресса пользователей, предоставление рецептов в зависимости от цели пользователя, а также возможность добавлять новые рецепты, отправка различных типов уведомлений с возможностью включения и выключения.

В результате выполнения работы создана полностью функциональная информационная система, соответствующая всем поставленным задачам. Внедрение данного решения позволит пользователям более эффективно достигать своих фитнес-целей, а бизнесу - оптимизировать процессы взаимодействия с клиентами и повысить качество предоставляемых услуг. Технологический стек решения включает: Python, Aiogram, Psycopg2, PostgreSQL и Telegram API. Система прошла полный цикл тестирования и готова к эксплуатации.

					ПГУ1.090301.38.001 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	Бизнес-проект «Разработка системы на основе telegram-бота для здоровья и фитнеса»	Лит.	Лист	Листов
Разраб.		Морозов К.В.		03.06.19				
Провер.		Карамышева НС		03.06.19				
Н. Контр		Бычков А.С.		03.06.19				
Утверд.		Митрохин М.А.		03.06.19				
						ФВТ, каф. ВТ гр. 21ВВП2		

Содержание

Перечень сокращений и обозначений.....	5
Введение.....	6
1 Бизнес-проект «Разработка системы на основе telegram-бота для здоровья и фитнеса».....	9
1.1 Анализ предметной области	9
1.2 Анализ решения поставленной задачи	10
1.3 Анализ актуальности разработки	12
1.4 Анализ аналогов информационных систем.....	13
1.5 Анализ архитектуры и структура бота	19
1.6 Постановка задачи	22
1.7 Выбор инструментов разработки	24
1.7.1 Основной стек инструментов разработки	24
1.8 Альтернативные инструменты разработки	24
1.8.1 Альтернативные языки программирования	24
1.8.2 Альтернативные фреймворки	27
1.8.3 Альтернативные базы данных	29
1.9 Реализация	32
1.9.1 Регистрация телеграм-бота	32
1.9.2 Реализация базы данных	36
1.9.3 Описание доступных функций	37
1.9.4 Описание программы.....	62
2 Безопасность жизнедеятельности.....	79
2.1 Правила безопасности при использовании ПК.....	79
2.2 Правила безопасности при использовании смартфона.....	80
3 Стартап проект	82
3.1 Сегменты ЦА	82
3.2 Ключевые процессы.....	83
3.3 Каналы взаимодействия	85
3.4 Источники дохода	86
3.5 Ключевые ресурсы.....	88
3.6 Риски проекта	90
Заключение	92

Список использованных источников	93
Приложение А	94
Приложение Б	100
Приложение В.....	209

					ПГУ1.090301.38.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		4

Перечень сокращений и обозначений

ОКВЭД — Общероссийский классификатор видов экономической деятельности

СУБД — Система управления базами данных

УСН — Упрощённая система налогообложения

API — Application Programming Interface (Интерфейс программирования приложений)

BMR — Basal Metabolic Rate (Основной обмен веществ)

FSM — Finite State Machine (Конечный автомат)

IT — Information Technology (Информационные технологии)

UML — Unified Modeling Language (Унифицированный язык моделирования)

					ПГУ1.090301.38.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		5

Введение

В современной жизни, такие вопросы как здоровье личностного и физического плана стоят все на более важном месте. Это связано отчасти с увеличением количества заболеваний, вызванных сидячим образом жизни и несбалансированным питанием. По этой причине создание новых методов поддержки здоровья и физической формы имеет первостепенное значение. Поэтому было принято решение создать систему на платформе Telegram в виде бота, который предоставит пользователям доступ к ценным ресурсам о здоровье, фитнесе и питании.

Telegram, будучи одним из самых популярных приложений для обмена сообщениями, предоставляет широкие перспективы для налаживания связей с аудиторией, что делает его прекрасной средой для интеграции фитнес-инструментов [1]. Автоматизированные программы в Telegram способны оптимизировать множество задач, обеспечивая пользователям комфортное и интуитивно понятное взаимодействие. Они могут генерировать персонализированные советы по упражнениям и питанию, вести учет достижений, а также отправлять уведомления о предстоящих тренировках, что, в свою очередь, стимулирует интерес пользователей и помогает им более осознанно подходить к заботе о своем здоровье.

Данная работа ставит перед собой задачу создания системы, где Telegram-бот выступает в роли индивидуального консультанта по вопросам здоровья и физической активности. В дипломной работе будут подробно рассмотрены принципы разработки и внедрения этой системы, включая ее функционал и достоинства. Предполагается, что разработанный бот значительно облегчит процесс достижения желаемых спортивных результатов и улучшит общее самочувствие пользователей, стимулируя их к формированию полезных привычек и ведению активного образа жизни.

Задачи выпускной квалификационной работы заключаются в следующем:

– приобретение навыков работы с современными методами разработки

					ПГУ1.090301.38.001 ПЗ	Лист
						6
Изм.	Лист	№ докум.	Подпись	Дата		

программного обеспечения, включая изучение Telegram API, платформ для создания ботов и средств интеграции с базами данных;

- анализ актуальных проблем в области здоровья и фитнеса, таких как недостаточная мотивация пользователей, отсутствие индивидуального подхода и трудности в отслеживании результатов;

- изучение потенциала telegram-ботов в сфере здоровья и фитнеса, анализ доступных решений, их сильных и слабых сторон;

- проектирование концепции и структуры системы, базирующейся на telegram-боте, с определением основных функций и вариантов взаимодействия с пользователем;

- формулирование требований к функционалу и интерфейсу бота, принимая во внимание потребность пользователей в простоте использования, персонализации и поддержании мотивации;

- создание прототипа системы с основными возможностями (составление плана тренировок, рекомендации по питанию, отслеживание прогресса) и его проверка на надежность, удобство и защищенность;

- оценка эффективности внедрения системы посредством пользовательского тестирования, анализ отзывов и показателей вовлеченности;

- сопоставительный анализ результатов с уже существующими аналогами, выявление преимуществ и недостатков разработанного бота;

- разработка предложений по дальнейшему совершенствованию системы, включая возможную интеграцию с другими сервисами (например, фитнес-браслетами), а также потенциальное расширение функциональности на основе пожеланий пользователей;

- подготовка общих советов по применению telegram-ботов для продвижения здорового образа жизни, увеличения мотивации и улучшения самочувствия пользователей.

Для успешной реализации намеченных целей требуется выполнение ряда конкретных шагов:

					ПГУ1.090301.38.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		7

- необходимо провести всесторонний анализ имеющихся фитнес-ботов, уделив особое внимание их функционалу;
- формулирование основных требований, предъявляемых к разрабатываемой системе;
- проектирование архитектуры будущего бота;
- реализация главных функций, включая составление тренировок, мониторинг рациона питания и отслеживание личного прогресса;
- осуществить проверку стабильности и корректности работы разработанной системы.

1 Бизнес-проект «Разработка системы на основе telegram-бота для здоровья и фитнеса»

1.1 Анализ предметной области

Для разработки эффективной и функциональной системы на основе Telegram-бота, ориентированной на здоровье и фитнес, необходимо провести комплексный анализ текущих процессов и определить потребности пользователей. В рамках данного анализа была проведена оценка существующих методов поддержки здоровья и физической активности, а также выявлены задачи, которые могут быть оптимизированы с помощью Telegram-бота.

Первым шагом анализа стало изучение ключевых процессов, связанных с фитнесом и здоровьем. Это включает в себя описание методов регистрации пользователей, предоставления рекомендаций по тренировкам и питанию, отслеживания прогресса, а также взаимодействия с тренерами и специалистами. Каждый из этих процессов был разбит на отдельные этапы и тщательно проанализирован. Например, для регистрации в фитнес-программе пользователю необходимо было заполнить анкету, выбрать подходящий план и дожидаться обратной связи от тренера. Однако с помощью Telegram-бота этот процесс можно значительно упростить: пользователю достаточно сделать несколько кликов для регистрации и получения рекомендаций.

Далее были определены потребности и требования пользователей. Ключевыми аспектами являются: удобство использования системы, безопасность данных, скорость обработки запросов, доступность информации и надежность работы платформы.

Также были выявлены проблемы, связанные с текущими подходами к фитнес-услугам. К ним относятся недостаток персонализированных рекомендаций, сложности в отслеживании прогресса, а также ограниченные возможности для обратной связи с тренерами. Все эти проблемы могут быть эффективно решены с помощью Telegram-бота, который обеспечивает

					ПГУ1.090301.38.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		9

интерактивное и персонализированное взаимодействие.

На основе проведенного анализа были сформулированы требования к новой системе. Это включает в себя простоту и интуитивность интерфейса, безопасность личных данных, доступность и надежность. Также подчеркивается необходимость автоматизации процессов, чтобы минимизировать ручной труд и ускорить выполнение рутинных задач.

Таким образом, результаты анализа подтвердили, что информационная система на основе Telegram-бота может значительно улучшить качество предоставляемых услуг в области здоровья и фитнеса. Разработка такого решения позволит повысить эффективность взаимодействия с пользователями, упростить процесс достижения фитнес-целей и способствовать формированию здорового образа жизни.

1.2 Анализ решения поставленной задачи

Telegram-бот представляет собой программное приложение, функционирующее на платформе Telegram и обеспечивающее взаимодействие с пользователем посредством сообщений и возможностей чата [1]. В современном бизнесе телеграм-боты активно применяются в различных сферах: от банковских услуг и ресторанного бизнеса до медицины, авиации и онлайн-торговли [2]. Их основная задача — автоматизация клиентского обслуживания и оптимизация внутренних процессов компаний. Рассмотрим ключевые преимущества использования телеграм-бота при разработке комплексной информационной системы для здоровья и фитнеса:

– доступность. Telegram-бот доступен всем пользователям мессенджера Telegram, что обеспечивает простоту и удобство взаимодействия без необходимости устанавливать дополнительное программное обеспечение;

– простота использования. Интуитивно понятный интерфейс позволяет пользователям работать с ботом без необходимости запоминать сложные команды, что снижает порог входа и повышает лояльность клиентов;

					ПГУ1.090301.38.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		10

– высокая скорость обработки запросов. Мгновенная реакция бота на запросы обеспечивает оперативное предоставление информации и услуг, что особенно важно для динамичных процессов фитнес-индустрии;

– автоматизация процессов. Телеграм-бот способен автоматизировать ключевые операции, такие как запись на тренировки, управление расписанием, обработка платежей и проведение тестирований, что существенно снижает нагрузку на персонал и повышает эффективность работы;

– безопасность. Использование протокола шифрования TLS гарантирует защиту передаваемых данных, что критично при работе с персональной информацией пользователей;

– надежность. Работа бота на серверах Telegram обеспечивает высокую стабильность и минимальные простои, что важно для поддержания непрерывного сервиса;

– многофункциональность и гибкость. Телеграм-бот можно адаптировать под различные задачи и требования, интегрировать с внешними системами и расширять функционал в зависимости от потребностей бизнеса;

– экономическая эффективность. Разработка телеграм-бота требует значительно меньше ресурсов и времени по сравнению с созданием полноценного мобильного приложения, что снижает затраты проекта;

– персонализация. Возможность настройки бота под индивидуальные предпочтения пользователя способствует улучшению пользовательского опыта и повышению вовлеченности;

– улучшение клиентского сервиса. Телеграм-бот обеспечивает круглосуточную поддержку, позволяя клиентам быстро получать ответы на вопросы, оформлять услуги и взаимодействовать с компанией в удобное время.

Таким образом, внедрение телеграм-бота в систему здоровья и фитнеса представляет собой эффективное решение поставленной задачи, позволяющее повысить качество обслуживания, оптимизировать внутренние процессы и обеспечить удобный доступ пользователей к необходимым сервисам.

1.3 Анализ актуальности разработки

Разработка системы на основе телеграм-бота для здоровья и фитнеса является актуальной в современном обществе по нескольким ключевым причинам.

Во-первых, здоровье и физическое благополучие становятся все более важными аспектами жизни людей. В условиях современного ритма жизни, когда многие сталкиваются с нехваткой времени, использование технологий для упрощения доступа к фитнес-услугам становится необходимостью. Телеграм-бот может стать удобным инструментом для пользователей, позволяя им быстро записываться на тренировки, получать рекомендации по питанию и отслеживать свои достижения.

Во-вторых, современные технологии значительно упрощают процессы управления и взаимодействия с клиентами. Также использование телеграм-бота для автоматизации процессов, таких как управление расписанием, оплата услуг и предоставление информации о тренировках, позволяет снизить нагрузку на персонал и повысить эффективность работы фитнес-центров. Это, в свою очередь, способствует улучшению качества обслуживания клиентов.

В-третьих, телеграм-боты становятся все более популярными среди пользователей, что делает их идеальным инструментом для бизнеса. Они обеспечивают мгновенный доступ к услугам и информации, а также возможность диалогового взаимодействия с клиентами. Это особенно важно в сфере фитнеса, где обратная связь и поддержка могут значительно повысить уровень мотивации пользователей.

Кроме того, все больше людей предпочитают осуществлять свои действия без необходимости прямого общения. В этом контексте, создание системы на базе телеграм-бота для здоровья и фитнеса предоставит пользователям удобный и безопасный способ управления своими тренировками и здоровьем. Это позволяет не только улучшить пользовательский опыт, но и адаптироваться к современным требованиям рынка.

					ПГУ1.090301.38.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		12

Таким образом, предложенный проект по разработке телеграм-бота для здоровья и фитнеса имеет высокую актуальность и значимость как для конечных пользователей, так и для развития фитнес-индустрии в целом. Он способствует оптимизации процессов, улучшению качества обслуживания и повышению доступности фитнес-услуг для широкой аудитории.

1.4 Анализ аналогов информационных систем

Обзор аналогов является важной частью разработки информационной системы, так как позволяет изучить подобные решения и определить их преимущества и недостатки. В нашем случае, был исследован рынок информационных систем, связанные с здоровьем и фитнесом, с целью выявления существующих аналогов на основе телеграм-ботов. Несмотря на то, что полноценной информационной системы на основе телеграм-бота не существует, можно провести сравнение с существующими мобильными приложениями, сайтами и некоторыми устройствами.

В качестве аналогов Telegram-бота для здоровья и фитнеса можно рассмотреть такие варианты, как мобильное приложение «MyFitnessPal», сайт «Bodybuilding.com», а также гаджеты и носимые устройства, например, Apple Watch. Каждый из этих инструментов имеет свои уникальные особенности, преимущества и недостатки.

MyFitnessPal.

MyFitnessPal — это популярное мобильное приложение, предназначенное для отслеживания питания и физической активности [3]. Оно предлагает пользователям обширную базу данных продуктов, что позволяет легко записывать потребляемые калории и макроэлементы. Приложение также включает функции социальной сети, где пользователи могут делиться своими успехами, получать поддержку и советы от других. Благодаря интеграции с множеством фитнес-приложений и устройств, MyFitnessPal помогает создать полную картину здоровья и физической активности. На рисунке 1 изображён логотип, а на рисунке 2 показан

интерфейс данного приложения.



Рисунок 1 - Логотип MyFitnessPal

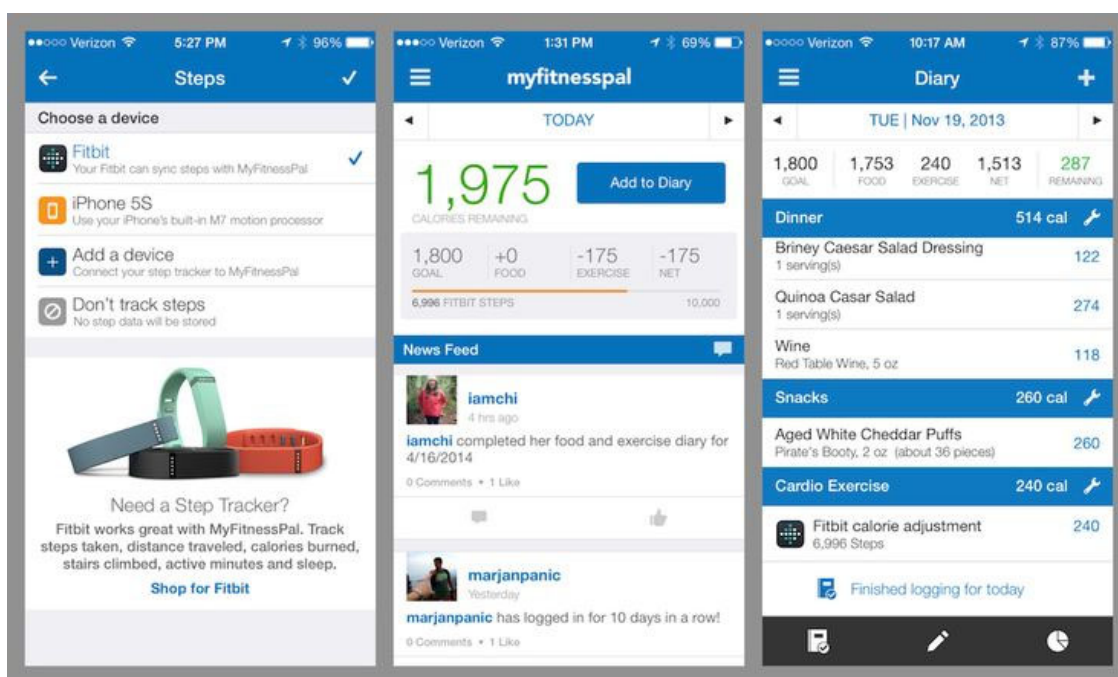


Рисунок 2 - Интерфейс приложения

Преимущества:

- MyFitnessPal предлагает обширную базу данных продуктов, что позволяет легко отслеживать калорийность и макроэлементы. Пользователи могут сканировать штрих-коды и добавлять свои блюда, что делает процесс удобным;
- приложение включает в себя функции социальной сети, где пользователи могут делиться своими успехами, получать поддержку и советы от других;
- MyFitnessPal поддерживает интеграцию с множеством фитнес-приложений и устройств, что позволяет получать полную картину здоровья и физической активности.

Недостатки:

- некоторые пользователи сообщают о неточностях в базе данных, что может привести к ошибкам в отслеживании калорийности;
- хотя приложение бесплатное, некоторые функции доступны только в платной версии, что может ограничивать пользователей с ограниченным бюджетом.

Bodybuilding.com.

Bodybuilding.com — это веб-сайт, который предоставляет обширную информацию о тренировках, питании и фитнесе [4]. Он предлагает пользователям множество статей, видео и руководств, что делает его ценным ресурсом для тех, кто хочет учиться и улучшать свои навыки. Сайт также включает активные форумы и сообщества, где пользователи могут обмениваться опытом, задавать вопросы и получать поддержку. Bodybuilding.com предлагает возможность создания персонализированных тренировочных планов и программ питания, что позволяет адаптировать подход к каждому пользователю. На рисунке 3 изображён логотип, а на рисунке 4 показан интерфейс данного сайта.

Преимущества:

- сайт предлагает множество статей, видео и руководств по тренировкам и питанию, что делает его ценным ресурсом для пользователей, стремящихся к



Рисунок 3 – Логотип Bodybuilding.com

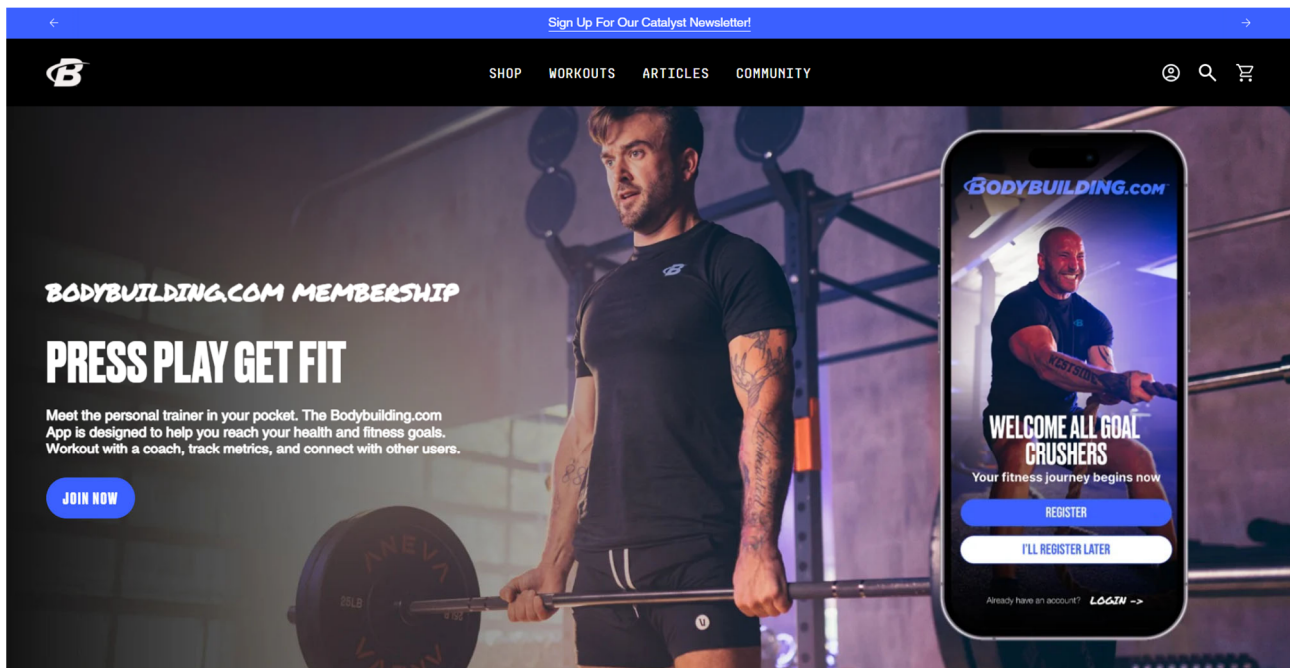


Рисунок 4 – Интерфейс сайта

обучению и улучшению своих навыков;

– Bodybuilding.com имеет активное сообщество, где пользователи могут обмениваться опытом, задавать вопросы и получать поддержку;

– платформа предлагает возможность создания индивидуальных тренировочных планов и программ питания, что позволяет адаптировать подход к каждому пользователю.

Недостатки:

– для новичков обилие информации может быть подавляющим, что затрудняет выбор правильного пути;

– хотя сайт оптимизирован для мобильных устройств, некоторые функции могут быть неудобны для использования на смартфонах.

Apple Watch.

Apple Watch — это носимое устройство, которое сочетает в себе функции умных часов и фитнес-трекера. Оно позволяет пользователям отслеживать физическую активность, сердечный ритм и другие показатели здоровья в реальном времени. Apple Watch интегрируется с множеством фитнес-приложений, предлагая пользователям уведомления о тренировках и

напоминания. Устройство также предлагает функции мониторинга здоровья, такие как отслеживание сна, уровень кислорода в крови и ЭКГ, что делает его универсальным инструментом для заботы о здоровье. Однако стоит отметить, что Apple Watch является дорогим устройством и требует iPhone для полноценного использования. На рисунке 5 изображён вид устройства, а на рисунке 6 показан приложение для тренировок на данном устройстве.



Рисунок 5 - Вид устройства



Рисунок 6 - Приложение на Apple Watch для тренировок

Преимущества:

– Apple Watch позволяет отслеживать физическую активность, сердечный ритм и другие показатели здоровья в режиме реального времени, что помогает пользователям оставаться мотивированными;

– устройство легко интегрируется с множеством фитнес-приложений и позволяет получать уведомления о тренировках и напоминания;

– Apple Watch предлагает функции мониторинга здоровья, такие как отслеживание сна, уровень кислорода в крови и ЭКГ, что делает его универсальным инструментом для заботы о здоровье.

Недостатки:

– Apple Watch является дорогим устройством, что может быть недоступно для многих пользователей;

– для полноценного использования устройства требуется iPhone, что ограничивает его доступность для пользователей других платформ.

После рассмотрения преимуществ и недостатков каждого из аналогов для здоровья и фитнеса, становится очевидным, что выбор подходящего инструмента зависит от индивидуальных предпочтений и целей. Краткое содержание всей информации о представленных аналогах описаны в таблице 1.

Таблица 1 - Краткое содержание анализа аналогов информационных систем

Название аналога	Преимущества аналога	Недостатки аналога
Мобильное приложение MyFitnessPal	Отслеживание питания. Наличие функции социальной сети. Возможность интеграции с другими приложениями.	Проблемы с точностью. Платные функции.
Веб-сайт Bodybuilder.com	Обширная информация. Наличие форумов и активного сообщества. Возможность настройки плана тренировок.	Перегруженность информации. Не оптимизирован для мобильных устройств.

Название аналога	Преимущества аналога	Недостатки аналога
Устройство Apple Watch	Отслеживание активности в реальном времени. Интеграция с другими приложениями. Наличие функции мониторинга здоровья.	Зависимость от экосистемы Apple. Стоимость устройств.

1.5 Анализ архитектуры и структура бота

Архитектура информационной системы, основанной на телеграм-боте для здоровья и бизнеса, включает три ключевых компонента: клиентское приложение (телеграм-бот), серверное приложение и базу данных.

Клиентское приложение представлено в виде телеграм-бота, который служит удобным интерфейсом для пользователей. Бот предоставляет множество возможностей, включая изменение количества шагов и калорий, предоставление видео с тренировками и так далее. Подробную диаграмму прецедентов можно изучить на рисунке 7.

Этот подход позволяет эффективно интегрировать функции здоровья и фитнеса, предлагая пользователям удобный и доступный способ взаимодействия с системой.

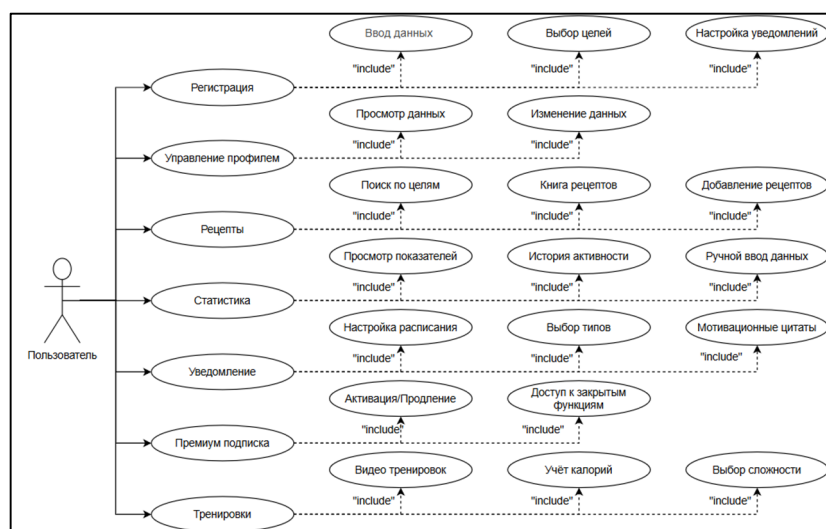


Рисунок 7 - UML-диаграмма прецедентов интерфейса пользователя

Для четкого понимания взаимодействия компонентов бота, необходимо создать UML-диаграмму последовательности. На рисунках 8-10 представлена данная диаграмма.

Также, помимо UML-диаграммы последовательности, были созданы диаграммы классов, которые также присутствуют в боте (рисунок 11).

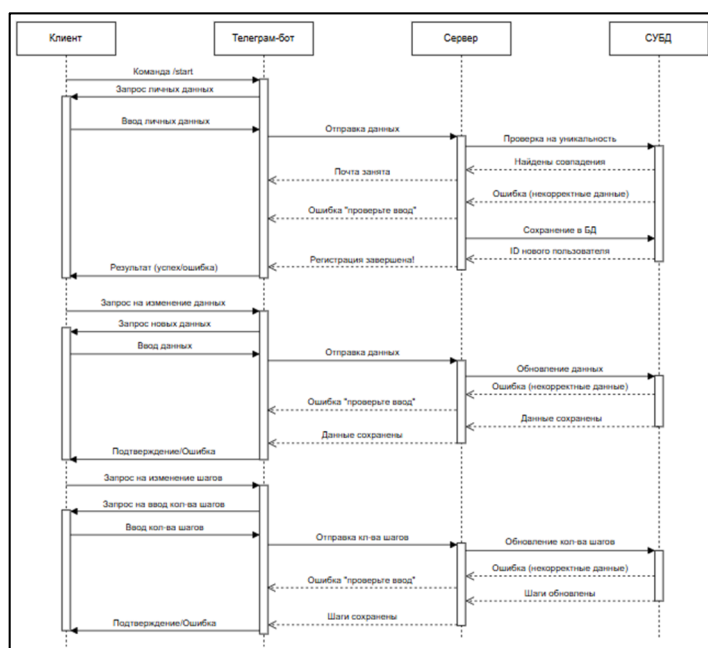


Рисунок 8 - UML-диаграмма последовательности (часть 1)

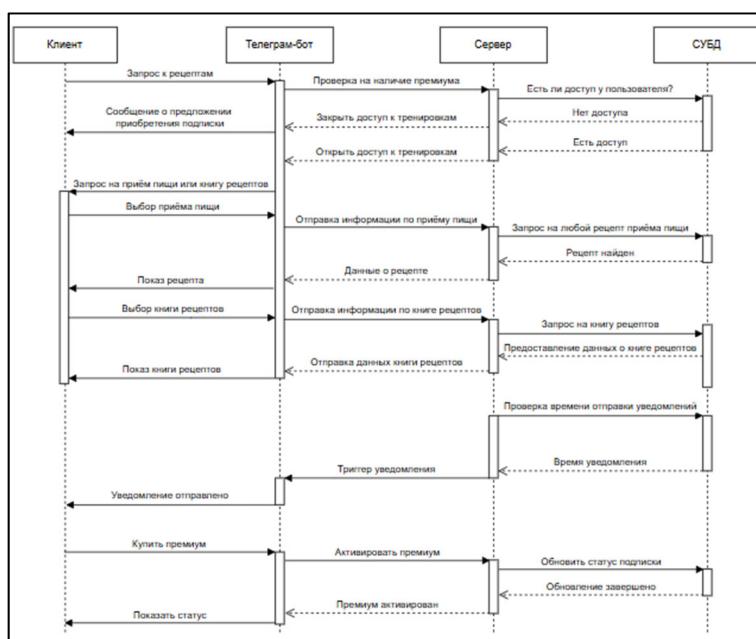


Рисунок 9 - UML-диаграмма последовательности (часть 2)

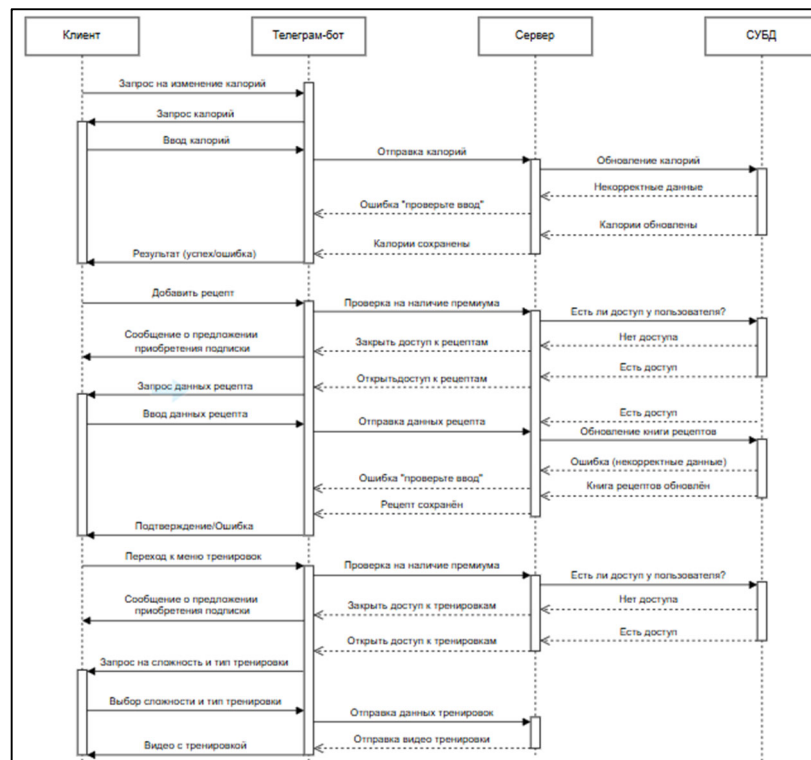


Рисунок 10 - UML-диаграмма последовательности (часть 3)

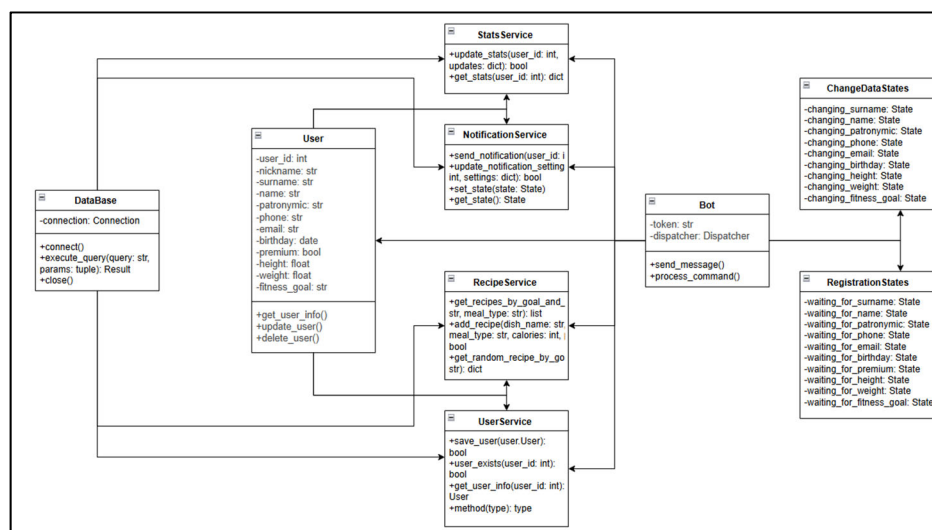


Рисунок 11 - UML-диаграмма классов телеграмм-бота

Описание классов:

- User. Класс, представляющий пользователя, с атрибутами, такими как user_id, nickname, surname, и методами для получения и обновления информации о пользователе;
- Bot. Класс, представляющий бота, который обрабатывает команды и

отправляет сообщения пользователям;

– User Service. Сервис, отвечающий за управление пользователями, включая сохранение, проверку существования и получение информации;

– StatsService. Сервис для работы со статистикой пользователей, включая обновление и получение статистики;

– NotificationService. Сервис для отправки уведомлений пользователям и обновления их настроек уведомлений;

– RecipeService. Сервис для работы с рецептами, включая получение рецептов по фитнес-целям и добавление новых рецептов;

– Database. Класс для работы с базой данных, включая подключение, выполнение запросов и закрытие соединения;

– состояния (RegistrationStates и ChangeDataStates). Классы, представляющие состояния, связанные с процессами регистрации и изменения данных пользователя.

1.6 Постановка задачи

Необходимо разработать информационную систему на основе телеграм-бота для здоровья и фитнеса, которая позволит пользователям эффективно управлять своими тренировками и питанием, а также отслеживать прогресс в достижении личных целей. Система должна обеспечить следующие ключевые функции:

– отслеживание активности:

1) запись количества шагов. Пользователь может вручную вводить количество пройденных шагов или синхронизировать данные с фитнес-трекером;

2) учёт калорий. Система позволяет отслеживать потребленные калории на основе введенной информации о еде. Пользователи могут добавлять продукты вручную или выбирать из базы данных;

– предоставление видео-тренировок:

1) доступ к видеоматериалам. Пользователи могут просматривать видео с

демонстрацией различных упражнений;

2) фильтрация по типу тренировки. Возможность выбирать видео по категориям, таким как кардио, силовые тренировки, растяжка и так далее;

– рецепты для здорового питания:

1) персонализированные рецепты. Система предлагает рецепты на основе предпочтений пользователя и его целей (похудение, набор массы и т.д.);

2) копирование рецептов. Пользователи могут копировать рецепты для удобного использования вне приложения;

– отправление мотивационных сообщений:

1) автоматические напоминания. Пользователи получают мотивационные сообщения и напоминания о тренировках;

2) настройка уведомлений. Возможность выбрать время и частоту отправки сообщений;

– обратная связь. Пользователи могут отправлять сообщения разработчику для устранения неполадок или предложения улучшений;

– управление профилем:

1) регистрация и вход. Пользователи могут создавать аккаунты и управлять своими профилями;

2) статистика. Доступ к истории активности и прогрессу;

– премиум функции:

1) доступ к закрытым функциям. Пользователи с премиум подпиской получают доступ к дополнительным функциям, таким как эксклюзивные тренировки и рецепты;

2) настройка расписания;

3) планирование тренировок и питания. Пользователи могут настраивать расписание для тренировок и приёмов пищи;

– история активности. Система сохраняет историю активности пользователя, позволяя анализировать достижения;

– ручной ввод данных. Пользователи могут вручную вводить данные, если

автоматическая синхронизация недоступна.

1.7 Выбор инструментов разработки

1.7.1 Основной стек инструментов разработки

Основным языком программирования для разработки Telegram-бота выбран Python [5]. Этот язык обладает рядом ключевых преимуществ, делающих его идеальным для данного проекта. Во-первых, Python отличается простым и читаемым синтаксисом, что значительно ускоряет процесс разработки и снижает порог входа для новых участников команды. Во-вторых, он поддерживает асинхронное программирование через библиотеки `asyncio` и `aiohttp`, что критически важно для обработки множества одновременных запросов пользователей. В-третьих, экосистема Python предлагает богатый набор специализированных библиотек для работы с Telegram API, включая современный асинхронный фреймворк `aiogram 3.x`.

Для backend-разработки используется фреймворк `aiogram 3.x` — наиболее современное и полное решение для создания Telegram-ботов на Python [6]. Его преимущества перед аналогами (`python-telegram-bot`, `Telebot`) включают: полноценную поддержку асинхронности, встроенный Finite State Machine (FSM) для сложных диалогов, а также активное сообщество разработчиков. В качестве базы данных выбрана PostgreSQL — надежная реляционная СУБД [7], идеально подходящая для структурированного хранения пользовательских данных, тренировок и рецептов.

1.8 Альтернативные инструменты разработки

1.8.1 Альтернативные языки программирования

JavaScript (Node.js).

Node.js с его JavaScript-экосистемой представляет собой мощную платформу для создания Telegram-ботов, особенно популярную среди веб-разработчиков благодаря своей универсальности.

					ПГУ1.090301.38.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		24

Преимущества:

- высокая производительность благодаря движку V8, который обеспечивает быструю обработку запросов;
- встроенная асинхронная модель (Promise, async/await) идеально подходит для обработки сообщений в реальном времени;
- идеальная интеграция с веб-технологиями особенно удобна, если бот является частью веб-приложения.

Недостатки:

- слабая поддержка машинного обучения отсутствие полноценных аналогов Pandas/NumPy затрудняет сложный анализ данных;
- ограниченные возможности для реализации сложных сценариев (например, конечные автоматы уступают Python-реализациям).

Java/Kotlin.

Java и Kotlin предлагают корпоративный подход к разработке Telegram-ботов, обеспечивая высокую надежность и стабильность работы.

Преимущества:

- стабильность и надежность JVM обеспечивает предсказуемое выполнение кода;
- эффективная многопоточность отличная поддержка параллельных вычислений;
- безопасность строгая типизация и зрелая экосистема уменьшают количество уязвимостей.

Недостатки:

- избыточный boilerplate-код требуется больше шаблонных конструкций по сравнению с Python/JS;
- медленный процесс прототипирования сложнее быстро тестировать новые идеи и концепции.

C (.NET).

C и платформа .NET предлагают сбалансированный вариант для разработки

					ПГУ1.090301.38.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		25

Telegram-ботов, особенно в экосистеме Microsoft.

Преимущества:

- отличная документация Microsoft обеспечивает подробные руководства и примеры;
- современная асинхронная модель полная поддержка `async/await`;
- глубокая интеграция с сервисами Microsoft удобно для развертывания в Azure.

Недостатки:

- ограниченность экосистемой Microsoft оптимально работает преимущественно в Windows-среде;
- меньшее сообщество по сравнению с Python/JS-разработчиками.

На основе того, что указано выше, была создана таблица сравнения с языком программирования Python (таблица 2).

Таблица 2 - сравнения языков программирования

Критерий	Python (основной выбор)	JavaScript/Node.js	Java/Kotlin	C (.NET)
Синтаксис	Простой и читаемый	Удобный, С-подобный	Многословн ый	Умеренно сложный
Асинхроннос ть	<code>asyncio</code> , <code>aiohttp</code>	<code>Promise</code> , <code>async/await</code>	Limited	<code>async/await</code>
Производител ьность	Средняя	Высокая (V8)	Высокая (JVM)	Высокая
Библиотеки для Telegram	<code>aiogram</code> , <code>PTB</code> (богатый выбор)	<code>Telegraf.js</code> , <code>grammY</code>	<code>TelegramBots</code> , <code>kotlin-telegram-bot</code>	<code>Telegram.Bot</code>
Интеграция с вебом	Средняя	Отличная	Средняя	Средняя

Критерий	Python (основной выбор)	JavaScript/Node.js	Java/Kotlin	C (.NET)
Сложные сценарии	Отличная поддержка (FSM и др.)	Ограниченная	Хорошая	Хорошая
Скорость разработки	Очень высокая	Высокая	Низкая	Средняя
Многопоточн ость	GIL ограничивает	Event-loop	Отличная	Хорошая
Лучший сценарий	Универсальные боты, ML- интеграция	Веб- интегрированные боты	Корпоративн ые решения	Windows- серверные приложения
Порог входа	Низкий	Средний	Высокий	Средний

1.8.2 Альтернативные фреймворки

Python-telegram-bot.

Python-telegram-bot (PTB) — один из старейших и наиболее надежных фреймворков для разработки Telegram-ботов на Python. Он официально рекомендован Telegram и активно поддерживается сообществом. Благодаря своей зрелости и стабильности, PTB подходит как для небольших ботов, так и для сложных проектов с высокой нагрузкой.

Преимущества:

- полная поддержка Telegram Bot API. Включает все актуальные методы и функции Telegram;
- гибкость работы. Начиная с версии 20+, поддерживает как синхронный, так и асинхронный (async/await) режимы, что позволяет оптимизировать производительность;
- отличная документация и активное сообщество. Легко найти ответы на вопросы, готовые примеры и дополнительные библиотеки;
- мощная система обработчиков (Handlers). Удобная фильтрация входящих

сообщений и команд, что упрощает логику бота.

Недостатки:

- до версии 2.0 был только синхронным. Старые проекты могут требовать модернизации для перехода на асинхронность;
- более многословный код. По сравнению с `aiogram`, требует больше строк кода для аналогичных задач, что может замедлять разработку.

`pyTelegramBotAPI` (`telebot`).

`pyTelegramBotAPI` (также известный как `telebot`) — это легковесный и удобный фреймворк, который отлично подходит для быстрой разработки Telegram-ботов. Его минималистичный подход делает его идеальным выбором для новичков и небольших проектов, где важна простота и скорость внедрения.

Преимущества:

- простота освоения. Интуитивно понятный синтаксис позволяет быстро начать работу даже без глубокого знания Python;
- минималистичный код. Требуется меньше boilerplate-кода по сравнению с `PTB` и `aiogram`;
- поддержка синхронных и асинхронных операций. Можно выбирать оптимальный подход в зависимости от задачи;
- встроенная поддержка `middleware`. Упрощает обработку запросов (например, логирование, проверку прав доступа).

Недостатки:

- менее гибкий, чем `PTB` или `aiogram`. Ограниченные возможности для сложных сценариев, таких как FSM (Finite State Machine) или кастомные фильтры;
- меньше возможностей для масштабирования. Может не подходить для высоконагруженных ботов с продвинутой бизнес-логикой.

На основе того, что указано выше, была создана таблица сравнения с фреймворком `Aiogram 3.x` (таблица 3).

Таблица 3 - сравнение фреймворков

					ПГУ1.090301.38.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		28

Критерий	Aiogram 3.x	Python-telegram-bot (PTB)	pyTelegramBotAPI (Telebot)
Тип	Асинхронный фреймворк	Синхронный/асинхронный (v20+)	Синхронный/асинхронный
Поддержка API	Полная (включая новые фичи)	Полная (официально рекомендован)	Полная
Производительность	Высокая (асинхронная архитектура)	Средняя/высокая	Средняя
FSM (машина состояний)	Встроенная поддержка	Требуется ручной реализации	Ограниченная поддержка
Сложность изучения	Средняя	Средняя	Низкая
Кодовая база	Компактная и выразительная	Многословная	Минималистичная
Сообщество	Активное русскоязычное	Большое международное	Среднее
Документация	Хорошая (улучшается в v3)	Отличная	Удовлетворительная
Лучший сценарий	Сложные боты с диалогами	Надежные production-решения	Прототипы и простые боты
Масштабируемость	Отличная	Хорошая	Ограниченная
Пример использования	async with aiogram	await bot.send_message()	bot.send_message()

1.8.3 Альтернативные базы данных

MySQL.

MySQL— это зрелая реляционная СУБД с открытым исходным кодом, которые широко используются в веб-разработке. Они предлагают хороший баланс между производительностью и функциональностью.

Преимущества:

– оптимизированы для операций чтения. Быстро работают с часто

запрашиваемыми данными (например, профилями пользователей);

- широкая поддержка хостинг-провайдеров. Доступны на большинстве shared и VPS-хостингов;

- хорошая интеграция с ORM. Совместимы с популярными системами объектно-реляционного отображения (SQLAlchemy, Django ORM, Sequelize);

- надежность. Проверенная временем архитектура с поддержкой ACID-транзакций.

Недостатки:

- ограниченная работа с JSON. Функциональность уступает PostgreSQL (важно для ботов с гибкими структурами данных);

- слабая поддержка геоданных. Сложнее реализовать функции, связанные с геолокацией;

- жесткая схема данных. Требуется тщательного проектирования структуры БД заранее.

SQLite.

SQLite — это встраиваемая реляционная СУБД, которая хранит всю базу данных в одном файле. Не требует отдельного серверного процесса, что делает ее исключительно простой в использовании.

Преимущества:

- простота развертывания. Не требует установки сервера, работает прямо из файла;

- идеально для небольших проектов. Хорошо справляется с нагрузкой до ~10 000 пользователей;

- нулевая настройка. Можно начать использовать сразу после установки библиотеки;

- низкие требования. Отлично работает даже на слабых серверах;

- полноценный SQL. Поддерживает большинство стандартных SQL-запросов.

Недостатки:

					ПГУ1.090301.38.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		30

– ограниченная параллельная запись. Только одна операция записи может выполняться в один момент времени;

– нет сетевого доступа. База доступна только локально на машине, где расположен файл;

– ограниченное масштабирование. Не подходит для высоконагруженных проектов;

– нет пользователей и прав доступа. Все или ничего с точки зрения доступа к данным.

На основе того, что указано выше, была создана таблица сравнения с языком программирования Python (таблица 4).

Таблица 4 - сравнение баз данных

Характеристика	PostgreSQL	MySQL	SQLite
Тип СУБД	Реляционная (объектно-реляционная)	Реляционная	Встраиваемая реляционная
Архитектура	Клиент-серверная	Клиент-серверная	Файловая
Масштабируемость	Отличная (для больших проектов)	Хорошая	Очень ограниченная
Производительность	Высокая (оптимизирована для сложных запросов)	Высокая (особенно на чтение)	Средняя (ограничена файловой системой)
Поддержка JSON	Полноценная (операции, индексы)	Ограниченная	Базовая
Геоданные	Расширения PostGIS	Ограниченная поддержка	Отсутствует
Параллельные запросы	Полная поддержка	Полная поддержка	Только чтение параллельно
Требования к серверу	Высокие	Средние	Минимальные

Характеристика	PostgreSQL	MySQL	SQLite
Сложность настройки	Средняя	Низкая	Нулевая
Безопасность	Расширенные механизмы доступа	Стандартные	Нет (все или ничего)
Лучший сценарий	Сложные боты с аналитикой	Веб-интегрированные боты	Прототипы и локальные приложения
Пример использования	Хранение сложных структур данных	Традиционные веб-приложения	Мобильные приложения, тестирование

1.9 Реализация

1.9.1 Регистрация телеграм-бота

Регистрация телеграм-бота является первым шагом для создания и интеграции бота в информационную систему. В этом разделе будет описана подробная процедура регистрации телеграм-бота.

Для начала нужно открыть официальное приложение Telegram на мобильном устройстве или компьютере и создать новый аккаунт, если его еще нет. Если уже есть аккаунт, то нужно войти в него. Главное меню показано на рисунке 12.

Далее нужно найти в поиске Telegram бота с именем "BotFather" — это официальный бот Telegram для создания новых ботов. Пример показан на рисунке 13.

После того, как чат с ботом открыт, необходимо начать диалог с ботом командой "/start". Это видно на рисунке 14.

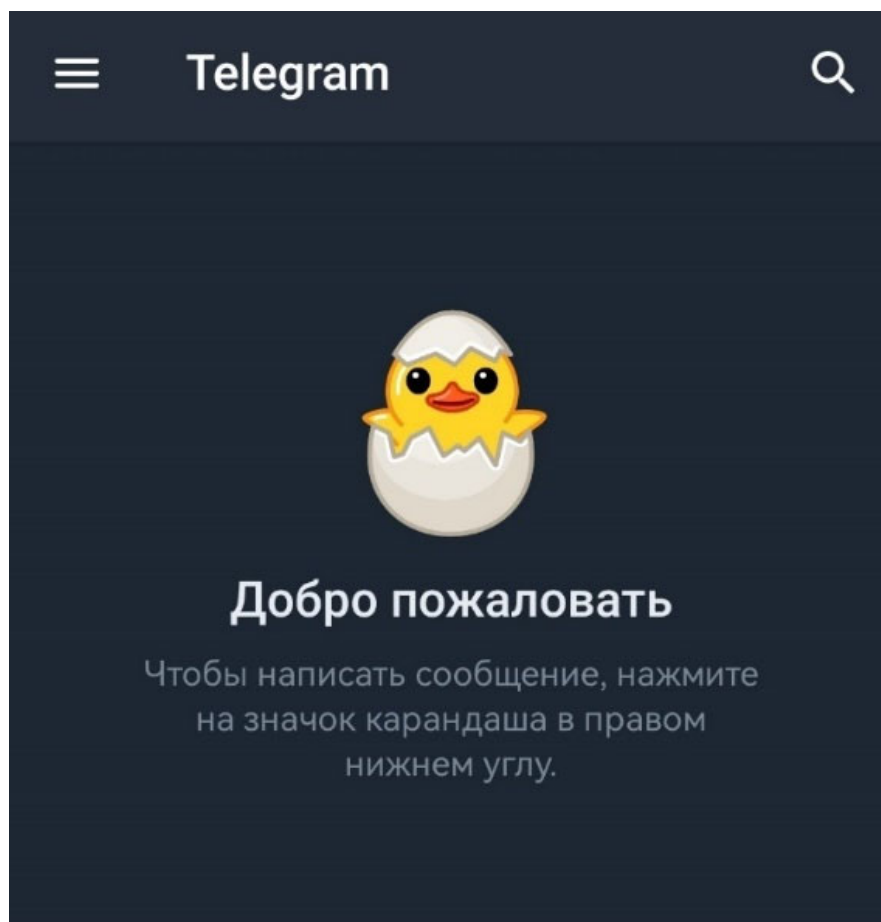


Рисунок 12 - Главное меню



Рисунок 13 - Поиск *BotFather*



Рисунок 14 - Начало диалога с BotFather

Для создания любого бота необходимо выполнить команду `/newbot`, а затем присвоить боту имя, которое будет отображаться в поиске, и логин, который будет отображаться в ссылке. Как это сделал я, показано на рисунке 15.

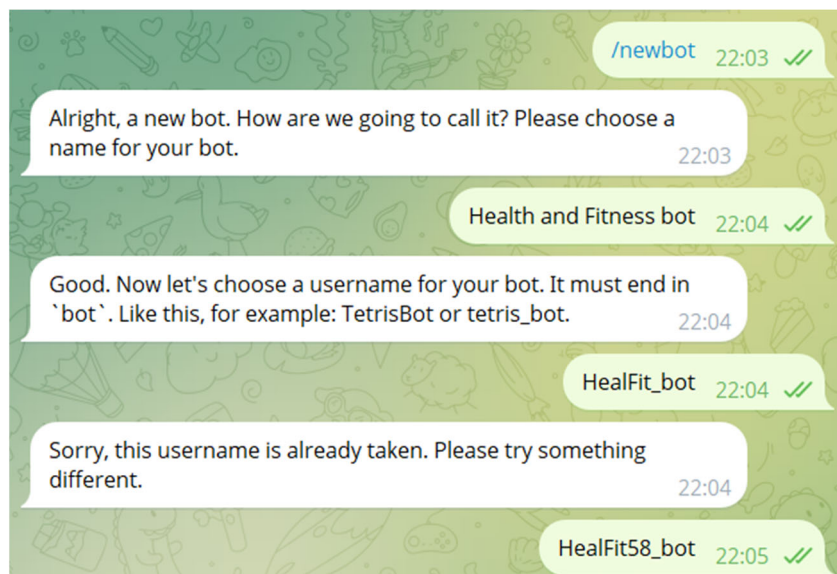


Рисунок 15 – Выбор имени бота

Далее BotFather предоставляет токен, который является ключом доступа к вашему телеграм-боту. Необходимо сохранить полученный от BotFather токен в надежном и безопасном месте. Токен будет использоваться для аутентификации вашего бота при взаимодействии с Telegram API. Информация, полученная от бота показана на рисунке 16.

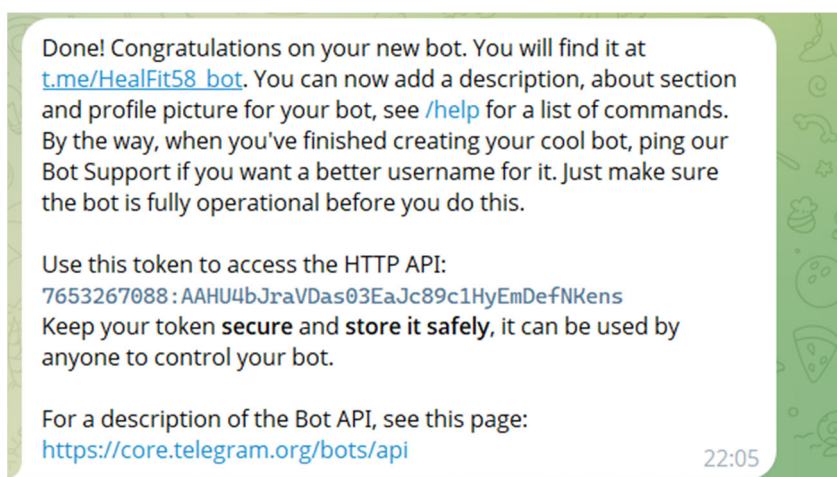


Рисунок 16 - Получение ссылки на бот

После завершения всех этих шагов телеграм-бот успешно зарегистрирован. Теперь можно использовать токен бота для настройки и интеграции бота в информационную систему автошколы.

1.9.2 Реализация базы данных

Для хранения информации необходимо реализовать базу данных

Основные таблицы:

– users основная таблица пользователей, содержащая:

- 1) персональные данные (ФИО, контакты);
- 2) физические параметры (рост, вес);
- 3) фитнес-цель и статус подписки;
- 4) дату регистрации;

– users_stats таблица статистики пользователей:

- 1) дневная активность (шаги, калории);
- 2) персональные цели;
- 3) настройки уведомлений;
- 4) связь с основной таблицей через user_id;

– recipes таблица рецептов:

- 1) блюда с группировкой по целям и типам питания;
- 2) калорийность и способ приготовления;
- 3) поддержка мультиязычности (русские названия типов питания);

– user_notification таблица уведомлений пользователей:

- 1) статусы уведомлений;
- 2) время отправки уведомлений;
- 3) время последнего отправленного уведомления;

– premium_subscription таблица состояния премиум-статуса пользователей:

- 1) описание подписки (1 месяц, 3 месяца, 6 месяцев, вечная);
- 2) дата окончания подписки.

Динамические таблицы:

Для каждого пользователя автоматически создаются таблицы истории формата user_[id]_history, содержащие:

- ежедневную статистику;
- историю изменений показателей.

Для наглядного понимания работы БД, на рисунке 17 представлена схема физической модели базы данных.

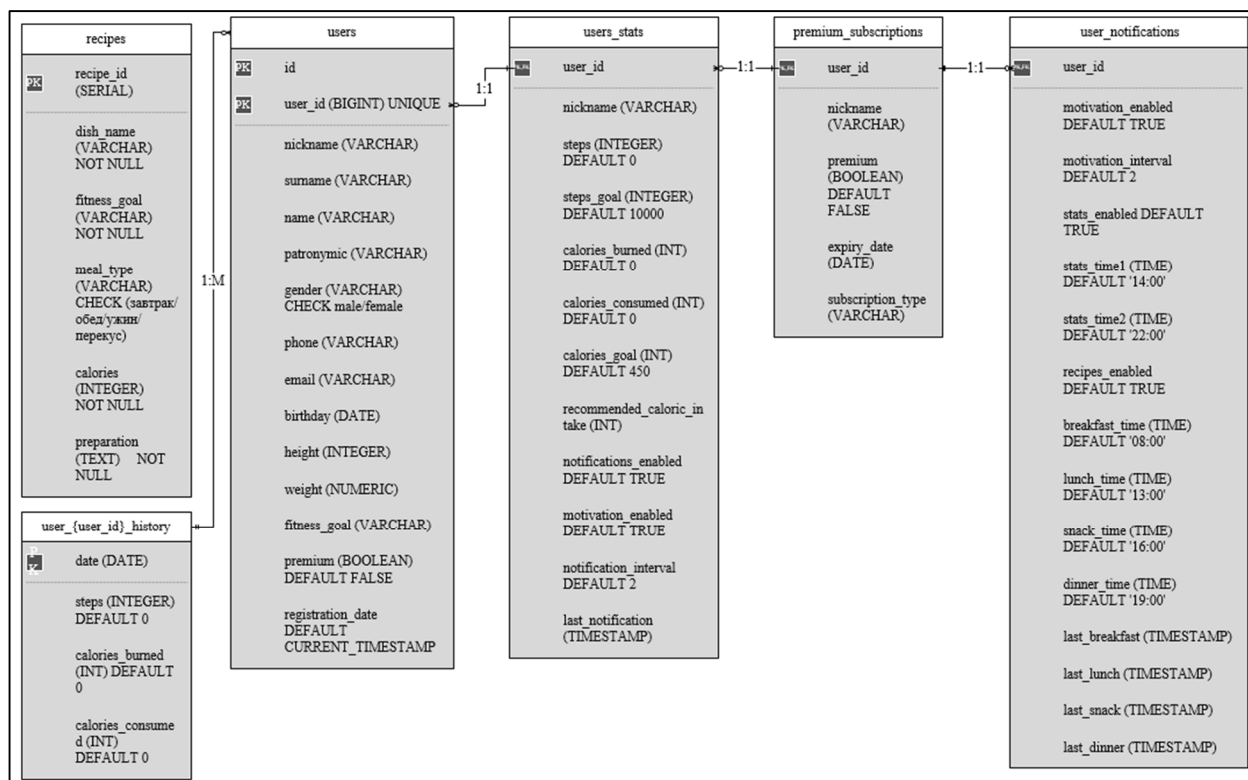


Рисунок 17 - Физическая модель базы данных

1.9.3 Описание доступных функций

1.9.3.1 Общие сведения

Информационная система представляет собой телеграм-бота, предназначенного для учета физической активности, контроля питания и управления персональными фитнес-целями. Информационная система фитнес-бота реализует следующие функции:

1) ч

Перед использованием бота, его нужно найти, это можно сделать 3 способами: перейти непосредственно по ссылке, найти в поиске, как показано на рисунке 18, или воспользоваться QR-кодом, показанным на рисунке 19.

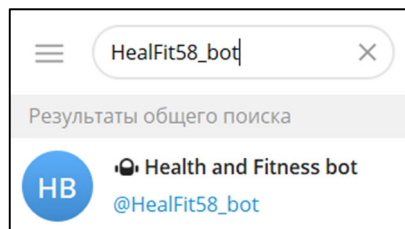


Рисунок 18 - Поиск бота в поисковике Telegram

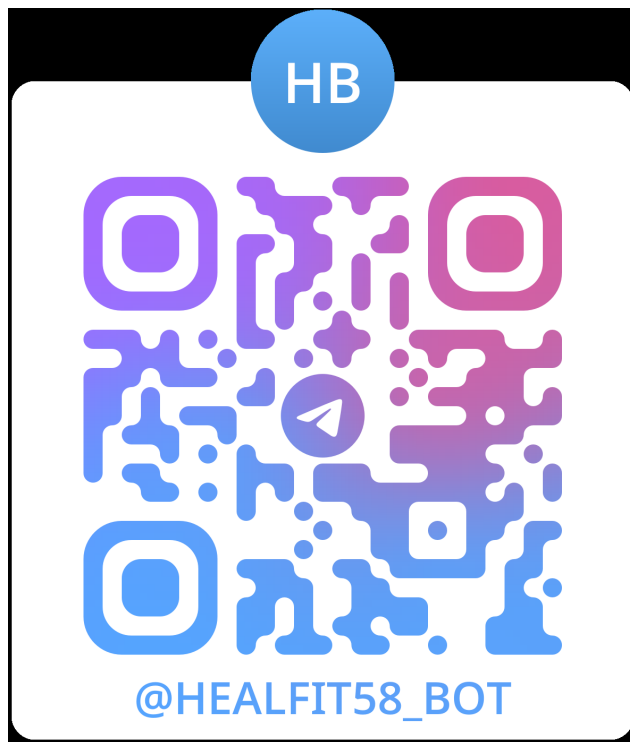


Рисунок 19 - Доступ к боту по QR

1.9.3.2 Регистрация пользователя

Далее нужно нажать на кнопку «Старт» для начала использования бота.

Пример начала работы показан на рисунке 20.

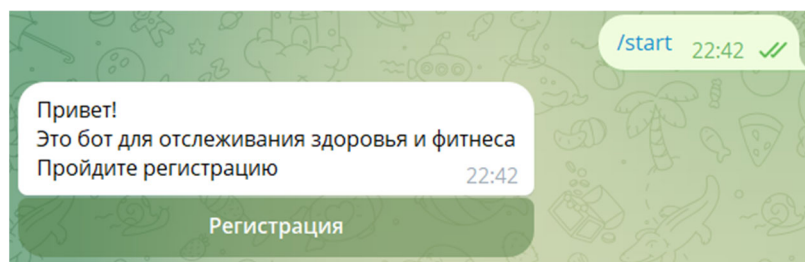


Рисунок 20 - Результат нажатия на /start

После старта работы бот приветствует вас после чего пользователю необходимо зарегистрироваться. Для этого необходимо ввести следующие данные:

- фамилия, имя, отчество (не более 90 символов, только буквенные символы) (рисунок 21);
- выбор пола (мужской или женский) (рисунок 22);
- номер телефона (формат +7XXX... или 8XXX..., проверка на уникальность) (рисунок 23);
- электронная почта (необязательное поле с валидацией формата) (рисунок 24);
- указание биометрических параметров (рисунок 25):
 - 1) дата рождения (проверка на возрастное ограничение 18+);
 - 2) рост (100-250 см);
 - 3) вес (30-200 кг);
- выбор фитнес-цели (рисунок 26):
 - 1) похудение;
 - 2) набор мышечной массы;
 - 3) поддержание здорового образа жизни;
- настройка уведомлений (рисунок 27):
 - 1) мотивационные уведомления;
 - 2) уведомления статистики;
 - 3) уведомления рецептов (доступно только с премиумом);
- настройка подписки (рисунки 28, 29):
 - 1) бесплатный базовый доступ;
 - 2) премиум-статус с расширенными возможностями.

После регистрации покажется сообщение об завершении регистрации вместе с сообщением, есть ли у пользователя подписка или нет (рисунок 30).

После нажатия на кнопку пользователь переходит в главное меню (рисунок 31).

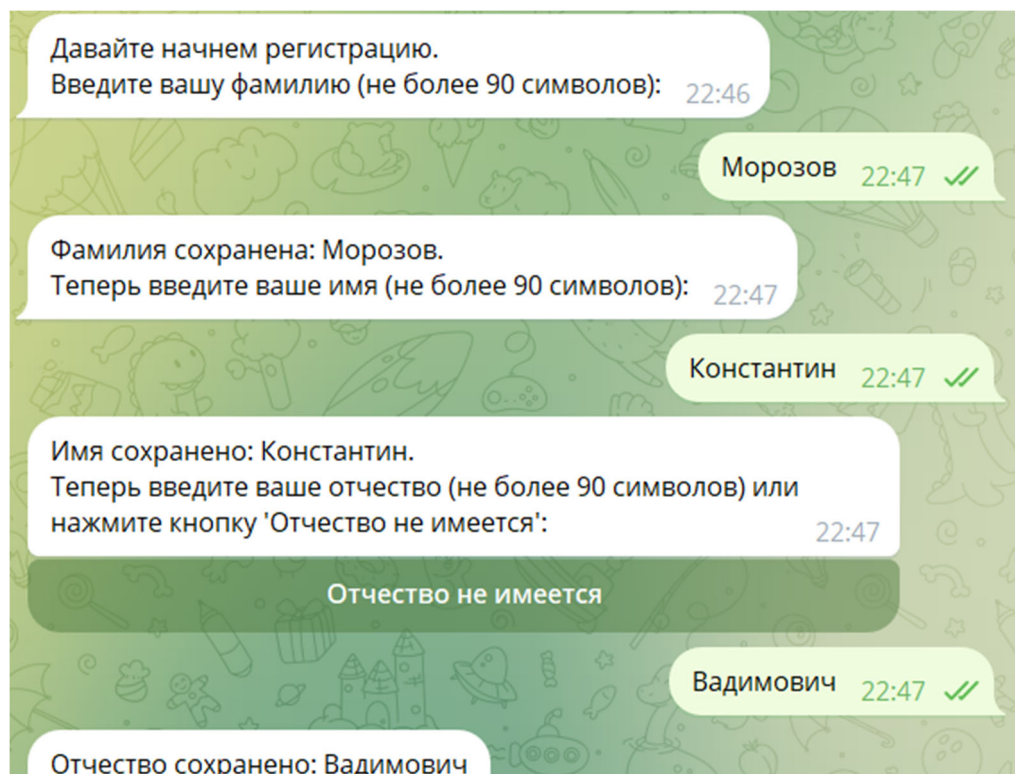


Рисунок 21 - Ввод ФИО

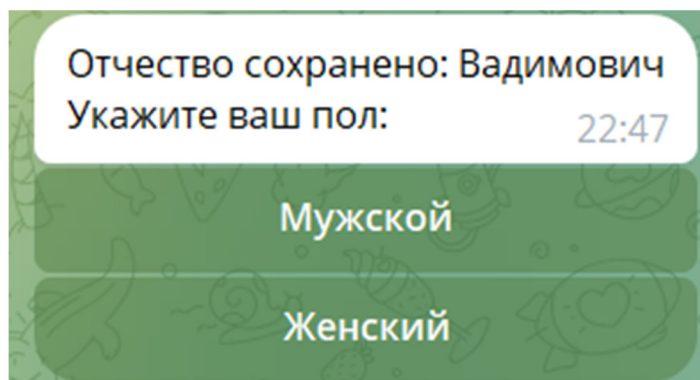


Рисунок 22 - Выбор пола

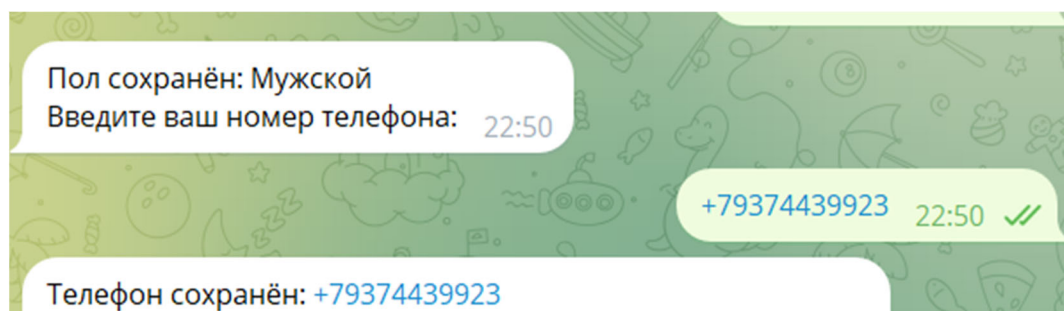


Рисунок 23 - Ввод телефона

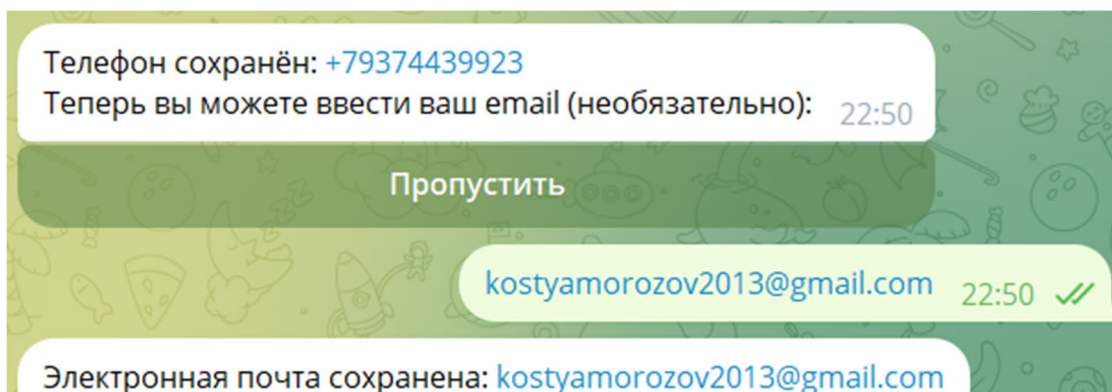


Рисунок 24 - Ввод электронной почты

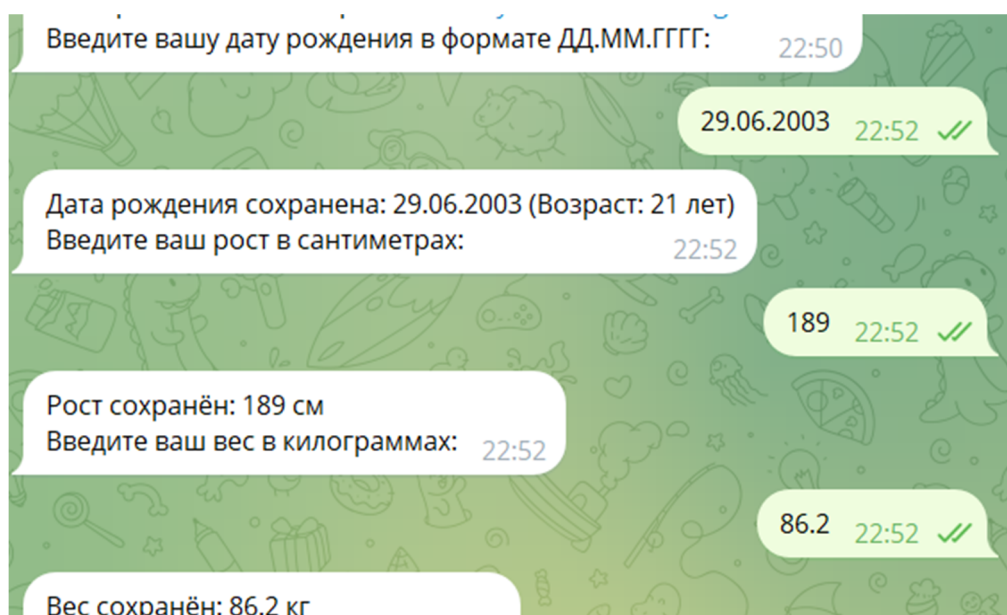


Рисунок 25 - Ввод биометрических параметров

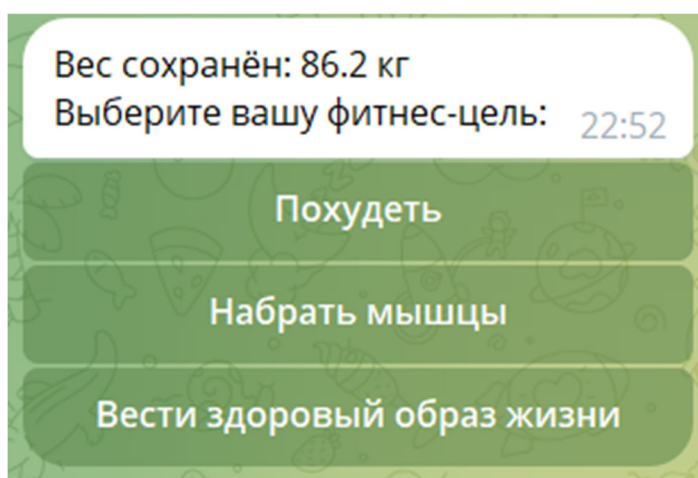


Рисунок 26 - Выбор фитнес-цели



Рисунок 27 - Выбор уведомлений

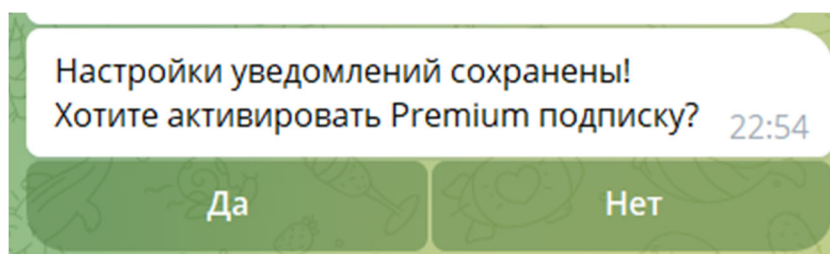


Рисунок 28 - Выбор между базовой и премиум подпиской

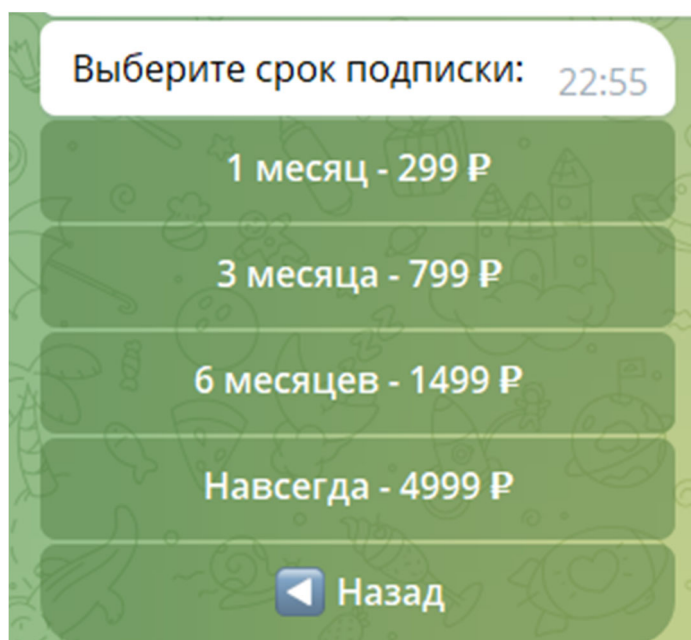


Рисунок 29 - Выбор срок премиума

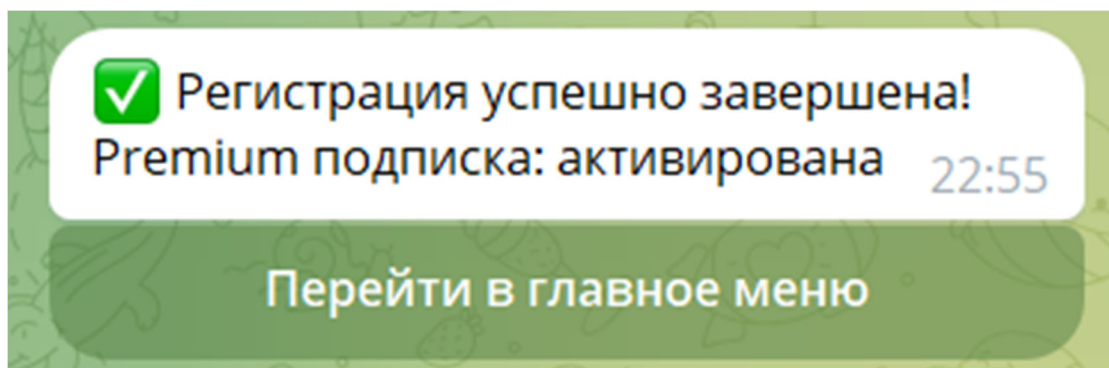


Рисунок 30 - Сообщение об окончании регистрации

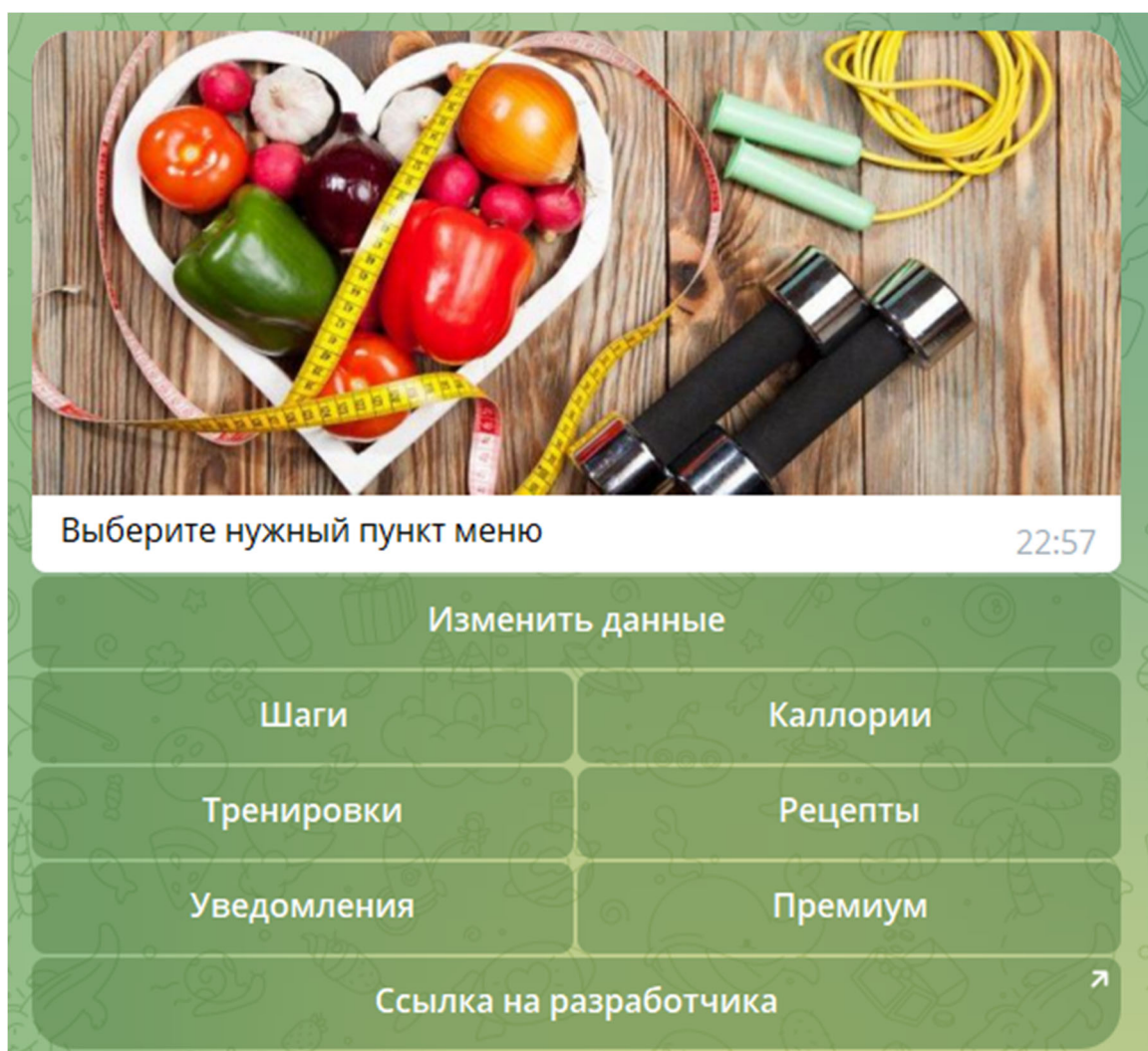


Рисунок 31 - Главное меню

1.9.3.3 Управление пользовательскими данными

По нажатию на «Изменить данные» появляется окно с данными пользователя (рисунок 32).

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.38.001 ПЗ

Лист

43

Ваши текущие данные:
 ФИО: Морозов Константин Вадимович
 Пол: Мужской
 Телефон: +79374439923
 Email: kostyamorozov2013@gmail.com
 Дата рождения: 2003-06-29
 Рост: 189 см
 Вес: 86.20 кг
 Фитнес-цель: Похудеть
 Выберите, что хотите изменить: 23:08

Фамилия Имя Отчество
 Пол Телефон Email
 Дата рождения
 Рост Вес
 Фитнес-цель
 Назад

Рисунок 32 - Окно с данными пользователя

По нажатию на любой из кнопок пользователь перейдёт на окно с изменением любой данной (например, изменении фамилии на рисунке 33). В данном окне также можно вернуться как изменениям данных, так и в главное меню. Если ввести новую фамилию, то появится сообщение об обновлении данных. При переходе в меню изменений данных можно увидеть, что фамилия поменялась (рисунок 34).

Текущая фамилия: Морозов
 Введите новую фамилию (только буквы, не более 90 символов): 23:08

Перейти в меню изменения данных
 Перейти в главное меню

Иванов 23:16 ✓

Данные успешно обновлены 23:16
 Перейти в меню изменения данных
 Перейти в главное меню

Рисунок 33 - Изменение фамилии

Ваши текущие данные:
ФИО: Иванов Константин Вадимович
Пол: Мужской
Телефон: +79374439923
Email: kostyamorozov2013@gmail.com
Дата рождения: 2003-06-29
Рост: 189 см
Вес: 86.20 кг
Фитнес-цель: Похудеть
Выберите, что хотите изменить:

23:19

Фамилия

Имя

Отчество

Пол

Телефон

Email

Дата рождения

Рост

Вес

Фитнес-цель

Назад

Рисунок 34 - Окно с данными пользователя после изменения фамилии

Так работают все остальные кнопки: по нажатию на любую из кнопок, кроме последней, можно изменить один из параметров, которые пользователь вводил при регистрации.

1.9.3.4 Трекинг активности

В меню есть 2 кнопки, связанные трекингом активности – это «Шаги» и «Калории» (см рисунок 31).

По нажатию на первую пользователь увидит окно статистики по шагам (рисунок 35, 38). Здесь можно как добавить шаги (рисунок 36), так и изменить цель (рисунок 37).

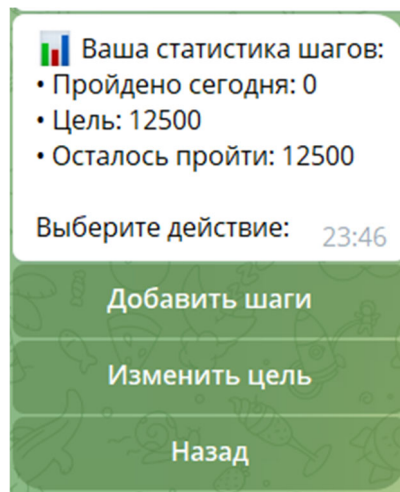


Рисунок 35 - Окно статистики шагов

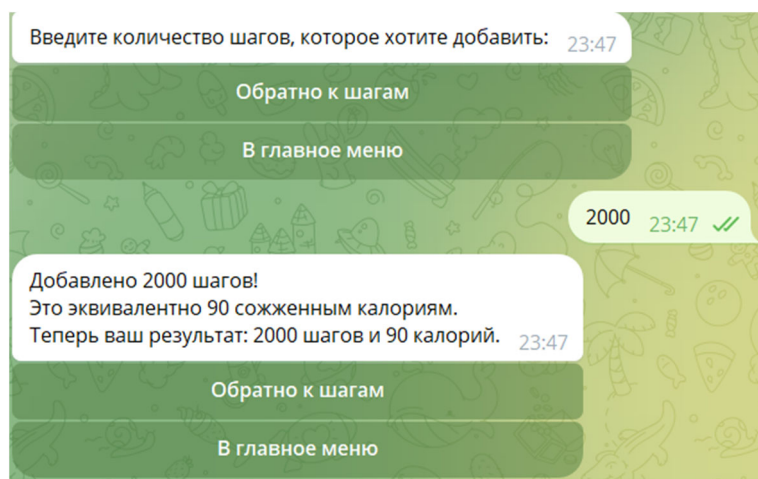


Рисунок 36 - Обновление кол-ва шагов

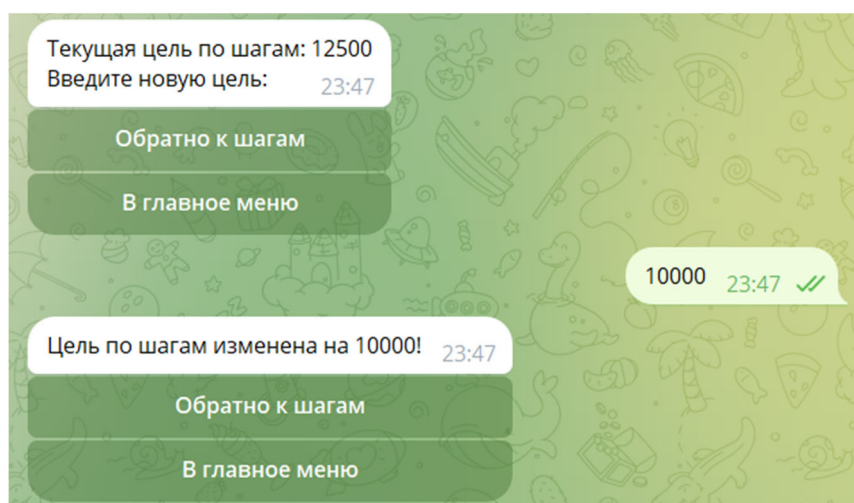


Рисунок 37 - Обновление цели шагов

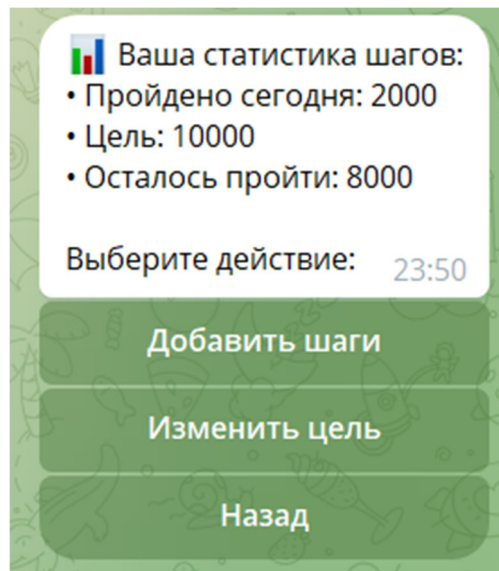


Рисунок 38 - Результат обновления данных

По нажатию на вторую кнопку пользователь увидит окно статистики по калориям (рисунок 39, 44). Здесь можно как добавить калории (рисунок 40, 41), так и изменить цель (рисунки 42, 43).

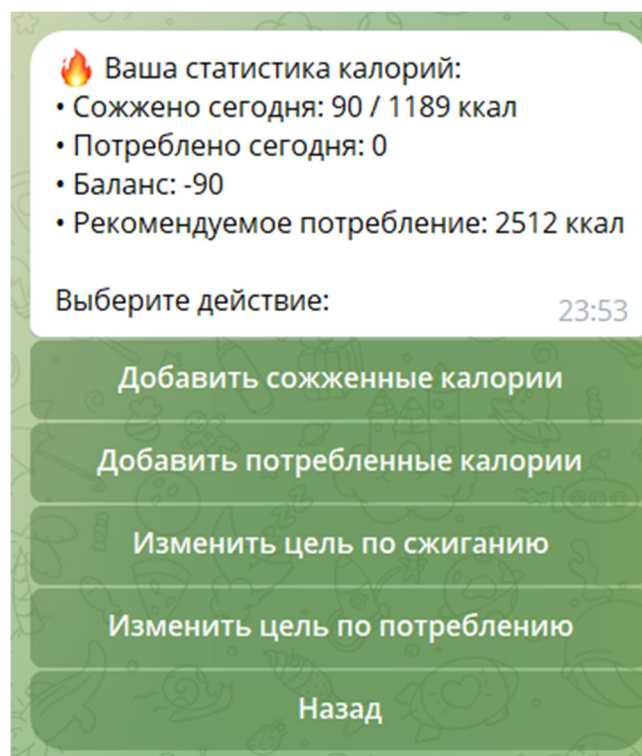


Рисунок 39 - Окно статистики калорий (так как мы добавили шаги, можно увидеть, что шаги конвертировались в сожжённые калории)

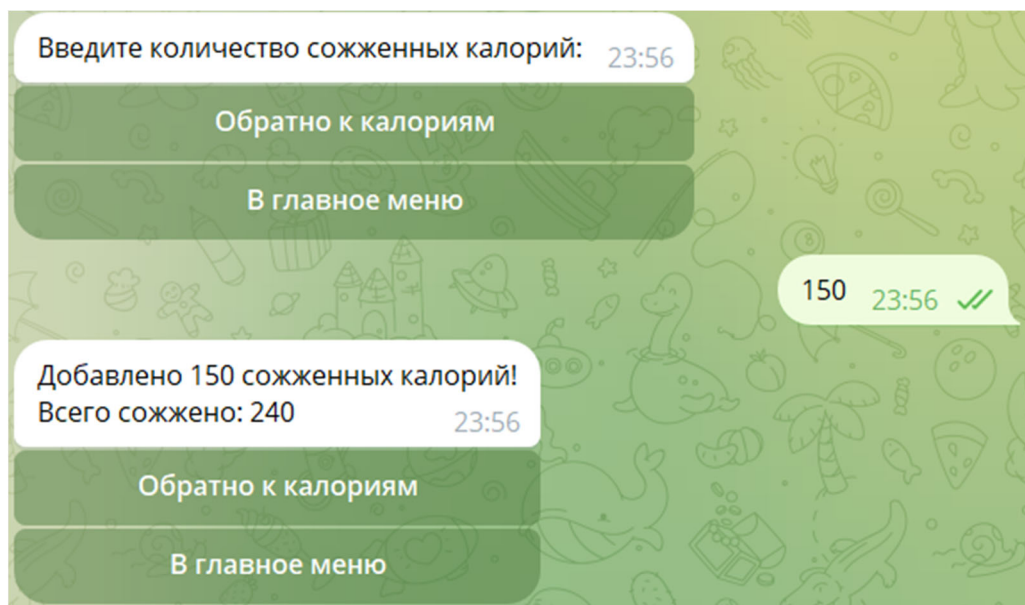


Рисунок 40 - Ввод сожжённых калорий

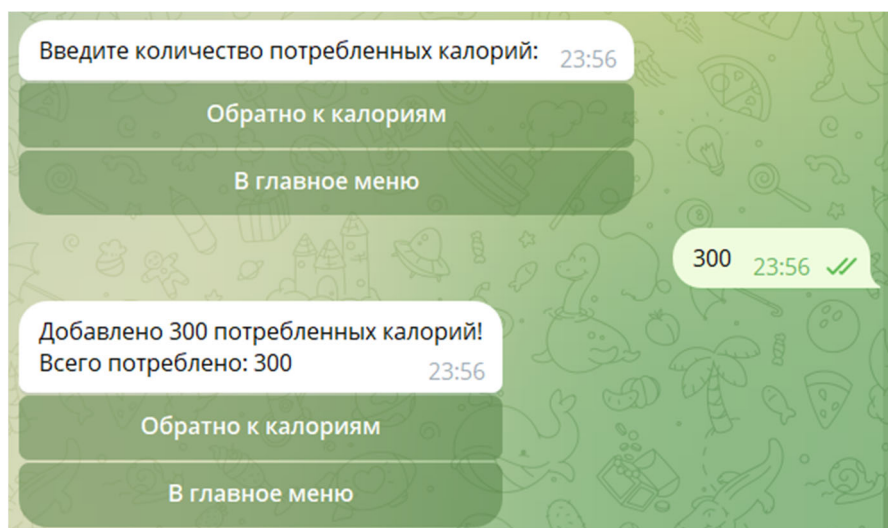


Рисунок 41 - Ввод потреблённых калорий

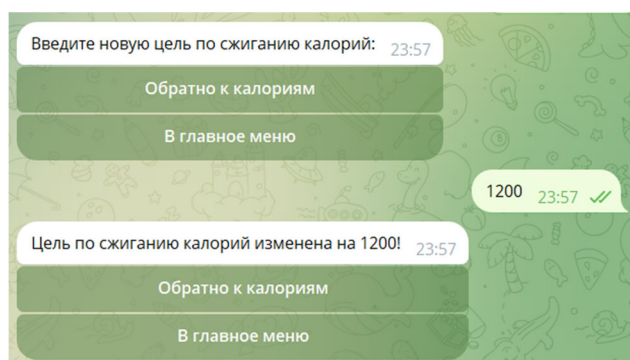


Рисунок 42 - Ввод цели на сожжённые калории

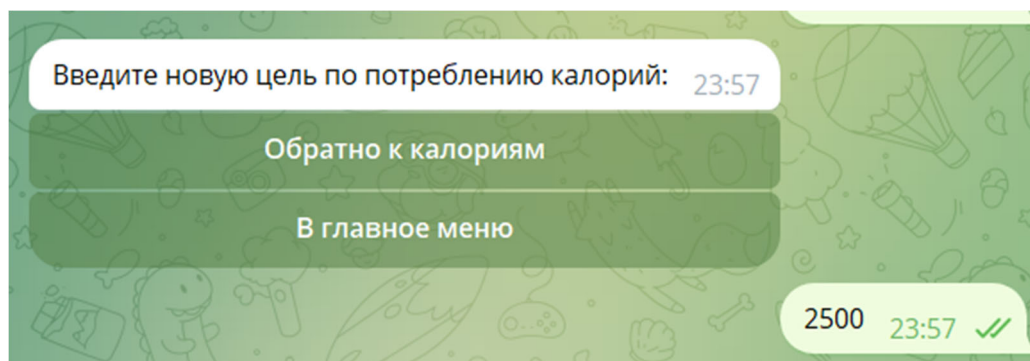


Рисунок 43 - Ввод цели на потреблённые калории

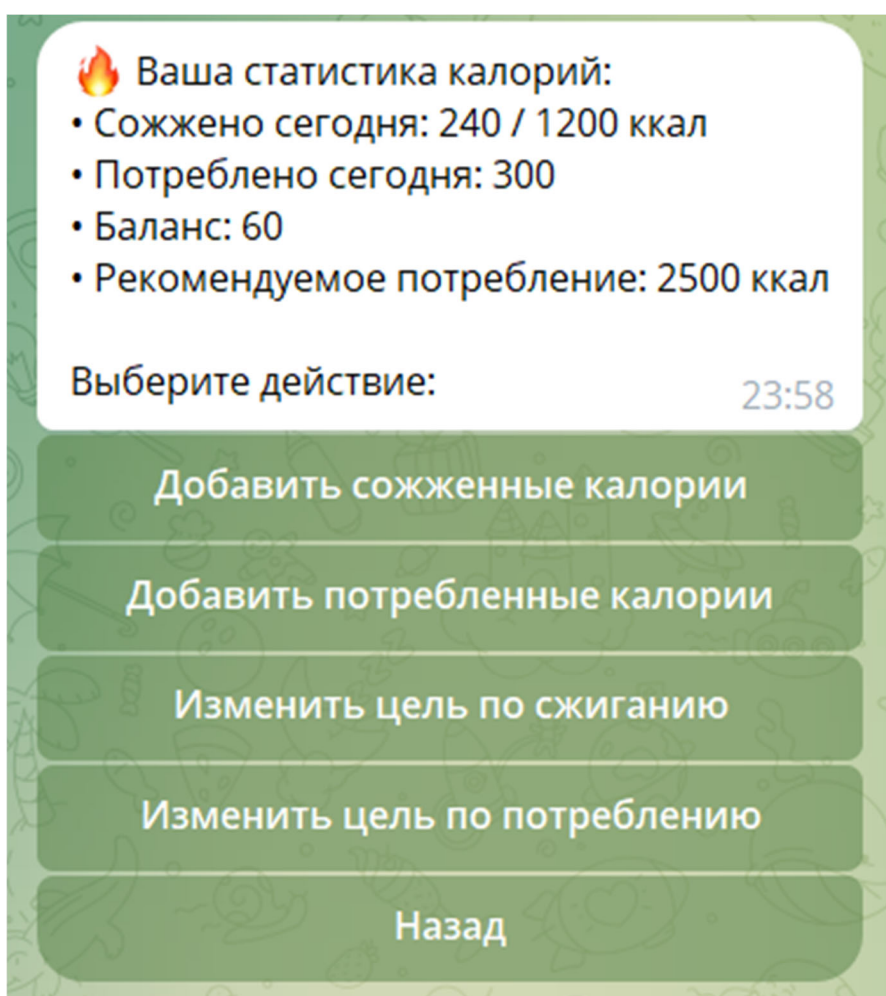


Рисунок 44 - Окно статистики калорий после ввода новых данных

1.9.3.5 Управление тренировками (премиум-функция)

Данная функция доступна только по подписке. Если пользователь имеет только базовый доступ, ему будет предложено приобретение премиум-подписки (рисунок 45).

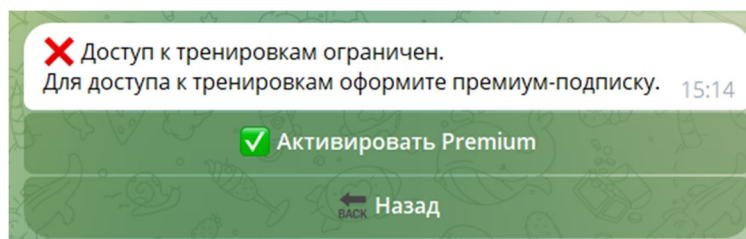


Рисунок 45 - Окно ограничения доступа

По нажатию на кнопку «Тренировки» пользователь перейдёт в окно, которое изображено на рисунке 46. При выборе любой из первых трёх кнопок он перейдёт на список тренировок (рисунок 47). В зависимости от сложности будут свои тренировки с фиксированным значением сжигания калорий:

- тренировки лёгкого уровня – 400 калорий;
- тренировки нормального уровня – 450 калорий;
- тренировки тяжёлого уровня – 500 калорий;
- тренировка «Растяжка» 400 калорий (фиксированная для каждой сложности).

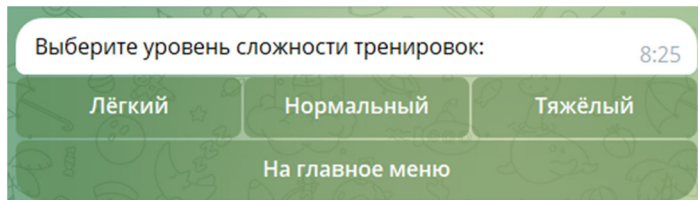


Рисунок 46 - Окно тренировок

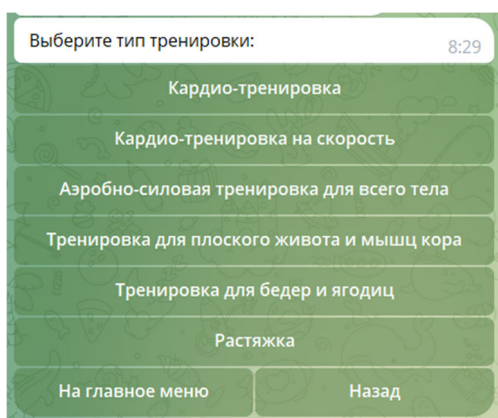


Рисунок 47 - Окно выбора тренировки

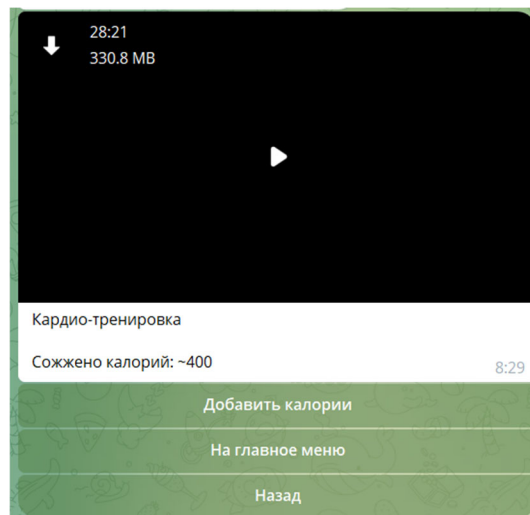


Рисунок 48 - Окно с видео-тренировкой

После того, как пользователь просмотрит тренировку он сможет по нажатию одной кнопки добавить значения сожжённых калорий за тренировку (рисунки 49, 50).

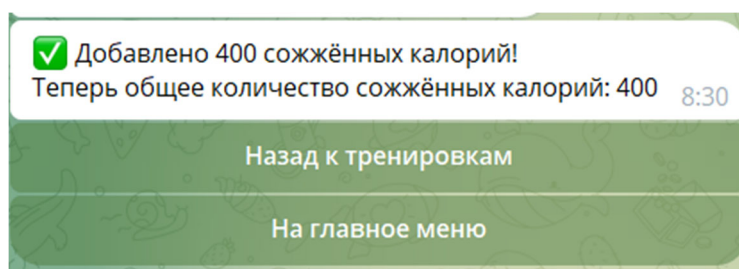


Рисунок 49 - Результат добавления калорий после тренировки

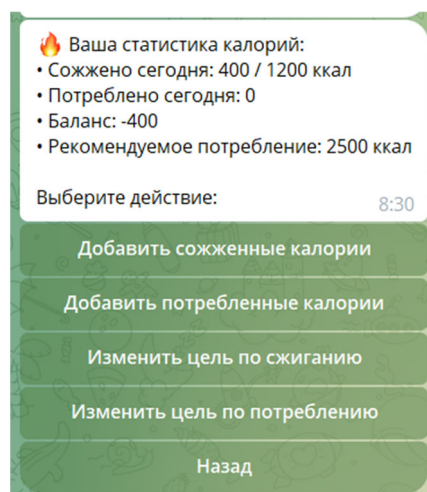


Рисунок 50 - Окно статистики калорий после тренировки

1.9.3.6 Работа с рецептами (премиум-функция)

Как и управление тренировками, данная функция доступна только по подписке. Если пользователь имеет только базовый доступ, ему будет предложено приобретение премиум-подписки (см. рисунок 45).

По нажатию на кнопку «Рецепты» пользователь перейдёт на окно, которое показано на рисунке 51. Первые 4 кнопки за тип питания: завтрак, обед, ужин и перекус. По нажатию на одну из них пользователю выдаст случайный рецепт из предварительно сделанной книги рецептов (рисунок 52). При выдаче рецепта можно добавить значение в потребление калорий (рисунок 53, 54). Также можно посмотреть книгу рецептов (рисунок 55). В зависимости какая у пользователя фитнес-цель будет свой список рецептов (рисунок 56).



Рисунок 51 - Окно меню рецептов

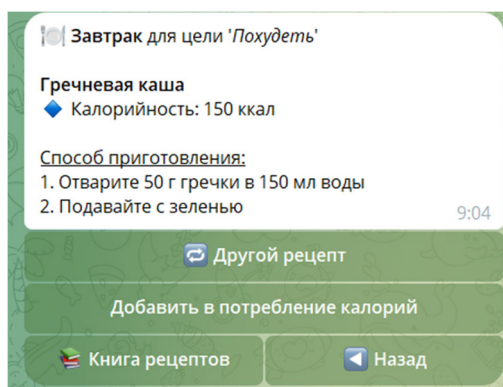


Рисунок 52 - окно при нажатии на «Завтрак»

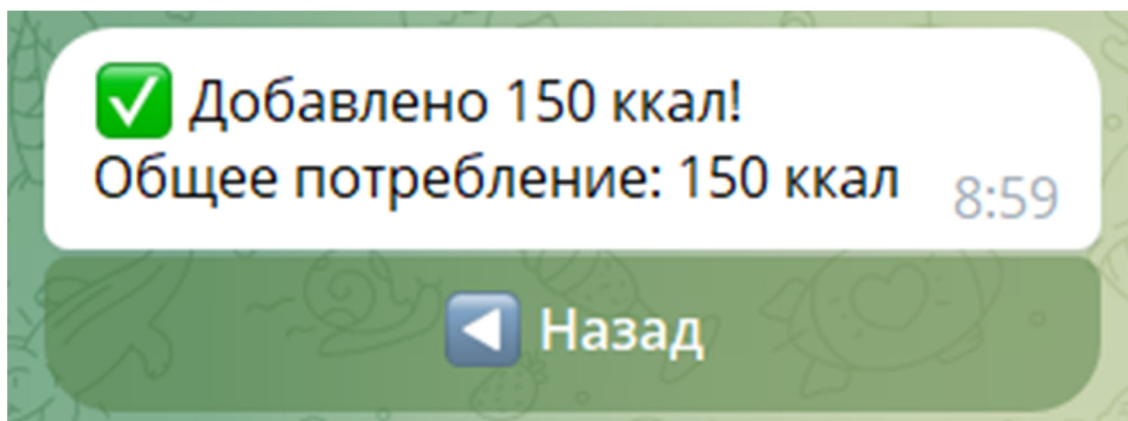


Рисунок 53 - Окно после нажатия «Добавить в потребление калорий»

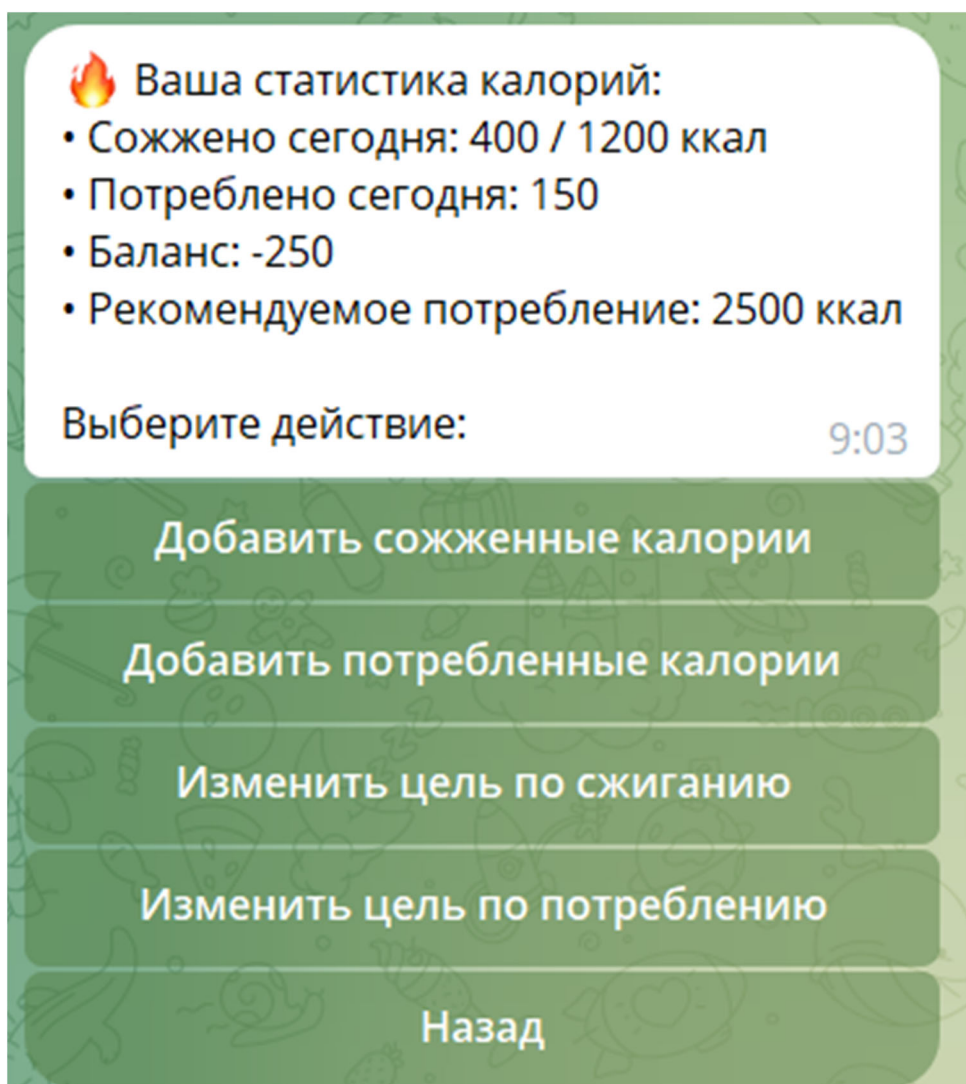


Рисунок 54 - Окно статистики калорий после добавления потреблённых калорий
через рецепты

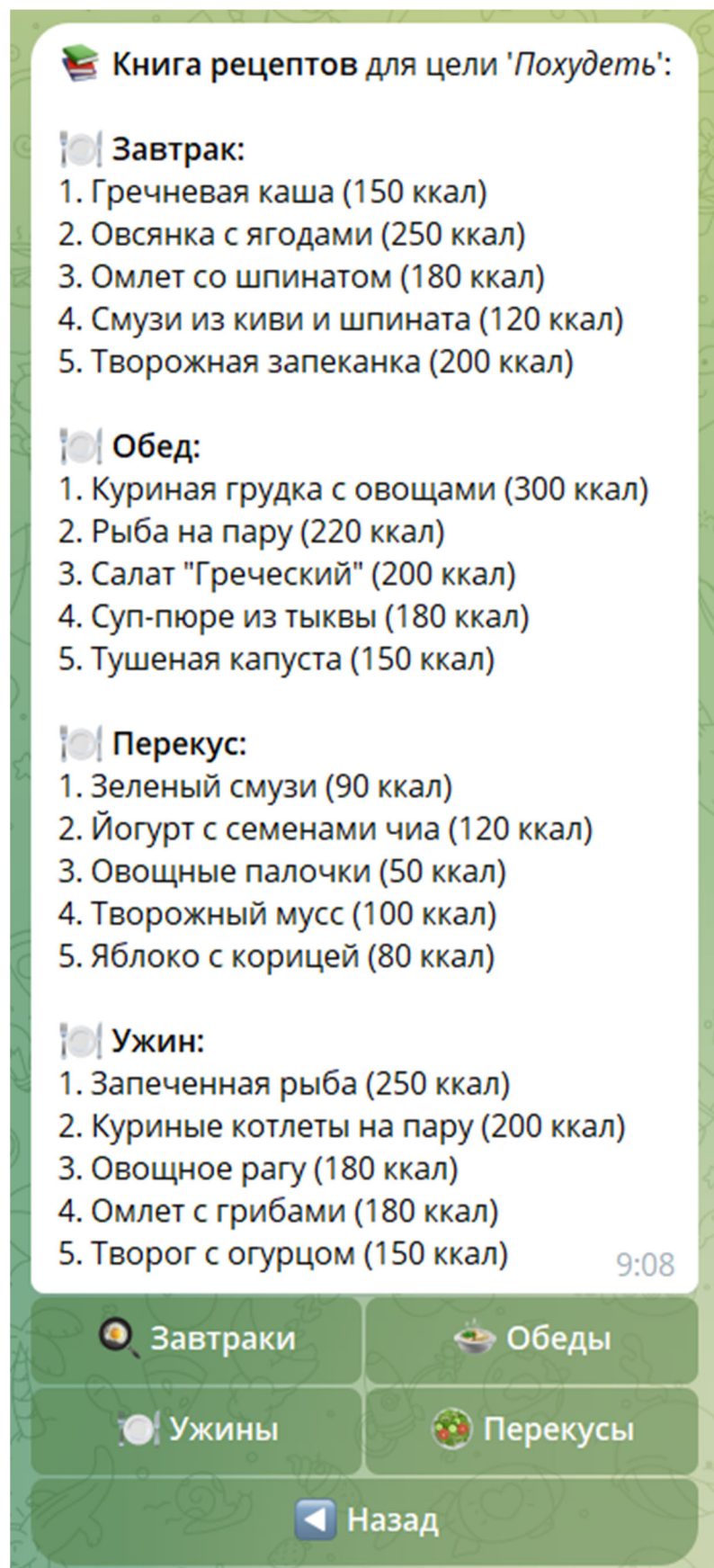


Рисунок 55 - Книги рецептов для похудения

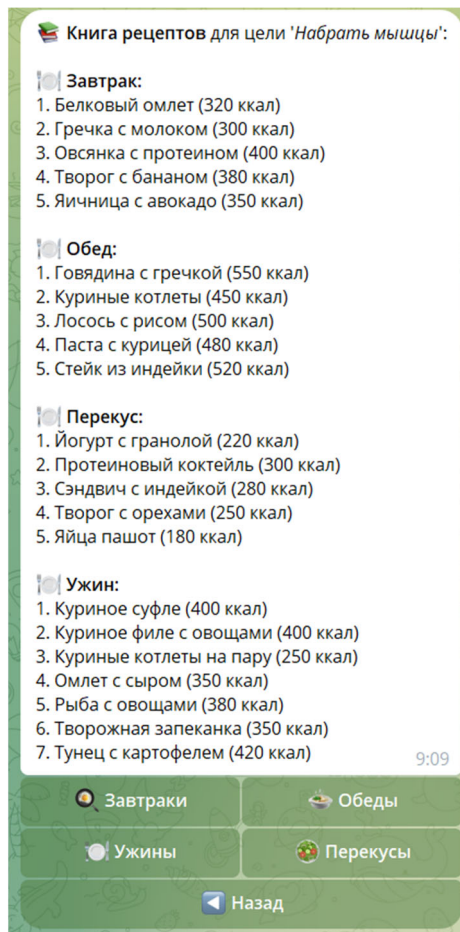


Рисунок 56 - Книги рецептов для набора мышечной массы

Если пользователю не будет устраивать блюда из рецептов, он может добавить свой рецепт блюда по одноимённой кнопке. Для рецепта нужны следующие данные: название, количество калорий, приём пищи, фитнес-цель и описание приготовления (рисунки 57-59). После всех введённых данных пользователь увидит окно с уже записанным рецептом в книгу (рисунки 60, 61).

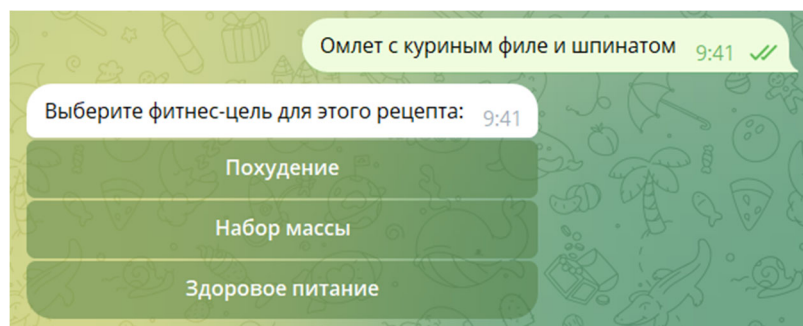


Рисунок 57 - Ввод фитнес-цели

Рисунок 58 - Ввод типа приёма пищи

Рисунок 59 - Ввод количества и описание приготовления блюда

Рисунок 60 - Результат добавления рецепта

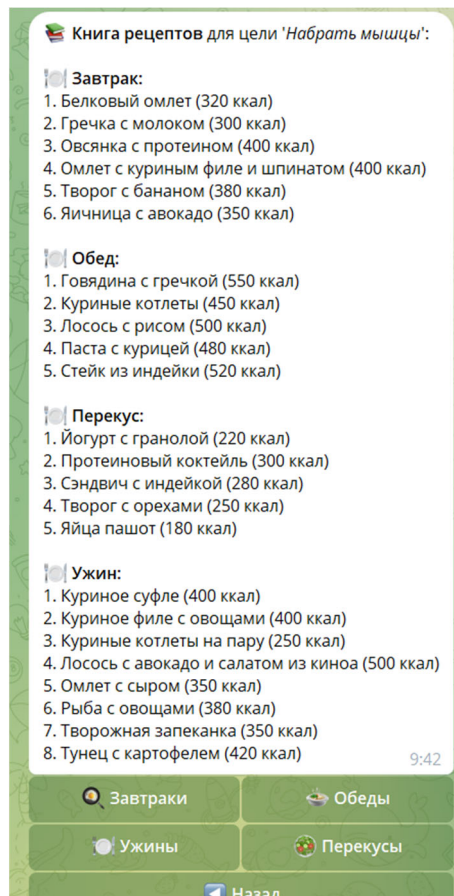


Рисунок 61 - Рецепт автоматически добавился в книгу рецептов

1.9.3.7 Система уведомлений

При нажатии на кнопку «Уведомления», пользователь перейдёт на окно настройки уведомлений (рисунок 62).

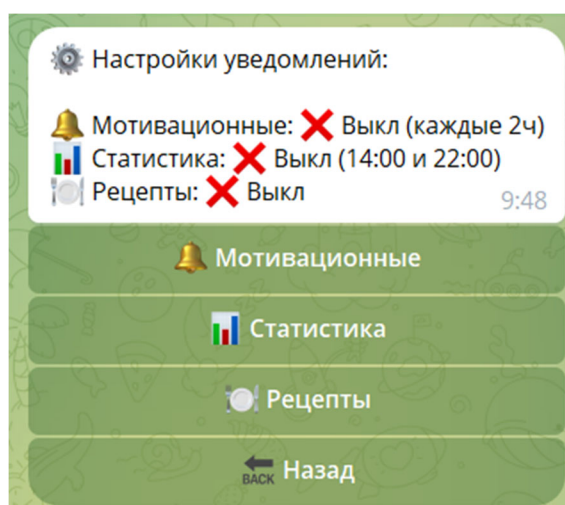


Рисунок 62 - Окно настроек уведомлений

Есть 3 типа уведомлений:

- мотивационные уведомления;
- уведомления статистики;
- уведомления рецептов.

Каждую из них можно включить или выключить по нажатию одной кнопки (рисунок 63, 64).

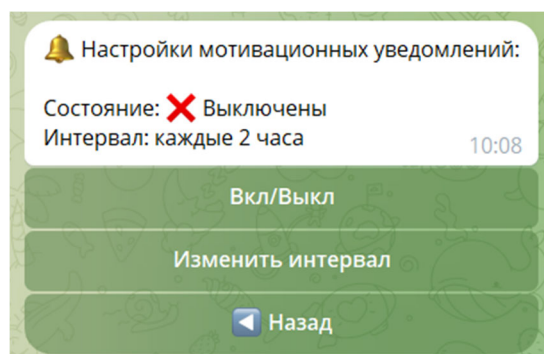


Рисунок 63 - - Состояние уведомления до нажатия на «Вкл/ Выкл»

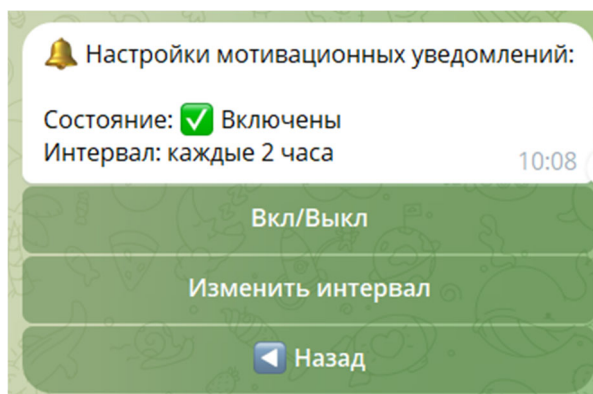


Рисунок 64 - Состояние уведомления после нажатия на «Вкл/ Выкл»

Пример работы уведомлений показаны на рисунке 65.

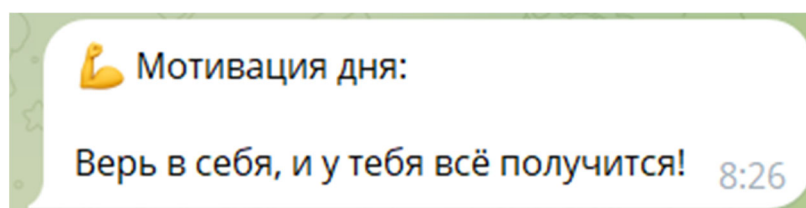


Рисунок 65 - Отправка мотивационного уведомления.

1.9.3.8 Аналитика

В системе есть автоматический расчёт количества шагов, Сожженных и потреблённых калорий в зависимости от возраста, пола, веса, роста и фитнес-цели.

Основные формулы расчета:

– BMR (Basal Metabolic Rate) - базовый метаболизм:

1) для мужчин: $BMR = 10 * \text{вес(кг)} + 6.25 * \text{рост(см)} - 5 * \text{возраст} + 5$;

2) для женщин: $BMR = 10 * \text{вес(кг)} + 6.25 * \text{рост(см)} - 5 * \text{возраст} - 161$.

– рекомендуемое суточное потребление калорий:

1) поддержание веса: $BMR * 1.55$ (средняя активность);

2) похудение: $(BMR * 1.55) - 500$ (дефицит 500 ккал);

3) набор массы: $(BMR * 1.55) + 300$ (профицит 300 ккал).

– цель по шагам. базовое значение: 8000 шагов.

– корректировки:

1) возраст <30 : +2000 шагов;

2) возраст >50 : -2000 шагов;

3) цель "похудеть": +2000 шагов;

4) цель "набрать мышцы": +1000 шагов.

– цель по сжиганию калорий:

1) похудение: $500 + (\text{вес} * 7)$;

2) набор массы: $300 + (\text{вес} * 5)$;

3) поддержание: $400 + (\text{вес} * 6)$.

Пример значений показаны на таблице 5.

Таблица 5 - примерные показатели шагов, калорий при различных биометрических данных

Параметры	Пример 1 (Похудение)	Пример 2 (Набор массы)	Пример 3 (Поддержание)
Пол	Мужчина	Женщина	Мужчина
Возраст	35	25	45
Рост (см)	180	165	175
Вес (кг)	85	55	75

Параметры	Пример 1 (Похудение)	Пример 2 (Набор массы)	Пример 3 (Поддержание)
Фитнес-цель	Похудение	Набор массы	Поддержание здорового образа жизни

Таблица 6 - Расчёты по данным из таблицы 5

Расчеты:	Пример 1 (Похудение)	Пример 2 (Набор массы)	Пример 3 (Поддержание)
BMR	1805 ккал	1289 ккал	1610 ккал
Рекомендуемое потребление	2300 ккал	2000 ккал	2500 ккал
Цель по шагам	10000 шагов	9000 шагов	8000 шагов
Цель по сжиганию калорий	1095 ккал	575 ккал	850 ккал

1.9.3.9 Премиум-функции

При нажатии на кнопку «Премиум» пользователь перейдёт в окно приобретения премиум доступа. Если у пользователя есть подписка, ему будет предложено продление премиума (рисунок 66, 67). В противном случае будет предлагаться приобретение подписки (рисунок 68, 69).

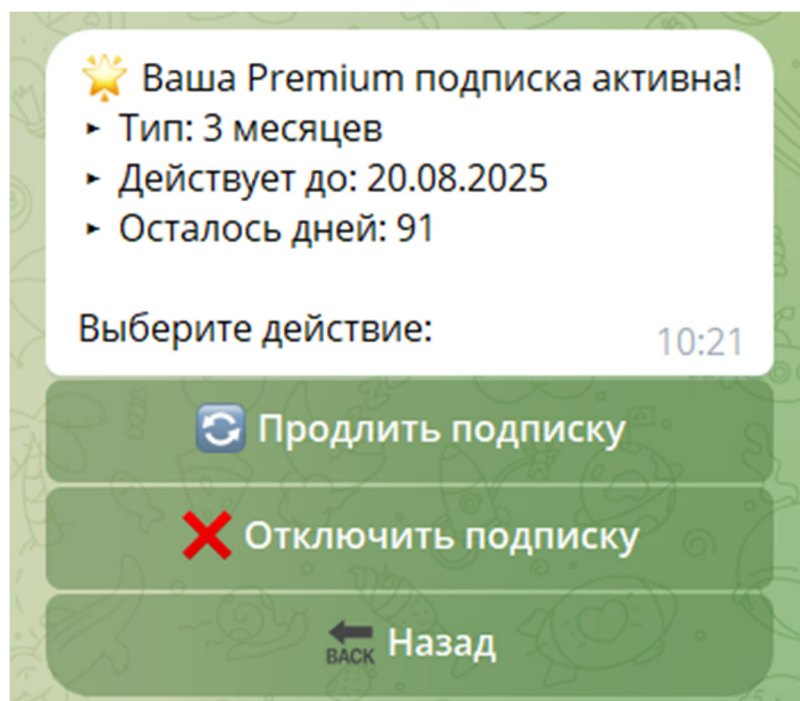


Рисунок 66 - Окно премиум-статуса при временной подписке

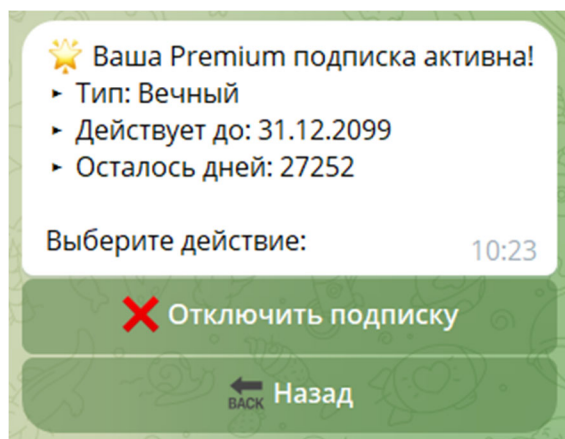


Рисунок 67 - Окно премиум-статуса при вечной подписке

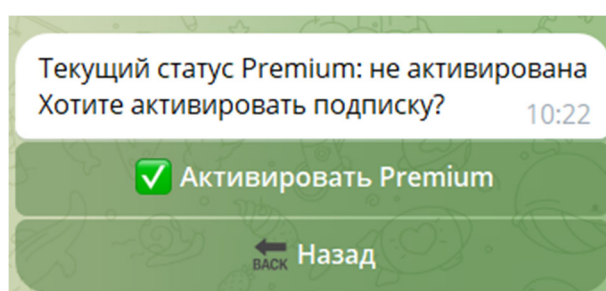


Рисунок 68 - Окна предложения и премиум-статуса

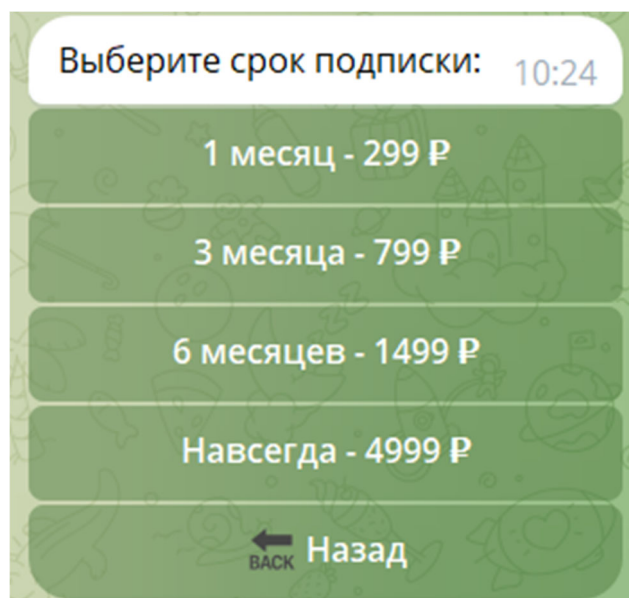


Рисунок 69 - Окна приобретения премиум-статуса

После отключения или приобретения премиума, бот будет вас уведомлять о вашем действии (рисунки 70, 71).

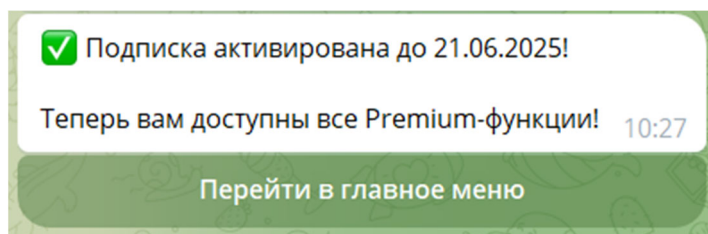


Рисунок 70 - Окна уведомления о приобретении подписки.

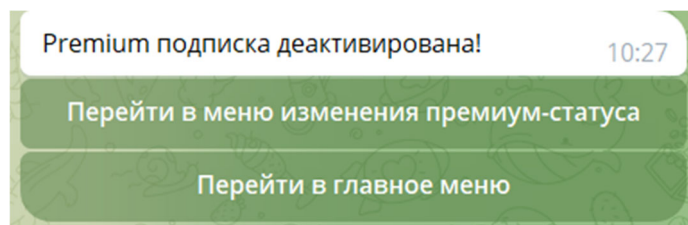


Рисунок 71 - Окна уведомления о деактивации подписки.

1.9.3.10 Технические функции

Бот по окончанию дня присылает статистику, которая сохраняется в истории пользователя (рисунок 72).

Также бот проверяет, не истекла ли премиум-подписка у пользователя.

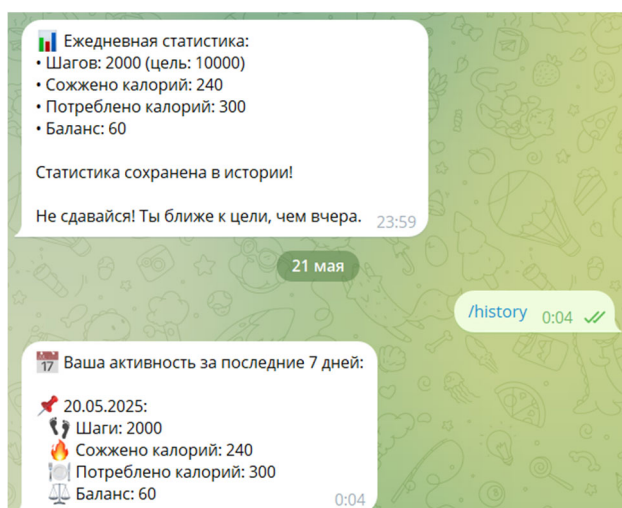


Рисунок 72 - Окна уведомления о сбросе статистики и сохранении их в истории пользователя

1.9.4 Описание программы

Диаграмма разворачивания, приведенная на рисунке 71, используется для

визуализации физического размещения компонентов системы и их взаимодействия на уровне аппаратного и программного обеспечения. Она позволяет представить архитектуру системы, указывая, на каких устройствах или серверах расположены компоненты приложения и как они связаны друг с другом.



Рисунок 73 - UML-диаграмма развёртывания

Файл `.gitignore` — это файл конфигурации, используемый в системе контроля версий *Git* для указания файлов и папок, которые должны быть проигнорированы и не отслеживаемы *Git* при выполнении операций коммита и других действий.

Файл `.env` — это конфигурационный файл, используемый для хранения переменных окружения, таких как API-ключи, пароли, настройки базы данных и другие чувствительные или зависящие от среды параметры. Он позволяет отделить конфигурацию от кода, что упрощает управление настройками в разных средах (разработка, тестирование, продакшн) и повышает безопасность, так как `.env` обычно исключается из системы контроля версий (например, *Git*) с помощью `.gitignore`.

Файл `photo.py` — файл, в котором хранятся ID всех фотографий, который можно использовать в боте.

Файл `video.py` — файл, похожий на `photo.py`, но вместо фотографий хранит в себе ID различных видео (в данном случае, видео с тренировками).

Файл `keyboard.py` — файл, хранящий все настройки, касаемо клавиатур для всех сообщений.

Файл `main.py` — файл с основной программой. Он хранит как функцию

запуска (main), так и все оставшиеся функции.

Визуальный интерфейс телеграм-бота в Python можно создавать с использованием Telegram Bot API и различных библиотек, таких как python-telegram-bot. Ниже перечислены некоторые виды визуальных элементов, которые можно использовать в интерфейсе телеграм-бота:

- текстовые сообщения. Бот может отправлять текстовые сообщения пользователю. Это может быть простой текстовый ответ или инструкции для пользователя;

- кнопки Inline. Кнопки могут использоваться для предоставления пользователю опций выбора или для выполнения определенных действий. Inline кнопки можно размещать непосредственно в тексте сообщения;

- Reply кнопки. Эти кнопки показываются внизу экрана в виде списка;

- инлайн-клавиатура. Инлайн-клавиатура — это набор кнопок, которые отображаются внизу экрана и позволяют пользователю выбирать опции или выполнять действия в контексте текущего сообщения. Таким образом сделан календарь, а также выбор категории для тестирования;

- всплывающие сообщения. Всплывающие сообщения обычно реализуются с использованием Inline-режима или Inline-уведомлений. Всплывающие сообщения могут содержать дополнительную информацию или предложения для пользователя и отображаться поверх текущего чата или сообщения;

- вложения (фото, видео, аудио, файлы). Бот может отправлять пользователю вложения, такие как фотографии, видео, аудио или файлы.

После того как пользователь отправляет какой-либо текст, команду или нажатие на кнопку, действие фильтруется и обрабатывается в определенных функциях. Ниже перечислены разделы, которые отвечают за определённый функционал данного бота.

1.9.4.1 Регистрация пользователя

Функция start_registration инициирует процесс регистрации, удаляет предыдущее сообщение и запрашивает у пользователя ввод фамилии,

устанавливая соответствующее состояние FSM (Finite State Machine) для обработки следующего шага.

Функция `process_surname` принимает и валидирует введенную фамилию (проверяет длину и допустимые символы), форматирует её (первая буква заглавная) и сохраняет в состоянии, после чего запрашивает имя пользователя.

Функция `process_name` обрабатывает введенное имя, проверяет его корректность аналогично фамилии, сохраняет отформатированное значение и предлагает ввести отчество или пропустить этот шаг.

Функция `process_patronymic` принимает отчество пользователя, проверяет его валидность и сохраняет, либо обрабатывает команду пропуска этого шага, после чего переходит к запросу пола.

Функция `skip_patronymic` обрабатывает ситуацию, когда пользователь решил не указывать отчество, явно устанавливает значение `None` для этого поля и переходит к следующему шагу регистрации.

Функция `process_gender` получает выбранный пользователем пол (мужской/женский) через `callback`, сохраняет значение и запрашивает номер телефона.

Функция `process_phone` принимает и проверяет номер телефона на соответствие форматам (начинается с + или 8, содержит нужное количество цифр), сохраняет его и запрашивает email.

Функция `process_email` валидирует введенный email (проверяет наличие @ и домена), проверяет его уникальность в системе и сохраняет, либо обрабатывает команду пропуска.

Функция `skip_email` обрабатывает ситуацию, когда пользователь решил не указывать email, сохраняет `None` для этого поля и переходит к следующему шагу - вводу даты рождения.

Функция `process_birthday` проверяет корректность введенной даты рождения (формат ДД.ММ.ГГГГ), валидирует возраст (18-100 лет) и сохраняет дату, после чего запрашивает рост.

Функция process_height принимает и проверяет значение роста (должно быть числом от 100 до 250 см), сохраняет его и запрашивает вес пользователя.

Функция process_weight обрабатывает введенный вес, проверяет что это число в диапазоне 30-200 кг, сохраняет значение и предлагает выбрать фитнес-цель.

Функция process_fitness_goal обрабатывает выбор цели (похудение, набор массы, поддержание формы) через callback, сохраняет значение и переходит к настройке уведомлений.

Функция process_notification_choice настраивает параметры уведомлений (мотивационные, статистика, рецепты) согласно выбору пользователя и предлагает оформить премиум-подписку.

Функция process_premium_choice обрабатывает решение пользователя о подключении премиум-подписки, либо сразу завершает регистрацию если выбран отказ.

Функция complete_registration финализирует процесс регистрации: сохраняет все собранные данные в БД, создаёт таблицу истории пользователя, инициализирует настройки уведомлений и статистики, отправляет приветственное сообщение.

Функция premium_choice_yes_registration обрабатывает согласие на премиум-подписку при регистрации, предлагая выбрать срок действия подписки.

Функция process_premium_duration_registration - Сохраняет выбранный срок премиум-подписки (1/3/6/12 месяцев или вечная), активирует соответствующие функции и завершает регистрацию.

Функция premium_choice_no_registration завершает регистрацию без активации премиум-подписки, сохраняя все введенные данные и отключая премиум-функции.

1.9.4.2 Управление пользовательскими данными

Функция change_surname_start запрашивает у пользователя новую фамилию, отображая текущее значение, и переводит бота в состояние ожидания

ввода нового значения с возможностью отмены.

Функция `process_new_surname` принимает и проверяет новую фамилию (длина, допустимые символы), сохраняет её в БД и уведомляет пользователя об успешном изменении или ошибке.

Функция `change_name_start` запрашивает новое имя пользователя, показывая текущее, и активирует состояние для обработки нового значения.

Функция `process_new_name` обрабатывает введённое имя, проверяет его корректность и обновляет в профиле, либо сообщает об ошибке и просит повторить ввод.

Функция `change_patronymic_start` предлагает ввести новое отчество или удалить текущее, отображая текущее значение (если есть).

Функция `process_new_patronymic` принимает новое отчество, проверяет его валидность и сохраняет в профиле, либо обрабатывает команду пропуска.

Функция `skip_patronymic_change` удаляет отчество из профиля пользователя и подтверждает изменение, возвращая его в меню настроек.

Функция `change_phone_start` запрашивает новый номер телефона, отображая текущий, и переводит бота в режим ожидания ввода.

Функция `process_new_phone` проверяет новый номер на соответствие формату (+ или 8, нужное количество цифр), сохраняет его в БД или сообщает об ошибке.

Функция `change_email_start` запрашивает новый email, показывая текущий, и предлагает либо ввести новое значение, либо пропустить шаг.

Функция `handle_skip_or_cancel` обрабатывает отмену ввода email (пропуск), сохраняя NULL в БД или оставляя текущее значение без изменений.

Функция `process_new_email` проверяет новый email на корректность и уникальность, обновляет его в профиле или просит исправить ошибки.

Функция `change_birthday_start` запрашивает новую дату рождения, отображая текущую, и переводит бота в режим ожидания ввода.

Функция `process_new_birthday` проверяет новый ввод (формат

ДД.ММ.ГГГГ, возраст 18–100 лет) и обновляет дату в профиле либо сообщает об ошибке.

Функция `change_height_start` запрашивает новый рост (в см), показывая текущее значение, и активирует состояние для обработки ввода.

Функция `process_new_height` принимает и проверяет рост (100–250 см), сохраняет его в БД и пересчитывает связанные параметры (цели по шагам/калориям).

Функция `change_weight_start` запрашивает новый вес (в кг), отображая текущий, и переводит бота в режим ожидания ввода.

Функция `process_new_weight` проверяет введенный вес (30–200 кг), обновляет его в профиле и автоматически корректирует рекомендации по калориям и активности.

Функция `change_fitness_goal_start` предлагает выбрать новую фитнес-цель (похудение, набор массы, поддержание формы), отображая текущую.

Функция `process_new_fitness_goal` обрабатывает выбор цели через кнопки, сохраняет его в БД и пересчитывает индивидуальные рекомендации (шаги, калории).

Функция `change_gender_start` запрашивает новый пол пользователя (мужской/женский), показывая текущий выбор, и активирует обработку callback.

Функция `process_new_gender` сохраняет выбранный пол в профиле и пересчитывает нормы калорий и активности с учётом гендерных различий.

Функция `cancel_handler` прерывает текущее изменение данных (например, ввод имени или телефона), сбрасывает состояние FSM и возвращает пользователя в главное меню.

1.9.4.3 Трекинг активности

Функция `add_steps_start` инициирует процесс добавления шагов, запрашивая у пользователя количество пройденных шагов для внесения в дневную статистику, и переводит бота в состояние ожидания ввода.

Функция `process_added_steps` принимает введенное количество шагов,

проверяет корректность данных (положительное число), добавляет их к текущему значению, автоматически рассчитывает и добавляет эквивалентные сожженные калории (примерно 450 ккал на 10000 шагов), после чего обновляет статистику пользователя.

Функция `add_calories_burned_start` начинает процесс учета сожженных калорий, запрашивая у пользователя ввод количества калорий, потраченных во время тренировки или активности.

Функция `process_added_calories` обрабатывает введенное количество сожженных калорий, проверяет валидность данных, суммирует с текущим значением и сохраняет обновленную статистику в базе данных.

Функция `add_calories_consumed_start` запускает процесс учета потребленных калорий, предлагая пользователю ввести количество калорий, полученных с пищей.

Функция `process_added_calories_consumed` принимает и проверяет введенные данные о потребленных калориях, добавляет их к текущему дневному значению и сохраняет обновленные данные, позволяя отслеживать энергетический баланс.

Функция `change_steps_goal_start` начинает процесс изменения дневной цели по шагам, отображая текущее значение цели и запрашивая новое.

Функция `process_steps_goal_change` обрабатывает новое значение дневной цели по шагам, проверяет его корректность (положительное число) и сохраняет в профиле пользователя, позволяя адаптировать программу под индивидуальные потребности.

Функция `change_calories_burned_goal_start` инициирует изменение цели по сжиганию калорий, показывая текущее значение и запрашивая новую цель.

Функция `process_changed_calories_goal` принимает и проверяет новое значение цели по сжиганию калорий, обновляет его в профиле пользователя и пересчитывает соответствующие рекомендации.

Функция `change_calories_consumed_goal_start` начинает процесс изменения

рекомендуемой нормы потребления калорий, отображая текущее значение и запрашивая новое.

Функция `process_changed_calories_consumed_goal` обрабатывает новое значение нормы потребления калорий, проверяет его валидность и сохраняет в профиле, что позволяет персонализировать диетические рекомендации.

1.9.4.4 Управление тренировками (премиум-функция)

Функция `training` основная функция меню тренировок, проверяет премиум-статус пользователя и предоставляет доступ к выбору уровня сложности тренировок (легкий, средний, сложный) для премиум-пользователей.

Функция `stretch` отправляет пользователю видео с комплексом упражнений на растяжку с описанием (~400 сожженных калорий) и кнопкой подтверждения выполнения тренировки.

Функция `easy` устанавливает состояние "легкий уровень сложности" и выводит меню с выбором типа легкой тренировки (кардио, аэробные и др.), доступное только премиум-пользователям.

Функция `medium` активирует режим среднего уровня сложности и показывает соответствующие варианты тренировок средней интенсивности (~450 ккал).

Функция `hard` включает режим сложных тренировок с высокоинтенсивными вариантами нагрузок (~500 ккал) для продвинутых пользователей.

Функция `easy_[x]` отправляет видео легкой тренировки с описанием (~400 ккал) и кнопкой возврата в меню.

Функция `medium_[x]` отправляет видео с тренировкой нормальной сложности (~450 ккал).

Функция `hard_[x]` отправляет видео с сложной тренировкой (~500 ккал)

Функция `add_training_calories` - автоматически добавляет калории (400-500 в зависимости от сложности) к сожженным после подтверждения выполнения тренировки.

Функция `add_stretch_calories` фиксирует 400 сожженных калорий после

выполнения комплекса растяжки и обновляет статистику пользователя.

1.9.4.5 Работа с рецептами (премиум-функция)

Функция `recipe_menu` показывает основное меню работы с рецептами, проверяющее премиум-статус пользователя и предоставляющее доступ к различным категориям рецептов (по типам приемов пищи) или полной книге рецептов, если подписка активна.

Функция `handle_recipe_request` обрабатывает запрос пользователя на получение рецепта, определяя тип блюда (завтрак, обед и т.д.), выбирая случайный рецепт из базы данных согласно фитнес-целям пользователя и выводя его с подробным описанием и кнопками действий.

Функция `show_another_recipe` показывает новый случайный рецепт из той же категории (типа приема пищи), что и предыдущий, обновляя информацию без возврата в меню, чтобы пользователь мог найти подходящий вариант.

Функция `show_recipe_book` отображает полную коллекцию рецептов, сгруппированных по типам приемов пищи, с краткой информацией о каждом блюде (название, калорийность) для удобного просмотра и выбора.

Функция `add_recipe_start` начинает процесс добавления пользовательского рецепта, запрашивая название блюда и последовательно собирая остальные параметры (тип, калорийность и т.д.) через цепочку вопросов.

Функция `process_recipe_name` принимает и сохраняет название нового рецепта, после чего запрашивает выбор фитнес-цели (похудение, поддержание веса и др.), для которой подходит блюдо.

Функция `process_recipe_goal` обрабатывает выбранную пользователем фитнес-цель для рецепта и переходит к запросу типа приема пищи (завтрак, обед, ужин или перекус).

Функция `process_recipe_meal_type` фиксирует тип приема пищи для нового рецепта и запрашивает информацию о калорийности блюда, необходимую для корректного учета в дневнике питания.

Функция `process_recipe_calories` принимает и проверяет введенную

калорийность рецепта (должна быть положительным числом), после чего запрашивает способ приготовления блюда.

Функция `process_recipe_preparation` завершает процесс добавления рецепта, принимая текст с описанием приготовления, сохраняя все данные в базу и выводя пользователю полную карточку созданного рецепта.

Функция `add_recipe_calories` позволяет быстро добавить калории из показанного рецепта в дневной рацион пользователя, автоматически суммируя значение с текущим потреблением и обновляя статистику.

1.9.4.6 Система уведомлений

Функция `notification_settings_menu` показывает основное меню настроек уведомлений, отображающее текущий статус (вкл/выкл) и интервалы для мотивационных сообщений, статистики и рецептов, с возможностью перехода к детальным настройкам каждого типа.

Функция `motivation_settings_menu` показывает текущие настройки мотивационных уведомлений (статус и интервал отправки) и предоставляет кнопки для их изменения или отключения.

Функция `stats_settings_menu` отображает параметры уведомлений со статистикой (включение/отключение и установленное время отправки), позволяя пользователю их настроить.

Функция `recipes_time_settings_menu`, которая показывает расписание уведомлений о рецептах для каждого приема пищи (завтрак, обед и т.д.) с возможностью изменения времени.

Функция `change_notification_setting` перенаправляет пользователя в соответствующий раздел настроек (мотивация, статистика или рецепты) в зависимости от выбранного типа уведомлений.

Функция `toggle_motivation` переключает статус мотивационных уведомлений (включение/выключение) и сразу применяет изменения в системе.

Функция `toggle_stats` активирует или деактивирует уведомления со статистикой активности, сохраняя новый статус в настройках пользователя.

Функция `toggle_recipes_notifications`, которая управляет рассылкой уведомлений о рецептах, с автоматической проверкой статуса подписки перед изменением настроек.

Функция `change_motivation_interval_start` инициирует процесс изменения интервала мотивационных уведомлений (1-24 часа), запрашивая у пользователя новое значение.

Функция `process_motivation_interval` принимает и проверяет введенный интервал для мотивационных сообщений, обновляя его в системе уведомлений.

Функция `change_stats_time_start` начинает процедуру изменения времени отправки статистических уведомлений, запрашивая новый временной параметр.

Функция `process_stats_time` обрабатывает введенное время для уведомлений со статистикой, проверяет формат (ЧЧ:ММ) и сохраняет в настройках (с автоматическим расчетом второго времени отправки через 8 часов).

Функция `change_recipes_time_start` запускает процесс настройки времени уведомлений о рецептах, предлагая выбрать тип приема пищи для изменения.

Функция `select_recipe_time_to_change` позволяет выбрать конкретный прием пищи (завтрак/обед и т.д.), для которого пользователь хочет изменить время уведомления.

Функция `process_recipe_time_change` принимает и сохраняет новое время для выбранного типа уведомлений о рецептах, автоматически включая рассылку если она была отключена.

Функция `set_notification_mode` устанавливает комплексный режим уведомлений ("только мотивация", "все включено" и др.) по выбору пользователя, применяя сразу группу настроек.

Функция `send_motivational_notification` отправляет индивидуальное мотивационное сообщение (случайную цитату) пользователю согласно установленному интервалу, удаляя предыдущее аналогичное уведомление.

Функция `send_stats_notification` формирует и отправляет персонализированное уведомление с текущей статистикой активности (шаги,

калории), включая прогресс к целям.

Функция `send_recipe_notification` премиум-функция, отправляющая уведомление с рецептом для определенного приема пищи в установленное время, с возможностью быстрого добавления калорий из рецепта в дневник.

1.9.4.7 Аналитика

Функция `show_history` отображает историю активности пользователя за последние 7 дней, включая количество шагов, сожженных и потребленных калорий, а также их баланс, представляя данные в удобном текстовом формате с разбивкой по дням для наглядного отслеживания прогресса.

Функция `show_my_data` выводит сводную информацию о текущих показателях пользователя (рост, вес, фитнес-цель), целях по шагам и калориям, а также рекомендуемом суточном потреблении калорий, предоставляя комплексный обзор ключевых параметров для контроля состояния здоровья и физической формы.

1.9.4.8 Премиум-функции

Функция `check_premium` проверяет текущий статус премиум-подписки пользователя, отображая тип подписки, дату окончания и оставшиеся дни действия, а также предлагая варианты управления подпиской (продление/отмена).

Функция `premium_settings_menu` показывает меню управления премиум-подпиской, включая текущий статус, срок действия и доступные действия (активация, продление, отмена), с проверкой актуальности подписки.

Функция `toggle_premium` переключает статус премиум-подписки (включение/выключение), обрабатывая выбор пользователя и перенаправляя на оформление подписки или ее отмену с подтверждением.

Функция `renew_premium` запускает процесс продления премиум-подписки, предлагая пользователю выбрать новый срок (1/3/6/12 месяцев или вечная подписка) и применяя изменения к текущей дате окончания.

Функция `save_premium_subscription` сохраняет данные о премиум-подписке в базу данных, включая тип подписки, срок действия и статус, обновляя

информацию как в основной таблице пользователей, так и в таблице подписок.

Функция `check_premium_status` проверяет активен ли премиум-доступ у пользователя, учитывая не только флаг в профиле, но и актуальность даты окончания подписки, возвращая `True/False`.

Функция `update_premium_status` обновляет статус премиум-подписки в базе данных (вкл/выкл), синхронизируя изменения между таблицами пользователей и подписок, а также автоматически отключая премиум-функции при истечении срока.

Функция `check_expired_subscriptions` ежедневно проверяет все активные подписки на истечение срока действия, автоматически отключает просроченные и уведомляет пользователей о необходимости продления.

1.9.4.9 Вспомогательные функции

Функция `init_user_stats` инициализирует статистику нового пользователя, рассчитывая рекомендуемые цели по шагам и калориям на основе его антропометрических данных и фитнес-целей.

Функция `init_user_notifications` устанавливает стандартные настройки уведомлений для нового пользователя (мотивация, статистика, рецепты) с предустановленными интервалами.

Функция `get_db_connection` создает и возвращает подключение к базе данных PostgreSQL, используя параметры из конфигурации.

Функция `user_exists` проверяет, зарегистрирован ли пользователь в системе, по его `user_id`.

Функция `get_user_info` получает полную информацию о пользователе из базы данных, включая личные данные, статистику и настройки уведомлений.

Функция `create_user_history_table` создает персональную таблицу истории активности для пользователя, если она не существует.

Функция `get_user_history` возвращает историю активности пользователя за указанное количество дней (шаги, калории).

Функция `save_user_history` сохраняет ежедневную статистику пользователя

в его таблицу истории, обновляя или добавляя запись.

Функция `reset_daily_stats` обнуляет дневные показатели активности всех пользователей (шаги, калории) и сохраняет их в историю.

Функция `save_user_to_db` сохраняет или обновляет основные данные пользователя в базе данных при регистрации или изменении профиля.

Функция `update_user_stats` обновляет статистические данные пользователя (шаги, калории) и связанные настройки уведомлений.

Функция `update_user_field` обновляет конкретное поле в профиле пользователя (например, вес или рост) с проверкой изменений.

Функция `is_valid_email` проверяет корректность формата email (наличие @ и домена).

Функция `is_email_unique` проверяет, не используется ли email другим пользователем (кроме текущего).

Функция `is_phone_unique` проверяет уникальность номера телефона в системе.

Функция `send_notification` формирует и отправляет персонализированное уведомление пользователю с текущей статистикой активности.

Функция `is_valid_phone` валидирует номер телефона (формат +7 или 8, количество цифр).

Функция `format_name` форматирует имя/фамилию (первая буква заглавная, остальные строчные).

Функция `is_valid_name` проверяет корректность имени (только буквы, апострофы или дефисы).

Функция `is_valid_birthday` проверяет, что дата рождения в допустимом диапазоне (возраст 18-100 лет).

Функция `calculate_age` вычисляет возраст пользователя по дате рождения.

Функция `calculate_bmr` рассчитывает базовый метаболизм (BMR) по формуле Миффлина-Сан Жеора с учетом пола.

Функция `calculate_daily_calories` определяет рекомендуемую норму калорий

на основе BMR, уровня активности и целей.

Функция `calculate_step_goal` вычисляет индивидуальную цель по шагам с учетом возраста, веса, роста и фитнес-цели.

Функция `calculate_calorie_goal` определяет цель по сжиганию калорий с поправкой на пол и вес.

Функция `format_fitness_goal` преобразует техническое название цели ("lose"/"gain") в читаемый формат.

Функция `get_recipes_by_goal_and_meal` возвращает рецепты, соответствующие фитнес-цели и типу приема пищи.

Функция `get_random_recipe_by_goal` выбирает случайный рецепт для указанной фитнес-цели.

Функция `get_last_notification_time` получает время последнего отправленного уведомления определенного типа.

Функция `update_last_notification_time` обновляет метку времени последнего уведомления для пользователя.

Функция `recalculate_goals` пересчитывает цели по шагам и калориям при изменении антропометрических данных.

Функция `add_steps_with_calories` добавляет шаги к статистике и автоматически рассчитывает эквивалентные сожженные калории.

Функция `get_user_notification_settings` возвращает текущие настройки уведомлений пользователя.

Функция `update_notification_settings` обновляет один или несколько параметров уведомлений в базе данных.

Функция `check_missing_notifications` проверяет и создает отсутствующие записи о настройках уведомлений для новых пользователей.

Функция `get_random_recipe_by_goal_and_meal` выбирает случайный рецепт, подходящий под цель и тип приема пищи.

Функция `get_recipes_grouped_by_meal` возвращает рецепты, сгруппированные по типам приемов пищи.

Функция `save_recipe_to_db` сохраняет пользовательский рецепт в базу данных.

Функция `get_recipe_by_id` получает данные рецепта по его идентификатору.

Функция `show_recipe_details` форматирует и отображает детальную информацию о рецепте с кнопками действий.

Функция `notification_scheduler` основной цикл планировщика, проверяющий необходимость отправки уведомлений каждую минуту.

Функция `check_and_send_recipe_notifications` проверяет расписание и отправляет уведомления с рецептами по времени.

Функция `check_and_send_notifications` координирует отправку всех типов уведомлений (мотивация, статистика, рецепты).

Функция `scheduler` фоновый процесс для ежедневного сброса статистики и проверки подписок.

Функция `on_shutdown` корректно завершает работу бота, отменяя все фоновые задачи.

2 Безопасность жизнедеятельности

2.1 Правила безопасности при использовании ПК

В данном разделе необходимо учесть риски, перечисленные в ГОСТ 12.0.003-2015 "Система стандартов безопасности труда. Электронно-вычислительная техника. Общие требования безопасности" [8]. Особое внимание следует уделить следующим аспектам:

– риски для зрения и повышенной утомляемости глаз:

- 1) продолжительная работа за компьютером без регулярных перерывов;
- 2) несоблюдение эргономических норм: неправильное расположение монитора, блики на экране, недостаточная освещенность рабочего пространства.

– риски возникновения заболеваний опорно-двигательного аппарата:

- 1) неправильная поза при работе: сутулость, неудобное положение рук или ног;
- 2) отсутствие специализированной мебели (эргономичных кресел, подставок для клавиатуры и т. д.);

– риски, связанные с электромагнитным излучением:

- 1) применение устаревшего или неисправного оборудования, генерирующего повышенные уровни электромагнитных полей;
- 2) неправильная расстановка техники, приводящая к избыточному воздействию на пользователя;

– риски поражения электрическим током и возникновения пожара:

- 1) нарушение правил эксплуатации электрооборудования: перегрузка сетей, использование поврежденных проводов;
- 2) несоблюдение норм пожарной безопасности: размещение легковоспламеняющихся материалов вблизи рабочих мест;

– риски нарушения информационной безопасности:

- 1) использование ненадежных паролей или их хранение в открытом доступе;
- 2) несанкционированный доступ к данным, ведущий к утечке

					ПГУ1.090301.38.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		79

конфиденциальной информации;

– риски, обусловленные повышенным уровнем шума. работа в условиях повышенного шума, вызванного компьютерным оборудованием или внешними факторами.

Учет указанных рисков позволит минимизировать потенциальные угрозы для здоровья пользователей и обеспечить безопасные условия труда при работе с электронно-вычислительной техникой.

2.2 Правила безопасности при использовании смартфона

В этом разделе следует уделить внимание рискам, связанным с безопасностью при использовании данного устройства. А также разработать свод правил, чтобы гарантировать предотвратить следующие риски:

– риск зависимости и негативного воздействия на психологическое и физическое здоровье. Чрезмерное использование смартфона, включая зависимость от социальных сетей, игр и постоянного онлайн-присутствия, что может негативно сказываться на здоровье и социальной активности.

– риск отрицательного влияния на сон и здоровье:

1) использование смартфона перед сном, из-за светового излучения экрана, может нарушать естественный сон и вызывать бессонницу;

2) чрезмерное использование смартфона в темное время суток может снижать уровень мелатонина, гормона сна;

– риск потери или кражи смартфона:

1) несанкционированное получение доступа к устройству, возможность потери или кражи смартфона, что может привести к утечке личных данных или использованию их в недобросовестных целях;

2) нарушение конфиденциальности и безопасности данных в случае потери или кражи смартфона;

– риск воздействия радиочастотного излучения:

1) возможные воздействия радиочастотного излучения на организм при длительном использовании смартфона, хотя точные последствия такого воздействия все еще находятся в стадии исследования;

2) при работе в плохо покрытых районах мобильной связи смартфон может выделять больше энергии для подключения к сети, что может привести к повышенному радиочастотному излучению;

– риск отвлечения водителя. Использование смартфона во время вождения может отвлекать водителя и повышать риск дорожно-транспортных происшествий. Это особенно актуально в случае чтения сообщений, отправки сообщений или проведения телефонных разговоров во время движения.

С учетом указанных рисков, следует разработать и рекомендовать следующие правила безопасности при использовании смартфона:

- ограничьте время использования смартфона, особенно перед сном;
- соблюдайте эргономику и правильное положение тела при работе со смартфоном;
- используйте оригинальные зарядные устройства и кабели, избегайте перегрева и короткого замыкания;
- устанавливайте надежные пароли и используйте функции блокировки экрана для защиты данных;
- храните смартфон в безопасном месте и будьте бдительны в общественных местах;
- регулярно делайте резервные копии данных и используйте антивирусное программное обеспечение для защиты от вредоносных программ;
- ограничьте использование смартфона во время вождения и соблюдайте правила безопасности на дороге.

Правильное соблюдение этих правил поможет минимизировать риски и обеспечить безопасное использование смартфона.

3 Стартап проект

Инициатор проекта планирует создать информационную систему в виде Telegram-бота, которая поможет автоматизировать процессы в сфере фитнеса и здорового образа жизни.

Бизнес будет зарегистрирован в г. Пенза, а офис разместится в помещении, соответствующем всем требованиям законодательства.

Основная цель проекта — получение стабильного дохода за счет предоставления удобного цифрового решения для фитнес-клубов, тренеров и их клиентов.

Целевой группой станут фитнес-центры, частные тренеры и wellness-специалисты, которые хотят упростить запись клиентов, управление расписанием, прием платежей и отслеживание прогресса. В будущем систему можно масштабировать на другие ниши, связанные с здоровьем.

3.1 Сегменты ЦА

Целевая аудитория проекта включает фитнес-клубы, спортивные центры и частных тренеров, а при дальнейшем развитии - другие wellness-организации. Для эффективного взаимодействия с каждым сегментом необходимо учитывать их специфические потребности и предпочтения.

Фитнес-центры стремятся предложить клиентам качественные тренировочные программы, комплексное сопровождение на всех этапах занятий и эффективное использование ресурсов. Они заинтересованы в гибкой системе управления расписанием тренировок, удобном взаимодействии с клиентами и тренерами, а также в функциональных инструментах для отслеживания прогресса посетителей и оценки результатов тренировок.

Каждому сегменту рынка будет предложено индивидуальное коммерческое предложение, максимально учитывающее их особенности. Стоимость создания фитнес-бота будет варьироваться в зависимости от требуемого функционала и

специфических пожеланий каждого заказчика.

Ключевыми партнерами проекта станут поставщики IT-оборудования, фитнес-клубы и wellness-центры. Технические партнеры обеспечат надежное оборудование и серверные мощности для стабильной работы системы. Основная выгода для них - долгосрочное сотрудничество и гарантированный спрос на их услуги.

Фитнес-центры и спортивные учреждения выступят основными клиентами системы. Сотрудничество с ними предоставит им современный инструмент для автоматизации бизнес-процессов и улучшения качества обслуживания. Взаимная выгода заключается в создании цифровой платформы, которая поможет оптимизировать работу клуба, повысить лояльность клиентов и увеличить доходы при сокращении операционных расходов.

3.2 Ключевые процессы

Ниже перечислены основные ключевые процессы при разработке стартап-проекта:

- создание ИП;
- поиск и создание команды;
- поиск средств;
- разработка ПО;
- поиск каналов сбыта и взаимодействие с целью продажи;
- поддержка проданного продукта.

Для легализации бизнеса требуется зарегистрироваться в качестве индивидуального предпринимателя. На этом этапе важно правильно подобрать коды ОКВЭД, соответствующие деятельности проекта. Наиболее подходящими являются 62.01 («Разработка компьютерного программного обеспечения») и 96.04 («Деятельность физкультурно-оздоровительная»). Далее необходимо выбрать систему налогообложения – оптимальным вариантом станет УСН с ставкой 6% от дохода, особенно в Пензенской области, где такая ставка действует. Подачу

документов удобнее всего осуществить через МФЦ или портал Госуслуги, чтобы избежать уплаты госпошлины. После подачи заявления в течение трёх дней на электронную почту придёт выписка из ЕГРИП, подтверждающая регистрацию.

Успешная реализация проекта невозможна без грамотно подобранной команды. На начальном этапе потребуются разработчики (программисты и тестировщики), отвечающие за создание и отладку бота. Также важно привлечь фитнес-экспертов, которые помогут сформировать контент и давать пользователям профессиональные рекомендации. Маркетолог займётся продвижением продукта, а бухгалтерский учёт можно доверить аутсорсинговой компании, например, сервису «Моё дело».

Разработка Telegram-бота начинается с анализа целевой аудитории, в которую входят спортсмены, приверженцы здорового образа жизни и новички в фитнесе. Затем проектируется функционал: планы тренировок, трекинг питания, напоминания о занятиях и чат-поддержка. Для технической реализации используются Python вместе с фреймворком Aiogram, API Telegram и база данных PostgreSQL. После завершения программирования проводится тестирование, и бот запускается в работу.

Для привлечения клиентов применяются различные маркетинговые инструменты. Коммерческие предложения рассылаются фитнес-клубам и персональным тренерам, заинтересованным в автоматизации своих услуг. Таргетированная реклама в социальных сетях (Instagram, ВКонтакте) помогает охватить широкую аудиторию. Дополнительным каналом продвижения становятся партнёрские программы с блогерами, специализирующимися на здоровом образе жизни.

Ключевыми преимуществами проекта являются доступность, удобство и уникальность. Telegram-бот обходится дешевле, чем разработка мобильного приложения, при этом мессенджер популярен среди разных возрастных групп. На рынке пока нет аналогов с аналогичным функционалом, что даёт проекту конкурентное преимущество. Ниже в таблице 7 приведены услуги, предложенные

					ПГУ1.090301.38.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		84

в коммерческом предложении.

Таблица 7 –наименование предложенных услуг

Наименование	Цена реализации
Разработка под организацию	25 000р
Настройка и внедрение	7000р
Поддержка	1000р/мес
Масштабируемость	5000р

После запуска важно продолжать развивать проект, собирая обратную связь от пользователей и регулярно обновляя контент. В перспективе можно добавить AI-ассистента, интегрировать бота с умными устройствами и расширить функционал за счёт новых фитнес-программ.

Таким образом, Telegram-бот для здоровья и фитнеса представляет собой перспективный стартап с относительно низкими затратами на вход и высоким потенциалом спроса. Грамотная реализация позволит занять устойчивую позицию на рынке digital-услуг для ЗОЖ.

3.3 Каналы взаимодействия

Важным инструментом привлечения клиентов станет создание сайта-визитки. Этот ресурс позволит потенциальным пользователям в любое время получать полную информацию о компании и предлагаемых услугах. На сайте будет подробно описан функционал фитнес-бота, его преимущества и возможности. Интерактивные элементы, такие как контактные формы и чат поддержки, значительно упростят процесс коммуникации с заинтересованными клиентами, позволяя оперативно отвечать на их вопросы.

Для более эффективного продвижения продукта планируется организация персональных встреч с владельцами фитнес-клубов, спортивных центров и wellness-студий. В ходе таких встреч можно будет наглядно продемонстрировать все возможности бота, показать его интерфейс и ответить на конкретные вопросы

потенциальных партнеров. Особое внимание будет уделено презентации уникальных функций, таких как персонализированные тренировочные программы, система мотивации и интеграция с популярными фитнес-трекерами.

После заключения договора о сотрудничестве наша команда возьмет на себя весь процесс внедрения бота. Это включает индивидуальную настройку системы под специфику работы конкретного фитнес-центра, интеграцию с существующими CRM-системами и обучение персонала. Для обеспечения плавного перехода будет разработан детальный план внедрения, учитывающий все технические и организационные аспекты.

Особое внимание будет уделено постпродажной поддержке клиентов. В течение гарантийного периода наша команда обязуется оперативно решать любые технические вопросы, вносить необходимые доработки и предоставлять консультации по использованию системы. Для поддержания долгосрочных отношений с клиентами планируется регулярно выпускать обновления функционала, а также организовывать тематические вебинары и онлайн-встречи для обсуждения новых возможностей и сбора обратной связи.

Для постоянных клиентов будет разработана программа лояльности, включающая специальные условия на обновление функционала и приоритетную техническую поддержку. Также планируется создание закрытого клубного сообщества пользователей бота, где они смогут обмениваться опытом, предлагать идеи для развития системы и получать эксклюзивный контент от фитнес-экспертов. Такой подход позволит не только поддерживать высокий уровень удовлетворенности клиентов, но и формировать лояльное сообщество вокруг продукта.

3.4 Источники дохода

Предлагаемый фитнес-бот будет генерировать доход через несколько каналов монетизации, обеспечивая стабильный денежный поток и возможности для масштабирования бизнеса.

					ПГУ1.090301.38.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		86

Основные источники дохода:

– разработка индивидуальных решений. Команда будет получать доход от создания персонализированных версий бота для фитнес-клубов и wellness-центров. Стоимость разработки индивидуального решения составляет 25 000 рублей за один проект. Эта сумма включает адаптацию функционала под специфику работы конкретного заведения, фирменный стиль и интеграцию с существующими системами клиента.

– внедрение и настройка. За процесс внедрения системы в работу организации предусмотрена отдельная плата в размере 7 000 рублей. Эта услуга включает установку бота, обучение персонала, первоначальную настройку и тестовый период сопровождения.

– абонентская плата за обслуживание. После внедрения клиентам будет предлагаться подписка на техническую поддержку и регулярные обновления системы стоимостью 1 500 рублей в месяц. В пакет поддержки входит решение технических вопросов, консультации и доступ к новым функциям.

– масштабирование на филиалы. Для сетевых фитнес-клубов предусмотрена возможность расширения системы на дополнительные филиалы. Стоимость подключения каждого нового филиала составляет 7 000 рублей, что включает адаптацию под особенности конкретного места и синхронизацию данных.

– прогноз доходности. Согласно данным аналитики фитнес-рынка Пензенской области, в регионе насчитывается около 120 фитнес-клубов и спортивных центров различного масштаба. При консервативной оценке, что 40% из них (48 организаций) проявят интерес к продукту, а каждая сеть имеет в среднем 3 филиала, можно рассчитать потенциальный доход.

Доход высчитывается по формуле

$$Д = К \times Ц \quad (1)$$

где Д – доход;

К – количество потребителей;

Ц – установленная цена;

На разработку индивидуальных решений будет выделяться 1 200 000 рублей (48 потребителей на 25 000 рублей).

Затраты на внедрение системы составят 336 000 рублей (48 потребителей на 7 000 рублей).

Подключение филиалов обойдется в 1 008 000 рублей (144 потребителей на 25 000 рублей).

Ежемесячная подписка будет стоить 72 000 рублей в месяц (48 потребителей на 1 500 рублей).

Таким образом, общий ожидаемый доход в г. Пенза составит 2 544 000 рублей единовременно и 72 000 рублей ежемесячно. Для наглядности на в таблице 2 представлена прогнозируемая программа производства по месяцам для коммерческих образовательных учреждений по г. Пенза.

Таблица 8 - Прогнозируемая программа производства

Услуга	1 мес.	2 мес.	3 мес.	4 мес.
Разработка	8	12	10	8
Внедрение	8	12	10	8
Подключение филиалов	15	24	30	25
Абонентское обслуживание	16	28	38	46

3.5 Ключевые ресурсы

Для успешной реализации проекта потребуется комплекс различных ресурсов. В первую очередь необходимы квалифицированные кадры: разработчики программного обеспечения со специализацией на создании Telegram-ботов, которые будут заниматься созданием и поддержкой ядра системы. В команду также войдет менеджер по продажам с опытом работы в b2b-сегменте фитнес-индустрии или IT-решений, отвечающий за привлечение клиентов и ведение переговоров. Дополнительно планируется привлечение фитнес-эксперта в качестве консультанта для разработки тренировочных программ и контента. Бухгалтерское сопровождение проекта будет

осуществляться через аутсорсинговую компанию.

Техническое оснащение проекта включает несколько важных компонентов. Для работы разработчиков потребуются два производительных ноутбука. Серверное оборудование необходимо для надежного хостинга бота, при этом важно предусмотреть резервные мощности для обеспечения бесперебойной работы. В перечень программного обеспечения войдут лицензии на среду разработки, подписки на необходимые сервисы и API, а также системы контроля версий. Инфраструктурная часть включает аренду офисного помещения площадью 30-40 квадратных метров, оснащенного необходимой оргтехникой и мебелью, а также системы видеосвязи для проведения удаленных переговоров с клиентами.

Финансовая модель проекта основана на бюджете в 1 000 000 рублей, который формируется из двух основных источников. Половина суммы (500 000 рублей) поступает из собственных средств инициатора проекта, еще 500 000 рублей планируется получить через государственный грант для молодых предпринимателей. Распределение бюджета осуществляется по нескольким направлениям.

Технические затраты включают несколько статей расходов. Аренда серверных мощностей на шесть месяцев обойдется в 18 000 рублей. На офисное помещение за этот же период планируется потратить 240 000 рублей. Закупка двух ноутбуков для разработчиков потребует 200 000 рублей бюджета. Оснащение офиса оргтехникой и мебелью оценивается в 150 000 рублей. Коммунальные платежи за полгода составят примерно 30 000 рублей. Таким образом, общие затраты на техническое обеспечение проекта достигают 638 000 рублей.

Расходы на персонал включают несколько позиций. Два разработчика в течение шести месяцев будут получать в общей сложности 600 000 рублей. Менеджеру по продажам за этот же период планируется выплатить 180 000 рублей. Бухгалтерское сопровождение обойдется в 60 000 рублей за полгода. В сумме затраты на человеческие ресурсы составят 840 000 рублей.

					ПГУ1.090301.38.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		89

Стоит отметить, что общая сумма планируемых затрат несколько превышает бюджет проекта. Эта ситуация будет решена за счет нескольких мер оптимизации. Часть оборудования может быть приобретена в б/у-состоянии, что позволит снизить затраты. Ведутся переговоры по поводу возможного снижения арендной платы за офисное помещение. Кроме того, предусмотрено поэтапное наращивание мощностей по мере поступления первых доходов от проекта.

3.6 Риски проекта

Реализация данного стартап-проекта сопряжена с рядом потенциальных рисков, которые могут повлиять на его успешность и жизнеспособность. К основным рискам можно отнести недостаточный рыночный спрос на предлагаемое решение, что может быть связано как с неверной оценкой целевой аудитории, так и с недостаточной информированностью потребителей о преимуществах продукта. Серьезную угрозу представляет конкуренция со стороны уже существующих фитнес-приложений и сервисов, которые могут предложить аналогичный функционал.

Значительным риском являются возможные изменения в законодательстве, регулирующем сферу цифровых медицинских и фитнес-услуг, что может потребовать дополнительных затрат на приведение продукта в соответствие с новыми требованиями. Финансовые риски включают в себя проблемы с привлечением инвестиций, нехватку оборотных средств на этапе разработки и вывода продукта на рынок, а также возможные ошибки в расчёте себестоимости и ценообразовании.

Технические риски проекта связаны с возможными сложностями при разработке и интеграции системы, включая проблемы с масштабируемостью решения, безопасностью данных пользователей и стабильностью работы бота при высокой нагрузке. Особое внимание следует уделить рискам, связанным с защитой персональных данных пользователей, так как бот будет обрабатывать информацию о здоровье и физическом состоянии клиентов.

					ПГУ1.090301.38.001 ПЗ	Лист
						90
Изм.	Лист	№ докум.	Подпись	Дата		

Помимо стандартных бизнес-рисков, проект может столкнуться с форс-мажорными обстоятельствами, такими как глобальные экономические кризисы, изменения валютных курсов, санкционные ограничения, влияющие на ИТ-сектор, или пандемии, которые могут изменить поведение целевой аудитории и рыночные условия. Природные катастрофы или техногенные аварии также могут нарушить нормальный ход реализации проекта.

Для снижения перечисленных рисков предусматривается:

- проведение детального маркетингового исследования перед запуском проекта;
- разработка гибкой бизнес-модели с возможностью быстрой адаптации к изменениям рынка;
- создание финансового резерва на непредвиденные расходы;
- поэтапное тестирование продукта с привлечением фокус-групп;
- заключение партнёрских соглашений с фитнес-центрами и нутрициологами;
- регулярный мониторинг изменений законодательства;
- разработка альтернативных сценариев развития проекта.

Особое внимание будет уделено созданию системы регулярного сбора и анализа обратной связи от пользователей, что позволит оперативно вносить коррективы в развитие продукта. Для защиты от форс-мажорных обстоятельств предусматривается диверсификация каналов монетизации и создание "подушки безопасности" в виде резервного фонда.

Заключение

Разработка Telegram-бота для поддержки здоровья и фитнеса представляет собой важный шаг в борьбе с последствиями сидячего образа жизни и неправильного питания. В условиях современного общества, где доступ к информации и поддержка здоровья становятся критически важными, такой бот может стать надежным помощником для пользователей, стремящихся улучшить свою физическую форму и общее состояние здоровья.

Интуитивный интерфейс и функциональные возможности бота, включая персонализированные рекомендации, отслеживание прогресса и напоминания о тренировках, создадут условия для более активного и ответственного подхода к здоровью. Ожидается, что система не только упростит путь к достижению фитнес-целей, но и станет катализатором формирования здоровых привычек, способствуя активному образу жизни и улучшению качества жизни пользователей. В конечном итоге, реализация данного проекта будет способствовать созданию более здорового и активного общества, что является важной целью в условиях современных вызовов.

					ПГУ1.090301.38.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		92

Список использованных источников

1. Habr.com: [Электронный ресурс]. URL: <https://habr.com/ru/articles/697052> (дата обращения: 27.04.2025)
2. Артем Демиденко. "Telegram Bot. Руководство по созданию бота в мессенджере Телеграм", 2023. - 23с.
3. MyFitnessPal: [Электронный ресурс]. URL: <https://www.myfitnesspal.com> (дата обращения: 01.06.2025)
4. Bodybuilding.com: [Электронный ресурс]. URL: <https://www.bodybuilding.com> (дата обращения: 01.06.2025)
5. Лучано Рамальо. "Python. К вершинам мастерства", 2018. – 770с.
6. Официальная документация Aiogram: [Электронный ресурс]. URL: <https://docs.aiogram.dev> (дата обращения: 01.06.2025)
7. Е. Рогов. "PostgreSQL изнутри", 2015. – 661с.
8. Consultant.ru: [Электронный ресурс]. URL: https://www.consultant.ru/document/cons_doc_LAW (дата обращения: 05.05.2025)

Приложение А

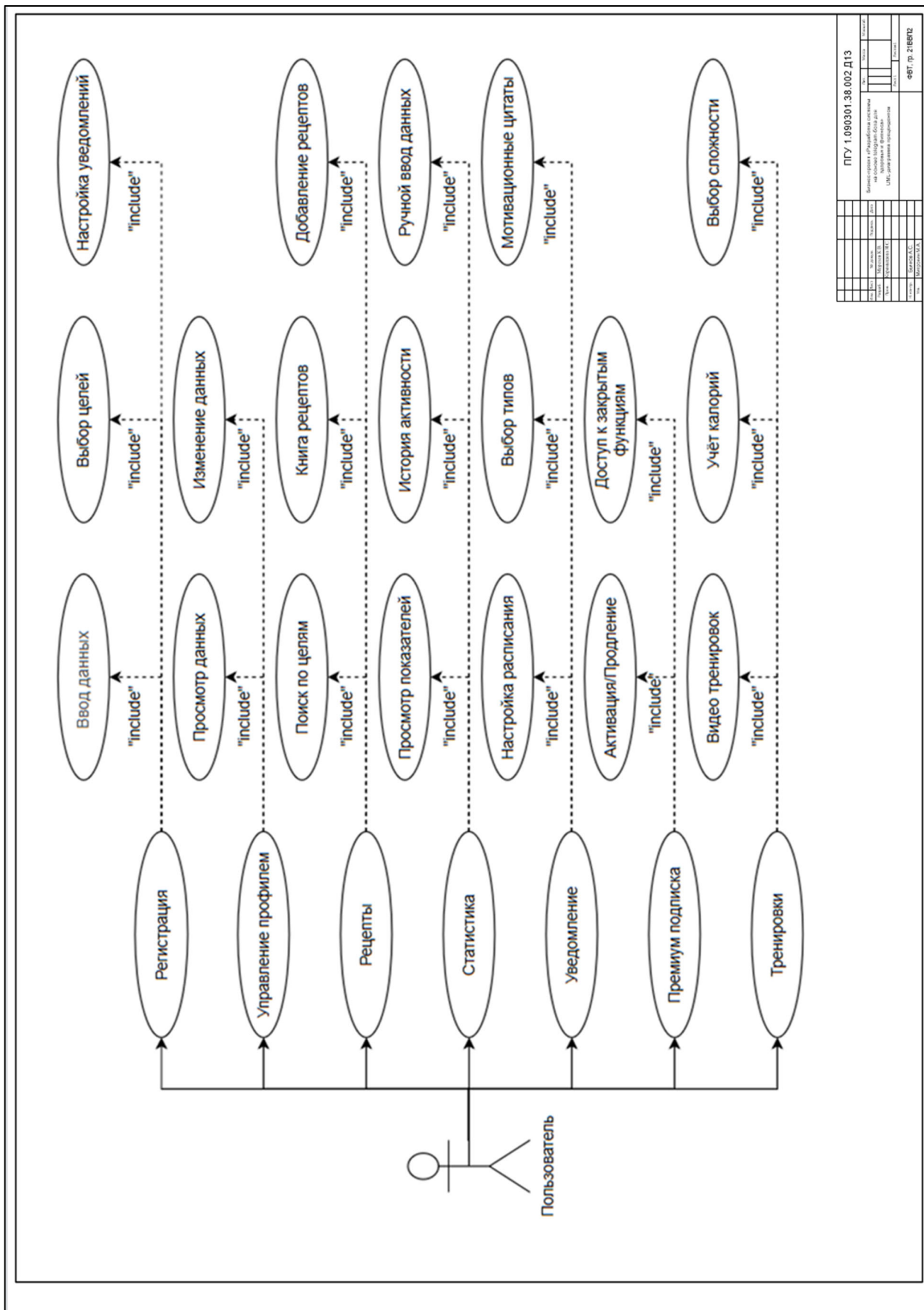
(Обязательное)

Бизнес-проект «Разработка системы на основе telegram-бота для здоровья и
фитнеса»

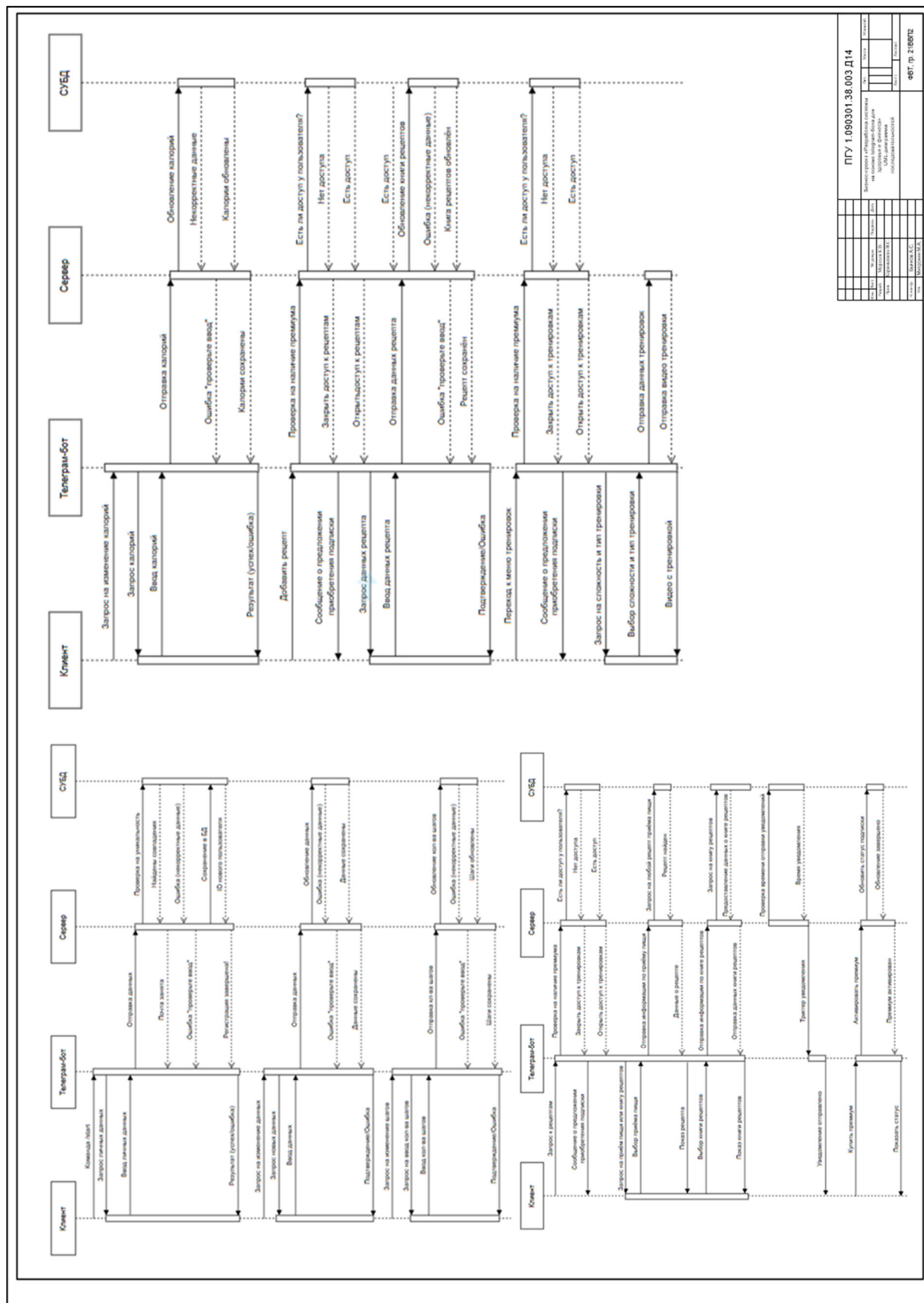
UML диаграммы

					ПГУ1.090301.38.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		94

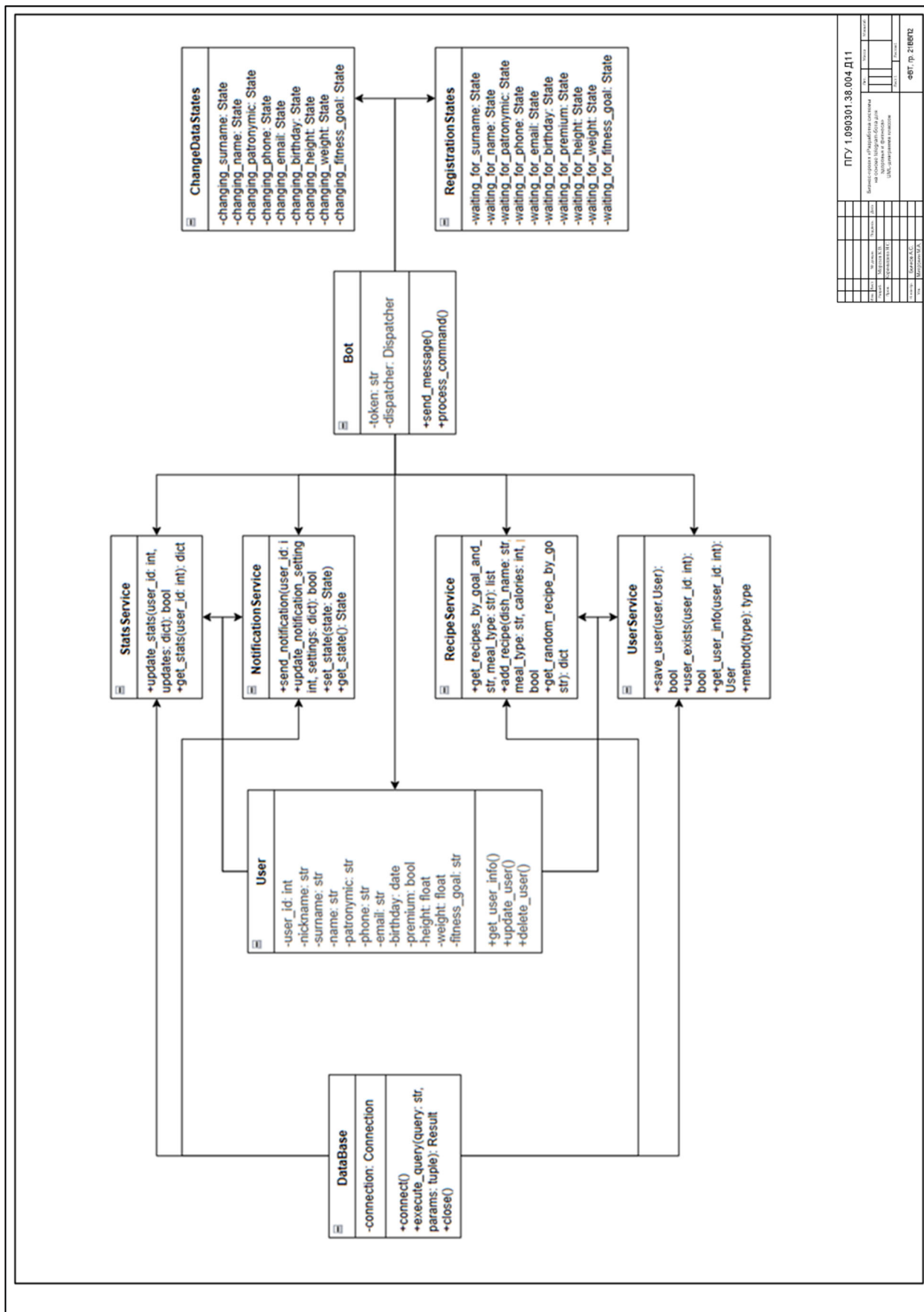
Приложение А.1 – UML – диаграмма прецедентов пользователя



Приложение А.2 – UML – диаграмма последовательностей



Приложение А.3 – UML – диаграмма классов



ПГУ 1.090301.38.004.Д11		Имя	Фамилия	Отчество
Бизнес-система «Облачная система управления персональными данными»		Имя	Фамилия	Отчество
УML-диаграмма классов		Имя	Фамилия	Отчество
487. (9. 2.08.02)		Имя	Фамилия	Отчество

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.38.001 ПЗ

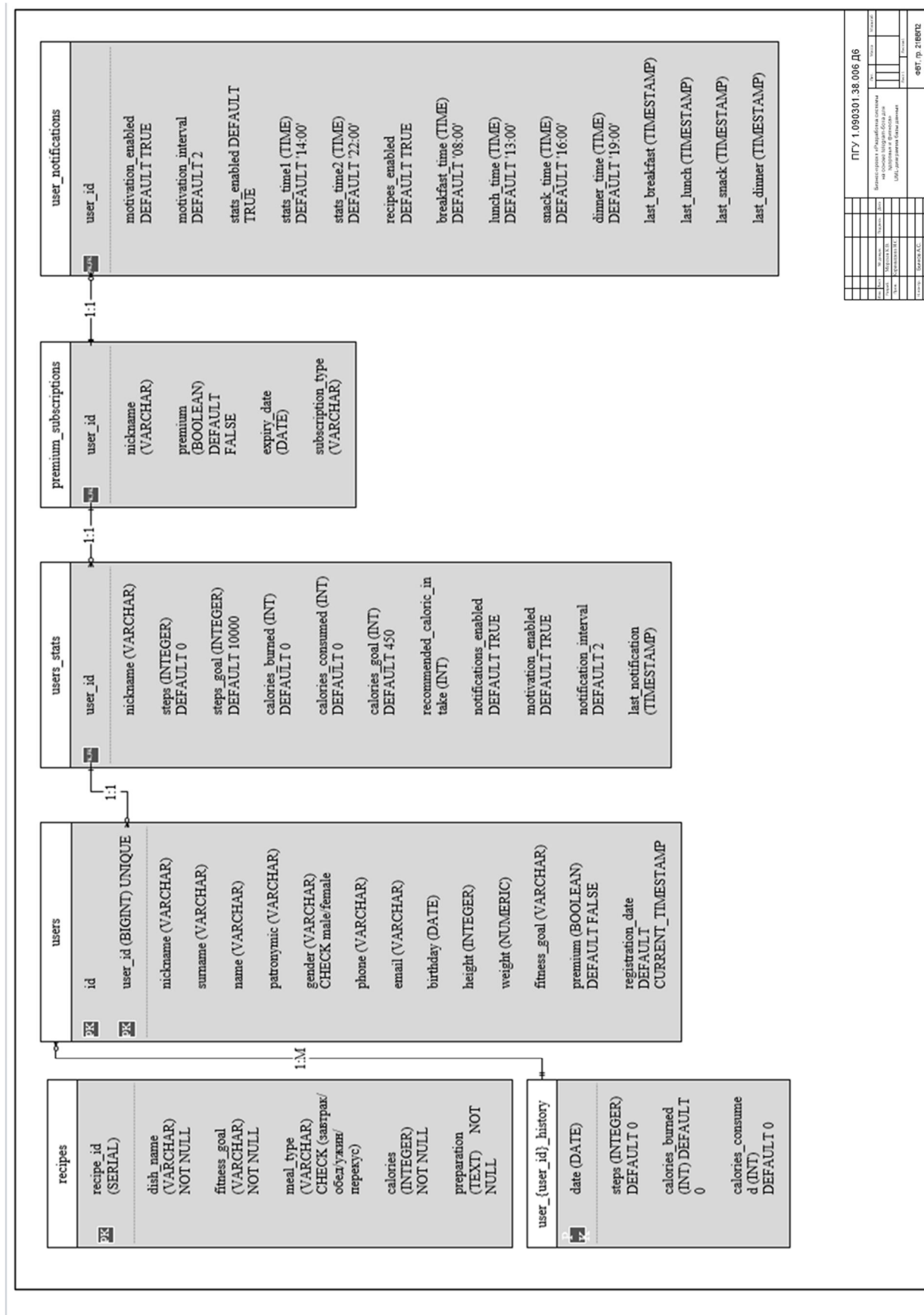
Лист

97

Приложение А.4 – UML – диаграмма равёртывания



Приложение А.5 – UML - диаграмма базы данных



Приложение Б

(Обязательное)

Бизнес-проект «Разработка системы на основе telegram-бота для здоровья и
фитнеса»

Листинг программы

					ПГУ1.090301.38.001 ПЗ	Лист
						100
Изм.	Лист	№ докум.	Подпись	Дата		

Файл «.gitignore»

```
.env  
__pycache__/  
*.pyc
```

Файл «.env»

```
# Telegram Bot Token  
TELEGRAM_TOKEN='7653267088:AAHU4bJraVDas03EaJc89c1HyEmDefNKens'  
# Database Configuration  
DB_HOST=localhost  
DB_USER=postgres  
DB_PASSWORD=1234  
DB_NAME=haf  
DB_PORT=5432
```

Файл «keyboard.py»

```
from aiogram.types import InlineKeyboardButton,  
InlineKeyboardMarkup, ReplyKeyboardMarkup, KeyboardButton  
from aiogram.utils.keyboard import InlineKeyboardBuilder,  
ReplyKeyboardBuilder  
  
registration_kb = InlineKeyboardBuilder()  
registration_kb.button(text='Регистрация',  
callback_data='register')  
  
start_kb = InlineKeyboardBuilder()  
start_kb.button(text='Перейти в главное меню',  
callback_data='Menu')  
  
menu_kb = InlineKeyboardBuilder()  
menu_kb.button(text='Изменить данные', callback_data='Change')  
menu_kb.button(text='Шаги', callback_data='Step'),  
menu_kb.button(text='Каллории', callback_data='Calory'),  
menu_kb.button(text='Тренировки', callback_data='Training'),  
menu_kb.button(text='Рецепты', callback_data='Recipe'),  
menu_kb.button(text='Уведомления',  
callback_data='notification_settings'),  
menu_kb.button(text='Премиум',  
callback_data='Premium_settings'),  
menu_kb.button(text='Ссылка на разработчика',  
url="https://t.me/Morozko122")  
menu_kb.adjust(1,2,2,2,1)  
  
# Клавиатуры для рецептов  
recipe_main_kb = InlineKeyboardBuilder()  
recipe_main_kb.button(text="☕ Завтраки",  
callback_data="recipe_breakfast")  
recipe_main_kb.button(text="🍲 Обеды",  
callback_data="recipe_lunch")
```

					ПГУ1.090301.38.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		101

```

recipe_main_kb.button(text="🍷 Ужины",
callback_data="recipe_dinner")
recipe_main_kb.button(text="🥪 Перекусы",
callback_data="recipe_snack")
recipe_main_kb.button(text="📖 Книга рецептов",
callback_data="recipe_book")
recipe_main_kb.button(text="◀ Главное меню",
callback_data="Menu")
recipe_main_kb.adjust(2, 2, 1, 1)

recipe_actions_kb = InlineKeyboardBuilder()
recipe_actions_kb.button(text="↻ Другой рецепт",
callback_data="recipe_another")
recipe_actions_kb.button(text="Добавить в потребление калорий",
callback_data="add_recipe_calories")
recipe_actions_kb.button(text="📖 Книга рецептов",
callback_data="recipe_book")
recipe_actions_kb.button(text="◀ Назад", callback_data="Recipe")
recipe_actions_kb.adjust(1, 1, 2)

recipe_type_kb = InlineKeyboardBuilder()
recipe_type_kb.button(text="🍳 Завтраки",
callback_data="recipe_breakfast")
recipe_type_kb.button(text="🍽 Обеды",
callback_data="recipe_lunch")
recipe_type_kb.button(text="🍷 Ужины",
callback_data="recipe_dinner")
recipe_type_kb.button(text="🥪 Перекусы",
callback_data="recipe_snack")
recipe_type_kb.button(text="◀ Назад", callback_data="Recipe")
recipe_type_kb.adjust(2, 2, 1)

back_recipe_kb = InlineKeyboardBuilder()
back_recipe_kb.button(text="Назад к рецептам",
callback_data="Recipe")
back_recipe_kb.button(text="В главное меню",
callback_data="Menu")

# Для регистрации
registration_fitness_goal_kb = InlineKeyboardBuilder()
registration_fitness_goal_kb.button(text="Похудеть",
callback_data="goal_lose")
registration_fitness_goal_kb.button(text="Набрать мышцы",
callback_data="goal_gain")
registration_fitness_goal_kb.button(text="Вести здоровый образ
жизни", callback_data="goal_healthy")
registration_fitness_goal_kb.adjust(1)

fitness_goal_kb = InlineKeyboardBuilder()

```

```

fitness_goal_kb.button(text="Похудеть",
callback_data="goal_lose_weight")
fitness_goal_kb.button(text="Набрать мышцы",
callback_data="goal_gain_muscle")
fitness_goal_kb.button(text="Вести здоровый образ жизни",
callback_data="goal_healthy_lifestyle")
fitness_goal_kb.button(text='Перейти в меню изменения данных',
callback_data='Change')
fitness_goal_kb.button(text='Перейти в главное меню',
callback_data='Menu')
fitness_goal_kb.adjust(1)

# Добавьте эту кнопку в классе с клавиатурами
add_recipe_kb = InlineKeyboardBuilder()
add_recipe_kb.button(text="Добавить рецепт",
callback_data="add_recipe")
add_recipe_kb.adjust(1)

# И модифицируем клавиатуру меню рецептов:
recipe_main_kb = InlineKeyboardBuilder()
recipe_main_kb.button(text="Завтрак",
callback_data="recipe_breakfast")
recipe_main_kb.button(text="Обед", callback_data="recipe_lunch")
recipe_main_kb.button(text="Ужин",
callback_data="recipe_dinner")
recipe_main_kb.button(text="Перекус",
callback_data="recipe_snack")
recipe_main_kb.button(text="Книга рецептов",
callback_data="recipe_book")
recipe_main_kb.button(text="Добавить рецепт",
callback_data="add_recipe")
recipe_main_kb.button(text="Назад", callback_data="Menu")
recipe_main_kb.adjust(2, 2, 1, 1, 1)

fitness_goal_for_recipe_kb = InlineKeyboardBuilder()
fitness_goal_for_recipe_kb.button(text="Похудение",
callback_data="add_recipe_goal_lose")
fitness_goal_for_recipe_kb.button(text="Набор массы",
callback_data="add_recipe_goal_gain")
fitness_goal_for_recipe_kb.button(text="Здоровое питание",
callback_data="add_recipe_goal_healthy")
fitness_goal_for_recipe_kb.adjust(1)

meal_type_kb = InlineKeyboardBuilder()
meal_type_kb.button(text="Завтрак",
callback_data="add_recipe_meal_breakfast")
meal_type_kb.button(text="Обед",
callback_data="add_recipe_meal_lunch")
meal_type_kb.button(text="Ужин",
callback_data="add_recipe_meal_dinner")

```

```

meal_type_kb.button(text="Перекус",
callback_data="add_recipe_meal_snack")
meal_type_kb.button(text='Назад к рецептам',
callback_data='Recipe'),
meal_type_kb.button(text='В главное меню',
callback_data='Menu'),
meal_type_kb.adjust(2)

change_separate_kb = InlineKeyboardBuilder()
change_separate_kb.button(text="Фамилия",
callback_data="change_surname")
change_separate_kb.button(text="Имя",
callback_data="change_name")
change_separate_kb.button(text="Отчество",
callback_data="change_patronymic")
change_separate_kb.button(text="Пол",
callback_data="change_gender")
change_separate_kb.button(text="Телефон", callback_data="Phone")
change_separate_kb.button(text="Email", callback_data="Email")
change_separate_kb.button(text="Дата рождения",
callback_data="Birthday")
change_separate_kb.button(text="Рост",
callback_data="change_height")
change_separate_kb.button(text="Вес",
callback_data="change_weight")
change_separate_kb.button(text="Фитнес-цель",
callback_data="change_fitness_goal")
change_separate_kb.button(text="Назад", callback_data="Menu")
change_separate_kb.adjust(3, 3, 1, 2, 1, 1)

skip_email_kb = InlineKeyboardBuilder()
skip_email_kb.button(text="Пропустить", callback_data ='Skip_E')

skip_or_cancel_kb = InlineKeyboardBuilder()
skip_or_cancel_kb.button(text="Удалить email", callback_data
='Skip_or_C_1')
skip_or_cancel_kb.button(text='Перейти в меню изменения данных',
callback_data='Change')
skip_or_cancel_kb.button(text='Перейти в главное меню',
callback_data='Menu')
skip_or_cancel_kb.adjust(1)

back_to_change_kb = InlineKeyboardBuilder()
back_to_change_kb.button(text='Перейти в меню изменения данных',
callback_data='Change')
back_to_change_kb.button(text='Перейти в главное меню',
callback_data='Menu')
back_to_change_kb.adjust(1)

# Для пропуска отчества
skip_patronymic_kb = InlineKeyboardBuilder()

```

```

skip_patronymic_kb.button(text="Удалить отчество",
callback_data="skip_patronymic")
skip_patronymic_kb.button(text='Перейти в меню изменения
данных', callback_data='Change')
skip_patronymic_kb.button(text='Перейти в главное меню',
callback_data='Menu')
skip_patronymic_kb.adjust(1)

Rskip_patronymic_kb = InlineKeyboardBuilder()
Rskip_patronymic_kb.button(text="Отчество не имеется",
callback_data="skip_patronymic")

# Для выбора премиума
premium_choice_kb = InlineKeyboardBuilder()
premium_choice_kb.button(text="Да", callback_data="premium_yes")
premium_choice_kb.button(text="Нет", callback_data="premium_no")

# Для настроек уведомлений
notifications_kb = InlineKeyboardBuilder()
notifications_kb.button(text="Только мотивационные",
callback_data="notif_motivation")
notifications_kb.button(text="Только статистика",
callback_data="notif_stats")
notifications_kb.button(text="Все уведомления",
callback_data="notif_all")
notifications_kb.button(text="Выключить все",
callback_data="notif_none")
notifications_kb.adjust(1)

backsc_kb = InlineKeyboardBuilder()
backsc_kb.button(text="Назад в меню", callback_data="Menu")

# Добавляем новые клавиатуры в Keyboard/keyboard.py
premium_toggle_kb = InlineKeyboardBuilder()
premium_toggle_kb.button(text="Активировать Premium",
callback_data="toggle_premium_on")
premium_toggle_kb.button(text="Отключить Premium",
callback_data="toggle_premium_off")
premium_toggle_kb.button(text="В главное меню",
callback_data="Menu")
premium_toggle_kb.adjust(1)

notifications_settings_kb = InlineKeyboardBuilder()
notifications_settings_kb.button(text="Только мотивационные",
callback_data="set_notif_motivation")
notifications_settings_kb.button(text="Только статистика",
callback_data="set_notif_stats")
notifications_settings_kb.button(text="Все уведомления",
callback_data="set_notif_all")
notifications_settings_kb.button(text="Выключить все",
callback_data="set_notif_none")

```

```

notifications_settings_kb.button(text="В главное меню",
callback_data="Menu")
notifications_settings_kb.adjust(1)

back_to_notification_kb = InlineKeyboardBuilder()
back_to_notification_kb.button(text='Перейти в меню изменения
уведомлений', callback_data='notification_settings')
back_to_notification_kb.button(text='Перейти в главное меню',
callback_data='Menu')
back_to_notification_kb.adjust(1)

back_to_Premium_kb = InlineKeyboardBuilder()
back_to_Premium_kb.button(text='Перейти в меню изменения
премиум-статуса', callback_data='Premium_settings')
back_to_Premium_kb.button(text='Перейти в главное меню',
callback_data='Menu')
back_to_Premium_kb.adjust(1)

training_kb = InlineKeyboardBuilder()
training_kb.button(text='Лёгкий', callback_data='Easy'),
training_kb.button(text='Нормальный', callback_data='Medium'),
training_kb.button(text='Тяжёлый', callback_data='Hard'),
training_kb.button(text='На главное меню',
callback_data='Menu'),
training_kb.adjust(3,1)

easy_kb = InlineKeyboardBuilder()
easy_kb.button(text="Кардио-тренировка", callback_data='easy_1')
easy_kb.button(text="Кардио-тренировка на скорость",
callback_data='easy_2')
easy_kb.button(text="Аэробно-силовая тренировка для всего тела",
callback_data='easy_3')
easy_kb.button(text="Тренировка для плоского живота и мышц
кора", callback_data='easy_4')
easy_kb.button(text="Тренировка для бедер и ягодиц",
callback_data='easy_5')
easy_kb.button(text="Растяжка", callback_data='stretch')
easy_kb.button(text="На главное меню", callback_data = "Menu")
easy_kb.button(text="Назад", callback_data='Training')
easy_kb.adjust(1,1,1,1,1,1,2)

medium_kb = InlineKeyboardBuilder()
medium_kb.button(text="Кардио-тренировка с упражнениями для
тонуса мышц", callback_data='medium_1')
medium_kb.button(text="Кардио-тренировка на скорость (Версия
№2)", callback_data='medium_2')
medium_kb.button(text="Скоростная тренировка для мышц кора",
callback_data='medium_3')
medium_kb.button(text="Аэробно-силовая тренировка для всего
тела", callback_data='medium_4')
medium_kb.button(text="Динамическая тренировка для плоского
живота и мышц кора", callback_data='medium_5')

```

```

medium_kb.button(text="Тренировка для рук, плеч и спины",
callback_data='medium_6')
medium_kb.button(text="Растяжка", callback_data='stretch')
medium_kb.button(text="На главное меню", callback_data = "Menu")
medium_kb.button(text="Назад", callback_data='Training')
medium_kb.adjust(1,1,1,1,1,1,2)

hard_kb = InlineKeyboardBuilder()
hard_kb.button(text="Кардио-тренировка на скорость (Версия №3)",
callback_data='hard_1')
hard_kb.button(text="Аэробно-силовая тренировка для всего тела
(Экстримальная версия)", callback_data='hard_2')
hard_kb.button(text="Аэробно-силовая тренировка для всего тела
(программа 'Пирамида')", callback_data='hard_3')
hard_kb.button(text="Тренировка для рук, плеч и спины (более
интенсивная нагрузка)", callback_data='hard_4')
hard_kb.button(text="Растяжка", callback_data='stretch')
hard_kb.button(text="На главное меню", callback_data = "Menu")
hard_kb.button(text="Назад", callback_data='Training')
hard_kb.adjust(1,1,1,1,1,2)

back_kb = InlineKeyboardBuilder()
back_kb.button(text="Добавить калории",
callback_data="add_training_calories")
back_kb.button(text="На главное меню", callback_data = "Menu")
back_kb.button(text="Назад", callback_data='Training')
back_kb.adjust(1)

stretch_kb = InlineKeyboardBuilder()
stretch_kb.button(text="Добавить 400 калорий",
callback_data="add_stretch_calories")
stretch_kb.button(text="На главное меню", callback_data =
"Menu")
stretch_kb.button(text="Назад", callback_data="Training") #
Возврат к выбору сложности
stretch_kb.adjust(1)

confirm_training_kb = InlineKeyboardBuilder()
confirm_training_kb.button(text="Назад к тренировкам",
callback_data="Training")
confirm_training_kb.button(text="На главное меню",
callback_data="Menu")
confirm_training_kb.adjust(1)

steps_actions_kb = InlineKeyboardBuilder()
steps_actions_kb.button(text="Добавить шаги",
callback_data="add_steps")
steps_actions_kb.button(text="Изменить цель",
callback_data="change_steps_goal")
steps_actions_kb.button(text="Назад", callback_data="Menu")
steps_actions_kb.adjust(1)

```

```

calories_actions_kb = InlineKeyboardBuilder()
calories_actions_kb.button(text="Добавить сожженные калории",
callback_data="add_calories_burned")
calories_actions_kb.button(text="Добавить потребленные калории",
callback_data="add_calories_consumed")
calories_actions_kb.button(text="Изменить цель по сжиганию",
callback_data="change_calories_burned_goal")
calories_actions_kb.button(text="Изменить цель по потреблению",
callback_data="change_calories_consumed_goal")
calories_actions_kb.button(text="Назад", callback_data="Menu")
calories_actions_kb.adjust(1, 1, 1, 1, 1)

```

```

step_kb = InlineKeyboardBuilder()
step_kb.button(text='Обратно к шагам', callback_data='Step'),
step_kb.button(text='В главное меню', callback_data='Menu'),
step_kb.adjust(1)

```

```

calory_kb = InlineKeyboardBuilder()
calory_kb.button(text='Обратно к калориям',
callback_data='Calory'),
calory_kb.button(text='В главное меню', callback_data='Menu'),
calory_kb.adjust(1)

```

```

premium_choice_registration_kb = InlineKeyboardBuilder()
premium_choice_registration_kb.button(text="Да",
callback_data="reg_premium_yes")
premium_choice_registration_kb.button(text="Нет",
callback_data="reg_premium_no")
premium_choice_registration_kb.adjust(2)

```

Клавиатура управления Premium

```

premium_manage_kb = InlineKeyboardBuilder()
premium_manage_kb.button(text="🔄 Продлить подписку",
callback_data="toggle_premium_renew")
premium_manage_kb.button(text="❌ Отключить подписку",
callback_data="toggle_premium_off")
premium_manage_kb.button(text="⬅️ Назад", callback_data="Menu")
premium_manage_kb.adjust(1)

```

Клавиатура выбора срока подписки

```

premium_duration_kb = InlineKeyboardBuilder()
premium_duration_kb.button(text="1 месяц - 299 ₽",
callback_data="premium_duration_1")
premium_duration_kb.button(text="3 месяца - 799 ₽",
callback_data="premium_duration_3")
premium_duration_kb.button(text="6 месяцев - 1499 ₽",
callback_data="premium_duration_6")
premium_duration_kb.button(text="Навсегда - 4999 ₽",
callback_data="premium_duration_lifetime")
premium_duration_kb.button(text="⬅️ Назад",
callback_data="Premium_settings")

```

```

premium_duration_kb.adjust(1)

premium_duration_registration_kb = InlineKeyboardBuilder()
premium_duration_registration_kb.button(text="1 месяц - 299 ₽",
callback_data="reg_premium_1")
premium_duration_registration_kb.button(text="3 месяца - 799 ₽",
callback_data="reg_premium_3")
premium_duration_registration_kb.button(text="6 месяцев - 1499
₽", callback_data="reg_premium_6")
premium_duration_registration_kb.button(text="Навсегда - 4999
₽", callback_data="reg_premium_lifetime")
premium_duration_registration_kb.button(text="◀ Назад",
callback_data="reg_premium_back")
premium_duration_registration_kb.adjust(1)

# Клавиатура для активации Premium
premium_upsell_kb = InlineKeyboardBuilder()
premium_upsell_kb.button(text="✔ Активировать Premium",
callback_data="toggle_premium_on")
premium_upsell_kb.button(text="⬅ Назад", callback_data="Menu")
premium_upsell_kb.adjust(1)

# Для меню премиум настроек
premium_activate_kb = InlineKeyboardBuilder()
premium_activate_kb.button(text="Активировать Premium",
callback_data="premium_action_activate")
premium_activate_kb.button(text="Назад", callback_data="Menu")
premium_activate_kb.adjust(1)

premium_extend_kb = InlineKeyboardBuilder()
premium_extend_kb.button(text="Продлить Premium",
callback_data="premium_action_extend")
premium_extend_kb.button(text="Назад", callback_data="Menu")
premium_extend_kb.adjust(1)

# Добавьте в Keyboard/keyboard.py

notification_settings_menu_kb = InlineKeyboardBuilder()
notification_settings_menu_kb.button(text="Мотивационные
уведомления", callback_data="notif_setting_motivation")
notification_settings_menu_kb.button(text="Уведомления
статистики", callback_data="notif_setting_stats")
notification_settings_menu_kb.button(text="Уведомления с
рецептами", callback_data="notif_setting_recipes")
notification_settings_menu_kb.button(text="Назад",
callback_data="Menu")
notification_settings_menu_kb.adjust(1)

# В Keyboard/keyboard.py
# В Keyboard/keyboard.py
notification_settings_kb = InlineKeyboardBuilder()

```

```

notification_settings_kb.button(text="🔔 Мотивационные",
callback_data="notif_motivation_menu")
notification_settings_kb.button(text="📊 Статистика",
callback_data="notif_stats_menu")
notification_settings_kb.button(text="🍳 Рецепты",
callback_data="notif_recipes_menu")
notification_settings_kb.button(text="◀ Назад",
callback_data="Menu")
notification_settings_kb.adjust(1)

```

```

motivation_settings_kb = InlineKeyboardBuilder()
motivation_settings_kb.button(text="Вкл/Выкл",
callback_data="toggle_motivation")
motivation_settings_kb.button(text="Изменить интервал",
callback_data="change_motivation_interval")
motivation_settings_kb.button(text="◀ Назад",
callback_data="notification_settings")
motivation_settings_kb.adjust(1)

```

```

stats_settings_kb = InlineKeyboardBuilder()
stats_settings_kb.button(text="Вкл/Выкл",
callback_data="toggle_stats")
stats_settings_kb.button(text="Изменить время",
callback_data="change_stats_time")
stats_settings_kb.button(text="◀ Назад",
callback_data="notification_settings")
stats_settings_kb.adjust(1)

```

```

recipes_settings_kb = InlineKeyboardBuilder()
recipes_settings_kb.button(text="Вкл/Выкл",
callback_data="toggle_recipes")
recipes_settings_kb.button(text="Изменить время",
callback_data="change_recipes_time")
recipes_settings_kb.button(text="◀ Назад",
callback_data="notification_settings")
recipes_settings_kb.adjust(1)

```

```

# В Keyboard/keyboard.py
registration_notifications_kb = InlineKeyboardBuilder()
registration_notifications_kb.button(text="Все уведомления",
callback_data="notif_all")
registration_notifications_kb.button(text="Выключить все",
callback_data="notif_none")
registration_notifications_kb.button(text="Только мотивация",
callback_data="notif_motivation")
registration_notifications_kb.button(text="Только статистика",
callback_data="notif_stats")
registration_notifications_kb.button(text="Только рецепты",
callback_data="notif_recipes")

```

```

registration_notifications_kb.button(text="Мотивация + рецепты",
callback_data="notif_motivation_recipes")
registration_notifications_kb.button(text="Статистика +
рецепты", callback_data="notif_stats_recipes")
registration_notifications_kb.button(text="Мотивация +
статистика", callback_data="notif_motivation_stats")
registration_notifications_kb.adjust(2, 2, 2, 2)

```

```

# В Keyboard/keyboard.py
recipe_time_change_kb = InlineKeyboardBuilder()
recipe_time_change_kb.button(text="Вкл/Выкл",
callback_data="toggle_recipes_notifications")
recipe_time_change_kb.button(text="Завтрак",
callback_data="change_time_breakfast")
recipe_time_change_kb.button(text="Обед",
callback_data="change_time_lunch")
recipe_time_change_kb.button(text="Перекус",
callback_data="change_time_snack")
recipe_time_change_kb.button(text="Ужин",
callback_data="change_time_dinner")
recipe_time_change_kb.button(text="◀ Назад",
callback_data="notification_settings")
recipe_time_change_kb.adjust(1,2)

```

```

back_to_notification_kb = InlineKeyboardBuilder()
back_to_notification_kb.button(text="◀ Назад к уведомлениям",
callback_data="notification_settings")
back_to_notification_kb.adjust(1)

```

```

gender_kb = InlineKeyboardBuilder()
gender_kb.button(text="Мужской", callback_data="gender_male")
gender_kb.button(text="Женский", callback_data="gender_female")
gender_kb.adjust(1)

```

```

gender_change_kb = InlineKeyboardBuilder()
gender_change_kb.button(text="Мужской",
callback_data="gender_male")
gender_change_kb.button(text="Женский",
callback_data="gender_female")
gender_change_kb.button(text="Перейти в меню изменения данных",
callback_data="Change")
gender_change_kb.button(text="Перейти в главное меню",
callback_data="Menu")
gender_change_kb.adjust(2,1,1)

```

```

# Клавиатура управления Premium без кнопки продления (для
подписки "навсегда")
premium_manage_no_renew_kb = InlineKeyboardBuilder()
premium_manage_no_renew_kb.button(text="✗ Отключить подписку",
callback_data="toggle_premium_off")

```

```
premium_manage_no_renew_kb.button(text="⬅ Назад",
callback_data="Menu")
premium_manage_no_renew_kb.adjust(1)
```

```
back_to_recipe_time_kb = InlineKeyboardBuilder()
back_to_recipe_time_kb.button(text="◀ Назад к настройкам",
callback_data="notif_recipes_time_menu")
back_to_recipe_time_kb.button(text="📁 В главное меню",
callback_data="Menu")
back_to_recipe_time_kb.adjust(1)
```

Файл «photo.py»

```
photo_start = 'AgACAgIAAxkBAAMKaD1-
6Z0avivFnsnphsrwwXLlYccAAldZMRsEP0hJaRH011UJh0ABAAMCAAN4AAM2BA'
```

Файл «video.py»

```
easy_1 =
"BAACAgIAAxkBAAMMaD2BLzREaIOJlIAs2Ekuf1D61rIAAhd5AAL4xvBJKh4AAWp
7ILP3NgQ"
easy_2 =
"BAACAgIAAxkBAAMQaD2EPKv_doTuXkLqmcWItMuI2rQAAnZ5AAL4xvBJBj8ymGP
ox0c2BA"
easy_3 = "BAACAgIAAxkBAAMSaD2GrwKBzOFQnKOPzIkKT-
Kb3M4AAsJwAAKwyOhJJR8AAW6lotvGNgQ"
easy_4 =
"BAACAgIAAxkBAAMUaD2IHWOHTTCa_YmBNvJdtYY8FzMAAsJ5AAL4xvBJX0bSwMC
d-uM2BA"
easy_5 =
"BAACAgIAAxkBAAMWaD2JwBosFoSKdh1wZpterD1eMfIAAtJ5AAL4xvBJ6V0Z2wk
cFfe2BA"
```

```
medium_1 =
"BAACAgIAAxkBAAMYaD2MVDjya0nufYKj7TUycYczKFsAAvx5AAL4xvBJGOLh3o5
X3OM2BA"
medium_2 = "BAACAgIAAxkBAAMaaD2OI_XTsuLLJ0cZI2-
Z70tQesYAAhN6AAL4xvBJLNDBrz9HZ782BA"
medium_3 = "BAACAgIAAxkBAAMdaD2Ps4fAUy-
GWXdvEelVhNn6C2sAAi16AAL4xvBJqfbzFPRArrU2BA"
medium_4 =
"BAACAgIAAxkBAAMfaD2Rgujvgkv7qf99BwEOSiMuEJ4AAkN6AAL4xvBJayQ59SZ
TDwk2BA"
medium_5 = "BAACAgIAAxkBAAMhaD2UDupytUU-
LghNSINQv1Zxk84AAmx6AAL4xvBJB5twIgeyVTY2BA"
medium_6 =
"BAACAgIAAxkBAAMjaD2VrbkcJQeMaQntYMHF4FoEwekAAoJ6AAL4xvBJMjfkOjk
EmRg2BA"
```

```
hard_1 = "BAACAgIAAxkBAAMlaD2XdMCG9RctLbhhDM4AASKs97mYAAKeegAC-
MbwsUkgctXQP0X8NgQ"
```

```

hard_2 =
"BAACAgIAAxkBAAMnaD2Z3f40qhbhVayln6jBXBrR0n8AAsV6AAL4xvBJ8X6rQNF
8zro2BA"
hard_3 = "BAACAgIAAxkBAAMpaD2cir-
Bcr5AcAVUuYWs8cejdh8AAvl6AAL4xvBJD7POPSDjvhA2BA"
hard_4 =
"BAACAgIAAxkBAAMraD2ehMRMg9udlTLAgbCR66QDxs4AAhd7AAL4xvBJjX9YOLL
i2nU2BA"

stretch = "BAACAgIAAxkBAAMtaD2gGAABrV1IxoJQwj5Q7T88M1qpAAI2ewAC-
MbwsSes9OF46bIxpNgQ"

```

Файл «main.py»

```

import asyncio
import psycpg2
import random
import math
import os
from dateutil.relativedelta import relativedelta
from datetime import datetime, date, timedelta, time
from typing import Optional
from dotenv import load_dotenv
from aiogram import Bot, Dispatcher, F
from aiogram.exceptions import TelegramBadRequest
from aiogram.filters import CommandStart
from aiogram.types import Message, CallbackQuery,
ReplyKeyboardRemove
from aiogram.fsm.state import State, StatesGroup
from aiogram.fsm.context import FSMContext
from aiogram.filters import Command
from aiogram.utils.keyboard import InlineKeyboardBuilder

import Photo.photo as ph
import Keyboard.keyboard as kb
import Video.video as vd

MIN_BIRTH_DATE = datetime(1925, 1, 1).date()
MAX_AGE = 100
MIN_AGE = 18
# Загружаем переменные окружения из .env
load_dotenv()
# Конфиг БД
DB_CONFIG = {
    'host': os.getenv("DB_HOST"),
    'user': os.getenv("DB_USER"),
    'password': os.getenv("DB_PASSWORD"),
    'database': os.getenv("DB_NAME"),
    'port': os.getenv("DB_PORT"),
}

```

```

# Токен бота
Token = os.getenv("TELEGRAM_TOKEN")
dp = Dispatcher()
bot = Bot(token=Token)

meal_type_mapping = {
    "breakfast": "завтрак",
    "lunch": "обед",
    "dinner": "ужин",
    "snack": "перекус"
}

db_meal_types = {
    "завтрак": "breakfast",
    "обед": "lunch",
    "ужин": "dinner",
    "перекус": "snack"
}

MOTIVATIONAL_QUOTES = [
    "Ты можешь больше, чем думаешь!",
    "Один маленький шаг каждый день - и ты достигнешь цели!",
    "Успех - это сумма небольших усилий, повторяемых изо дня в день.",
    "Не сдавайся! Ты ближе к цели, чем вчера.",
    "Верь в себя, и у тебя всё получится!"
]

last_notification_ids = {
    'stats': {}, # Для статистики
    'motivation': {}, # Для мотивационных
    'breakfast': {}, # Для рецептов
    'lunch': {},
    'snack': {},
    'dinner': {}
}

class RegistrationStates(StatesGroup):
    waiting_for_surname = State()
    waiting_for_name = State()
    waiting_for_patronymic = State()
    waiting_for_gender = State() # Перенесли пол сразу после
ФИО
    waiting_for_phone = State()
    waiting_for_email = State()
    waiting_for_birthday = State()
    waiting_for_height = State()
    waiting_for_weight = State()
    waiting_for_fitness_goal = State()
    waiting_for_premium = State()
    waiting_for_options = State()

```

```

class ChangeDataStates(StatesGroup):
    changing_surname = State()
    changing_name = State()
    changing_patronymic = State()
    changing_phone = State()
    changing_email = State()
    changing_birthday = State()
    changing_gender = State() # Добавляем новое состояние
    changing_height = State()
    changing_weight = State()
    changing_fitness_goal = State()

class GoalsStates(StatesGroup):
    changing_steps_goal = State()
    adding_steps = State() # New state for adding steps
    adding_calories = State() # New state for adding calories
    adding_calories_consumed = State()
    changing_calories_goal = State()
    changing_calories_consumed_goal = State()
    adding_recipe_name = State()
    adding_recipe_goal = State()
    adding_recipe_meal_type = State()
    adding_recipe_calories = State()
    adding_recipe_preparation = State()

class NotificationStates(StatesGroup):
    changing_motivation_interval = State()
    changing_stats_time = State()
    changing_recipes_time = State()

class PremiumStates(StatesGroup):
    choosing_subscription = State() # Для выбора подписки при
    регистрации
    renewal_subscription = State() # Для продления подписки

async def init_user_stats(user_id: int):
    conn = None
    try:
        conn = await get_db_connection()
        with conn.cursor() as cursor:
            # Get user data for goal calculation
            cursor.execute("""
                SELECT height, weight, fitness_goal, birthday,
gender
                FROM users
                WHERE user_id = %s
            """, (user_id,))
            user_data = cursor.fetchone()

            if not user_data:
                raise ValueError("User not found")

```

```

        height, weight, fitness_goal, birthday, gender =
user_data
        age = calculate_age(birthday)

        # Calculate goals with gender consideration
        bmr = calculate_bmr(weight, height, age, gender)
        daily_calories = calculate_daily_calories(bmr, 1.55,
fitness_goal)
        step_goal = calculate_step_goal(age, float(weight),
height, fitness_goal, gender)
        calorie_goal = calculate_calorie_goal(float(weight),
fitness_goal, gender)

        # Get username from users table
        cursor.execute("SELECT nickname FROM users WHERE
user_id = %s", (user_id,))
        nickname = cursor.fetchone()[0] or f"user_{user_id}"

        # Create record in users_stats
        cursor.execute("""
            INSERT INTO users_stats
            (user_id, nickname, steps_goal, calories_goal,
recommended_caloric_intake)
            VALUES (%s, %s, %s, %s, %s)
            ON CONFLICT (user_id) DO UPDATE SET
            nickname = EXCLUDED.nickname,
            steps_goal = EXCLUDED.steps_goal,
            calories_goal = EXCLUDED.calories_goal,
            recommended_caloric_intake =
EXCLUDED.recommended_caloric_intake
            """, (user_id, nickname, step_goal, calorie_goal,
int(daily_calories)))

        conn.commit()
    except Exception as e:
        print(f"Ошибка при инициализации статистики
пользователя: {e}")
        if conn:
            conn.rollback()
        raise
    finally:
        if conn:
            conn.close()

async def init_user_notifications(user_id: int):
    """Initialize notification settings for new user with
default values"""
    conn = None
    try:
        conn = await get_db_connection()
        with conn.cursor() as cursor:
            cursor.execute("""

```

```

        INSERT INTO user_notifications
        (user_id, motivation_enabled,
motivation_interval,
        stats_enabled, stats_time1, stats_time2,
        recipes_enabled, breakfast_time, lunch_time,
        snack_time, dinner_time)
        VALUES (%s, TRUE, 2, TRUE, '14:00', '22:00',
        TRUE, '08:00', '13:00', '16:00',
'19:00')

        ON CONFLICT (user_id) DO NOTHING
        """ , (user_id,))
        conn.commit()
    except Exception as e:
        print(f"Ошибка при инициализации уведомлений
пользователя: {e}")
        if conn:
            conn.rollback()
    finally:
        if conn:
            conn.close()

async def get_db_connection():
    # Убираем await, так как psycopg2.connect не асинхронный
    return psycopg2.connect(**DB_CONFIG)

async def user_exists(user_id: int) -> bool:
    conn = None
    try:
        conn = await get_db_connection()
        with conn.cursor() as cursor:
            cursor.execute("SELECT 1 FROM users WHERE user_id =
%s", (user_id,))
            return cursor.fetchone() is not None
    finally:
        if conn:
            conn.close()

async def get_user_info(user_id: int) -> dict:
    conn = None
    try:
        conn = await get_db_connection()
        with conn.cursor() as cursor:
            cursor.execute("""
                SELECT u.surname, u.name, u.patronymic,
u.gender, u.phone, u.email, u.birthday,
                u.height, u.weight, u.fitness_goal,
u.premium, u.registration_date,
                us.steps, us.steps_goal,
us.calories_burned, us.calories_consumed,
                us.calories_goal,
us.recommended_caloric_intake,
            """

```

```

        us.notifications_enabled,
us.motivation_enabled, us.notification_interval,
        us.last_notification
    FROM users u
    LEFT JOIN users_stats us ON u.user_id =
us.user_id
    WHERE u.user_id = %s
    """ , (user_id,))

result = cursor.fetchone()
if result:
    return {
        'surname': result[0],
        'name': result[1],
        'patronymic': result[2],
        'gender': result[3],
        'phone': result[4],
        'email': result[5],
        'birthday': result[6],
        'height': result[7],
        'weight': result[8],
        'fitness_goal': result[9],
        'premium': result[10],
        'registration_date': result[11],
        'steps': result[12],
        'steps_goal': result[13],
        'calories_burned': result[14],
        'calories_consumed': result[15],
        'calories_goal': result[16],
        'recommended_caloric_intake': result[17],
        'notifications_enabled': result[18],
        'motivation_enabled': result[19],
        'notification_interval': result[20],
        'last_notification': result[21]
    }
    return None
finally:
    if conn:
        conn.close()

async def create_user_history_table(user_id: int):
    conn = None
    try:
        conn = await get_db_connection()
        with conn.cursor() as cursor:
            cursor.execute(f"""
                CREATE TABLE IF NOT EXISTS
user_{user_id}_history (
                    date DATE PRIMARY KEY,
                    steps INTEGER DEFAULT 0,
                    calories_burned INTEGER DEFAULT 0,
                    calories_consumed INTEGER DEFAULT 0

```

```

        )
        """)
        conn.commit()
    except Exception as e:
        print(f"Ошибка при создании таблицы истории для
пользователя {user_id}: {e}")
        if conn:
            conn.rollback()
    finally:
        if conn:
            conn.close()

async def get_user_history(user_id: int, days: int = 7) -> list:
    conn = None
    try:
        conn = await get_db_connection()
        with conn.cursor() as cursor:
            cursor.execute(f"""
                SELECT date, steps, calories_burned,
calories_consumed
                FROM user_{user_id}_history
                ORDER BY date DESC
                LIMIT %s
            """, (days,))
            return cursor.fetchall()
    except Exception as e:
        print(f"Ошибка при получении истории пользователя
{user_id}: {e}")
        return []
    finally:
        if conn:
            conn.close()

async def save_user_history(user_id: int, steps: int,
calories_burned: int, calories_consumed: int, target_date:
date):
    conn = None
    try:
        conn = await get_db_connection()
        with conn.cursor() as cursor:
            cursor.execute(f"""
                INSERT INTO user_{user_id}_history (date, steps,
calories_burned, calories_consumed)
                VALUES (%s, %s, %s, %s)
                ON CONFLICT (date) DO UPDATE SET
                    steps = EXCLUDED.steps,
                    calories_burned = EXCLUDED.calories_burned,
                    calories_consumed =
EXCLUDED.calories_consumed
            """, (target_date, steps, calories_burned,
calories_consumed))

```

```

        conn.commit()
        return True
    except Exception as e:
        print(f"Ошибка при сохранении истории для пользователя
{user_id}: {e}")
        if conn:
            conn.rollback()
        return False
    finally:
        if conn:
            conn.close()

async def reset_daily_stats():
    conn = None
    try:
        conn = await get_db_connection()
        with conn.cursor() as cursor:
            # Получаем пользователей с ненулевой статистикой
            cursor.execute("""
                SELECT user_id, steps, calories_burned,
calories_consumed
                FROM users_stats
                WHERE steps > 0 OR calories_burned > 0 OR
calories_consumed > 0
            """)
            users = cursor.fetchall()

            yesterday = date.today() - timedelta(days=1) #
Используем вчерашнюю дату

            for user_id, steps, calories_burned,
calories_consumed in users:
                # Сохраняем историю за вчера
                await save_user_history(user_id, steps,
calories_burned, calories_consumed, yesterday)

                # Сбрасываем статистику
                cursor.execute("""
                    UPDATE users_stats
                    SET steps = 0, calories_burned = 0,
calories_consumed = 0
                    WHERE user_id = %s
                """, (user_id,))

            conn.commit()
            print(f"Ежедневный сброс выполнен для {len(users)}
пользователей")

            # Отправляем уведомления
            for user_id, steps, calories_burned,
calories_consumed in users:
                user_info = await get_user_info(user_id)

```

```

        if user_info and
user_info['notifications_enabled']:
            notification_text = (
                f"📊 Ежедневная статистика:\n"
                f"• Шагов: {steps} (цель:
{user_info['steps_goal']})\n"
                f"• Сожжено калорий:
{calories_burned}\n"
                f"• Потреблено калорий:
{calories_consumed}\n"
                f"• Баланс: {calories_consumed -
calories_burned}\n\n"
                f"Статистика сохранена в истории!"
            )

            if user_info['motivation_enabled']:
                quote =
random.choice(MOTIVATIONAL_QUOTES)
                notification_text += f"\n\n{quote}"

            try:
                await bot.send_message(user_id,
notification_text)
            except Exception as e:
                print(f"Не удалось отправить уведомление
пользователю {user_id}: {e}")

    except Exception as e:
        print(f"Ошибка при ежедневном сбросе: {e}")
        if conn:
            conn.rollback()
    finally:
        if conn:
            conn.close()

async def save_user_to_db(user_data: dict):
    """Сохраняет пользователя в базу данных"""
    required_fields = ['user_id', 'surname', 'name', 'gender',
'phone', 'birthday', 'height', 'weight', 'fitness_goal']
    # Проверяем наличие обязательных полей
    for field in required_fields:
        if field not in user_data:
            print(f"Отсутствует обязательное поле: {field}")
            return False
    conn = None
    try:
        conn = await get_db_connection()
        with conn.cursor() as cursor:
            sql = """
                INSERT INTO users

```

```

        (user_id, nickname, surname, name, patronymic,
gender, phone, email,
        birthday, height, weight, fitness_goal,
premium, registration_date)
VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s,
%s, %s, %s, %s)
ON CONFLICT (user_id) DO UPDATE SET
    nickname = EXCLUDED.nickname,
    surname = EXCLUDED.surname,
    name = EXCLUDED.name,
    patronymic = EXCLUDED.patronymic,
    gender = EXCLUDED.gender,
    phone = EXCLUDED.phone,
    email = EXCLUDED.email,
    birthday = EXCLUDED.birthday,
    height = EXCLUDED.height,
    weight = EXCLUDED.weight,
    fitness_goal = EXCLUDED.fitness_goal,
    premium = EXCLUDED.premium
"""
cursor.execute(sql, (
    user_data['user_id'],
    user_data.get('nickname'),
    user_data.get('surname'),
    user_data.get('name'),
    user_data.get('patronymic'),
    user_data.get('gender'), # Добавлено сохранение
    user_data.get('phone'),
    user_data.get('email'),
    user_data.get('birthday'),
    user_data.get('height'),
    user_data.get('weight'),
    user_data.get('fitness_goal'),
    user_data.get('premium', False),
    datetime.now() # registration_date
))
conn.commit()
return True
except Exception as e:
    print(f"ОШИБКА сохранения пользователя: {str(e)}")
    if conn:
        conn.rollback()
    return False
finally:
    if conn:
        conn.close()

async def update_user_stats(user_id: int, updates: dict):
    """Update user stats and sync notification settings"""
    conn = None
    try:

```

пола

```

        conn = await get_db_connection()
        with conn.cursor() as cursor:
            # Обновляем users_stats
            stats_updates = {k: v for k, v in updates.items() if
not k.endswith('_enabled')}
            if stats_updates:
                set_clause = ", ".join([f"{field} = %s" for
field in stats_updates.keys()])
                values = list(stats_updates.values()) +
[user_id]

                cursor.execute(f"""
                    UPDATE users_stats
                    SET {set_clause}
                    WHERE user_id = %s
                    """, values)

            # Обновляем user_notifications если есть
соответствующие поля
            notif_updates = {k: v for k, v in updates.items() if
k.endswith('_enabled')}
            if notif_updates:
                set_clause = ", ".join([f"{field} = %s" for
field in notif_updates.keys()])
                values = list(notif_updates.values()) +
[user_id]

                cursor.execute(f"""
                    UPDATE user_notifications
                    SET {set_clause}
                    WHERE user_id = %s
                    """, values)

        conn.commit()
        return True
    except Exception as e:
        print(f"Ошибка при обновлении статистики: {e}")
        if conn:
            conn.rollback()
        return False
    finally:
        if conn:
            conn.close()

async def update_user_field(user_id: int, field: str, value):
    conn = None
    try:
        conn = await get_db_connection()
        with conn.cursor() as cursor:
            # Для полей с подчеркиванием используем кавычки
            column = f'"{field}"' if field in ['fitness_goal']
    else field

        # Проверяем текущее значение

```

```

        cursor.execute(f"SELECT {column} FROM users WHERE
user_id = %s", (user_id,))
        result = cursor.fetchone()
        current_value = result[0] if result else None

        if str(current_value) == str(value):
            return False, "Значение не изменилось",
kb.back_to_change_kb.as_markup()

        # Обновляем поле
        cursor.execute(f"""
            UPDATE users
            SET {column} = %s
            WHERE user_id = %s
        """, (value, user_id))

        conn.commit()
        return True, "Данные успешно обновлены", None
except Exception as e:
    print(f"Ошибка при обновлении {field}: {e}")
    if conn:
        conn.rollback()
    return False, f"Ошибка при обновлении: {e}", None
finally:
    if conn:
        conn.close()

def is_valid_email(email: str) -> bool:
    if not email:
        return True
    return '@' in email and '.' in email.split('@')[-1] and
len(email.split('@')[0]) > 0

async def is_email_unique(email: str, current_user_id: int =
None) -> bool:
    if not email:
        return True

    conn = None
    try:
        conn = await get_db_connection()
        with conn.cursor() as cursor:
            if current_user_id:
                cursor.execute("""
                    SELECT 1 FROM users
                    WHERE email = %s AND user_id != %s AND email
IS NOT NULL
                """, (email, current_user_id))
            else:
                cursor.execute("""
                    SELECT 1 FROM users
                    WHERE email = %s AND email IS NOT NULL

```

```

        """", (email,))

        return cursor.fetchone() is None
    except Exception as e:
        print(f"Ошибка при проверке уникальности email: {e}")
        return False
    finally:
        if conn:
            conn.close()

async def is_phone_unique(phone: str, current_user_id: int =
None) -> bool:
    conn = None
    try:
        conn = await get_db_connection()
        with conn.cursor() as cursor:
            if current_user_id:
                cursor.execute("""
                    SELECT 1 FROM users
                    WHERE phone = %s AND user_id != %s
                    """, (phone, current_user_id))
            else:
                cursor.execute("""
                    SELECT 1 FROM users
                    WHERE phone = %s
                    """, (phone,))

            return cursor.fetchone() is None
    except Exception as e:
        print(f"Ошибка при проверке уникальности телефона: {e}")
        return False
    finally:
        if conn:
            conn.close()

async def send_notification(user_id: int):
    """Отправляет уведомление пользователю с текущей
    статистикой"""
    user_info = await get_user_info(user_id)
    if not user_info:
        return

    if user_info['notifications_enabled']:
        balance = user_info['calories_consumed'] -
user_info['calories_burned']
        notification_text = (
            f"■ Текущая статистика:\n"
            f"• Шагов:
{user_info['steps']}/{user_info['steps_goal']}\n"
            f"• Сожжено калорий:
{user_info['calories_burned']}\n"

```

```

        f"• Потреблено калорий:
{user_info['calories_consumed']}\n"
        f"• Баланс: {balance}\n"
        f"• Рекомендуемое потребление:
{user_info.get('recommended_caloric_intake', 'не рассчитано')}
ккал"
    )

    if user_info['motivation_enabled']:
        quote = random.choice(MOTIVATIONAL_QUOTES)
        notification_text += f"\n\n💪 {quote}"

    try:
        await bot.send_message(user_id, notification_text)
    except Exception as e:
        print(f"Не удалось отправить уведомление
пользователю {user_id}: {e}")

async def periodic_notifications():
    while True:
        try:
            conn = await get_db_connection()
            with conn.cursor() as cursor:
                # Получаем всех пользователей с включенными
уведомлениями
                cursor.execute("""
                    SELECT user_id FROM users_stats
                    WHERE notifications_enabled = TRUE
                """)
                users = cursor.fetchall()

                for (user_id,) in users:
                    await send_notification(user_id)

        except Exception as e:
            print(f"Ошибка при отправке периодических
уведомлений: {e}")

            # Интервал между уведомлениями (например, каждые 2 часа)
            await asyncio.sleep(2 * 60 * 60)

def is_valid_phone(phone: str) -> bool:
    if not phone:
        return False
    if phone[0] not in ('+', '8'):
        return False

    # Проверяем что остальные символы - цифры
    digits = phone[1:] if phone[0] == '+' else phone
    if not digits.isdigit():
        return False

```

```

# Проверяем длину в зависимости от первого символа
if phone[0] == '+' and len(digits) < 11: # Изменено с <= на
<
    return False
if phone[0] == '8' and len(digits) < 11: # Изменено с <= на
< (8 + 10 цифр)
    return False

return True

def format_name(name: str) -> str:
    if not name:
        return name
    return name[0].upper() + name[1:].lower() if len(name) > 0
else name

def is_valid_name(name: str) -> bool:
    if not name:
        return False
    # Проверяем, что все символы - буквы, апострофы или дефисы
    # И что количество слов не превышает 2
    words = name.split()
    if len(words) > 2:
        return False
    return all(c.isalpha() or c in ("'", "-") for c in
name.replace(" ", ""))

def is_valid_birthday(birth_date: date) -> tuple[bool, str]:
    today = date.today()

    # Проверка что дата не в будущем
    if birth_date > today:
        return False, "Дата рождения не может быть в будущем"

    # Проверка минимальной даты (1925 год)
    if birth_date < MIN_BIRTH_DATE:
        return False, f"Дата рождения не может быть раньше
{MIN_BIRTH_DATE.strftime('%d.%m.%Y')}"

    # Проверка минимального возраста (18 лет)
    age = today.year - birth_date.year - ((today.month,
today.day) < (birth_date.month, birth_date.day))
    if age < MIN_AGE:
        return False, f"Регистрация доступна только лицам старше
{MIN_AGE} лет"

    # Проверка максимального возраста
    if age > MAX_AGE:
        return False, f"Возраст не может превышать {MAX_AGE}
лет"

```

```

    return True, ""

def calculate_age(birth_date: date) -> int:
    today = date.today()
    return today.year - birth_date.year - ((today.month,
today.day) < (birth_date.month, birth_date.day))

def calculate_bmr(weight: float, height: int, age: int, gender:
str) -> float:
    """Calculate Basal Metabolic Rate using Mifflin-St Jeor
Equation"""
    try:
        # Явное преобразование веса к float
        weight_float = float(weight)
        if gender == 'male':
            return float(10 * weight_float + 6.25 * height - 5 *
age + 5)
        else: # female
            return float(10 * weight_float + 6.25 * height - 5 *
age - 161)
    except Exception as e:
        print(f"Ошибка в calculate_bmr: {e}, weight={weight},
type={type(weight)}")
        raise

def calculate_daily_calories(bmr: float, activity_level: float,
goal: str) -> int:
    """Calculate daily caloric needs based on goal"""
    maintenance = bmr * activity_level

    if goal == 'lose':
        return int(maintenance - 500) # 500 calorie deficit for
weight loss
    elif goal == 'gain':
        return int(maintenance + 300) # 300 calorie surplus for
muscle gain
    else: # healthy_lifestyle
        return int(maintenance)

def calculate_step_goal(age: int, weight: float, height: int,
goal: str, gender: str) -> int:
    """Calculate recommended daily step goal with gender
consideration"""
    base_steps = 8000

    # Gender adjustments
    if gender == 'male':
        base_steps += 500 # Мужчины обычно имеют более высокие
цели
    else:
        base_steps += 300 # Женщины обычно имеют немного
меньшие цели

```

```

    # Adjust based on age
    if age < 30:
        base_steps += 2000
    elif age > 50:
        base_steps -= 2000

    # Adjust based on goal
    if goal == 'lose':
        base_steps += 2000
    elif goal == 'gain':
        base_steps += 1000

    return base_steps

def calculate_calorie_goal(weight: float, goal: str, gender:
str) -> int:
    """Calculate recommended calorie burn goal with gender
    consideration"""
    if gender == 'male':
        base_calories = weight * 8 # Более высокий базовый
уровень для мужчин
    else:
        base_calories = weight * 7 # Немного ниже для женщин

    if goal == 'lose':
        return int(base_calories + 500) # More aggressive
calorie burn for weight loss
    elif goal == 'gain':
        return int(base_calories + 300) # Moderate calorie burn
for muscle gain
    else: # healthy_lifestyle
        return int(base_calories + 400)

def format_fitness_goal(goal: Optional[str]) -> str:
    if not goal:
        return "не указана"

    goal_mapping = {
        'lose': "Похудеть",
        'gain': "Набрать мышцы",
        'healthy': "Вести здоровый образ жизни"
    }
    return goal_mapping.get(goal, goal)

async def get_recipes_by_goal_and_meal(fitness_goal: str,
meal_type: str) -> list:
    """Получить рецепты по фитнес-цели и типу приема пищи"""
    conn = None
    try:
        conn = await get_db_connection()
        with conn.cursor() as cursor:

```

```

        cursor.execute("""
            SELECT dish_name, calories, preparation
            FROM recipes
            WHERE fitness_goal = %s AND meal_type = %s
            ORDER BY RANDOM()
            LIMIT 10
        """, (fitness_goal, meal_type))
        return cursor.fetchall()
    except Exception as e:
        print(f"Ошибка при получении рецептов: {e}")
        return []
    finally:
        if conn:
            conn.close()

async def get_random_recipe_by_goal(fitness_goal: str) ->
Optional[dict]:
    """Получить случайный рецепт для цели"""
    conn = None
    try:
        conn = await get_db_connection()
        with conn.cursor() as cursor:
            cursor.execute("""
                SELECT dish_name, meal_type, calories,
preparation
                FROM recipes
                WHERE fitness_goal = %s
                ORDER BY RANDOM()
                LIMIT 1
            """, (fitness_goal,))
            result = cursor.fetchone()
            if result:
                return {
                    'dish_name': result[0],
                    'meal_type': result[1],
                    'calories': result[2],
                    'preparation': result[3]
                }
            return None
    except Exception as e:
        print(f"Ошибка при получении рецепта: {e}")
        return None
    finally:
        if conn:
            conn.close()

async def periodic_notifications():
    while True:
        try:
            conn = await get_db_connection()
            with conn.cursor() as cursor:

```

```

        # Получаем всех пользователей с включенными
уведомлениями и их интервалами
        cursor.execute("""
            SELECT user_id, notification_interval FROM
users_stats
            WHERE notifications_enabled = TRUE
        """)
        users = cursor.fetchall()

        for user_id, interval in users:
            # Проверяем, нужно ли отправлять уведомление
сейчас
            last_notification = await
get_last_notification_time(user_id)
            now = datetime.now()

            if last_notification is None or (now -
last_notification).total_seconds() >= interval * 3600:
                await send_notification(user_id)
                await
update_last_notification_time(user_id, now)

        except Exception as e:
            print(f"Ошибка при отправке периодических
уведомлений: {e}")

        # Проверяем каждые 5 минут вместо фиксированного
интервала
        await asyncio.sleep(60)

async def get_last_notification_time(user_id: int) ->
Optional[datetime]:
    conn = None
    try:
        conn = await get_db_connection()
        with conn.cursor() as cursor:
            cursor.execute("""
                SELECT last_notification FROM users_stats
                WHERE user_id = %s
            """, (user_id,))
            result = cursor.fetchone()
            return result[0] if result else None
    except Exception as e:
        print(f"Ошибка при получении времени последнего
уведомления: {e}")
        return None
    finally:
        if conn:
            conn.close()

async def update_last_notification_time(user_id: int, time:
datetime):

```

```

conn = None
try:
    conn = await get_db_connection()
    with conn.cursor() as cursor:
        cursor.execute("""
            UPDATE users_stats
            SET last_notification = %s
            WHERE user_id = %s
            """, (time, user_id))
        conn.commit()
except Exception as e:
    print(f"Ошибка при обновлении времени уведомления: {e}")
    if conn:
        conn.rollback()
finally:
    if conn:
        conn.close()

async def recalculate_goals(user_id: int):
    """Пересчитывает цели по шагам, калориям и потреблению при
    изменении данных"""
    try:
        user_info = await get_user_info(user_id)
        if not user_info or not user_info.get('height') or not
        user_info.get('weight') or not user_info.get('fitness_goal') or
        not user_info.get('gender'):
            print(f"Недостаточно данных для пересчета целей:
{user_info}")
            return

        age = calculate_age(user_info['birthday'])
        bmr = calculate_bmr(
            weight=user_info['weight'],
            height=user_info['height'],
            age=age,
            gender=user_info['gender']
        )

        daily_calories = calculate_daily_calories(bmr, 1.55,
        user_info['fitness_goal'])
        step_goal = calculate_step_goal(age,
        float(user_info['weight']), user_info['height'],
        user_info['fitness_goal'], user_info['gender'])
        calorie_goal =
        calculate_calorie_goal(float(user_info['weight']),
        user_info['fitness_goal'], user_info['gender'])

        await update_user_stats(user_id, {
            'steps_goal': step_goal,
            'calories_goal': calorie_goal,
            'recommended_caloric_intake': int(daily_calories)
        })

```

```

except Exception as e:
    print(f"Ошибка в recalculate_goals: {e}")

async def add_steps_with_calories(user_id: int, steps_to_add:
int):
    # Рассчитываем калории и округляем в большую сторону
    calories_to_add = math.ceil(steps_to_add * 450 / 10000) #
Используем math.ceil для округления вверх

    user_info = await get_user_info(user_id)
    if user_info:
        new_steps = user_info['steps'] + steps_to_add
        new_calories = user_info['calories_burned'] +
calories_to_add

        success = await update_user_stats(user_id, {
            'steps': new_steps,
            'calories_burned': new_calories
        })

    return success, steps_to_add, calories_to_add
return False, 0, 0

@dp.message(CommandStart())
async def start(message: Message, state: FSMContext):
    user_id = message.from_user.id
    if await user_exists(user_id):
        user_info = await get_user_info(user_id)
        premium_status = "✔ Premium" if user_info['premium']
else "✗ Basic"
        await message.answer(
            text=f'Привет!\nРады снова вас видеть!\nСтатус:
{premium_status}',

reply_markup=kb.start_kb.as_markup(resize_keyboard=True))
    else:
        await message.answer(
            text=f'Привет!\nЭто бот для отслеживания здоровья и
фитнеса\nПройдите регистрацию',

reply_markup=kb.registration_kb.as_markup(resize_keyboard=True))

@dp.callback_query(F.data == "register")
async def start_registration(call: CallbackQuery, state:
FSMContext):
    await call.message.delete()
    await call.message.answer("Давайте начнем
регистрацию.\nВведите вашу фамилию (не более 90 символов):")
    await
state.set_state(RegistrationStates.waiting_for_surname)

```

```

@dp.message(RegistrationStates.waiting_for_surname)
async def process_surname(message: Message, state: FSMContext):
    surname = message.text.strip()

    # Проверка количества слов
    if len(surname.split()) > 2:
        await message.answer("Фамилия должна содержать не более
90 символов.\nПопробуйте еще раз:")
        return

    if not is_valid_name(surname):
        await message.answer("Фамилия должна содержать только
буквы.\nПопробуйте еще раз:")
        return

    formatted_surname = format_name(surname)
    await state.update_data(surname=formatted_surname)
    await message.answer(f"Фамилия сохранена:
{formatted_surname}.\nТеперь введите ваше имя (не более 90
символов):")
    await state.set_state(RegistrationStates.waiting_for_name)

@dp.message(RegistrationStates.waiting_for_name)
async def process_name(message: Message, state: FSMContext):
    name = message.text.strip()

    # Проверка количества слов
    if len(name.split()) > 2:
        await message.answer("Имя должно содержать не более 90
символов.\nПопробуйте еще раз:")
        return

    if not is_valid_name(name):
        await message.answer("Имя должно содержать только
буквы.\nПопробуйте еще раз:")
        return

    formatted_name = format_name(name)
    await state.update_data(name=formatted_name)
    await message.answer(
        f"Имя сохранено: {formatted_name}.\nТеперь введите ваше
отчество (не более 90 символов) или нажмите кнопку 'Отчество не
имеется':",
        reply_markup=kb.Rskip_patronymic_kb.as_markup()
    )
    await
state.set_state(RegistrationStates.waiting_for_patronymic)

@dp.message(RegistrationStates.waiting_for_patronymic)
async def process_patronymic(message: Message, state:
FSMContext):
    if message.text: # Если пользователь ввел текст

```

```

patronymic = message.text.strip()

if len(patronymic.split()) > 2:
    await message.answer(
        "Отчество должно содержать не более 2
слов.\nПопробуйте еще раз или нажмите кнопку 'Пропустить':",
        reply_markup=kb.Rskip_patronymic_kb.as_markup()
    )
    return

if not is_valid_name(patronymic):
    await message.answer(
        "Отчество должно содержать только
буквы.\nПопробуйте еще раз или нажмите кнопку 'Пропустить':",
        reply_markup=kb.Rskip_patronymic_kb.as_markup()
    )
    return

formatted_patronymic = format_name(patronymic)
await state.update_data(patronymic=formatted_patronymic)

# Добавляем подтверждение сохранения отчества
await message.answer(
    f"Отчество сохранено: {formatted_patronymic}\n"
    "Укажите ваш пол:",
    reply_markup=kb.gender_kb.as_markup()
)
await
state.set_state(RegistrationStates.waiting_for_gender)
else:
    await message.answer(
        "Пожалуйста, введите отчество (не более 2 слов) или
нажмите кнопку 'Пропустить'",
        reply_markup=kb.Rskip_patronymic_kb.as_markup()
    )

@dp.callback_query(F.data == "skip_patronymic",
RegistrationStates.waiting_for_patronymic)
async def skip_patronymic(call: CallbackQuery, state:
FSMContext):
    await call.message.delete()
    await state.update_data(patronymic=None)
    # Добавляем сообщение о том, что отчество не указано
    await call.message.answer(
        "Отчество: не указано\n"
        "Укажите ваш пол:",
        reply_markup=kb.gender_kb.as_markup()
    )
    await state.set_state(RegistrationStates.waiting_for_gender)

@dp.callback_query(RegistrationStates.waiting_for_gender,
F.data.startswith("gender_"))

```

```

async def process_gender(call: CallbackQuery, state:
FSMContext):
    gender = call.data.split("_")[1] # male или female
    gender_display = "Мужской" if gender == 'male' else
"Женский"

    await state.update_data(gender=gender)
    await call.message.delete()

    # Добавляем подтверждение сохранения пола
    await call.message.answer(
        f"Пол сохранён: {gender_display}\n"
        "Введите ваш номер телефона:",
        reply_markup=ReplyKeyboardRemove()
    )
    await state.set_state(RegistrationStates.waiting_for_phone)

@dp.message(RegistrationStates.waiting_for_phone)
async def process_phone(message: Message, state: FSMContext):
    phone = message.text
    await state.update_data(phone=phone)
    await message.answer(
        f"Телефон сохранён: {phone}\n"
        "Теперь вы можете ввести ваш email (необязательно):",
        reply_markup=kb.skip_email_kb.as_markup()
    )
    await state.set_state(RegistrationStates.waiting_for_email)

@dp.message(RegistrationStates.waiting_for_email)
async def process_email(message: Message, state: FSMContext):
    email = message.text.strip() if message.text else ""

    # Обработка пропуска email
    if email.lower() in ["пропустить", "skip"]:
        email = ""

    if email: # Если email введен (не пустой)
        if not is_valid_email(email):
            await message.answer(
                "Некорректный email. Email должен содержать @ и
домен.\nПопробуйте еще раз или нажмите 'Пропустить':",
                reply_markup=kb.skip_email_kb.as_markup()
            )
            return

        if not await is_email_unique(email):
            await message.answer(
                "Этот email уже используется другим
пользователем.\nПожалуйста, введите другой email или нажмите
'Пропустить':",
                reply_markup=kb.skip_email_kb.as_markup()
            )

```

```

        return

    # Сохраняем email (может быть None если пропустили)
    await state.update_data(email=email if email else None)

    # Формируем сообщение подтверждения
    email_display = email if email else "не указан"
    await message.answer(
        f"Электронная почта сохранена: {email_display}\n"
        "Введите вашу дату рождения в формате ДД.ММ.ГГГГ:",
        reply_markup=ReplyKeyboardRemove()
    )
    await
    state.set_state(RegistrationStates.waiting_for_birthday)

@dp.callback_query(F.data == 'Skip_E')
async def skip_email(call: CallbackQuery, state: FSMContext):
    await call.message.delete()
    await state.update_data(email=None)
    # Сообщение с подтверждением пропуска email
    await call.message.answer(
        "Электронная почта: не указана\n"
        "Введите вашу дату рождения в формате ДД.ММ.ГГГГ:",
        reply_markup=ReplyKeyboardRemove()
    )
    await
    state.set_state(RegistrationStates.waiting_for_birthday)

@dp.message(RegistrationStates.waiting_for_birthday)
async def process_birthday(message: Message, state: FSMContext):
    try:
        birth_date = datetime.strptime(message.text,
"%d.%m.%Y").date()
        is_valid, error_msg = is_valid_birthday(birth_date)

        if not is_valid:
            today = date.today()
            min_date = date(today.year - MIN_AGE, today.month,
today.day)
            await message.answer(
                f"{error_msg}\n"
                f"Пожалуйста, введите дату между
{MIN_BIRTH_DATE.strftime('%d.%m.%Y')} "
                f"и {min_date.strftime('%d.%m.%Y')}:"
            )
            return

        await state.update_data(birthday=birth_date)
        age = calculate_age(birth_date)

        await message.answer(

```

```

        f"Дата рождения сохранена:
        {birth_date.strftime('%d.%m.%Y')} (Возраст: {age} лет)\n"
        "Введите ваш рост в сантиметрах:"
    )
    await
state.set_state(RegistrationStates.waiting_for_height)

    except ValueError:
        await message.answer(
            "Неверный формат даты.\nПожалуйста, введите дату в
формате ДД.ММ.ГГГГ"
        )

@dp.message(RegistrationStates.waiting_for_height)
async def process_height(message: Message, state: FSMContext):
    try:
        height = int(message.text)
        if height < 100 or height > 250:
            await message.answer("Пожалуйста, введите корректный
рост (100-250 см):")
            return

        await state.update_data(height=height)
        await message.answer(
            f"Рост сохранён: {height} см\n"
            "Введите ваш вес в килограммах:"
        )
        await
state.set_state(RegistrationStates.waiting_for_weight)

    except ValueError:
        await message.answer("Пожалуйста, введите число для
роста:")

@dp.message(RegistrationStates.waiting_for_weight)
async def process_weight(message: Message, state: FSMContext):
    try:
        weight = float(message.text)
        if weight < 30 or weight > 200:
            await message.answer("Пожалуйста, введите корректный
вес (30-200 кг):")
            return

        await state.update_data(weight=weight)
        await message.answer(
            f"Вес сохранён: {weight} кг\n"
            "Выберите вашу фитнес-цель:",

reply_markup=kb.registration_fitness_goal_kb.as_markup()
        )
        await
state.set_state(RegistrationStates.waiting_for_fitness_goal)

```

```

except ValueError:
    await message.answer("Пожалуйста, введите число для веса
(можно с десятичной точкой):")

@dp.callback_query(RegistrationStates.waiting_for_fitness_goal,
F.data.startswith("goal_"))
async def process_fitness_goal(call: CallbackQuery, state:
FSMContext):
    await call.message.delete()
    goal = call.data.split("_")[1]
    goal_display = format_fitness_goal(goal)

    await state.update_data(fitness_goal=goal)

    # Получаем все сохраненные данные для вывода сводки
    user_data = await state.get_data()
    height = user_data.get('height')
    weight = user_data.get('weight')

    await call.message.answer(
        f"Фитнес-цель сохранена: {goal_display}\n\n"
        f"Сводка данных:\n"
        f"► Рост: {height} см\n"
        f"► Вес: {weight} кг\n"
        f"► Цель: {goal_display}\n\n"
        "Продолжаем настройку уведомлений...",
        reply_markup=ReplyKeyboardRemove()
    )

    # Переходим к настройке уведомлений
    await call.message.answer(
        "🔔 Настройте уведомления:\n\n"
        "• Мотивационные - вдохновляющие цитаты\n"
        "• Статистика - отчет о вашей активности\n"
        "• Рецепты - идеи для приемов пищи\n\n"
        "Выберите предпочитаемый вариант:",

reply_markup=kb.registration_notifications_kb.as_markup()
    )
    await
state.set_state(RegistrationStates.waiting_for_options)

@dp.callback_query(RegistrationStates.waiting_for_options,
F.data.startswith("notif_"))
async def process_notification_choice(call: CallbackQuery,
state: FSMContext):
    await call.message.delete()
    choice = call.data

    # Проверяем премиум статус

```

```

has_premium = await check_premium_status(call.from_user.id)

# Определяем настройки для каждого варианта
settings = {
    "notif_all": {
        "motivation_enabled": True,
        "stats_enabled": True,
        "recipes_enabled": True if has_premium else False
    },
    "notif_motivation": {
        "motivation_enabled": True,
        "stats_enabled": False,
        "recipes_enabled": False
    },
    "notif_stats": {
        "motivation_enabled": False,
        "stats_enabled": True,
        "recipes_enabled": False
    },
    "notif_recipes": {
        "motivation_enabled": False,
        "stats_enabled": False,
        "recipes_enabled": True if has_premium else False
    },
    "notif_motivation_recipes": {
        "motivation_enabled": True,
        "stats_enabled": False,
        "recipes_enabled": True if has_premium else False
    },
    "notif_stats_recipes": {
        "motivation_enabled": False,
        "stats_enabled": True,
        "recipes_enabled": True if has_premium else False
    },
    "notif_motivation_stats": {
        "motivation_enabled": True,
        "stats_enabled": True,
        "recipes_enabled": False
    },
    "notif_none": {
        "motivation_enabled": False,
        "stats_enabled": False,
        "recipes_enabled": False
    }
}

if choice in settings:
    # Сохраняем настройки
    await update_notification_settings(call.from_user.id,
settings[choice])

```

```

        # Если выбраны рецепты, но нет премиума - сообщаем об
этом
        if choice in ["notif_all", "notif_recipes",
"notif_motivation_recipes",
                    "notif_stats_recipes"] and not
has_premium:
            await call.message.answer(
                "❗ Уведомления о рецептах доступны только с
Premium подпиской. "
                "Они были автоматически отключены.",
reply_markup=kb.premium_choice_registration_kb.as_markup()
            )

            await call.message.answer(
                "Настройки уведомлений сохранены!\n"
                "Хотите активировать Premium подписку?",
reply_markup=kb.premium_choice_registration_kb.as_markup()
            )
            await
state.set_state(RegistrationStates.waiting_for_premium)
        else:
            await call.message.answer(
                "Неизвестный вариант. Попробуйте еще раз:",
reply_markup=kb.registration_notifications_kb.as_markup()
            )

@dp.callback_query(F.data.startswith("premium_"),
RegistrationStates.waiting_for_premium)
async def process_premium_choice(call: CallbackQuery, state:
FSMContext):
    await call.message.delete()
    if call.data == "premium_yes":
        await call.message.answer(
            "Выберите срок премиум-подписки:",
            reply_markup=kb.premium_duration_kb.as_markup()
        )
        await
state.set_state(PremiumStates.choosing_subscription)
    else:
        # Завершаем регистрацию без премиума
        user_data = await state.get_data()
        print("Нет подписки\n")
        # Проверяем, были ли выбраны уведомления о рецептах
        notification_settings =
user_data.get('notification_settings', {})
        if notification_settings.get('recipes_enabled', False):
            # Если выбраны рецепты, но нет премиума - отключаем
их

```

```

        print("Нет рецептов")
        notification_settings['recipes_enabled'] = False
        await call.message.answer(
            "❗ Уведомления о рецептах доступны только с
Premium подпиской. "
            "Они были автоматически отключены.",
            reply_markup=kb.start_kb.as_markup()
        )

    await complete_registration(call, state, user_data,
premium=False)

async def complete_registration(call: CallbackQuery, state:
FSMContext, user_data: dict, premium: bool):
    """Завершает регистрацию и показывает сводную информацию"""
    try:
        # Добавляем обязательные поля, если их нет
        user_data['user_id'] = call.from_user.id
        user_data['nickname'] = call.from_user.username or
f"user_{call.from_user.id}"
        user_data['premium'] = premium
        user_data['registration_date'] = datetime.now()

        # Сохраняем пользователя в БД
        if not await save_user_to_db(user_data):
            await call.message.answer("❌ Не удалось сохранить
данные пользователя")
            return

        # Создаем таблицу истории
        await create_user_history_table(call.from_user.id)

        # Инициализируем настройки уведомлений
        await init_user_notifications(call.from_user.id)

        # Инициализируем статистику
        await init_user_stats(call.from_user.id)

        # Формируем сообщение о завершении регистрации
        await call.message.answer(
            f"✅ Регистрация успешно завершена!\n"
            f"Premium подписка: {'активирована' if premium else
'не активирована'}",
            reply_markup=kb.start_kb.as_markup(resize_keyboard=True)
        )

        await state.clear()
    except Exception as e:
        print(f"Ошибка при завершении регистрации: {e}")
        await call.message.answer(

```

```

        "Произошла ошибка при завершении регистрации",
        reply_markup=kb.start_kb.as_markup()
    )

@dp.callback_query(F.data.startswith("premium_duration_"))
async def process_premium_duration(call: CallbackQuery, state:
FSMContext):
    try:
        await call.message.delete()
        duration = call.data.split("_")[2]
        today = date.today()
        user_id = call.from_user.id
        nickname = call.from_user.username or f"user_{user_id}"

        # Удаляем предыдущее состояние
        await state.clear()

        if duration == "lifetime":
            expiry_date = date(2099, 12, 31)
            subscription_type = "lifetime"
        else:
            months = int(duration)
            expiry_date = today + relativedelta(months=+months)
            subscription_type = f"{months} месяцев"

        # Проверяем существующую подписку
        conn = None
        try:
            conn = await get_db_connection()
            with conn.cursor() as cursor:
                cursor.execute("""
                    SELECT expiry_date FROM
premium_subscriptions
                    WHERE user_id = %s
                    """, (user_id,))
                existing_sub = cursor.fetchone()

                if existing_sub and existing_sub[0] > today:
                    # Продление существующей подписки
                    expiry_date = existing_sub[0] +
relativedelta(months=+months)
                    subscription_type = f"Продление +{months}
месяцев"

                message = f"✓ Подписка продлена до
{expiry_date.strftime('%d.%m.%Y')}!"
                else:
                    # Новая подписка
                    message = f"✓ Подписка активирована до
{expiry_date.strftime('%d.%m.%Y')}!"
            except Exception as e:
                print(f"Ошибка при проверке подписки: {e}")

```

```

        message = f"✓ Подписка активирована до
{expiry_date.strftime('%d.%m.%Y')}!"
    finally:
        if conn:
            conn.close()

    # Сохраняем подписку
    if await save_premium_subscription(user_id, nickname,
True, expiry_date, subscription_type):
        await update_user_field(user_id, 'premium', True)
        # Создаем таблицу истории, если её нет
        await create_user_history_table(user_id)
        await call.message.answer(
            f"{message}\n\nТеперь вам доступны все Premium-
функции!",
            reply_markup=kb.start_kb.as_markup()
        )
    else:
        await call.message.answer(
            "✗ Не удалось оформить подписку. Попробуйте
позже.",
            reply_markup=kb.start_kb.as_markup()
        )

except Exception as e:
    print(f"Ошибка в process_premium_duration: {e}")
    await call.message.answer(
        "Произошла ошибка при обработке запроса.",
        reply_markup=kb.start_kb.as_markup()
    )

@dp.callback_query(F.data == "reg_premium_yes",
RegistrationStates.waiting_for_premium)
async def premium_choice_yes_registration(call: CallbackQuery,
state: FSMContext):
    await call.message.delete()
    await call.message.answer(
        "Выберите срок подписки:",

reply_markup=kb.premium_duration_registration_kb.as_markup()
    )
    await state.set_state(PremiumStates.choosing_subscription)

@dp.callback_query(PremiumStates.choosing_subscription,
F.data.startswith("reg_premium_"))
async def process_premium_duration_registration(call:
CallbackQuery, state: FSMContext):
    try:
        duration = call.data.split("_")[2]

        if duration == "back":

```

```

        await call.message.delete()
        await call.message.answer(
            "Хотите активировать Premium подписку?",

reply_markup=kb.premium_choice_registration_kb.as_markup()
        )
        await
state.set_state(RegistrationStates.waiting_for_premium)
        return

# Сначала сохраняем пользователя
user_data = await state.get_data()
user_data['user_id'] = call.from_user.id
user_data['nickname'] = call.from_user.username
user_data['premium'] = True

if not await save_user_to_db(user_data):
    await call.message.answer("❌ Не удалось сохранить
данные пользователя")
    return

# Затем создаем подписку
today = date.today()
if duration == "lifetime":
    expiry_date = date(2099, 12, 31)
    duration_text = "навсегда"
    subscription_type = "lifetime" # В базу сохраняем
"lifetime"
    display_type = "Вечный" # Пользователю показываем
"Вечный"
    premium_kb = kb.start_kb
else:
    months = int(duration)
    expiry_date = today + relativedelta(months=+months)
    duration_text = f"на {months} месяцев"
    subscription_type = f"{months} месяцев"
    display_type = subscription_type
    premium_kb = kb.start_kb

# Сохраняем подписку в БД
success = await save_premium_subscription(
    user_id=call.from_user.id,
    nickname=call.from_user.username or "",
    premium=True,
    expiry_date=expiry_date,
    subscription_type=subscription_type # Сохраняем
техническое значение
)

if not success:

```

```

        await call.message.answer("✗ Не удалось оформить
подписку. Попробуйте позже.")
        return

    await complete_registration(call, state, user_data,
premium=True)

    except Exception as e:
        print(f"Ошибка при выборе подписки: {e}")
        await call.message.answer(
            "Произошла ошибка, попробуйте выбрать подписку
снова",

reply_markup=kb.premium_duration_registration_kb.as_markup()
)

@dp.callback_query(F.data == "reg_premium_no",
RegistrationStates.waiting_for_premium)
async def premium_choice_no_registration(call: CallbackQuery,
state: FSMContext):
    try:
        await call.message.delete()
        user_data = await state.get_data()
        await complete_registration(call, state, user_data,
premium=False)
    except Exception as e:
        print(f"Ошибка при отказе от премиума: {e}")
        await call.message.answer(
            "Произошла ошибка при завершении регистрации",
            reply_markup=kb.start_kb.as_markup()
        )

@dp.callback_query(F.data == "Menu")
async def menu(call: CallbackQuery):
    await call.message.delete()
    photo = ph.photo_start
    await call.message.answer_photo(photo=photo,
caption=f'Выберите нужный
пункт меню',

reply_markup=kb.menu_kb.as_markup(resize_keyboard=True))

@dp.callback_query(F.data == 'Change')
async def options_menu(call: CallbackQuery):
    await call.message.delete()
    user_info = await get_user_info(call.from_user.id)
    if user_info:
        # Форматируем отчество и email
        patronymic_display = user_info['patronymic'] if
user_info.get('patronymic') else ''

```

```

        email_display = user_info['email'] if
user_info.get('email') else 'не указан'
        gender_display = "Мужской" if user_info.get('gender') ==
'male' else "Женский" if user_info.get('gender') else 'не
указан'

        await call.message.answer(
            text=f"Ваши текущие данные:\n"
                f"ФИО: {user_info['surname']}\n"
{user_info['name']} {patronymic_display}\n"
                f"Пол: {gender_display}\n"
                f"Телефон: {user_info['phone']}\n"
                f"Email: {email_display}\n"
                f"Дата рождения: {user_info['birthday']}\n"
                f"Рост: {user_info.get('height', 'не указан')}\n"
            см\n"
                f"Вес: {user_info.get('weight', 'не указан')}\n"
            кг\n"
                f"Фитнес-цель:
{format_fitness_goal(user_info.get('fitness_goal'))}\n"
                f"Выберите, что хотите изменить:",

            reply_markup=kb.change_separate_kb.as_markup(resize_keyboard=True))
        else:
            await call.message.answer("Сначала пройдите
регистрацию!")

@dp.callback_query(F.data == 'change_surname')
async def change_surname_start(call: CallbackQuery, state:
FSMContext):
    try:
        user_info = await get_user_info(call.from_user.id)
        if not user_info:
            await call.answer("Сначала пройдите регистрацию!")
            return

        current_surname = user_info.get('surname', 'не указана')
        await call.message.edit_text(
            f"Текущая фамилия: {current_surname}\n"
            "Введите новую фамилию (только буквы, не более 90
СИМВОЛОВ):",
            reply_markup=kb.back_to_change_kb.as_markup()
        )
        await state.set_state(ChangeDataStates.changing_surname)

    except Exception as e:
        print(f"Ошибка в change_surname_start: {e}")
        await call.answer("Произошла ошибка")

@dp.message(ChangeDataStates.changing_surname)

```

```

async def process_new_surname(message: Message, state:
FSMContext):
    surname = message.text.strip()

    if len(surname.split()) > 2:
        await message.answer("Фамилия должна содержать не более
90 символов.\nПопробуйте еще раз:",

reply_markup=kb.back_to_change_kb.as_markup())
        return

    if not is_valid_name(surname):
        await message.answer("Фамилия должна содержать только
буквы.\nПопробуйте еще раз:",

reply_markup=kb.back_to_change_kb.as_markup())
        return

    formatted_surname = format_name(surname)
    success, response, reply_markup = await
update_user_field(message.from_user.id, 'surname',
formatted_surname)
    await message.answer(response, reply_markup=reply_markup if
reply_markup else kb.back_to_change_kb.as_markup())
    await state.clear()

@dp.callback_query(F.data == 'change_name')
async def change_name_start(call: CallbackQuery, state:
FSMContext):
    try:
        user_info = await get_user_info(call.from_user.id)
        if not user_info:
            await call.answer("Сначала пройдите регистрацию!")
            return

        current_name = user_info.get('name', 'не указано')
        await call.message.edit_text(
            f"Текущее имя: {current_name}\n"
            "Введите новое имя (только буквы, не более 90
СИМВОЛОВ):",
            reply_markup=kb.back_to_change_kb.as_markup()
        )
        await state.set_state(ChangeDataStates.changing_name)

    except Exception as e:
        print(f"Ошибка в change_name_start: {e}")
        await call.answer("Произошла ошибка")

@dp.message(ChangeDataStates.changing_name)
async def process_new_name(message: Message, state: FSMContext):
    name = message.text.strip()

```

```

        if len(name.split()) > 2:
            await message.answer("Имя должно содержать не более 90
символов.\nПопробуйте еще раз:",

reply_markup=kb.back_to_change_kb.as_markup())
            return

        if not is_valid_name(name):
            await message.answer("Имя должно содержать только
буквы.\nПопробуйте еще раз:",

reply_markup=kb.back_to_change_kb.as_markup())
            return

        formatted_name = format_name(name)
        success, response, reply_markup = await
update_user_field(message.from_user.id, 'name', formatted_name)
        await message.answer(response, reply_markup=reply_markup if
reply_markup else kb.back_to_change_kb.as_markup())
        await state.clear()

@dp.callback_query(F.data == 'change_patronymic')
async def change_patronymic_start(call: CallbackQuery, state:
FSMContext):
    try:
        user_info = await get_user_info(call.from_user.id)
        if not user_info:
            await call.answer("Сначала пройдите регистрацию!")
            return

        current_patronymic = user_info.get('patronymic', 'не
указано')
        await call.message.edit_text(
            f"Текущее отчество: {current_patronymic}\n"
            "Введите новое отчество или нажмите кнопку 'Удалить
отчество':",
            reply_markup=kb.skip_patronymic_kb.as_markup()
        )
        await
state.set_state(ChangeDataStates.changing_patronymic)

    except Exception as e:
        print(f"Ошибка в change_patronymic_start: {e}")
        await call.answer("Произошла ошибка")

@dp.message(ChangeDataStates.changing_patronymic)
async def process_new_patronymic(message: Message, state:
FSMContext):
    patronymic = message.text.strip()

    if len(patronymic.split()) > 2:

```

```

        await message.answer("Отчество должно содержать не более
90 символов.\nПопробуйте еще раз:",

reply_markup=kb.skip_patronymic_kb.as_markup())
        return

    if not is_valid_name(patronymic):
        await message.answer("Отчество должно содержать только
буквы. Попробуйте еще раз:",

reply_markup=kb.skip_patronymic_kb.as_markup())
        return

    formatted_patronymic = format_name(patronymic)
    success, response, reply_markup = await
update_user_field(message.from_user.id, 'patronymic',
formatted_patronymic)
    await message.answer(response, reply_markup=reply_markup if
reply_markup else kb.start_kb.as_markup())
    await state.clear()

@dp.callback_query(F.data == "skip_patronymic",
ChangeDataStates.changing_patronymic)
async def skip_patronymic_change(call: CallbackQuery, state:
FSMContext):
    await call.message.delete()
    success, response = await
update_user_field(call.from_user.id, 'patronymic', None)
    if success:
        await call.message.answer(
            "Отчество удалено!\n Данные успешно сохранены!",
            reply_markup=kb.start_kb.as_markup()
        )
    else:
        await call.message.answer(response)
    await state.clear()

@dp.callback_query(F.data == 'Phone')
async def change_phone_start(call: CallbackQuery, state:
FSMContext):
    try:
        user_info = await get_user_info(call.from_user.id)
        if not user_info:
            await call.answer("Сначала пройдите регистрацию!")
            return

        current_phone = user_info.get('phone', 'не указан')
        await call.message.edit_text(
            f"Текущий телефон: {current_phone}\n"
            "Введите новый номер телефона (начинается с + и 11
цифр или с 8 и 10 цифр):",
            reply_markup=kb.back_to_change_kb.as_markup()

```

```

    )
    await state.set_state(ChangeDataStates.changing_phone)

except Exception as e:
    print(f"Ошибка в change_phone_start: {e}")
    await call.answer("Произошла ошибка")

@dp.message(ChangeDataStates.changing_phone)
async def process_new_phone(message: Message, state:
FSMContext):
    if message.text.lower() in ["отмена", "/cancel"]:
        await cancel_handler(message, state)
        return

    phone = message.text

    if not is_valid_phone(phone):
        await message.answer("Некорректный номер телефона.
Попробуйте еще раз:",

reply_markup=kb.back_to_change_kb.as_markup())
        return

    if not await is_phone_unique(phone, message.from_user.id):
        await message.answer("Этот номер уже используется.
Введите другой:",

reply_markup=kb.back_to_change_kb.as_markup())
        return

    success, response, reply_markup = await
update_user_field(message.from_user.id, 'phone', phone)
    await message.answer(response, reply_markup=reply_markup if
reply_markup else kb.start_kb.as_markup())
    await state.clear()

@dp.callback_query(F.data == 'Email')
async def change_email_start(call: CallbackQuery, state:
FSMContext):
    try:
        user_info = await get_user_info(call.from_user.id)
        if not user_info:
            await call.answer("Сначала пройдите регистрацию!")
            return

        current_email = user_info.get('email', 'не указан')
        await call.message.edit_text(
            f"Текущий email: {current_email}\n"
            "Введите новый email или нажмите кнопку 'Удалить
email':",

            reply_markup=kb.skip_or_cancel_kb.as_markup()
        )
    )

```

```

        await state.set_state(ChangeDataStates.changing_email)

    except Exception as e:
        print(f"Ошибка в change_email_start: {e}")
        await call.answer("Произошла ошибка")

@dp.callback_query(F.data.startswith('Skip_or_C'))
async def handle_skip_or_cancel(call: CallbackQuery, state:
FSMContext):
    action = call.data.split('_')[-1]

    if action == '1': # Пропустить
        user_info = await get_user_info(call.from_user.id)
        current_email = user_info.get('email') if user_info else
None

        if current_email is None:
            await call.message.delete()
            await call.message.answer(
                "Email остается не указанным.",
                reply_markup=kb.back_to_change_kb.as_markup()
            )
        else:
            success, _ = await
update_user_field(call.from_user.id, 'email', None)
            if success:
                await call.message.delete()
                await call.message.answer(
                    "Email удален!",
                    reply_markup=kb.back_to_change_kb.as_markup()
                )
            else:
                await call.message.delete()
                await call.message.answer(
                    "Произошла ошибка при удалении email.",
                    reply_markup=kb.back_to_change_kb.as_markup()
                )
            await state.clear()

@dp.message(ChangeDataStates.changing_email)
async def process_new_email(message: Message, state:
FSMContext):
    email = message.text.strip() if message.text else ""

    if not email:
        await message.answer("Пожалуйста, введите email или
используйте кнопки:",
        reply_markup=kb.skip_or_cancel_kb.as_markup())
    return

```

```

        if not is_valid_email(email):
            await message.answer("Некорректный email. Попробуйте еще
раз:",

reply_markup=kb.skip_or_cancel_kb.as_markup())
        return

        if not await is_email_unique(email, message.from_user.id):
            await message.answer("Этот email уже используется.
Введите другой:",

reply_markup=kb.skip_or_cancel_kb.as_markup())
        return

        success, response, reply_markup = await
update_user_field(message.from_user.id, 'email', email)
        await message.answer(response, reply_markup=reply_markup if
reply_markup else kb.back_to_change_kb.as_markup())
        await state.clear()

@dp.callback_query(F.data == 'Birthday')
async def change_birthday_start(call: CallbackQuery, state:
FSMContext):
    try:
        user_info = await get_user_info(call.from_user.id)
        if not user_info:
            await call.answer("Сначала пройдите регистрацию!")
            return

        current_birthday = user_info.get('birthday', 'не
указана')
        if isinstance(current_birthday, date):
            current_birthday =
current_birthday.strftime('%d.%m.%Y')

        await call.message.edit_text(
            f"Текущая дата рождения: {current_birthday}\n"
            "Введите новую дату рождения в формате ДД.ММ.ГГГГ:",
            reply_markup=kb.skip_or_cancel_kb.as_markup()
        )
        await
state.set_state(ChangeDataStates.changing_birthday)

    except Exception as e:
        print(f"Ошибка в change_birthday_start: {e}")
        await call.answer("Произошла ошибка")

@dp.message(ChangeDataStates.changing_birthday)
async def process_new_birthday(message: Message, state:
FSMContext):
    try:

```

```

        new_birth_date = datetime.strptime(message.text,
"%d.%m.%Y").date()
        is_valid, error_msg = is_valid_birthday(new_birth_date)

        if not is_valid:
            await message.answer(f"{error_msg}\nПопробуйте еще
раз:",

reply_markup=kb.skip_or_cancel_kb.as_markup())
            return

        success, response, reply_markup = await
update_user_field(message.from_user.id, 'birthday',
new_birth_date)
        if success:
            age = calculate_age(new_birth_date)
            await message.answer(f"{response}! Вам {age} лет",

reply_markup=kb.skip_or_cancel_kb.as_markup())
        else:
            await message.answer(response,
reply_markup=reply_markup if reply_markup else None)
        except ValueError:
            await message.answer("Неверный формат. Используйте
ДД.ММ.ГГГГ",

reply_markup=kb.skip_or_cancel_kb.as_markup())
        finally:
            await state.clear()

@dp.callback_query(F.data == 'change_height')
async def change_height_start(call: CallbackQuery, state:
FSMContext):
    try:
        user_info = await get_user_info(call.from_user.id)
        if not user_info:
            await call.answer("Сначала пройдите регистрацию!")
            return

        current_height = user_info.get('height', 'не указан')
        await call.message.edit_text(
            f"Текущий рост: {current_height} см\n"
            "Введите новый рост (100-250 см):",
            reply_markup=kb.back_to_change_kb.as_markup()
        )
        await state.set_state(ChangeDataStates.changing_height)

    except Exception as e:
        print(f"Ошибка в change_height_start: {e}")
        await call.answer("Произошла ошибка")

@dp.message(ChangeDataStates.changing_height)

```

```

async def process_new_height(message: Message, state:
FSMContext):
    try:
        new_height = int(message.text)
        if new_height < 100 or new_height > 250:
            await message.answer("Рост должен быть от 100 до 250
см. Попробуйте еще раз:",

reply_markup=kb.back_to_change_kb.as_markup())
        return

        success, response, reply_markup = await
update_user_field(message.from_user.id, 'height', new_height)
        await message.answer(response, reply_markup=reply_markup
if reply_markup else kb.back_to_change_kb.as_markup())
        if success:
            await recalculate_goals(message.from_user.id)
    except ValueError:
        await message.answer("Пожалуйста, введите число:",

reply_markup=kb.back_to_change_kb.as_markup())
    finally:
        await state.clear()

@dp.callback_query(F.data == 'change_weight')
async def change_weight_start(call: CallbackQuery, state:
FSMContext):
    try:
        user_info = await get_user_info(call.from_user.id)
        if not user_info:
            await call.answer("Сначала пройдите регистрацию!")
            return

        current_weight = user_info.get('weight', 'не указан')
        await call.message.edit_text(
            f"Текущий вес: {current_weight} кг\n"
            "Введите новый вес (30-200 кг):",
            reply_markup=kb.back_to_change_kb.as_markup()
        )
        await state.set_state(ChangeDataStates.changing_weight)

    except Exception as e:
        print(f"Ошибка в change_weight_start: {e}")
        await call.answer("Произошла ошибка")

@dp.message(ChangeDataStates.changing_weight)
async def process_new_weight(message: Message, state:
FSMContext):
    try:
        new_weight = float(message.text)
        if new_weight < 30 or new_weight > 200:

```

```

        await message.answer("Вес должен быть от 30 до 200
кг. Попробуйте еще раз:",

reply_markup=kb.back_to_change_kb.as_markup())
        return

        success, response, reply_markup = await
update_user_field(message.from_user.id, 'weight', new_weight)
        await message.answer(response, reply_markup=reply_markup
if reply_markup else kb.back_to_change_kb.as_markup())
        if success:
            await recalculate_goals(message.from_user.id)
        except ValueError:
            await message.answer("Пожалуйста, введите число:",

reply_markup=kb.back_to_change_kb.as_markup())
        finally:
            await state.clear()

@dp.callback_query(F.data == 'change_fitness_goal')
async def change_fitness_goal_start(call: CallbackQuery, state:
FSMContext):
    try:
        user_info = await get_user_info(call.from_user.id)
        if not user_info:
            await call.answer("Сначала пройдите регистрацию!")
            return

        current_goal =
format_fitness_goal(user_info.get('fitness_goal'))
        await call.message.edit_text(
            f"Текущая фитнес-цель: {current_goal}\n"
            "Выберите новую цель:",
            reply_markup=kb.fitness_goal_kb.as_markup()
        )
        await
state.set_state(ChangeDataStates.changing_fitness_goal)

    except Exception as e:
        print(f"Ошибка в change_fitness_goal_start: {e}")
        await call.answer("Произошла ошибка")

@dp.callback_query(ChangeDataStates.changing_fitness_goal,
F.data.startswith("goal_"))
async def process_new_fitness_goal(call: CallbackQuery, state:
FSMContext):
    try:
        # Удаляем сообщение с кнопками сразу
        await call.message.delete()

        # Получаем текущие данные
        user_data = await state.get_data()

```

```

current_goal = user_data.get('current_goal')
new_goal = call.data.split("_")[1]

# Проверяем, изменилась ли цель
if current_goal == new_goal:
    await bot.send_message(
        chat_id=call.from_user.id,
        text="Значение не изменилось",
        reply_markup=kb.back_to_change_kb.as_markup()
    )
    await state.clear()
    return

# Если цель изменилась
goal_display = format_fitness_goal(new_goal)
success, response, _ = await
update_user_field(call.from_user.id, 'fitness_goal', new_goal)

if success:
    await bot.send_message(
        chat_id=call.from_user.id,
        text=f"Данные успешно обновлены. Теперь цель:
{goal_display}",
        reply_markup=kb.back_to_change_kb.as_markup()
    )
    await recalculate_goals(call.from_user.id)
else:
    await bot.send_message(
        chat_id=call.from_user.id,
        text=response or "Ошибка при обновлении цели",
        reply_markup=kb.back_to_change_kb.as_markup()
    )

await state.clear()

except Exception as e:
    print(f"Ошибка в process_new_fitness_goal: {e}")
    await bot.send_message(
        chat_id=call.from_user.id,
        text="Произошла ошибка при изменении цели",
        reply_markup=kb.back_to_change_kb.as_markup()
    )

@dp.callback_query(F.data == 'change_gender')
async def change_gender_start(call: CallbackQuery, state:
FSMContext):
    try:
        user_info = await get_user_info(call.from_user.id)
        if not user_info:
            await call.answer("Сначала пройдите регистрацию!")
            return

```

```

        current_gender = user_info.get('gender')
        gender_display = "Мужской" if current_gender == 'male'
else "Женский" if current_gender else 'не указан'

# Сохраняем текущее значение для последующей проверки
await state.update_data(current_gender=current_gender)

await call.message.edit_text(
    f"Текущий пол: {gender_display}\n"
    "Выберите новый пол:",
    reply_markup=kb.gender_change_kb.as_markup()
)
await state.set_state(ChangeDataStates.changing_gender)

except Exception as e:
    print(f"Ошибка в change_gender_start: {e}")
    await call.answer("Произошла ошибка")

@dp.callback_query(ChangeDataStates.changing_gender,
F.data.startswith("gender_"))
async def process_new_gender(call: CallbackQuery, state:
FSMContext):
    try:
        user_data = await state.get_data()
        current_gender = user_data.get('current_gender')
        new_gender = call.data.split("_")[1]

        # Проверяем, изменился ли пол
        if current_gender == new_gender:
            await call.message.delete() # Удаляем сообщение с
выбором пола
            await call.message.answer(
                "Значение не изменилось",
                reply_markup=kb.back_to_change_kb.as_markup()
            )
            await state.clear()
            return

        gender_display = "Мужской" if new_gender == 'male' else
"Женский"

        success, response, _ = await
update_user_field(call.from_user.id, 'gender', new_gender)

        if success:
            await call.message.edit_text(
                f"Данные успешно обновлены. Теперь пол:
{gender_display}",
                reply_markup=kb.back_to_change_kb.as_markup()
            )
            await recalculate_goals(call.from_user.id)
        else:

```

```

        await call.answer(response or "Ошибка при обновлении
пола", show_alert=True)

        await state.clear()

    except Exception as e:
        print(f"Ошибка в process_new_gender: {e}")
        await call.answer("Произошла ошибка при изменении пола",
show_alert=True)

@dp.message(Command("cancel"))
@dp.message(F.text.casefold() == "отмена")
async def cancel_handler(message: Message, state: FSMContext):
    current_state = await state.get_state()
    if not current_state:
        await message.answer("Нет активных действий для
отмены.")
        return

    # Проверяем, в каком состоянии находится пользователь
    if current_state in [
        ChangeDataStates.changing_surname,
        ChangeDataStates.changing_name,
        ChangeDataStates.changing_patronymic,
        ChangeDataStates.changing_phone,
        ChangeDataStates.changing_email,
        ChangeDataStates.changing_birthday
    ]:
        await state.clear()
        await message.answer(
            "Изменение данных отменено.",
            reply_markup=kb.start_kb.as_markup()
        )
    elif current_state in [
        RegistrationStates.waiting_for_surname,
        RegistrationStates.waiting_for_name,
        RegistrationStates.waiting_for_patronymic,
        RegistrationStates.waiting_for_phone,
        RegistrationStates.waiting_for_email,
        RegistrationStates.waiting_for_birthday,
        RegistrationStates.waiting_for_premium,
        RegistrationStates.waiting_for_options
    ]:
        await state.clear()
        await message.answer(
            "Регистрация отменена.",
            reply_markup=kb.start_kb.as_markup()
        )
    else:
        await state.clear()
        await message.answer(
            "Действие отменено.",

```

```

        reply_markup=kb.start_kb.as_markup()
    )

@dp.message(Command("history"))
async def show_history(message: Message):
    user_id = message.from_user.id
    history = await get_user_history(user_id, 7)

    if not history:
        await message.answer("У вас пока нет истории активности.")
        return

    history_text = "📅 Ваша активность за последние 7 дней:\n\n"
    for record in history:
        date_str = record[0].strftime("%d.%m.%Y")
        balance = record[3] - record[2] # Потреблено - сожжено
        history_text += f"✳️ {date_str}:\n"
        history_text += f"👣 Шаги: {record[1]}\n"
        history_text += f"🔥 Сожжено калорий: {record[2]}\n"
        history_text += f"🥗 Потреблено калорий: {record[3]}\n"
        history_text += f"⚖️ Баланс: {balance}\n\n"

    await message.answer(history_text)

@dp.message(Command("my_data"))
async def show_my_data(message: Message):
    user_info = await get_user_info(message.from_user.id)
    if user_info:
        await message.answer(
            f"Ваши данные:\n"
            f"Рост: {user_info.get('height', 'не указан')}\n"
            f"Вес: {user_info.get('weight', 'не указан')}\n"
            f"Фитнес-цель: {format_fitness_goal(user_info.get('fitness_goal'))}\n"
            f"Цель по шагам: {user_info.get('steps_goal', 'не указана')}\n"
            f"Рекомендуемое потребление калорий: {user_info.get('recommended_caloric_intake', 'не рассчитано')}"
        )
    else:
        await message.answer("Вы еще не зарегистрированы.")

@dp.message(Command("premium"))
async def check_premium(message: Message):
    user_id = message.from_user.id
    is_premium = await check_premium_status(user_id)

    if is_premium:

```

```

conn = None
try:
    conn = await get_db_connection()
    with conn.cursor() as cursor:
        cursor.execute("""
            SELECT expiry_date, subscription_type
            FROM premium_subscriptions
            WHERE user_id = %s
            """, (user_id,))
        result = cursor.fetchone()

        if result:
            expiry_date, sub_type = result
            display_type = "Вечный" if sub_type ==
"lifetime" else sub_type
            is_lifetime = sub_type == "lifetime"
            days_left = (expiry_date -
date.today()).days

            await message.answer(
                f"✓ Ваша премиум-подписка активна!\n"
                f"Тип подписки: {display_type}\n"
                f"Дата окончания:
{expiry_date.strftime('%d.%m.%Y')}\n"
                f"Осталось дней: {days_left}",

reply_markup=kb.premium_manage_no_renew_kb.as_markup() if
is_lifetime

                else kb.premium_manage_kb.as_markup()
            )
        except Exception as e:
            print(f"Ошибка при проверке подписки: {e}")
            await message.answer(
                "Произошла ошибка при проверке подписки.",
                reply_markup=kb.start_kb.as_markup()
            )
        finally:
            if conn:
                conn.close()
    else:
        await message.answer(
            "✗ У вас нет активной премиум-подписки.\n",
            reply_markup=kb.premium_upsell_kb.as_markup()
        )

@dp.callback_query(F.data == 'Premium_settings')
async def premium_settings_menu(call: CallbackQuery):
    try:
        await call.message.delete()

        user_info = await get_user_info(call.from_user.id)

```

```

        if not user_info:
            await call.message.answer("✗ Не удалось загрузить
данные пользователя")
            return

        if user_info['premium']:
            conn = None
            try:
                conn = await get_db_connection()
                with conn.cursor() as cursor:
                    cursor.execute("""
                        SELECT expiry_date, subscription_type
                        FROM premium_subscriptions
                        WHERE user_id = %s
                    """, (call.from_user.id,))
                    result = cursor.fetchone()

                    if result:
                        expiry_date, sub_type = result
                        # Определяем тип для отображения
                        display_type = "Вечный" if sub_type ==
"lifetime" else sub_type
                        # Определяем клавиатуру
                        is_lifetime = sub_type == "lifetime"

                        days_left = (expiry_date -
date.today()).days

                        await call.message.answer(
                            f"★ Ваша Premium подписка
активна!\n"

                            f"► Тип: {display_type}\n"
                            f"► Действует до:
{expiry_date.strftime('%d.%m.%Y')}\n"
                            f"► Осталось дней: {days_left}\n\n"
                            "Выберите действие:",

                            reply_markup=kb.premium_manage_no_renew_kb.as_markup() if
is_lifetime
                                else
kb.premium_manage_kb.as_markup()
                                )
                        else:
                            await call.message.answer(
                                "Текущий статус Premium: не
активирована\n"

                                "Выберите действие:",

                                reply_markup=kb.premium_toggle_kb.as_markup()
                                )
            except Exception as e:

```

```

        print(f"Ошибка при проверке подписки: {e}")
        await call.message.answer(
            "Текущий статус Premium: не активирована\n"
            "Выберите действие:",

reply_markup=kb.premium_toggle_kb.as_markup()
        )
        finally:
            if conn:
                conn.close()
        else:
            await call.message.answer(
                "Текущий статус Premium: не активирована\n"
                "Хотите активировать подписку?",
                reply_markup=kb.premium_upsell_kb.as_markup()
            )
    except Exception as e:
        print(f"Ошибка в premium_settings_menu: {e}")
        await call.message.answer(
            "Произошла ошибка при обработке запроса",
            reply_markup=kb.start_kb.as_markup()
        )

@dp.callback_query(F.data.startswith("toggle_premium_"))
async def toggle_premium(call: CallbackQuery, state:
FSMContext):
    await call.message.delete()
    action = call.data.split("_")[-1]

    if action == "renew":
        # Обработка продления подписки
        await call.message.answer(
            "Выберите срок продления подписки:",
            reply_markup=kb.premium_duration_kb.as_markup()
        )
        return

    new_status = action == "on"
    user_id = call.from_user.id

    if new_status:
        # Активация новой подписки
        await call.message.answer(
            "Выберите срок подписки:",
            reply_markup=kb.premium_duration_kb.as_markup()
        )
        await
    state.set_state(PremiumStates.choosing_subscription)
    else:
        # Деактивация подписки
        success, response, _ = await update_user_field(user_id,
'premium', False)

```

```

    if success:
        # Также обновляем подписку в отдельной таблице
        await save_premium_subscription(
            user_id,
            call.from_user.username or "",
            False,
            date.today(), # Устанавливаем текущую дату как
дату окончания
            "deactivated"
        )
        await call.message.answer(
            "Premium подписка деактивирована!",
            reply_markup=kb.back_to_Premium_kb.as_markup()
        )
    else:
        await call.message.answer(
            response,
            reply_markup=kb.back_to_Premium_kb.as_markup()
        )

@dp.callback_query(F.data == "toggle_premium_renew")
async def renew_premium(call: CallbackQuery, state: FSMContext):
    try:
        await call.message.delete()
        # Устанавливаем состояние ПЕРЕД показом клавиатуры
        await
state.set_state(PremiumStates.choosing_subscription)
        await call.message.answer(
            "Выберите срок продления подписки:",
            reply_markup=kb.premium_duration_kb.as_markup()
        )
    except Exception as e:
        print(f"Ошибка в renew_premium: {e}")
        await call.message.answer(
            "Произошла ошибка при обработке запроса.",
            reply_markup=kb.start_kb.as_markup()
        )

async def save_premium_subscription(user_id: int, nickname: str,
premium: bool, expiry_date: date,
                                subscription_type: str) ->
bool:
    conn = None
    try:
        conn = await get_db_connection()
        with conn.cursor() as cursor:
            # Update premium status in users table
            cursor.execute("""
                UPDATE users SET premium = %s WHERE user_id = %s
            """, (premium, user_id))

```

```

        # Add/update record in premium_subscriptions
        cursor.execute("""
            INSERT INTO premium_subscriptions
            (user_id, nickname, premium, expiry_date,
subscription_type)
            VALUES (%s, %s, %s, %s, %s)
            ON CONFLICT (user_id) DO UPDATE SET
                nickname = EXCLUDED.nickname,
                premium = EXCLUDED.premium,
                expiry_date = EXCLUDED.expiry_date,
                subscription_type =
EXCLUDED.subscription_type
            """, (user_id, nickname or "", premium, expiry_date,
subscription_type))

        conn.commit()
        return True
    except Exception as e:
        print(f"Ошибка при сохранении подписки: {e}")
        if conn:
            conn.rollback()
        return False
    finally:
        if conn:
            conn.close()

async def check_premium_status(user_id: int) -> bool:
    conn = None
    try:
        conn = await get_db_connection()
        with conn.cursor() as cursor:
            # Проверяем как в основной таблице users, так и в
premium_subscriptions
            cursor.execute("""
                SELECT u.premium, ps.expiry_date
                FROM users u
                LEFT JOIN premium_subscriptions ps ON u.user_id
= ps.user_id
                WHERE u.user_id = %s
            """, (user_id,))
            result = cursor.fetchone()

            if result:
                premium, expiry_date = result
                if not premium:
                    return False
                if expiry_date and expiry_date < date.today():
                    # Подписка истекла, обновляем статус
                    await update_premium_status(user_id, False)
                    return False
                return True
            return False

```

```

except Exception as e:
    print(f"Ошибка при проверке премиум-статуса: {e}")
    return False
finally:
    if conn:
        conn.close()

async def update_premium_status(user_id: int, status: bool):
    conn = None
    try:
        conn = await get_db_connection()
        with conn.cursor() as cursor:
            # Обновляем обе таблицы для согласованности
            cursor.execute("""
                UPDATE users SET premium = %s WHERE user_id = %s
            """, (status, user_id))

            cursor.execute("""
                UPDATE premium_subscriptions SET premium = %s
WHERE user_id = %s
            """, (status, user_id))

            # Если статус премиума снят - отключаем уведомления
            # о рецептах
            if not status:
                # Получаем текущие настройки
                cursor.execute("""
                    SELECT recipes_enabled FROM
user_notifications
                    WHERE user_id = %s
                """, (user_id,))
                result = cursor.fetchone()

                if result and result[0]: # Если уведомления о
                    # рецептах были включены
                    cursor.execute("""
                        UPDATE user_notifications
                        SET recipes_enabled = FALSE
                        WHERE user_id = %s
                    """, (user_id,))
                    try:
                        await bot.send_message(
                            user_id,
                            "❗ Уведомления о рецептах отключены,
так как ваша Premium подписка закончилась."
                        )
                    except Exception as e:
                        print(f"Не удалось уведомить
пользователя {user_id}: {e}")

            conn.commit()

```

```

except Exception as e:
    print(f"Ошибка при обновлении премиум-статуса: {e}")
    if conn:
        conn.rollback()
finally:
    if conn:
        conn.close()

async def check_expired_subscriptions():
    conn = None
    try:
        conn = await get_db_connection()
        with conn.cursor() as cursor:
            # Находим подписки, которые истекли, но еще помечены
            как активные
            cursor.execute("""
                SELECT user_id FROM premium_subscriptions
                WHERE premium = TRUE AND expiry_date < %s
            """, (date.today(),))
            expired_users = cursor.fetchall()

            for (user_id,) in expired_users:
                await update_premium_status(user_id, False)
                try:
                    await bot.send_message(
                        user_id,
                        "❌ Ваша премиум-подписка истекла.\n"
                        "Для продления доступа оформите подписку
снова.",
                        reply_markup=kb.premium_upsell_kb.as_markup()
                    )
                except Exception as e:
                    print(f"Не удалось уведомить пользователя
{user_id}: {e}")

            print(f"Проверка подписок завершена. Найдено
{len(expired_users)} истекших подписок.")
    except Exception as e:
        print(f"Ошибка при проверке подписок: {e}")
    finally:
        if conn:
            conn.close()

async def send_motivational_notification(user_id: int) -> bool:
    """Отправляет мотивационное уведомление с обработкой
ошибок"""
    try:
        print(f"\n🔔 Попытка отправки мотивации для {user_id}")

```

```

# Проверяем, существует ли пользователь и есть ли
настройки
settings = await get_user_notification_settings(user_id)
if not settings:
    print("✗ Настройки уведомлений не найдены")
    return False

if not settings['motivation_enabled']:
    print("⚙ Мотивационные уведомления отключены
пользователем")
    return False

print(f"⌚ Интервал мотивации:
{settings['motivation_interval']} ч")

# Проверяем время последней отправки
last_sent = settings['last_motivation']
now = datetime.now()

if last_sent:
    hours_passed = (now - last_sent).total_seconds() /
3600

    print(f"⌚ С последней отправки прошло:
{hours_passed:.1f} ч")

    if hours_passed < settings['motivation_interval']:
        print(f"⌚ Рано отправлять (требуется
{settings['motivation_interval']} ч)")
        return False

# Выбираем случайную цитату
quote = random.choice(MOTIVATIONAL_QUOTES)
print(f"📄 Выбрана цитата: {quote[:50]}...")

# Удаляем предыдущее уведомление
prev_msg_id =
last_notification_ids['motivation'].get(user_id)
if prev_msg_id:
    print(f"🗑 Удаление предыдущего уведомления (ID:
{prev_msg_id})")
    try:
        await bot.delete_message(chat_id=user_id,
message_id=prev_msg_id)
    except Exception as e:
        print(f"⚠ Не удалось удалить: {str(e)}")

# Отправляем новое уведомление
print(f"✉ Отправка мотивационного сообщения...")
try:
    msg = await bot.send_message(

```

```

        user_id,
        f"👉 Мотивация дня:\n\n{quote}",
        parse_mode="HTML"
    )
    print(f"✅ Успешно отправлено (ID:
{msg.message_id})")

    # Обновляем данные
    last_notification_ids['motivation'][user_id] =
msg.message_id
    await update_notification_settings(
        user_id,
        {'last_motivation': now}
    )
    return True

except TelegramBadRequest as e:
    if "chat not found" in str(e):
        print(f"❌ Чат с пользователем {user_id} не
найден (возможно, заблокировал бота)")
        # Можно обновить статус пользователя в БД
        return False
    raise # Перебрасываем другие ошибки

except Exception as e:
    print(f"❌ Критическая ошибка: {str(e)}")
    import traceback
    traceback.print_exc()
    return False

async def send_stats_notification(user_id: int):
    """Отправляет уведомление со статистикой с подробным
логированием"""
    try:
        print(f"📤 Попытка отправки статистики для {user_id}")

        user_info = await get_user_info(user_id)
        if not user_info:
            print("❌ Пользователь не найден")
            return False

        print(f"📊 Данные пользователя:
steps={user_info.get('steps')},
calories={user_info.get('calories_burned')}")

        # Создаем клавиатуру
        kb = InlineKeyboardBuilder()
        kb.button(text='👣 Шаги', callback_data='Step')
        kb.button(text='🔥 Калории', callback_data='Calory')
        kb.adjust(2)

```

```

        balance = user_info['calories_consumed'] -
user_info['calories_burned']
        text = (
            f"📊 <b>Ваша статистика</b>\n\n"
            f"• Шагов:
<b>{user_info['steps']}/{user_info['steps_goal']}</b>\n"
            f"• Сожжено: <b>{user_info['calories_burned']}</b>
ккал\n"
            f"• Потреблено: <b>{user_info['calories_consumed']}</b>
ккал\n"
            f"• Баланс: <b>{balance}</b> ккал\n"
            f"• Рекомендуемое потребление:
<b>{user_info.get('recommended_caloric_intake', 'N/A')}</b>
ккал\n"
        )

        # Удаление предыдущего уведомления
        prev_msg_id = last_notification_ids.get('stats',
        {}).get(user_id)
        if prev_msg_id:
            print(f"🗑️ Удаление предыдущего уведомления (ID:
        {prev_msg_id})")
            try:
                await bot.delete_message(chat_id=user_id,
        message_id=prev_msg_id)
            except Exception as e:
                print(f"⚠️ Не удалось удалить предыдущее
        сообщение: {str(e)}")

        # Отправка нового уведомления
        print("✉️ Отправка нового уведомления...")
        msg = await bot.send_message(
            user_id,
            text,
            parse_mode="HTML",
            reply_markup=kb.as_markup()
        )

        print(f"✅ Уведомление отправлено (ID:
        {msg.message_id})")

        # Сохраняем ID сообщения
        last_notification_ids.setdefault('stats', {})[user_id] =
        msg.message_id
        await update_notification_settings(user_id,
        {'last_stats': datetime.now()})
        return True

    except Exception as e:
        print(f"❌ Критическая ошибка при отправке: {str(e)}")

```

```

import traceback
traceback.print_exc()
return False

async def send_recipe_notification(user_id: int, meal_type:
str):
    """Отправляет уведомление с рецептом"""
    try:
        if not await check_premium_status(user_id):
            # Отключаем уведомления о рецептах, если они были
включены
            settings = await
get_user_notification_settings(user_id)
            if settings and settings['recipes_enabled']:
                await update_notification_settings(user_id,
{'recipes_enabled': False})
            try:
                await bot.send_message(
                    user_id,
                    "❗ Уведомления о рецептах отключены, так
как у вас нет активной Premium подписки."
                )
            except Exception as e:
                print(f"Не удалось уведомить пользователя
{user_id}: {e}")
            return False

        meal_type_in_db = meal_type_mapping.get(meal_type,
meal_type)
        user_info = await get_user_info(user_id)

        if not user_info or not user_info.get('fitness_goal'):
            return False

        recipe = await get_random_recipe_by_goal_and_meal(
            user_info['fitness_goal'],
            meal_type_in_db
        )

        if not recipe:
            print(f"Рецепт не найден для {user_id},
{meal_type_in_db}")
            return False

        # Создаем клавиатуру
        recipe_kb = InlineKeyboardBuilder()
        recipe_kb.button(
            text="➕ Добавить калории",
            callback_data=f"add_recipe_calories_{meal_type}"
        )
        recipe_kb.button(

```

```

        text="◀ К рецептам",
        callback_data="Recipe"
    )
    recipe_kb.button(
        text="🏠 Главное меню",
        callback_data="Menu"
    )
    recipe_kb.adjust(1) # Кнопки вертикально

    meal_name = meal_type_mapping.get(meal_type, meal_type)
    goal_name =
format_fitness_goal(user_info['fitness_goal'])

    message_text = (
        f"🍷 <b>Идея для {meal_name}</b> (цель:
<i>{goal_name}</i>)\n\n"
        f"<b>{recipe['dish_name']}</b>\n"
        f"💎 Калорийность: {recipe['calories']} ккал\n\n"
        f"<u>Способ приготовления:</u>\n"
        f"{recipe['preparation']}"
    )

    # Удаляем предыдущее уведомление
    prev_msg_id =
last_notification_ids[meal_type].get(user_id)
    if prev_msg_id:
        try:
            await bot.delete_message(chat_id=user_id,
message_id=prev_msg_id)
        except Exception:
            pass

    # Отправляем новое сообщение
    msg = await bot.send_message(
        user_id,
        message_text,
        parse_mode="HTML",
        reply_markup=recipe_kb.as_markup()
    )

    last_notification_ids[meal_type][user_id] =
msg.message_id
    return True

except Exception as e:
    print(f"Ошибка при отправке рецепта: {e}")
    return False

@dp.callback_query(F.data == "notification_settings")
async def notification_settings_menu(call: CallbackQuery):
    await call.message.delete()

```

```

        settings = await
get_user_notification_settings(call.from_user.id)
    if not settings:
        await call.answer("Настройки уведомлений не найдены")
        return

    status = {
        True: "✓ Вкл",
        False: "✗ Выкл"
    }

    # Создаем клавиатуру
    builder = InlineKeyboardBuilder()

    builder.button(text="🔔 Мотивационные",
callback_data="notif_motivation_menu")
    builder.button(text="📊 Статистика",
callback_data="notif_stats_menu")
    builder.button(text="🍳 Рецепты",
callback_data="notif_recipes_time_menu") # Теперь ведет прямо к
меню рецептов
    builder.button(text="⬅️ Назад", callback_data="Menu")
    builder.adjust(1, 1, 1, 1)

    await call.message.answer(
        f"⚙️ Настройки уведомлений:\n\n"
        f"🔔 Мотивационные:
{status[settings['motivation_enabled']] (каждые
{settings['motivation_interval']}ч)\n"
        f"📊 Статистика: {status[settings['stats_enabled']]
(({settings['stats_time1'].strftime('%H:%M')) и
{settings['stats_time2'].strftime('%H:%M')})\n"
        f"🍳 Рецепты: {status[settings['recipes_enabled']]}",
        reply_markup=builder.as_markup()
    )

@dp.callback_query(F.data == "notif_motivation_menu")
async def motivation_settings_menu(call: CallbackQuery):
    await call.message.delete()
    settings = await
get_user_notification_settings(call.from_user.id)
    if not settings:
        await call.answer("Настройки уведомлений не найдены")
        return

    status = "✓ Включены" if settings['motivation_enabled'] else
"✗ Выключены"

```

```

        await call.message.answer(
            f"🔔 Настройки мотивационных уведомлений:\n\n"
            f"Состояние: {status}\n"
            f"Интервал: каждые {settings['motivation_interval']}
часа",
            reply_markup=kb.motivation_settings_kb.as_markup()
        )

@dp.callback_query(F.data == "notif_stats_menu")
async def stats_settings_menu(call: CallbackQuery):
    await call.message.delete()
    settings = await
get_user_notification_settings(call.from_user.id)
    if not settings:
        await call.answer("Настройки уведомлений не найдены")
        return

    status = "✔ Включены" if settings['stats_enabled'] else "✗
Выключены"

    await call.message.answer(
        f"📊 Настройки уведомлений статистики:\n\n"
        f"Состояние: {status}\n"
        f"Время отправки:
{settings['stats_time1'].strftime('%H:%M')} и
{settings['stats_time2'].strftime('%H:%M')}",
        reply_markup=kb.stats_settings_kb.as_markup()
    )

@dp.callback_query(F.data == "notif_recipes_time_menu")
async def recipes_time_settings_menu(call: CallbackQuery):
    await call.message.delete()

    # Проверяем премиум статус
    if not await check_premium_status(call.from_user.id):
        await call.message.answer(
            "✗ Настройки рецептов доступны только с Premium
подпиской",
            reply_markup=kb.premium_upsell_kb.as_markup()
        )
        return

    settings = await
get_user_notification_settings(call.from_user.id)
    if not settings:
        await call.answer("Настройки уведомлений не найдены")
        return

    # Формируем текст с текущими настройками
    status_icon = "✔" if settings['recipes_enabled'] else "✗"

```

```

text = (
    f"{status_icon} Настройки уведомлений о рецептах:\n\n"
    f"☞ Завтрак:
{settings['breakfast_time'].strftime('%H:%M') if
settings['breakfast_time'] else 'не установлено'}\n"
    f"☞ Обед: {settings['lunch_time'].strftime('%H:%M') if
settings['lunch_time'] else 'не установлено'}\n"
    f"☞ Перекус: {settings['snack_time'].strftime('%H:%M')
if settings['snack_time'] else 'не установлено'}\n"
    f"☞ Ужин: {settings['dinner_time'].strftime('%H:%M') if
settings['dinner_time'] else 'не установлено'}\n\n"
    "Выберите действие:"
)

await call.message.answer(
    text,
    reply_markup=kb.recipe_time_change_kb.as_markup()
)

@dp.callback_query(F.data.startswith("notif_setting_"))
async def change_notification_setting(call: CallbackQuery,
state: FSMContext):
    setting_type = call.data.split("_")[2]
    await call.message.delete()

    if setting_type == "motivation":
        await call.message.answer(
            "Настройки мотивационных уведомлений:\n"
            "1. Включить/выключить\n"
            "2. Изменить интервал",
            reply_markup=kb.motivation_settings_kb.as_markup()
        )
    elif setting_type == "stats":
        await call.message.answer(
            "Настройки уведомлений статистики:\n"
            "1. Включить/выключить\n"
            "2. Изменить время",
            reply_markup=kb.stats_settings_kb.as_markup()
        )
    elif setting_type == "recipes":
        await call.message.answer(
            "Настройки уведомлений с рецептами:\n"
            "1. Включить/выключить\n"
            "2. Изменить время",
            reply_markup=kb.recipes_settings_kb.as_markup()
        )

@dp.callback_query(F.data == "toggle_motivation")
async def toggle_motivation(call: CallbackQuery):
    settings = await
get_user_notification_settings(call.from_user.id)

```

```

    if not settings:
        await call.answer("Настройки уведомлений не найдены")
        return

    new_status = not settings['motivation_enabled']
    await update_notification_settings(call.from_user.id,
{'motivation_enabled': new_status})

    status = "включены" if new_status else "выключены"
    await call.answer(f"Мотивационные уведомления {status}")
    await motivation_settings_menu(call) # Обновляем меню

@dp.callback_query(F.data == "toggle_stats")
async def toggle_stats(call: CallbackQuery):
    settings = await
get_user_notification_settings(call.from_user.id)
    if not settings:
        await call.answer("Настройки уведомлений не найдены")
        return

    new_status = not settings['stats_enabled']
    await update_notification_settings(call.from_user.id,
{'stats_enabled': new_status})

    status = "включены" if new_status else "выключены"
    await call.answer(f"Уведомления статистики {status}")
    await stats_settings_menu(call)

@dp.callback_query(F.data == "toggle_recipes_notifications")
async def toggle_recipes_notifications(call: CallbackQuery):
    # Проверяем премиум статус
    if not await check_premium_status(call.from_user.id):
        await call.answer(
            "❌ Уведомления о рецептах доступны только с Premium
подпиской",
            show_alert=True
        )
        return

    # Получаем текущие настройки
    settings = await
get_user_notification_settings(call.from_user.id)
    if not settings:
        await call.answer("Ошибка: настройки не найдены",
show_alert=True)
        return

    # Меняем статус уведомлений
    new_status = not settings['recipes_enabled']
    await update_notification_settings(
        call.from_user.id,

```

```

        {'recipes_enabled': new_status}
    )

    # Обновляем сообщение
    await call.answer(f"Уведомления о рецептах {'включены' if
new_status else 'выключены'} ✓")
    await recipes_time_settings_menu(call) # Обновляем меню

@dp.callback_query(F.data == "change_motivation_interval")
async def change_motivation_interval_start(call: CallbackQuery,
state: FSMContext):
    await call.message.delete()
    await call.message.answer(
        "Введите новый интервал для мотивационных уведомлений (в
часах, от 1 до 24):",
        reply_markup=kb.back_to_notification_kb.as_markup()
    )
    await
state.set_state(NotificationStates.changing_motivation_interval)

@dp.message(NotificationStates.changing_motivation_interval)
async def process_motivation_interval(message: Message, state:
FSMContext):
    try:
        interval = int(message.text)
        if interval < 1 or interval > 24:
            raise ValueError

        await update_notification_settings(message.from_user.id,
{'motivation_interval': interval})
        await message.answer(
            f"Интервал мотивационных уведомлений изменен на
{interval} часов",
            reply_markup=kb.back_to_notification_kb.as_markup()
        )
        await state.clear()
    except ValueError:
        await message.answer(
            "Пожалуйста, введите число от 1 до 24",
            reply_markup=kb.back_to_notification_kb.as_markup()
        )

@dp.callback_query(F.data == "change_stats_time")
async def change_stats_time_start(call: CallbackQuery, state:
FSMContext):
    await call.message.delete()
    await call.message.answer(
        "Введите новое время для уведомлений статистики в
формате ЧЧ:ММ (например, 14:00):",
        reply_markup=kb.back_to_notification_kb.as_markup()
    )

```

```

        await
state.set_state(NotificationStates.changing_stats_time)

@dp.message(NotificationStates.changing_stats_time)
async def process_stats_time(message: Message, state:
FSMContext):
    try:
        time_obj = datetime.strptime(message.text,
"%H:%M").time()
        second_time = (datetime.combine(date.today(), time_obj)
+ timedelta(hours=8)).time()

        await update_notification_settings(message.from_user.id,
{
            'stats_time1': time_obj,
            'stats_time2': second_time
        })

        await message.answer(
            f"Время уведомлений статистики изменено на
{time_obj.strftime('%H:%M')} и "
            f"{second_time.strftime('%H:%M')}",
            reply_markup=kb.back_to_notification_kb.as_markup()
        )
        await state.clear()
    except ValueError:
        await message.answer(
            "Неверный формат времени. Пожалуйста, используйте
ЧЧ:ММ (например, 14:00)",
            reply_markup=kb.back_to_notification_kb.as_markup()
        )

@dp.callback_query(F.data == "change_recipes_time")
async def change_recipes_time_start(call: CallbackQuery, state:
FSMContext):
    await call.message.delete()
    await call.message.answer(
        "Выберите тип приема пищи для изменения времени:",
        reply_markup=kb.recipe_time_change_kb.as_markup()
    )
    await
state.set_state(NotificationStates.changing_recipes_time)

@dp.callback_query(F.data.startswith("change_time_"))
async def select_recipe_time_to_change(call: CallbackQuery,
state: FSMContext):
    meal_type = call.data.split("_")[2] # breakfast, lunch,
snack, dinner
    await state.update_data(meal_type=meal_type)

    await call.message.edit_text(

```

```

        f"Введите новое время для {meal_type_mapping[meal_type]}
в формате ЧЧ:ММ (например, 08:30):",
        reply_markup=kb.back_to_notification_kb.as_markup()
    )
    await
state.set_state(NotificationStates.changing_recipes_time)

@dp.message(NotificationStates.changing_recipes_time)
async def process_recipe_time_change(message: Message, state:
FSMContext):
    data = await state.get_data()
    meal_type = data.get('meal_type')

    if not meal_type:
        await message.answer("Ошибка: тип приёма пищи не
определён")
        await state.clear()
        return

    try:
        time_obj = datetime.strptime(message.text,
"%H:%M").time()
        field_name = f"{meal_type}_time"

        # Включаем уведомления, если они были выключены
        settings = await
get_user_notification_settings(message.from_user.id)
        updates = {field_name: time_obj}
        if not settings['recipes_enabled']:
            updates['recipes_enabled'] = True

        success = await update_notification_settings(
            message.from_user.id,
            updates
        )

        if success:
            await message.answer(
                f"✓ Время для {meal_type_mapping[meal_type]}
установлено на {time_obj.strftime('%H:%M')}\n"
                f"Уведомления о рецептах автоматически
включены",

reply_markup=kb.recipe_time_change_kb.as_markup()
            )
        else:
            await message.answer(
                "✗ Не удалось изменить время",

reply_markup=kb.recipe_time_change_kb.as_markup()
            )

```

```

except ValueError:
    await message.answer(
        "☹ Неверный формат времени. Используйте ЧЧ:ММ  

(например, 08:30)",
        reply_markup=kb.back_to_recipe_time_kb.as_markup()
    )

await state.clear()

@dp.callback_query(F.data.startswith("set_notif_"))
async def set_notification_mode(call: CallbackQuery):
    await call.message.delete()
    mode = call.data.split("_")[-1]

    notification_settings = {
        "motivation": {"notifications_enabled": False,
        "motivation_enabled": True},
        "stats": {"notifications_enabled": True,
        "motivation_enabled": False},
        "all": {"notifications_enabled": True,
        "motivation_enabled": True},
        "none": {"notifications_enabled": False,
        "motivation_enabled": False}
    }

    success, response = await
    update_user_stats(call.from_user.id,
    notification_settings[mode])
    if success:
        status_text = {
            "motivation": "только мотивационные уведомления",
            "stats": "только уведомления статистики",
            "all": "все уведомления",
            "none": "все уведомления выключены"
        }[mode]

        await call.message.answer(
            f"Настройки уведомлений успешно изменены:
{status_text}",
            reply_markup=kb.back_to_notification_kb.as_markup()
        )
    else:
        await call.message.answer(
            f"Ошибка при изменении настроек: {response}",
            reply_markup=kb.back_to_notification_kb.as_markup()
        )

async def init_user_notifications(user_id: int):
    """Initialize notification settings for new user"""
    conn = None
    try:

```

```

        conn = await get_db_connection()
        with conn.cursor() as cursor:
            cursor.execute("""
                INSERT INTO user_notifications (user_id)
                VALUES (%s)
                ON CONFLICT (user_id) DO NOTHING
            """, (user_id,))
            conn.commit()
    except Exception as e:
        print(f"Ошибка при инициализации уведомлений
пользователя: {e}")
        if conn:
            conn.rollback()
    finally:
        if conn:
            conn.close()

async def get_user_notification_settings(user_id: int) -> dict:
    """Get user notification settings"""
    conn = None
    try:
        conn = await get_db_connection()
        with conn.cursor() as cursor:
            cursor.execute("""
                SELECT * FROM user_notifications
                WHERE user_id = %s
            """, (user_id,))
            result = cursor.fetchone()
            if result:
                return {
                    'user_id': result[0],
                    'motivation_enabled': result[1],
                    'motivation_interval': result[2],
                    'stats_enabled': result[3],
                    'stats_time1': result[4],
                    'stats_time2': result[5],
                    'recipes_enabled': result[6],
                    'breakfast_time': result[7],
                    'lunch_time': result[8],
                    'snack_time': result[9],
                    'dinner_time': result[10],
                    'last_motivation': result[11],
                    'last_stats': result[12],
                    'last_breakfast': result[13],
                    'last_lunch': result[14],
                    'last_snack': result[15],
                    'last_dinner': result[16]
                }
            return None
    except Exception as e:
        print(f"Ошибка при получении настроек уведомлений: {e}")
        return None

```

```

        finally:
            if conn:
                conn.close()

async def update_notification_settings(user_id: int, updates:
dict):
    """Update user notification settings with proper time
handling"""
    conn = None
    try:
        conn = await get_db_connection()
        with conn.cursor() as cursor:
            set_clause = ", ".join([f"{field} = %s" for field in
updates.keys()])
            values = list(updates.values())
            values.append(user_id)

            cursor.execute(f"""
                UPDATE user_notifications
                SET {set_clause}
                WHERE user_id = %s
            """, values)
            conn.commit()
            return True
    except Exception as e:
        print(f"Ошибка при обновлении настроек уведомлений:
{e}")
        if conn:
            conn.rollback()
        return False
    finally:
        if conn:
            conn.close()

async def check_missing_notifications():
    """Check and fix users missing notification settings"""
    conn = None
    try:
        conn = await get_db_connection()
        with conn.cursor() as cursor:
            # Find users without notification settings
            cursor.execute("""
                SELECT u.user_id FROM users u
                LEFT JOIN user_notifications un ON u.user_id =
un.user_id
                WHERE un.user_id IS NULL
            """)
            users = cursor.fetchall()

            for (user_id,) in users:
                await init_user_notifications(user_id)

```

```

        print(f"Проверка уведомлений завершена. Добавлено
{len(users)} записей.")
    except Exception as e:
        print(f"Ошибка при проверке уведомлений: {e}")
    finally:
        if conn:
            conn.close()

@dp.callback_query(RegistrationStates.waiting_for_options,
F.data.startswith("enable_"))
async def process_notification_choice(call: CallbackQuery,
state: FSMContext):
    await call.message.delete()
    choice = call.data

    settings = {
        "enable_all_notifications": {
            "motivation_enabled": True,
            "stats_enabled": True,
            "recipes_enabled": True
        },
        "enable_motivation_only": {
            "motivation_enabled": True,
            "stats_enabled": False,
            "recipes_enabled": False
        },
        "enable_stats_only": {
            "motivation_enabled": False,
            "stats_enabled": True,
            "recipes_enabled": False
        },
        "disable_all_notifications": {
            "motivation_enabled": False,
            "stats_enabled": False,
            "recipes_enabled": False
        }
    }

    if choice in settings:
        await update_notification_settings(call.from_user.id,
settings[choice])

    # Переходим к выбору премиума
    await call.message.answer(
        "Хотите активировать Premium подписку?",

reply_markup=kb.premium_choice_registration_kb.as_markup()
    )
    await
state.set_state(RegistrationStates.waiting_for_premium)

@dp.callback_query(F.data == 'Step')

```

```

async def steps_menu(call: CallbackQuery):
    await call.message.delete()
    # Проверяем и создаем таблицу истории, если её нет
    await create_user_history_table(call.from_user.id)
    user_info = await get_user_info(call.from_user.id)
    if user_info:
        steps = user_info.get('steps', 0)
        steps_goal = user_info.get('steps_goal', 10000)
        remaining = max(0, steps_goal - steps) if steps_goal is
not None else 0

        await call.message.answer(
            f"📊 Ваша статистика шагов:\n"
            f"• Пройдено сегодня: {steps}\n"
            f"• Цель: {steps_goal}\n"
            f"• Осталось пройти: {remaining}\n\n"
            "Выберите действие:",
            reply_markup=kb.steps_actions_kb.as_markup()
        )

@dp.callback_query(F.data == 'Calory')
async def calories_menu(call: CallbackQuery):
    await call.message.delete()
    user_info = await get_user_info(call.from_user.id)
    if user_info:
        burned = user_info.get('calories_burned', 0)
        consumed = user_info.get('calories_consumed', 0)
        balance = consumed - burned if None not in (consumed,
burned) else 0
        calories_goal = user_info.get('calories_goal', 'не
указана')

        await call.message.answer(
            f"🔥 Ваша статистика калорий:\n"
            f"• Сожжено сегодня: {burned} / {calories_goal}
ккал\n"
            f"• Потреблено сегодня: {consumed}\n"
            f"• Баланс: {balance}\n"
            f"• Рекомендуемое потребление:
{user_info.get('recommended_caloric_intake', 'не рассчитано')}
ккал\n\n"
            "Выберите действие:",
            reply_markup=kb.calories_actions_kb.as_markup()
        )

@dp.callback_query(F.data == 'add_steps')
async def add_steps_start(call: CallbackQuery, state:
FSMContext):
    await call.message.delete()
    await call.message.answer("Введите количество шагов, которое
хотите добавить:",

```

```

        reply_markup=kb.step_kb.as_markup())
    await state.set_state(GoalsStates.adding_steps) # Changed
to adding_steps

@dp.callback_query(F.data == 'add_calories_burned')
async def add_calories_burned_start(call: CallbackQuery, state:
FSMContext):
    await call.message.delete()
    await call.message.answer(
        "Введите количество сожженных калорий:",
        reply_markup=kb.calory_kb.as_markup()
    )
    await state.set_state(GoalsStates.adding_calories)

@dp.message(GoalsStates.adding_calories)
async def process_added_calories(message: Message, state:
FSMContext):
    try:
        calories = int(message.text)
        if calories <= 0:
            await message.answer(
                "Введите положительное число:",
                reply_markup=kb.calory_kb.as_markup()
            )
            return

        user_info = await get_user_info(message.from_user.id)
        if user_info:
            new_total = user_info['calories_burned'] + calories
            success = await
update_user_stats(message.from_user.id, {'calories_burned':
new_total})

            if success:
                await message.answer(
                    f"Добавлено {calories} сожженных калорий!\n"
                    f"Всего сожжено: {new_total}",
                    reply_markup=kb.calory_kb.as_markup()
                )
            else:
                await message.answer(
                    "Ошибка при обновлении данных",
                    reply_markup=kb.calory_kb.as_markup()
                )
            await state.clear()
    except ValueError:
        await message.answer(
            "Пожалуйста, введите число:",
            reply_markup=kb.calory_kb.as_markup()
        )

@dp.message(GoalsStates.adding_steps)

```

```

async def process_added_steps(message: Message, state:
FSMContext):
    try:
        steps_to_add = int(message.text)
        if steps_to_add <= 0:
            await message.answer(
                "Пожалуйста, введите положительное число.",
                reply_markup=kb.step_kb.as_markup()
            )
            return # Оставляем состояние активным, НЕ очищаем
его

        success, added_steps, added_calories = await
add_steps_with_calories(message.from_user.id, steps_to_add)
        if success:
            await message.answer(
                f"Добавлено {added_steps} шагов!\n"
                f"Это эквивалентно {added_calories} сожженным
калориям.\n"
                f"Теперь ваш результат: {added_steps} шагов и
{added_calories} калорий.",
                reply_markup=kb.step_kb.as_markup()
            )
        else:
            await message.answer("Произошла ошибка при
обновлении данных.")

            # Очищаем состояние ТОЛЬКО после успешной обработки
            await state.clear()

    except ValueError:
        await message.answer(
            "Пожалуйста, введите корректное число.",
            reply_markup=kb.step_kb.as_markup()
        )
        # Не очищаем состояние, чтобы бот продолжал ждать ввод
        return

@dp.callback_query(F.data == 'change_steps_goal')
async def change_steps_goal_start(call: CallbackQuery, state:
FSMContext):
    await call.message.delete()
    user_info = await get_user_info(call.from_user.id)
    if user_info:
        await call.message.answer(f"Текущая цель по шагам:
{user_info['steps_goal']}\nВведите новую
цель:", reply_markup=kb.step_kb.as_markup())
        await state.set_state(GoalsStates.changing_steps_goal)

@dp.callback_query(F.data == 'change_calories_burned_goal')
async def change_calories_burned_goal_start(call: CallbackQuery,
state: FSMContext):

```

```

        await call.message.delete()
        await call.message.answer(
            "Введите новую цель по сжиганию калорий:",
            reply_markup=kb.calory_kb.as_markup()
        )
        await state.set_state(GoalsStates.changing_calories_goal)

@dp.message(GoalsStates.changing_calories_goal)
async def process_changed_calories_goal(message: Message, state:
FSMContext):
    try:
        new_goal = int(message.text)
        if new_goal <= 0:
            await message.answer(
                "Цель должна быть положительным числом:",
                reply_markup=kb.calory_kb.as_markup()
            )
            return

        success = await update_user_stats(message.from_user.id,
{'calories_goal': new_goal})
        if success:
            await message.answer(
                f"Цель по сжиганию калорий изменена на
{new_goal}!",
                reply_markup=kb.calory_kb.as_markup()
            )
        else:
            await message.answer(
                "Ошибка при обновлении цели",
                reply_markup=kb.calory_kb.as_markup()
            )
        await state.clear()
    except ValueError:
        await message.answer(
            "Пожалуйста, введите число:",
            reply_markup=kb.calory_kb.as_markup()
        )

@dp.message(GoalsStates.changing_steps_goal)
async def process_steps_goal_change(message: Message, state:
FSMContext):
    try:
        new_goal = int(message.text)
        if new_goal <= 0:
            await message.answer(
                "Цель должна быть положительным числом.",
                reply_markup=kb.step_kb.as_markup()
            )
        return # Оставляем состояние активным

```

```

        success = await update_user_stats(message.from_user.id,
{'steps_goal': new_goal})
        if success:
            await message.answer(
                f"Цель по шагам изменена на {new_goal}!",
                reply_markup=kb.step_kb.as_markup()
            )
        else:
            await message.answer(
                "Произошла ошибка при обновлении цели.",
                reply_markup=kb.step_kb.as_markup()
            )

# Очищаем состояние только после успешной обработки
await state.clear()

except ValueError:
    await message.answer(
        "Пожалуйста, введите корректное число.",
        reply_markup=kb.step_kb.as_markup()
    )
    # Не очищаем состояние, чтобы бот продолжал ждать ввод
    return

@dp.callback_query(F.data == 'Recipe')
async def recipe_menu(call: CallbackQuery):
    if not await check_premium_status(call.from_user.id):
        # Проверяем, были ли включены уведомления о рецептах
        settings = await
get_user_notification_settings(call.from_user.id)
        if settings and settings['recipes_enabled']:
            await
update_notification_settings(call.from_user.id,
{'recipes_enabled': False})
            await call.message.answer(
                "❗ Уведомления о рецептах отключены, так как у
вас нет активной Premium подписки."
            )
            await call.message.delete()
            await call.message.answer(
                "❌ Доступ к рецептам ограничен.\n"
                "Для доступа к рецептам оформите премиум-подписку.",
                reply_markup=kb.premium_upsell_kb.as_markup()
            )
        return

    await call.message.delete()
    await call.message.answer(
        "📖 Выберите тип приема пищи или просмотрите книгу
рецептов:",
        reply_markup=kb.recipe_main_kb.as_markup()
    )

```

```

)

@dp.callback_query(F.data.startswith("recipe_"))
async def handle_recipe_request(call: CallbackQuery, state:
FSMContext):
    # Разбираем callback_data
    parts = call.data.split("_")
    if len(parts) < 2:
        await call.answer("Неверный запрос")
        return

    action = parts[1]

    if action == "book":
        await show_recipe_book(call)
        return
    elif action == "another":
        await show_another_recipe(call, state)
        return

    # Если это не "book" и не "another", то это тип приема пищи
    meal_type = action
    meal_type_in_db = meal_type_mapping.get(meal_type,
meal_type)

    if not await check_premium_status(call.from_user.id):
        await call.message.answer(
            "❌ Доступ к рецептам ограничен\n"
            "Для доступа к рецептам оформите премиум-подписку.",
            reply_markup=kb.premium_upsell_kb.as_markup()
        )
        return

    user_info = await get_user_info(call.from_user.id)
    if not user_info or not user_info.get('fitness_goal'):
        await call.message.answer("Сначала установите фитнес-
цель в профиле")
        return

    fitness_goal = user_info['fitness_goal']
    recipe = await
get_random_recipe_by_goal_and_meal(fitness_goal,
meal_type_in_db)

    if not recipe:
        await call.message.answer(f"Рецепты для {meal_type} не
найжены")
        return

    # Сохраняем данные рецепта и тип приема пищи в FSM
    await state.update_data(

```

```

        recipe_calories=recipe['calories'],
        current_recipe=recipe,
        current_meal_type=meal_type_in_db,
        current_fitness_goal=fitness_goal
    )

    goal_display = format_fitness_goal(fitness_goal)
    response = (
        f"🍽️ <b>{meal_type_mapping.get(meal_type,
meal_type).capitalize()}</b> для цели
'<i>{goal_display}</i>'\\n\\n"
        f"<b>{recipe['dish_name']}</b>\\n"
        f"💎 Калорийность: {recipe['calories']} ккал\\n\\n"
        f"<u>Способ приготовления:</u>\\n"
        f"{recipe['preparation']}"
    )
    recipe_actions_kb = InlineKeyboardBuilder()
    recipe_actions_kb.button(
        text="🔄 Другой рецепт",
        callback_data="recipe_another"
    )
    recipe_actions_kb.button(
        text="Добавить в потребление калорий",
        callback_data=f"add_recipe_calories_{meal_type}" #
Добавляем meal_type
    )
    recipe_actions_kb.button(
        text="📖 Книга рецептов",
        callback_data="recipe_book"
    )
    recipe_actions_kb.button(
        text="◀ Назад",
        callback_data="Recipe"
    )
    recipe_actions_kb.adjust(1, 1, 2)

    # Удаляем предыдущее сообщение и отправляем новое
    await call.message.delete()
    await call.message.answer(
        response,
        reply_markup=recipe_actions_kb.as_markup(),
        parse_mode="HTML"
    )

@dp.callback_query(F.data == 'recipe_another')
async def show_another_recipe(call: CallbackQuery, state:
FSMContext):
    # Получаем данные из состояния (тип приема пищи и цель)
    data = await state.get_data()
    meal_type = data.get('current_meal_type')
    fitness_goal = data.get('current_fitness_goal')

```

```

    if not meal_type or not fitness_goal:
        await call.answer("Не удалось определить тип рецепта")
        return

    # Получаем новый случайный рецепт
    recipe = await
get_random_recipe_by_goal_and_meal(fitness_goal, meal_type)

    if not recipe:
        await call.answer("Рецепты не найдены")
        return

    # Обновляем данные в состоянии
    await state.update_data(
        recipe_calories=recipe['calories'],
        current_recipe=recipe
    )

    # Формируем текст сообщения
    meal_name = meal_type_mapping.get(meal_type, meal_type)
    goal_display = format_fitness_goal(fitness_goal)
    response = (
        f"📌 <b>{meal_name.capitalize()}</b> для цели
'<i>{goal_display}</i>'\\n\\n"
        f"<b>{recipe['dish_name']}</b>\\n"
        f"💎 Калорийность: {recipe['calories']} ккал\\n\\n"
        f"<u>Способ приготовления:</u>\\n"
        f"{recipe['preparation']}"
    )
    recipe_actions_kb = InlineKeyboardBuilder()
    recipe_actions_kb.button(
        text="🔄 Другой рецепт",
        callback_data="recipe_another"
    )
    recipe_actions_kb.button(
        text="Добавить в потребление калорий",
        callback_data=f"add_recipe_calories_{meal_type}" #
Добавляем meal_type
    )
    recipe_actions_kb.button(
        text="📖 Книга рецептов",
        callback_data="recipe_book"
    )
    recipe_actions_kb.button(
        text="◀ Назад",
        callback_data="Recipe"
    )
    recipe_actions_kb.adjust(1, 1, 2)
    # Редактируем существующее сообщение
    await call.message.delete()

```

```

        await call.message.answer(
            response,
            reply_markup=recipe_actions_kb.as_markup(),
            parse_mode="HTML"
        )

    async def show_recipe_book(call: CallbackQuery):
        user_info = await get_user_info(call.from_user.id)
        if not user_info or not user_info.get('fitness_goal'):
            await call.message.answer("Сначала установите фитнес-цель в профиле")
            return

        fitness_goal = user_info['fitness_goal']
        goal_display = format_fitness_goal(fitness_goal)
        recipes_by_meal = await
        get_recipes_grouped_by_meal(fitness_goal)

        if not recipes_by_meal:
            await call.message.answer("Рецепты не найдены")
            return

        response = f"📖 <b>Книга рецептов</b> для цели  
'<i>{goal_display}</i>':\n\n"

        for meal_type, recipes in recipes_by_meal.items():
            response += f"<b>🍽️ {meal_type.capitalize()}:</b>\n"
            for i, recipe in enumerate(recipes, 1):
                response += f"{i}. {recipe['dish_name']}\n"
            response += f"({recipe['calories']} ккал)\n"
            response += "\n"

        await call.message.edit_text(
            response,
            reply_markup=kb.recipe_type_kb.as_markup(),
            parse_mode="HTML"
        )

    async def get_random_recipe_by_goal_and_meal(fitness_goal: str,
        meal_type: str) -> Optional[dict]:
        """Получить случайный рецепт для цели и типа приема пищи"""
        conn = None
        try:
            conn = await get_db_connection()
            with conn.cursor() as cursor:
                cursor.execute("""
                    SELECT dish_name, calories, preparation
                    FROM recipes
                    WHERE fitness_goal = %s AND meal_type = %s
                    ORDER BY RANDOM()
                    LIMIT 1
                """)

```

```

        """", (fitness_goal, meal_type))
    result = cursor.fetchone()
    if result:
        # Заменяем \n на настоящие переносы строк
        preparation = result[2].replace('\n', '\n')
        return {
            'dish_name': result[0],
            'calories': result[1],
            'preparation': preparation
        }
    return None
except Exception as e:
    print(f"Ошибка при получении рецепта: {e}")
    return None
finally:
    if conn:
        conn.close()

async def get_recipes_grouped_by_meal(fitness_goal: str) -> dict:
    conn = None
    try:
        conn = await get_db_connection()
        with conn.cursor() as cursor:
            cursor.execute("""
                SELECT meal_type, dish_name, calories
                FROM recipes
                WHERE fitness_goal = %s
                ORDER BY meal_type, dish_name
            """, (fitness_goal,))

            recipes = cursor.fetchall()
            result = {}
            for meal_type, dish_name, calories in recipes:
                if meal_type not in result:
                    result[meal_type] = []
                result[meal_type].append({
                    'dish_name': dish_name,
                    'calories': calories
                })
            return result
    except Exception as e:
        print(f"Ошибка при получении рецептов: {e}")
        return {}
    finally:
        if conn:
            conn.close()

@dp.callback_query(F.data == 'add_recipe')
async def add_recipe_start(call: CallbackQuery, state: FSMContext):
    await call.message.delete()

```

```

        await call.message.answer(
            "Введите название блюда:",
            reply_markup=kb.back_recipe_kb.as_markup() # Правильный
формат передачи клавиатуры
        )
        await state.set_state(GoalsStates.adding_recipe_name)

@dp.message(GoalsStates.adding_recipe_name)
async def process_recipe_name(message: Message, state:
FSMContext):
    await state.update_data(dish_name=message.text)
    await message.answer(
        "Выберите фитнес-цель для этого рецепта:",
        reply_markup=kb.fitness_goal_for_recipe_kb.as_markup()
    )
    await state.set_state(GoalsStates.adding_recipe_goal)

@dp.callback_query(GoalsStates.adding_recipe_goal,
F.data.startswith("add_recipe_goal_"))
async def process_recipe_goal(call: CallbackQuery, state:
FSMContext):
    goal = call.data.split("_")[3] # теперь получаем из
add_recipe_goal_{goal}
    await state.update_data(fitness_goal=goal)
    await call.message.delete()
    await call.message.answer(
        "Выберите тип приема пищи:",
        reply_markup=kb.meal_type_kb.as_markup()
    )
    await state.set_state(GoalsStates.adding_recipe_meal_type)

@dp.callback_query(GoalsStates.adding_recipe_meal_type,
F.data.startswith("add_recipe_meal_"))
async def process_recipe_meal_type(call: CallbackQuery, state:
FSMContext):
    meal_type = call.data.split("_")[3] # теперь получаем из
add_recipe_meal_{meal_type}
    meal_type_mapping = {
        "breakfast": "завтрак",
        "lunch": "обед",
        "dinner": "ужин",
        "snack": "перекус"
    }

    if meal_type not in meal_type_mapping:
        await call.message.answer("Неизвестный тип приема пищи")
        return

    await
state.update_data(meal_type=meal_type_mapping[meal_type])
    await call.message.delete()

```

```

        await call.message.answer("Введите количество калорий в
блюде (число):", reply_markup=kb.back_recipe_kb.as_markup())
        await state.set_state(GoalsStates.adding_recipe_calories)

@dp.message(GoalsStates.adding_recipe_calories)
async def process_recipe_calories(message: Message, state:
FSMContext):
    try:
        calories = int(message.text)
        if calories <= 0:
            raise ValueError
        await state.update_data(calories=calories)
        await message.answer("Введите способ приготовления
(можно с переносами строк):",
reply_markup=kb.back_recipe_kb.as_markup() )
        await
state.set_state(GoalsStates.adding_recipe_preparation)
    except ValueError:
        await message.answer("Пожалуйста, введите корректное
число калорий (больше 0).",
reply_markup=kb.back_recipe_kb.as_markup() )

@dp.message(GoalsStates.adding_recipe_preparation)
async def process_recipe_preparation(message: Message, state:
FSMContext):
    recipe_data = await state.get_data()
    preparation = message.text

    # Сохраняем рецепт в базу данных
    success = await save_recipe_to_db(
        dish_name=recipe_data['dish_name'],
        fitness_goal=recipe_data['fitness_goal'],
        meal_type=recipe_data['meal_type'],
        calories=recipe_data['calories'],
        preparation=preparation
    )
    recipe_actions_kb = InlineKeyboardBuilder()
    recipe_actions_kb.button(
        text="📖 Книга рецептов",
        callback_data="recipe_book"
    )
    recipe_actions_kb.button(
        text="◀ Назад",
        callback_data="Recipe"
    )
    recipe_actions_kb.adjust(1, 1, 2)
    if success:
        await message.answer(
            "✅ Рецепт успешно добавлен!\n\n"
            f"<b>{recipe_data['dish_name']}</b>\n"

```

```

        f"◆ Для цели:
{format_fitness_goal(recipe_data['fitness_goal'])}\n"
        f"◆ Прием пищи: {recipe_data['meal_type']}\n"
        f"◆ Калорийность: {recipe_data['calories']}
ккал\n\n"
        f"<u>Способ приготовления:</u>\n{preparation}",
        reply_markup=recipe_actions_kb.as_markup(),
        parse_mode="HTML"
    )
else:
    await message.answer(
        "✗ Не удалось добавить рецепт. Попробуйте позже.",
        reply_markup=kb.start_kb.as_markup()
    )

await state.clear()

async def save_recipe_to_db(dish_name: str, fitness_goal: str,
meal_type: str, calories: int, preparation: str) -> bool:
    conn = None
    try:
        conn = await get_db_connection()
        with conn.cursor() as cursor:
            cursor.execute("""
                INSERT INTO recipes
                (dish_name, fitness_goal, meal_type, calories,
preparation)
                VALUES (%s, %s, %s, %s, %s)
            """, (dish_name, fitness_goal, meal_type, calories,
preparation))
            conn.commit()
            return True
    except Exception as e:
        print(f"Ошибка при добавлении рецепта: {e}")
        if conn:
            conn.rollback()
        return False
    finally:
        if conn:
            conn.close()

async def get_recipe_by_id(recipe_id: int) -> Optional[dict]:
    """Получить рецепт по ID"""
    conn = None
    try:
        conn = await get_db_connection()
        with conn.cursor() as cursor:
            cursor.execute("""
                SELECT dish_name, calories
                FROM recipes
                WHERE recipe_id = %s
            """)

```

```

        """", (recipe_id,))
        result = cursor.fetchone()
        if result:
            return {
                'dish_name': result[0],
                'calories': result[1]
            }
        return None
    except Exception as e:
        print(f"Ошибка при получении рецепта: {e}")
        return None
    finally:
        if conn:
            conn.close()

async def show_recipe_details(call: CallbackQuery, recipe:
dict):
    """Показать детали рецепта с кнопкой добавления калорий"""
    kb = InlineKeyboardBuilder()
    kb.button(
        text=f"Добавить {recipe['calories']} ккал к
потреблению",

callback_data=f"add_recipe_calories_{recipe['recipe_id']}"
    )
    kb.button(text="Назад", callback_data="Recipe")
    kb.adjust(1)

    await call.message.answer(
        f"🍴 <b>{recipe['dish_name']}</b>\n"
        f"💎 Калорийность: {recipe['calories']} ккал\n"
        f"💎 Тип: {recipe['meal_type']}\n\n"
        f"<u>Способ
приготовления:</u>\n{recipe['preparation']}",
        reply_markup=kb.as_markup(),
        parse_mode="HTML"
    )

@dp.callback_query(F.data == 'add_calories_consumed')
async def add_calories_consumed_start(call: CallbackQuery,
state: FSMContext):
    await call.message.delete()
    await call.message.answer(
        "Введите количество потребленных калорий:",
        reply_markup=kb.calory_kb.as_markup()
    )
    await state.set_state(GoalsStates.adding_calories_consumed)

@dp.message(GoalsStates.adding_calories_consumed)
async def process_added_calories_consumed(message: Message,
state: FSMContext):

```

```

try:
    calories = int(message.text)
    if calories <= 0:
        await message.answer(
            "Введите положительное число:",
            reply_markup=kb.calory_kb.as_markup()
        )
        return

    user_info = await get_user_info(message.from_user.id)
    if user_info:
        new_total = user_info['calories_consumed'] +
calories
        success = await
update_user_stats(message.from_user.id, {'calories_consumed':
new_total})

        if success:
            await message.answer(
                f"Добавлено {calories} потребленных
калорий!\n"
                f"Всего потреблено: {new_total}",
                reply_markup=kb.calory_kb.as_markup()
            )
        else:
            await message.answer(
                "Ошибка при обновлении данных",
                reply_markup=kb.calory_kb.as_markup()
            )
        await state.clear()
except ValueError:
    await message.answer(
        "Пожалуйста, введите число:",
        reply_markup=kb.calory_kb.as_markup()
    )

@dp.callback_query(F.data == 'change_calories_consumed_goal')
async def change_calories_consumed_goal_start(call:
CallbackQuery, state: FSMContext):
    await call.message.delete()
    await call.message.answer(
        "Введите новую цель по потреблению калорий:",
        reply_markup=kb.calory_kb.as_markup()
    )
    await
state.set_state(GoalsStates.changing_calories_consumed_goal)

@dp.message(GoalsStates.changing_calories_consumed_goal)
async def process_changed_calories_consumed_goal(message:
Message, state: FSMContext):
    try:
        new_goal = int(message.text)

```

```

        if new_goal <= 0:
            await message.answer(
                "Цель должна быть положительным числом:",
                reply_markup=kb.calory_kb.as_markup()
            )
            return

        success = await update_user_stats(message.from_user.id,
{'recommended_caloric_intake': new_goal})
        if success:
            await message.answer(
                f"Цель по потреблению калорий изменена на
{new_goal}!",
                reply_markup=kb.calory_kb.as_markup()
            )
        else:
            await message.answer(
                "Ошибка при обновлении цели",
                reply_markup=kb.calory_kb.as_markup()
            )
        await state.clear()
    except ValueError:
        await message.answer(
            "Пожалуйста, введите число:",
            reply_markup=kb.calory_kb.as_markup()
        )

@dp.callback_query(F.data.startswith("add_recipe_calories_"))
async def add_recipe_calories(call: CallbackQuery, state:
FSMContext):
    meal_type = call.data.split("_")[-1] # Получаем тип приема
пищи
    user_info = await get_user_info(call.from_user.id)

    if user_info:
        # Получаем калорийность последнего рецепта
        recipe = await get_random_recipe_by_goal_and_meal(
            user_info['fitness_goal'],
            meal_type_mapping.get(meal_type, meal_type)
        )

        if recipe:
            new_calories = user_info['calories_consumed'] +
recipe['calories']
            await update_user_stats(call.from_user.id, {
                'calories_consumed': new_calories
            })

            await call.message.edit_text(
                f"✓ Добавлено {recipe['calories']} ккал!\n"
                f"Общее потребление: {new_calories} ккал",

```

```

        reply_markup=InlineKeyboardBuilder()
        .button(text="◀ Назад", callback_data="Recipe")
        .as_markup()
    )
    else:
        await call.answer("Не удалось добавить калории",
show_alert=True)
    else:
        await call.answer("Сначала пройдите регистрацию",
show_alert=True)

@dp.callback_query(F.data == 'Training')
async def training(call: CallbackQuery):
    if not await check_premium_status(call.from_user.id):
        await call.message.delete()
        await call.message.answer(
            "❌ Доступ к тренировкам ограничен.\n"
            "Для доступа к тренировкам оформите премиум-
подписку.",
            reply_markup=kb.premium_upsell_kb.as_markup()
        )
    return

    await call.message.delete()
    await call.message.answer(text=f"Выберите уровень сложности
тренировок:",

reply_markup=kb.training_kb.as_markup(resize_keyboard=True))

@dp.callback_query(F.data == 'stretch')
async def stretch(call: CallbackQuery):
    await call.message.delete()
    video = vd.stretch
    await call.message.answer_video(
        video=video,
        caption='Растяжка\n\nСожжено калорий: ~400',

reply_markup=kb.stretch_kb.as_markup(resize_keyboard=True)
    )

@dp.callback_query(F.data == 'Easy')
async def easy(call: CallbackQuery, state: FSMContext):
    await call.message.delete()
    await state.update_data(difficulty='Easy') # Сохраняем
уровень сложности
    await call.message.answer(text='Выберите тип тренировки:',

reply_markup=kb.easy_kb.as_markup(resize_keyboard=True))

@dp.callback_query(F.data == 'Medium')
async def medium(call: CallbackQuery, state: FSMContext):

```

```

        await call.message.delete()
        await state.update_data(difficulty='Medium')
        await call.message.answer(text='Выберите тип тренировки:',

reply_markup=kb.medium_kb.as_markup(resize_keyboard=True))

@dp.callback_query(F.data == 'Hard')
async def hard(call: CallbackQuery, state: FSMContext):
    await call.message.delete()
    await state.update_data(difficulty='Hard')
    await call.message.answer(text='Выберите тип тренировки:',

reply_markup=kb.hard_kb.as_markup(resize_keyboard=True))

@dp.callback_query(F.data == 'easy_1')
async def easy_1(call: CallbackQuery):
    await call.message.delete()
    video = vd.easy_1
    await call.message.answer_video(video=video,
                                    caption=f'Кардио-
тренировка\n\nСожжено калорий: ~400',

reply_markup=kb.back_kb.as_markup(resize_keyboard=True))

@dp.callback_query(F.data == 'easy_2')
async def easy_2(call : CallbackQuery):
    await call.message.delete()
    video = vd.easy_2
    await call.message.answer_video(video=video,
                                    caption=f'Кардио-тренировка
на скорость\n\nСожжено калорий: ~400',

reply_markup=kb.back_kb.as_markup(resize_keyboard=True))

@dp.callback_query(F.data == 'easy_3')
async def easy_3(call : CallbackQuery):
    await call.message.delete()
    video = vd.easy_3
    await call.message.answer_video(video=video,
                                    caption=f'Аэробно-силовая
тренировка для всего тела\n\nСожжено калорий: ~400',

reply_markup=kb.back_kb.as_markup(resize_keyboard=True))

@dp.callback_query(F.data == 'easy_4')
async def easy_4(call : CallbackQuery):
    await call.message.delete()
    video = vd.easy_4
    await call.message.answer_video(video=video,
                                    caption=f'Тренировка для
плоского живота и мышц кора\n\nСожжено калорий: ~400',

```

```

reply_markup=kb.back_kb.as_markup(resize_keyboard=True))

@dp.callback_query(F.data == 'easy_5')
async def easy_5(call : CallbackQuery):
    await call.message.delete()
    video = vd.easy_5
    await call.message.answer_video(video=video,
                                    caption=f'Тренировка для
бедер и ягодич\н\нСожжено калорий: ~400',

reply_markup=kb.back_kb.as_markup(resize_keyboard=True))

@dp.callback_query(F.data == 'medium_1')
async def easy_1(call : CallbackQuery):
    await call.message.delete()
    video = vd.medium_1
    await call.message.answer_video(video=video,
                                    caption=f'Кардио-тренировка
с упражнениями для тонуса мышц\н\нСожжено калорий: ~450',

reply_markup=kb.back_kb.as_markup(resize_keyboard=True))

@dp.callback_query(F.data == 'medium_2')
async def easy_1(call : CallbackQuery):
    await call.message.delete()
    video = vd.medium_2
    await call.message.answer_video(video=video,
                                    caption=f'Кардио-тренировка
на скорость (Версия №2)\н\нСожжено калорий: ~450',

reply_markup=kb.back_kb.as_markup(resize_keyboard=True))

@dp.callback_query(F.data == 'medium_3')
async def easy_1(call : CallbackQuery):
    await call.message.delete()
    video = vd.medium_3
    await call.message.answer_video(video=video,
                                    caption=f'Скоростная
тренировка для мышц кора\н\нСожжено калорий: ~450',

reply_markup=kb.back_kb.as_markup(resize_keyboard=True))

@dp.callback_query(F.data == 'medium_4')
async def easy_1(call : CallbackQuery):
    await call.message.delete()
    video = vd.medium_4
    await call.message.answer_video(video=video,
                                    caption=f'Аэробно-силовая
тренировка для всего тела\н\нСожжено калорий: ~450',

reply_markup=kb.back_kb.as_markup(resize_keyboard=True))

```

```

@dp.callback_query(F.data == 'medium_5')
async def easy_1(call : CallbackQuery):
    await call.message.delete()
    video = vd.medium_5
    await call.message.answer_video(video=video,
                                    caption=f'Динамическая
тренировка для плоского живота и мышц кора\n\nСожжено калорий:
~450',

reply_markup=kb.back_kb.as_markup(resize_keyboard=True))

@dp.callback_query(F.data == 'medium_6')
async def easy_1(call : CallbackQuery):
    await call.message.delete()
    video = vd.medium_6
    await call.message.answer_video(video=video,
                                    caption=f'Тренировка для
рук, плеч и спины\n\nСожжено калорий: ~450',

reply_markup=kb.back_kb.as_markup(resize_keyboard=True))

@dp.callback_query(F.data == 'hard_1')
async def hard_1(call : CallbackQuery):
    await call.message.delete()
    video = vd.hard_1
    await call.message.answer_video(video=video,
                                    caption=f'Кардио-тренировка
на скорость (Версия №3)\n\nСожжено калорий: ~500',

reply_markup=kb.back_kb.as_markup(resize_keyboard=True))

@dp.callback_query(F.data == 'hard_2')
async def hard_2(call : CallbackQuery):
    await call.message.delete()
    video = vd.hard_2
    await call.message.answer_video(video=video,
                                    caption=f'Аэробно-силовая
тренировка для всего тела (Экстримальная версия)\n\nСожжено
калорий: ~500',

reply_markup=kb.back_kb.as_markup(resize_keyboard=True))

@dp.callback_query(F.data == 'hard_3')
async def hard_3(call : CallbackQuery):
    await call.message.delete()
    video = vd.hard_3
    await call.message.answer_video(video=video,
                                    caption=f'Аэробно-силовая
тренировка для всего тела (программа "Пирамида")\n\nСожжено
калорий: ~500',

```

```

reply_markup=kb.back_kb.as_markup(resize_keyboard=True))

@dp.callback_query(F.data == 'hard_4')
async def hard_4(call : CallbackQuery):
    await call.message.delete()
    video = vd.hard_4
    await call.message.answer_video(video=video,
                                    caption=f'Тренировка для
рук, плеч и спины (более интенсивная нагрузка)\n\nСожжено
калорий: ~500',

reply_markup=kb.back_kb.as_markup(resize_keyboard=True))

@dp.callback_query(F.data == "add_training_calories")
async def add_training_calories(call: CallbackQuery, state:
FSMContext):
    data = await state.get_data()
    difficulty = data.get('difficulty')

    if difficulty == 'Easy':
        calories = 400
    elif difficulty == 'Medium':
        calories = 450
    else: # Hard
        calories = 500

    user_info = await get_user_info(call.from_user.id)
    if user_info:
        new_calories = user_info['calories_burned'] + calories
        success = await update_user_stats(call.from_user.id,
{'calories_burned': new_calories})

        if success:
            await call.message.delete()
            await call.message.answer(
                f"✓ Добавлено {calories} сожжённых калорий!\n"
                f"Теперь общее количество сожжённых калорий:
{new_calories}",
                reply_markup=kb.confirm_training_kb.as_markup()
            )
        else:
            await call.answer("Ошибка при обновлении данных",
show_alert=True)
        else:
            await call.answer("Сначала пройдите регистрацию",
show_alert=True)

@dp.callback_query(F.data == "add_stretch_calories")
async def add_stretch_calories(call: CallbackQuery):
    calories = 400 # Фиксированное значение для растяжки

```

```

user_info = await get_user_info(call.from_user.id)
if user_info:
    new_calories = user_info['calories_burned'] + calories
    success = await update_user_stats(call.from_user.id,
{'calories_burned': new_calories})

    if success:
        await call.message.delete()
        await call.message.answer(
            f"✔ Добавлено {calories} сожжённых калорий за
растяжку!\n"
            f"Теперь общее количество сожжённых калорий:
{new_calories}",
            reply_markup=kb.confirm_training_kb.as_markup()
        )
    else:
        await call.answer("Ошибка при обновлении данных",
show_alert=True)
    else:
        await call.answer("Сначала пройдите регистрацию",
show_alert=True)

@dp.callback_query()
async def debug_callback(call: CallbackQuery):
    print(f"Получен callback: {call.data}") # Проверьте, что
приходит
    await call.answer()

async def on_shutdown(notification_tasks=None):
    # Отменяем все задачи при выключении
    for task in notification_tasks.values():
        task.cancel()

async def notification_scheduler():
    """Основной планировщик уведомлений"""
    while True:
        try:
            print("\n🕒 Запуск проверки уведомлений...")
            await check_and_send_notifications()
        except Exception as e:
            print(f"❌ Ошибка в планировщике: {str(e)}")

            await asyncio.sleep(60) # Проверяем каждую минуту

async def check_and_send_recipe_notifications():
    """Проверяет и отправляет уведомления с рецептами по
расписанию"""
    now = datetime.now()
    current_time = now.time()

```

```

conn = None
try:
    conn = await get_db_connection()
    with conn.cursor() as cursor:
        cursor.execute("""
            SELECT user_id, breakfast_time, lunch_time,
snack_time, dinner_time
            FROM user_notifications
            WHERE recipes_enabled = TRUE
        """)
        users = cursor.fetchall()

        for user_id, breakfast_time, lunch_time, snack_time,
dinner_time in users:
            # Проверяем каждый прием пищи
            for meal_type, meal_time in [
                ('breakfast', breakfast_time),
                ('lunch', lunch_time),
                ('snack', snack_time),
                ('dinner', dinner_time)
            ]:
                if (current_time.hour == meal_time.hour and
                    current_time.minute == meal_time.minute):
                    await send_recipe_notification(user_id,
meal_type)

                    await asyncio.sleep(0.5)

except Exception as e:
    print(f"Ошибка в проверке рецептов: {str(e)}")
finally:
    if conn:
        conn.close()

async def check_and_send_notifications():
    """Проверяет и отправляет все типы уведомлений"""
    now = datetime.now()
    current_time = now.time()
    print(f"\n🔍 Проверка уведомлений в {current_time}")

    conn = None
    try:
        conn = await get_db_connection()
        with conn.cursor() as cursor:
            cursor.execute("""
                SELECT user_id,
                    motivation_enabled, motivation_interval,
                    stats_enabled, stats_time1, stats_time2,
lunch_time,
                    snack_time, dinner_time
                FROM user_notifications
                WHERE motivation_enabled = TRUE
            """)

```

```

        OR stats_enabled = TRUE
        OR recipes_enabled = TRUE
    """)
    users_settings = cursor.fetchall()

    for settings in users_settings:
        user_id = settings[0]
        print(f"\n👤 Обработка пользователя {user_id}")

        # Мотивационные уведомления
        if settings[1]: # motivation_enabled
            print("🔔 Проверка мотивационных уведомлений...")
            await send_motivational_notification(user_id)

        # Проверка статистики (если включена)
        if settings[3] and isinstance(settings[4], time) and
isinstance(settings[5],
time): # stats_enabled и проверка типа времени
            for stats_time in [settings[4], settings[5]]: #
time1 и time2
                if (current_time.hour == stats_time.hour and
                    current_time.minute ==
stats_time.minute):
                    await send_stats_notification(user_id)
                    await asyncio.sleep(0.3)

        # Проверка рецептов (если включены)
        if settings[6]: # recipes_enabled
            for meal_type, meal_time in zip(
                ['breakfast', 'lunch', 'snack',
'dinner'],
                [settings[7], settings[8], settings[9],
settings[10]]
            ):
                if isinstance(meal_time, time): # Проверка
типа времени
                    if (current_time.hour == meal_time.hour
and
                        current_time.minute ==
meal_time.minute):
                        await
send_recipe_notification(user_id, meal_type)
                        await asyncio.sleep(0.3)

    except Exception as e:
        print(f"❌ Ошибка проверки уведомлений: {str(e)}")
    finally:
        if conn:
            conn.close()

```

```

async def scheduler():
    while True:
        now = datetime.now()
        # Запланировать сброс на 00:00
        target_time = now.replace(hour=0, minute=0, second=0,
microsecond=0) + timedelta(days=1)
        wait_seconds = (target_time - now).total_seconds()

        await asyncio.sleep(wait_seconds)
        await reset_daily_stats()
        await check_expired_subscriptions() # Добавляем
проверку подписок

@dp.message(F.photo | F.video)
async def media_id(message: Message):
    if message.photo:
        file_id = message.photo[-1].file_id
        media_type = "Фото"
    elif message.video:
        file_id = message.video.file_id
        media_type = "Видео"

    await message.answer(f"{media_type}
ID:\n<code>{file_id}</code>")

async def main():
    await check_missing_notifications() # Проверяем при старте
    asyncio.create_task(scheduler())
    asyncio.create_task(notification_scheduler()) # Запускаем
планировщик уведомлений
    await bot.delete_webhook(drop_pending_updates=True)
    await dp.start_polling(bot)

if __name__ == "__main__":
    try:
        asyncio.run(main())
    finally:
        asyncio.run(on_shutdown())

```

Приложение В

(Обязательное)

Бизнес-проект «Разработка системы на основе telegram-бота для здоровья и
фитнеса»

Презентация

					ПГУ1.090301.38.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		209

Бизнес-проект: «Разработка системы на основе telegram-бота для здоровья и фитнеса»

Автор работы: студент группы 21ВВП2

Морозов К.В.

Научный руководитель: к.т.н., доцент кафедры ВТ

Карамышева Н.С.

Отсканируйте для демонстрации работы бота!



ПЕРЕД НАЧАЛОМ РАЗРАБОТКИ НЕОБХОДИМО:

- Проанализировать предметную область
- Выбрать стек технологий для разработки для телеграм-бота
- Расписать функционал для телеграм-бота

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.38.001 ПЗ

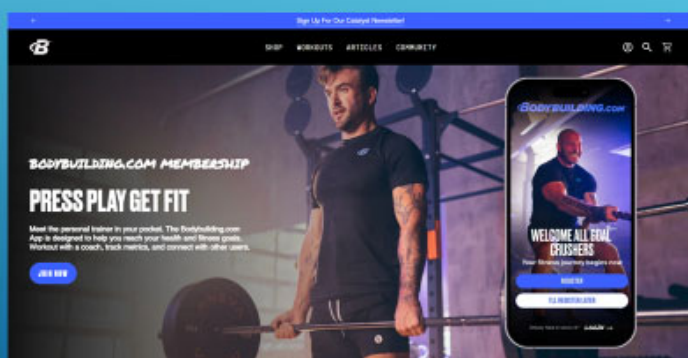
Лист

210

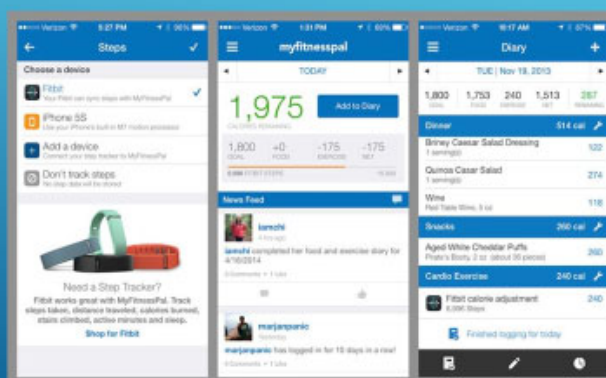
АНАЛОГИ

- Сайт «Bodybuilding.com»
- Приложение MyFitnessPal
- Устройство Apple Watch

САЙТ «BODYBUILDING.COM»



ПРИЛОЖЕНИЕ MYFITNESSPAL



Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.38.001 ПЗ

Лист

211

УСТРОЙСТВО APPLE WATCH



СТЕК ТЕХНОЛОГИЙ



Python

Язык
программирования



PostgreSQL

Система
управления
базами данных



PyCharm

Среда разработки



Telegram

Мессенджер



Aiogram

Библиотека

В ПРОЦЕССЕ РАЗРАБОТКИ ДОЛЖНЫ БЫТЬ РЕШЕНЫ СЛЕДУЮЩИЕ ЗАДАЧИ:

- Реализовать регистрацию с указанием личных данных, физических параметров и целей.
- Добавить возможность редактирования профиля и настройки целей (шаги, калории).
- Внедрить подсчет шагов, сожженных и потребленных калорий.
- Обеспечить хранение истории активности (7 дней) и расчет рекомендуемых норм.
- Создать базу видео-тренировок (3 уровня сложности) с расчетом калорий.
- Добавить библиотеку рецептов с фильтрацией по целям и учетом калорий.
- Настроить отправку мотивационных цитат, статистики и напоминаний (премиум).
- Внедрить ежедневные отчеты о прогрессе.
- Реализовать выбор тарифов, оплату и автоматическое ограничение функционала при отмене премиума.
- Обеспечить защиту от спама и неавторизованных действий.
- Настроить ежедневный сброс статистики и проверку подписок.
- Создать удобные разделы для управления данными, тренировками и питанием.

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.38.001 ПЗ

Лист

212

ФУНКЦИОНАЛ БОТА

- регистрация пользователей с указанием;
 - управление пользовательскими данными;
 - трекинг активности;
 - управление тренировками (премиум-функция);
 - работа с рецептами (премиум-функция);
 - система уведомлений;
 - аналитика:
- 1) расчет рекомендуемого потребления калорий;
 - 2) автоматический пересчет целей при изменении параметров;
- приобретение премиума:
 - 1) различные виды подписок;
 - 2) автоматическое изменение некоторых данных при отключении премиума;
 - технические функции:
 - 1) ежедневный автоматический сброс статистики;
 - 2) проверка истечения подписок.

ПРИ РАЗРАБОТКЕ ГРАФИЧЕСКОГО ИНТЕРФЕЙСА БОТА БЫЛИ ИСПОЛЬЗОВАНЫ СЛЕДУЮЩИЕ ЭЛЕМЕНТЫ:

- Текстовые сообщения
- Inline кнопки
- Всплывающие сообщения
- Вложения

ДИАГРАММА ПРЕЦЕДЕНТОВ ПОЛЬЗОВАТЕЛЯ



ДИАГРАММА ПОСЛЕДОВАТЕЛЬНОСТИ

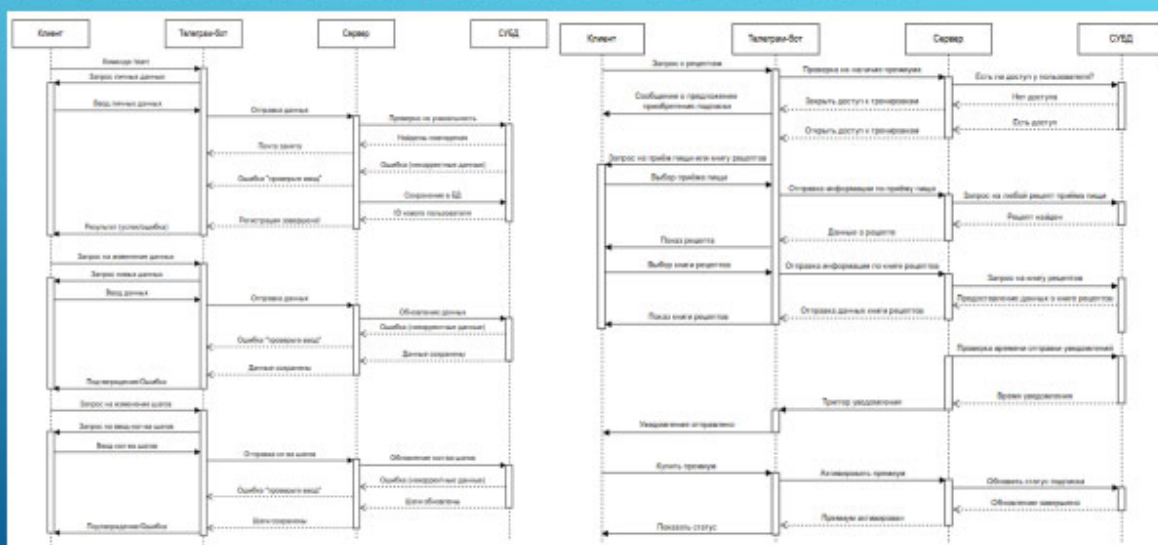


ДИАГРАММА ПОСЛЕДОВАТЕЛЬНОСТИ



ДИАГРАММА КЛАССОВ

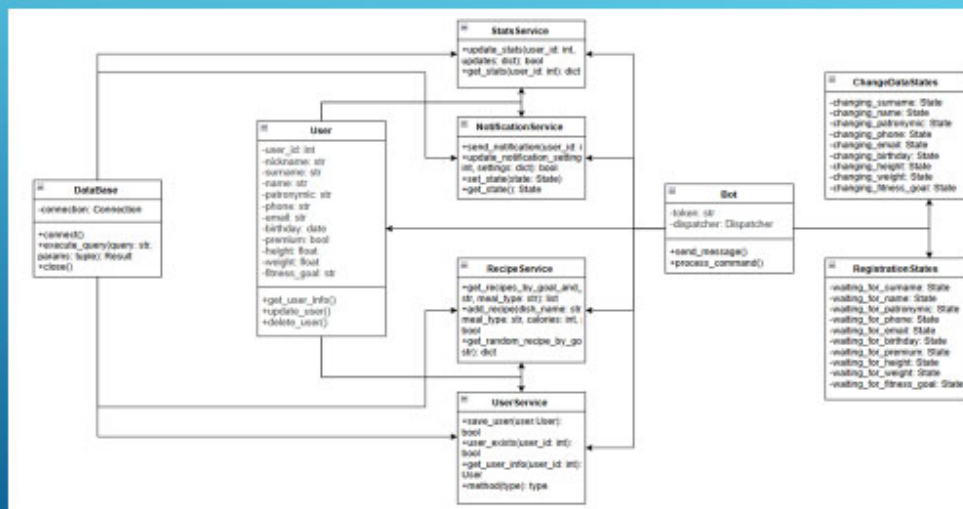


СХЕМА БАЗЫ ДАННЫХ

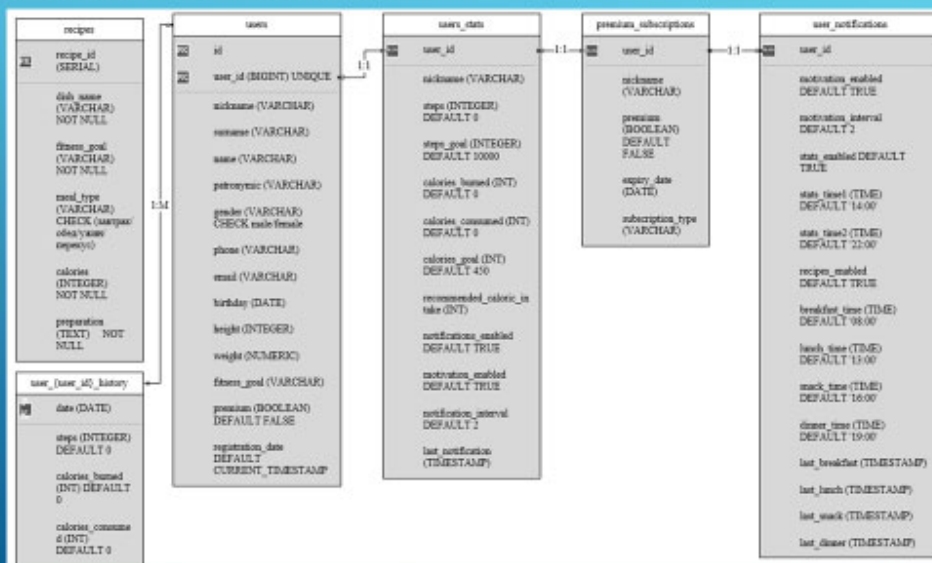
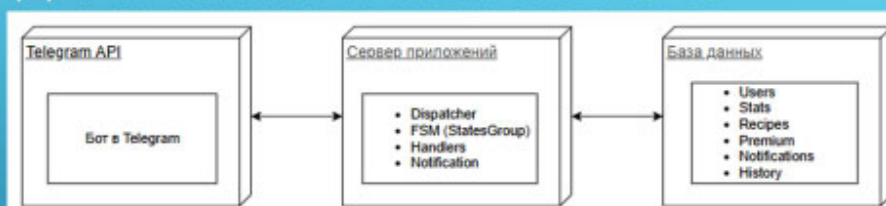


ДИАГРАММА РАЗВЁРТЫВАНИЯ



- Telegram API:
 - Бот — точка входа для пользователей, обрабатывает команды (/start, /menu и т.д.).
- Сервер приложений:
 - Python-приложение на фреймворке Aiogram:
 - Dispatcher — маршрутизация сообщений.
 - FSM (Finite State Machine) — управление состояниями (регистрация, настройки).
 - Handlers — обработчики команд (старт, тренировки, рецепты).
- База данных (PostgreSQL):
 - users — профили пользователей (вес, рост, цели).
 - recipes — база рецептов с фильтрацией по целям.
 - stats — статистика (шаги, калории).
 - premium — отслеживание премиум подписки.
 - notifications — настройки уведомлений.
 - history — хранение ежедневной статистики.

ЗАКЛЮЧЕНИЕ

Разработанная ИС способна выполнять основные требуемые функции.

Также возможно усовершенствовать разработанную автоматизированную систему, добавив:

- Взаимодействие бота с приложениями (Здоровье на iOS, fatsecret на Android);
- Составление собственных тренировок и возможность добавить свои тренировки;
- Добавление уведомлений тренировок;
- Возможность по наличию у пользователя определённых продуктов выдавать рецепты;
- Добавить новые параметры в рецептах. Например, название продуктов для рецепта, белки, жиры, углеводы.

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.38.001 ПЗ

Лист

216