

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

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

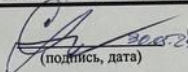
Направление подготовки: 09.03.01 «Информатика и вычислительная техника»
Профиль подготовки: «Программное обеспечение средств вычислительной
техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

**ТЕЛЕГРАММ-БОТ «ИНДИВИДУАЛЬНЫЙ
ФИТНЕС-ТРЕНЕР»**

Студент


(подпись, дата)

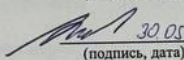
Сагатов Артём Каренович
(ФИО полностью)

Руководитель


(подпись, дата)

Зинкин С.А.
(фамилия, инициалы)

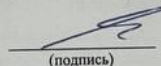
Нормоконтролёр


(подпись, дата)

Бычков А.С.
(фамилия, инициалы)

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

Заведующий кафедрой

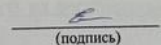

(подпись)

Митрохин М.А.
(фамилия, инициалы)

Работа защищена с отметкой отлично (протокол заседания ГЭК от

6 июня 2025 г. № 11)

Секретарь ГЭК


(подпись)

Гурин Е.И.
(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
« 6 » мая 2025 г.

ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ

1. Студент Сагатов Артём Каренович гр. 21ВВП1 факультета ФВТ

2. Тема работы: Телеграмм-бот «Индивидуальный фитнес-тренер»

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

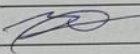
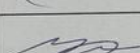
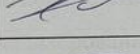
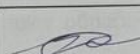
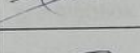
3. Руководитель бакалаврской работы Зинкин С.А.

4. Задание на работу (назначение разработки, исходные данные и т.п.)
Разработать автоматизированную информационную систему «Индивидуальный фитнес-тренер», использующую телеграм-бот. Язык программирования python, среда разработки Visual Studio Code. Программа должна обладать интуитивно понятным интерфейсом в мессенджере telegram и использовать следующие технологии:

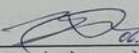
1. Telegram
2. Python-telegram-bot
3. Асинхронное программирование
4. Сетевое программирование
5. BeautifulSoup
6. SMTP

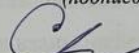
5. Перечень подлежащих разработке вопросов
Требуется произвести анализ предметных областей: изучить существующие информационные системы, выделить их достоинства и недостатки. Выявить потребности пользователей. Выбрать язык программирования, среду разработки, СУБД, хостинг которые подойдут для создания 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	5%	06.05.2025	
2	Анализ требований	9%	07.05.2025 – 08.05.2025	
3	Создание макета графического интерфейса	7%	09.05.2025 – 10.05.2025	
4	Выбор технологий, применяемых при проектировании и разработке	15%	11.05.2025 – 13.05.2025	
5	Реализация графического интерфейса	40%	14.05.2025 – 24.05.2025	
6	Разработка основной части программы	10%	25.05.2025 – 27.05.2025	
7	Оформление пояснительной записки	14%	28.05.2025 – 30.05.2025	

Дата выдачи задания "06" мая 2025 г.

Руководитель бакалаврской работы  06.05.2025 Зинкин С.А.
(подпись, дата)

Задание к исполнению принял студент  06.05.2025 Сагатов А.К.
(подпись, дата)

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

Реферат

Бакалаврская работа содержит 100 листов, 29 рисунков, 2 таблицы, 7 источников и 3 приложения.

ТЕЛЕГРАМ-БОТ, ФИТНЕС-КОНСУЛЬТАНТ, АВТОМАТИЗАЦИЯ, ОБРАБОТКА СОБЫТИЙ, ЛОГИРОВАНИЕ, БАЗА ДАННЫХ, API, ORM, SQLITE, ШИФРОВАНИЕ, PYTHON

Цель работы – разработка Telegram-бота для персонального фитнес-консультирования, позволяющего пользователю получать персонализированные тренировочные программы с целью помощи начинающим спортсменам. Для реализации данной цели разработана программа, которая объединяет обработку входящих сообщений, генерацию индивидуальных планов тренировок и динамичное взаимодействие с пользователями внутри мессенджера.

В рамках проекта были реализованы следующие задачи: организация работы системы с использованием учетных записей пользователей, формирование индивидуальных программ тренировок, реализация механизма обработки входящих команд и событий, обеспечение защиты конфигурационных данных посредством их шифрования, создание системы логирования.

В процессе разработки использовались современные технологии: язык программирования Python, ORM для взаимодействия с базой данных SQLite, API Telegram для организации обмена сообщениями, а также криптографические библиотеки для защиты конфиденциальной информации. В результате проведенной работы разработана чат-бот система с клиент-серверной архитектурой.

ПГУ1.090301.19.001 ПЗ

Изм.	Лист	№ докум.	Подпись	Дата	Лит.	Лист	Листов
Разраб.		Сагитов А.К.		30.05.21			
Провер.		Зинкин С.А.		30.05.21		2	100
Н. Контр		Бычков А.С.		30.05.21			
Утверд.		Митрохин М.А.		30.05.21			

Telegramm-бот «Индивидуальный фитнес-тренер»
Пояснительная записка

ФВТ, каф. ВТ
гр. 21ВВП1

Содержание

Перечень сокращений и обозначений	7
Введение	8
1 Телеграмм-бот «Индивидуальный фитнес-тренер».....	10
1.1 Анализ потребности	10
1.2 Анализ аналогов информационных систем	11
1.3 Постановка задачи	15
1.4 Анализ архитектуры и структуры бота.....	17
1.4.1 Общая концепция приложения.....	17
1.4.2 Функциональные требования	18
1.4.3 Требования к СУБД	24
1.5 Выбор инструментов разработки	26
1.5.1 Выбор языка программирования	27
1.5.2 Выбор СУБД.....	28
1.5.3 Выбранные технологии.....	30
1.5.4 Выбор IDE	31
1.6 Установка и настройка инструментов разработки	32
1.6.1 Установка Visual Studio Code	32
1.6.2 Настройка IDE	33
1.6.3 Создание проекта	33
1.6.4 Регистрация бота	36
1.7 Реализация бота.....	39

1.7.1 Создание БД	39
1.7.2 Описание функций приложения.....	42
1.8 Перенос проекта на виртуальный сервер.....	53
1.9 Описание финального архитектурного решения	56
1.10 Описание интерфейса приложения	57
2 Безопасность жизнедеятельности	65
2.1 Правила безопасной работы с персональным компьютером.....	65
2.2 Правила безопасного использования смартфона	66
Заключение.....	68
Список использованных источников.....	71
Приложение А – UML-диаграммы	72
Приложение Б – Листинг программы.....	83
Приложение В – Презентация	117

Перечень сокращений и обозначений

БД – база данных

СУБД – система управления базами данных

ACID (Atomicity, Consistency, Isolation, Durability) – свойства транзакций

API (Application Programming Interface) – интерфейс прикладного программирования

BMI (Body Mass Index) – индекс массы тела

ID – идентификатор

IDE (Integrated Development Environment) – интегрированная среда разработки

Inline-кнопки – интерактивные элементы управления

JSON (JavaScript Object Notation) – формат обмена данными

ORM (Object-Relational Mapping) – объектно-реляционное отображение

PHP (Hypertext Preprocessor) – препроцессор гипертекста

SQL (Structured Query Language) – язык структурированных запросов

SSH (Secure Shell)– криптографический сетевой протокол

UML (Unified Modeling Language) – унифицированный язык моделирования

URL (Uniform Resource Locator) – универсальный указатель ресурса

Ver (version) – версия

VS – Visual Studio

WinSCP (Windows Secure Copy) – программа для безопасной передачи файлов

Введение

В современном мире цифровые технологии становятся неотъемлемой частью повседневной жизни. С развитием мессенджеров и появлением новых платформ, таких как Telegram, у разработчиков появляется возможность создавать интерактивные решения, упрощающие выполнение различных задач и способствующие улучшению качества жизни. Одним из актуальных направлений является создание ботов, которые помогают пользователям получать персонализированные рекомендации в режиме реального времени.

Данная выпускная квалификационная работа посвящена разработке телеграм-бота «Индивидуальный фитнес-тренер», основная задача которого – предоставить пользователям удобный инструмент для составления индивидуальных фитнес-программ. Бот позволяет людям самостоятельно планировать тренировки, получать пошаговые инструкции по выполнению упражнений, а также предоставляет наглядное сопровождение в виде изображений, демонстрирующих правильную технику. Такой подход способствует более эффективному достижению поставленных целей и делает тренировки доступными вне зависимости от наличия персонального тренера.

С учетом растущей популярности здорового образа жизни и увеличивающегося спроса на персонализированные тренировки, данное решение отвечает потребности современного пользователя. Телеграм-бот «Индивидуальный фитнес-тренер» позволяет заниматься спортом в удобное время, самостоятельно настраивать интенсивность и режим тренировок, а также получать оперативную обратную связь. Приложение учитывает индивидуальные особенности пользователя, такие как рост и вес, и на основе этих данных формирует программу занятий, адаптированную под его физические возможности и цели.

Разработка данного приложения была выполнена с использованием Visual Studio Code, что позволило обеспечить гибкость и удобство в работе над проектом, а также интегрировать современные библиотеки и инструменты для создания надежного и масштабируемого решения. Такой выбор среды разработки гарантирует качественную реализацию проекта и возможность дальнейшего расширения функционала бота.

Цель данной работы заключается в создании функционального и удобного телеграм-бота, который станет незаменимым помощником для тех, кто стремится к здоровому образу жизни и эффективной организации тренировочного процесса.

1 Телеграмм-бот «Индивидуальный фитнес-тренер»

1.1 Анализ потребности

В современном мире становится всё больше людей, которые заботятся о своем здоровье и состоянии своего тела. Одним из важных факторов, весомо влияющих на самочувствие человека, является занятие спортом. Многие начинающие спортсмены, не имеющие абсолютно никакого опыта в этом деле, сталкиваются с трудностью составления плана индивидуальных занятий. Порой бывает сложно разобраться в обилии различных упражнений, выбрать самые необходимые для себя и правильно их скомпоновать друг с другом. Решить данную проблему и упростить вход в мир спорта для новичков помогут информационные технологии.

Одним из инструментов для составления индивидуального плана занятий спортсмена может быть телеграм-бот. Телеграм-бот является удобным решением, которое используется в различных сферах нашей жизни, в том числе в обучении, бизнесе и повседневных делах, помогая решать огромное количество различных задач. Кроме того, это отличная альтернатива мобильным приложениям. Во-первых, для пользователя нет необходимости устанавливать отдельное приложение, занимающее большое пространство в памяти на каждое свое устройство, достаточно иметь аккаунт в телеграм и доступ в интернет, чтобы использовать чат-бот. Во-вторых, чат-бот может быть удобно интегрирован в привычный интерфейс телеграма, что поможет начинающим спортсменам быстрее адаптироваться. Кроме того, чат-бот требует меньше финансовых вложений и усилий со стороны разработчиков, чем создание отдельного полноценного приложения под различные устройства. Также, чат-бот устраняет необходимость каждый раз обновлять приложение на девайсе. Это значит, что

обновления могут производиться незаметно для пользователя, отсутствует необходимость постоянно выгружать новые версии на различные платформы и отслеживать вытекающие отсюда жалобы пользователей на баги системы после обновления приложения.

Чат-бот система, которую я выполнил в данной работе, представляет собой удобный инструмент, позволяющий воплощать различные задачи спортсмена. Прежде всего, она позволяет подбирать программу тренировок по личным параметрам и желаниям пользователя, сохранять результаты каждой тренировки, корректировать план и объяснять правильную технику выполнения различных упражнений, что будет очень важно для начинающих пользователей.

1.2 Анализ аналогов информационных систем

Рассмотрение аналогичных решений является важной частью разработки любой программы. Это помогает выделить различные преимущества и недостатки, которые будут учтены во время создания собственной системы.

В качестве аналогов для нашего чат-бота я выбрал чат-бот «@gymcoach_bot» и мобильные приложения: «F&V», «ProFit», «Тренинг Про». Эти аналоги предоставляют возможность получить готовую программу тренировок, просматривать правильную технику выполнения упражнений, составлять собственную программу занятий.

В качестве первого аналога рассмотрим чат-бот «@gymcoach_bot» (рисунок 1.1). Сразу после запуска бота нас встречают приветственные сообщения. Затем, у нас запрашиваются такие параметры как: пол, место проведения занятий, уровень. Далее мы можем выбрать одну из трех готовых программ занятий. Этот бот не предоставляет функции по подбору плана занятий по физиологическим параметрам человека. Также, нет возможности корректировать программу, добавлять и убирать упражнения, есть только готовые

программы, заранее записанные на видео. Кроме того, нет возможности отслеживать прогресс занятий.

Приложение «F&V» (рисунок 1.2) дает возможность просмотреть большой список различных упражнений, выбрать большее количество готовых программ, создать личный план занятий. Главным недостатком является то, что доступ к готовым программам и составлению индивидуального плана занятий можно получить только после оплаты подписки. Также, как и в первом примере, в этом приложении нет ориентира на физиологические особенности человека.

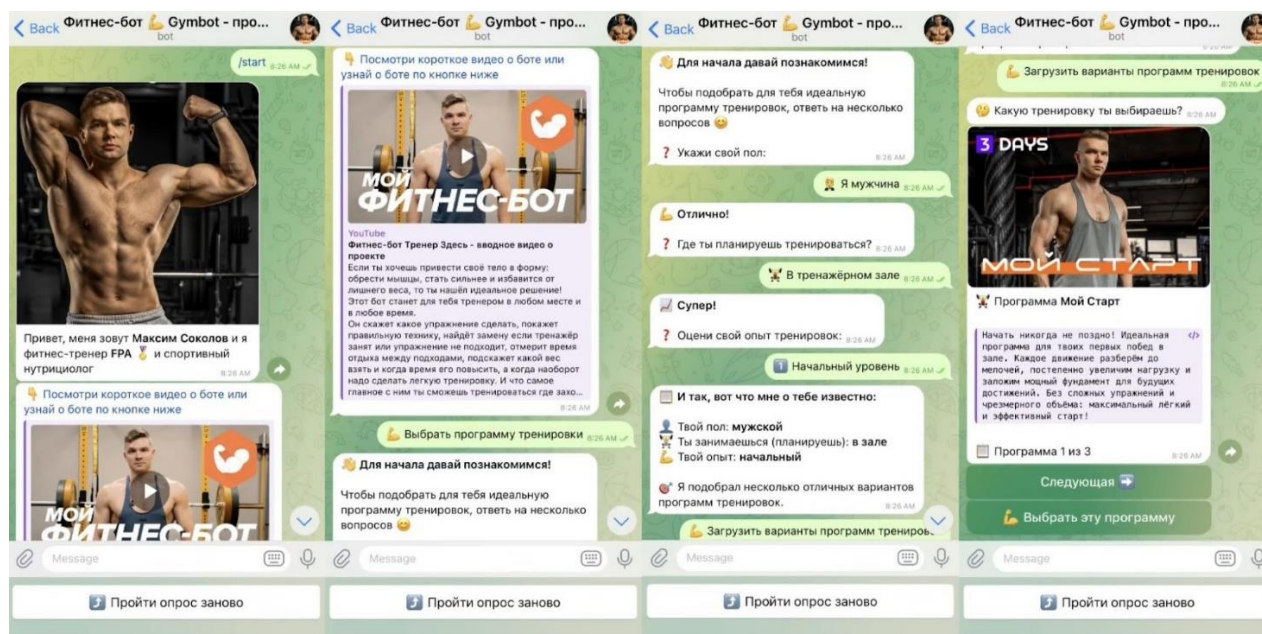


Рисунок 1.1 — Обзор аналога «@gymcoach_bot»

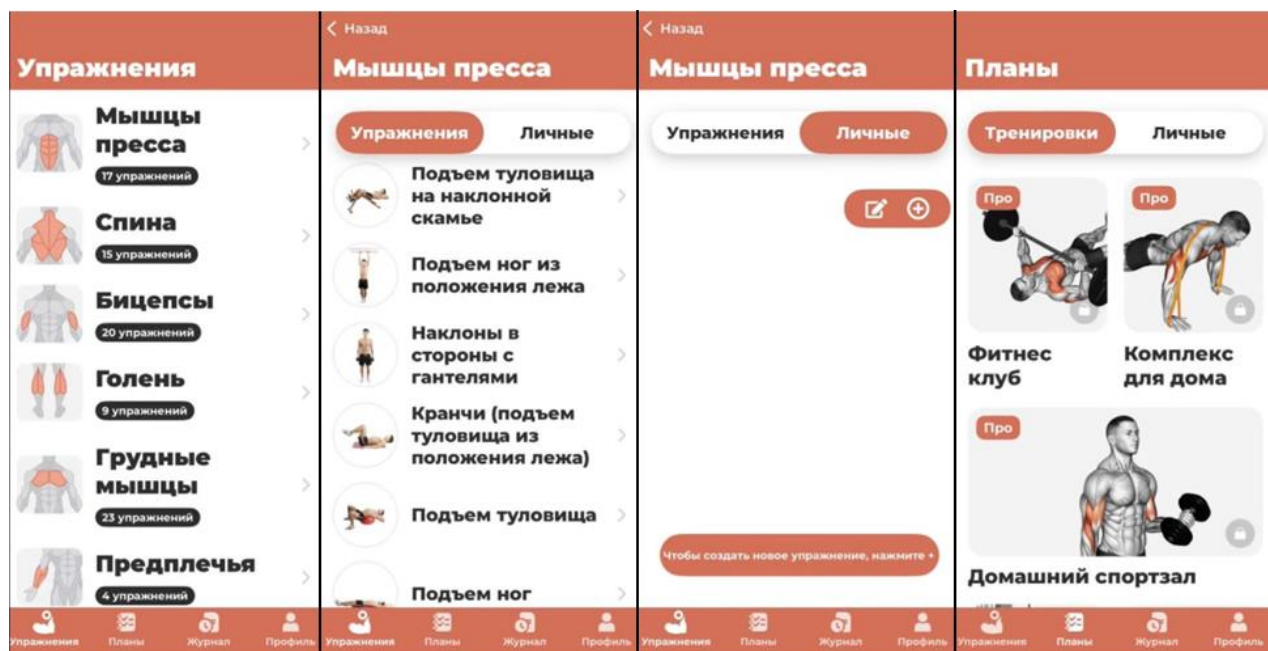


Рисунок 1.2 — Обзор аналога «F&V»

Следующий аналог — мобильное приложение «ProFit» (рисунок 1.3). Функционал этого приложения схож с «F&V», добавлена лишь возможность вести статистику прогресса занятий. Недостатками также является то, что готовый план занятий и возможность вести статистику открывается только после покупки подписки на приложение, программы не ориентируются на параметры человека.

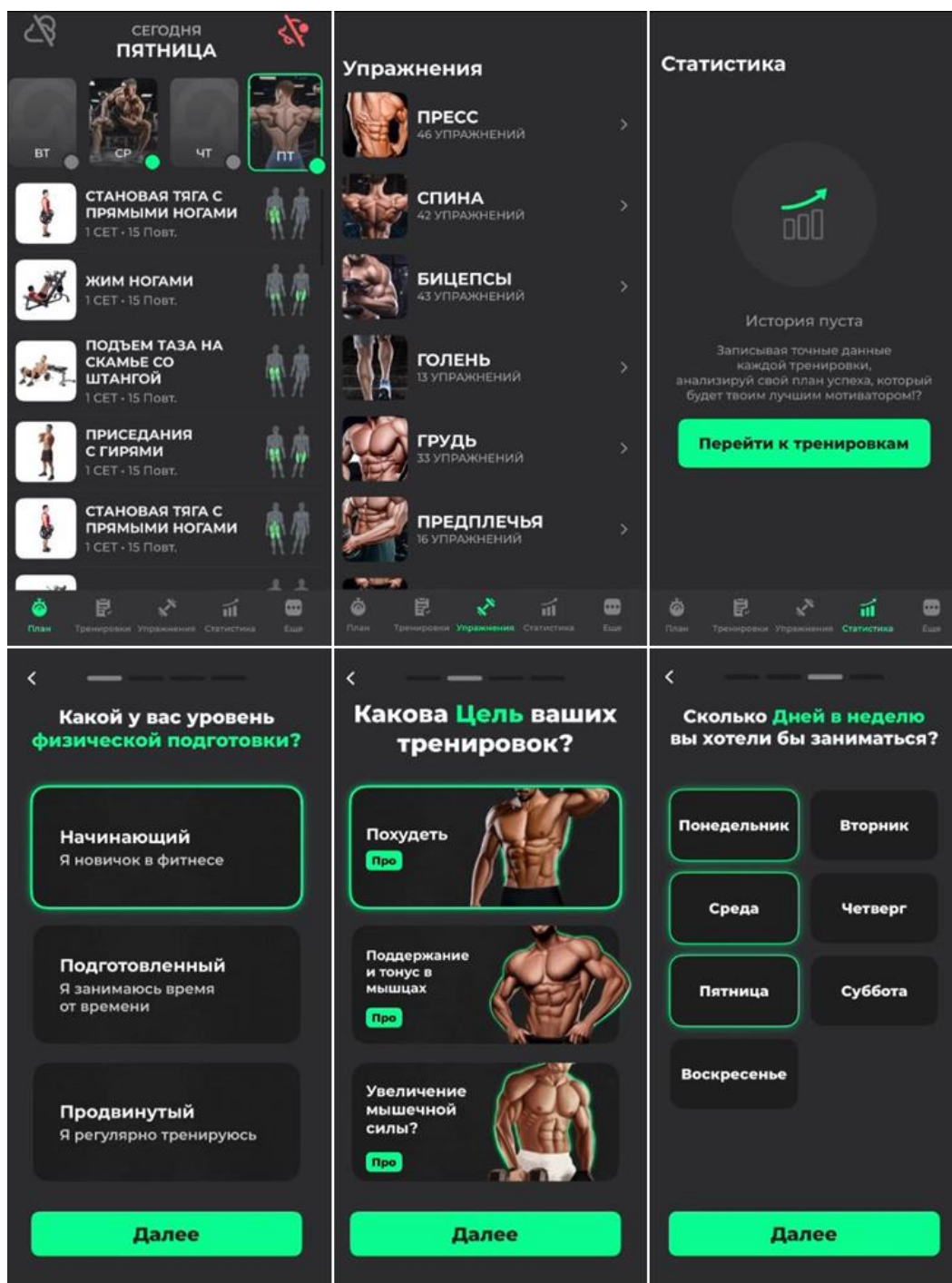


Рисунок 1.3 — Обзор аналога «ProFit»

В качестве следующего аналога мной было выбрано мобильное приложение «Тренинг Про» (рисунок 1.4). Здесь, как и в предыдущих аналогах, есть возможность выбрать программу, просмотреть список упражнений. Кроме того,

добавлена функция калькуляции различных параметров тела и рациона. Недостатком также является недоступность плана тренировок без покупки подписки на приложение. Кроме этого, планы тренировок, как и в прошлых аналогах, строятся на общих принципах построения программ, а не опирается на физиологические особенности конкретного человека.

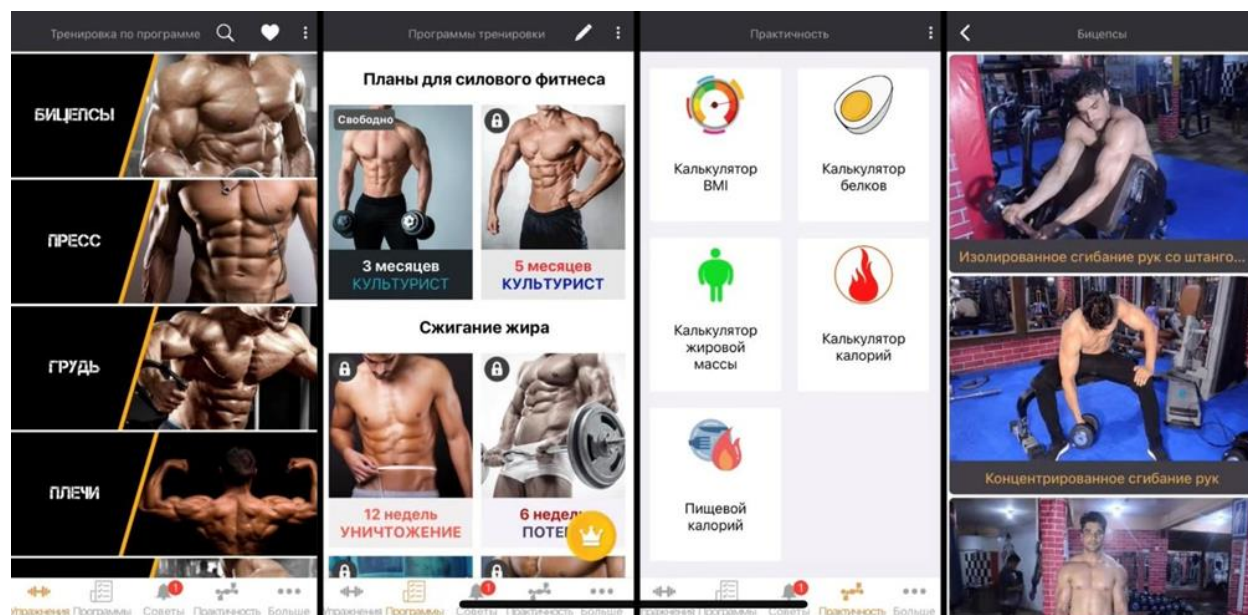


Рисунок 1.4 — Обзор аналога «Тренинг Про»

1.3 Постановка задачи

Необходимо разработать информационную систему в виде чат-бота для тренировок, который позволяет начинающим спортсменам выстраивать программу индивидуальных занятий. Система должна подбирать программу занятий на основе параметров человека. Это подразумевает за собой, что пользователь будет вводить свой рост, вес, тип телосложения и цель тренировок, а программа по этим вводным данным будет подбирать оптимальную схему занятий. Также приложение должно иметь возможность демонстрировать технику

выполнения упражнений, что очень важно для новичков, которые не знают, как правильно выполнять то или иное упражнение, что может вызвать травмы. Помимо этого, пользователь сможет корректировать программу по своему желанию, если ему некомфортно выполнять какое-то из движений или наоборот хочется добавить свое упражнение.

Основная цель системы — это помощь неопытным спортсменам в быстром и понятном старте занятий. Чат-бот должен быть интуитивно понятным и простым в использовании, чтобы новички не распространяли свое внимание на ненужные им вещи, которые очень часто мешают на начальных этапах для понимания основ построения программы тренировок.

Помимо этого, система должна быть простой в обслуживании, эффективной, и масштабируемой, чтобы в будущем можно было ее легко расширить, если это потребуется.

Исходя из глобальных задач, которые описывают общий функционал приложения, можно выделить более подробные задачи, которые предстоит учесть при разработке приложения:

- создание интуитивно понятного интерфейса для пользователя;
- нормализация текста для корректного сравнения и поиска данных;
- поиск локальных изображений и, при необходимости, обращение к внешнему API;
- формирование индивидуальной тренировочной программы на основе параметров пользователя;
- генерация интерактивных элементов (кнопок) для выбора упражнений;
- обработка запросов через Telegram API для динамического общения с пользователем;
- очистка временных данных для поддержания удобного интерфейса.

1.4 Анализ архитектуры и структуры бота

1.4.1 Общая концепция приложения

Архитектура телеграм-бота, представленного в данной работе, состоит из клиентской части, сервера и базы данных.

Клиентская часть представлена в виде интерфейса, с которым взаимодействует пользователь. Интерфейс предоставляет пользователю возможность внести свои параметры, выбрать предложенную программу тренировок, редактировать ее, вести статистику занятий и своих параметров. Подробный сценарий взаимодействия пользователя с клиентской частью бота можно увидеть на UML-диаграмме прецедентов (рисунок 1.5).

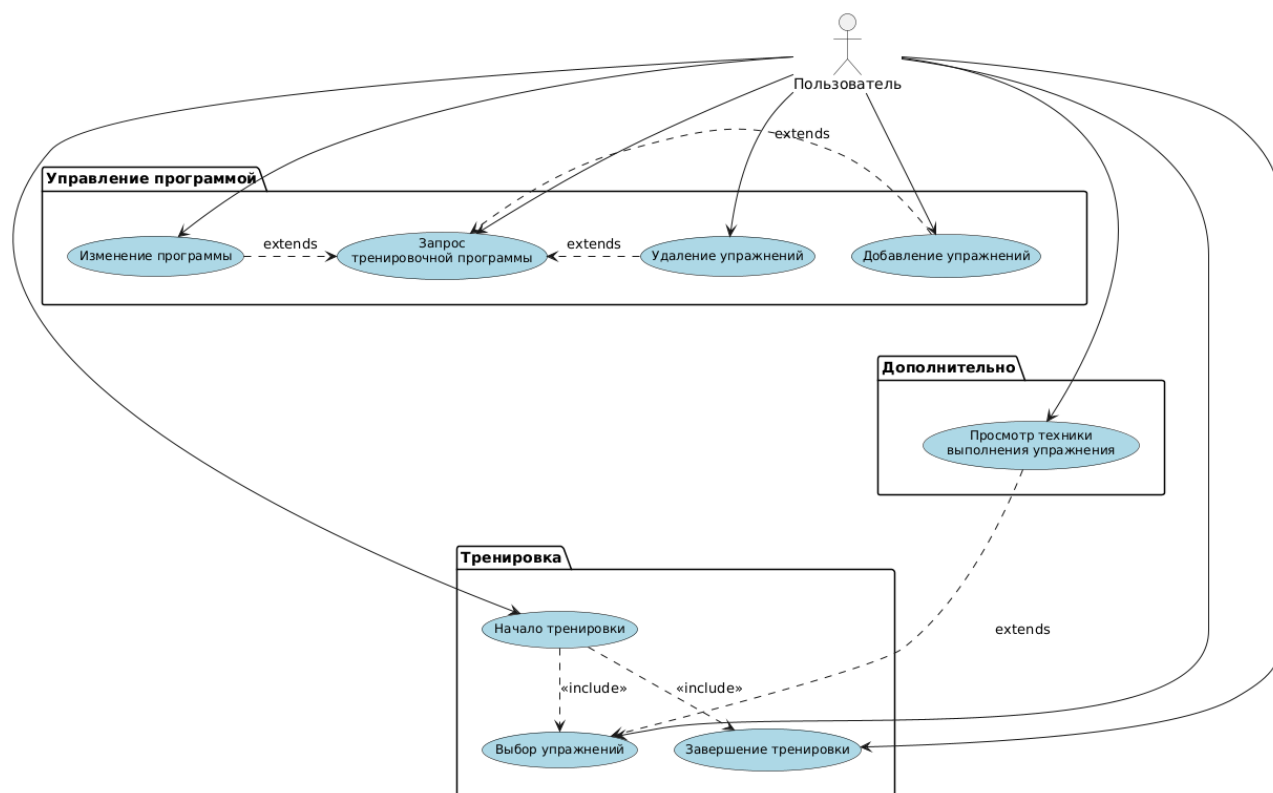


Рисунок 1.5 — UML-диаграмма прецедентов

Серверная часть отвечает за обработку данных, которые ввел пользователь. Далее сервер конвертирует полученные данные в нужный формат и запрашивает у базы данных подходящую программу упражнений, после чего отправляет ее в клиентскую часть приложения. Для более точного понимания взаимодействия частей бота создана UML-диаграмма последовательности (рисунок 1.6), которая описывает связь между клиентом, сервером и базой данных.

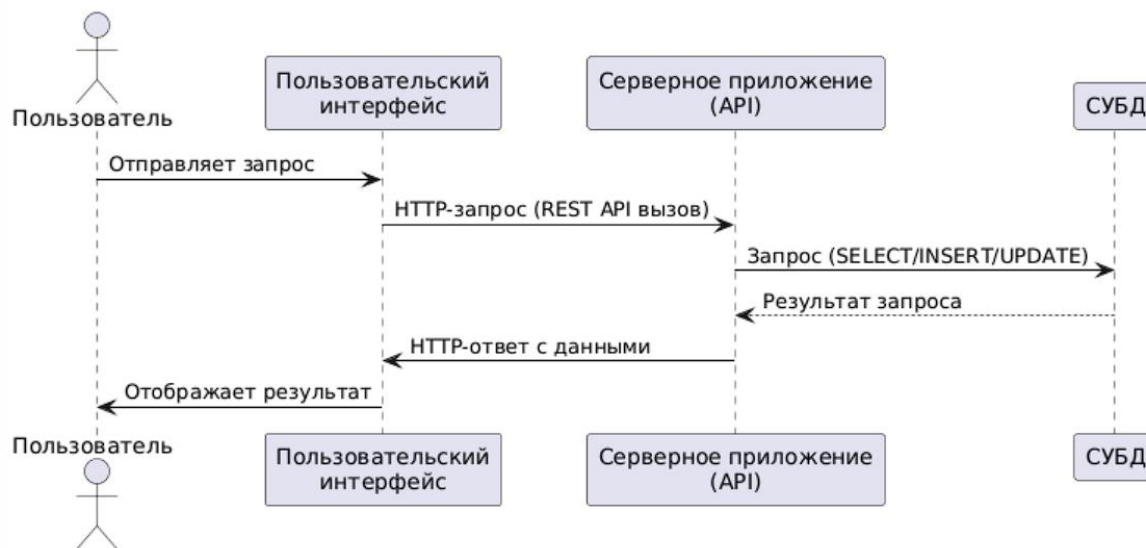


Рисунок 1.6 — UML-диаграмма последовательности

База данных хранит программы тренировок, которые сервер запрашивает по определенным параметрам, генерируемым исходя из параметров пользователя.

1.4.2 Функциональные требования

Создание функциональных требований помогает нам четко определить, что именно должен уметь делать наш фитнес-бот. Это позволяет зафиксировать конечные цели и задачи проекта. Такие требования служат ориентиром для дальнейшей разработки, позволяют оперативно оценивать, правильно ли реализованы поставленные функции, и помогают избежать недоразумений в

процессе интеграции отдельных компонентов. Взяв за основу общую архитектуру приложения, были выделены следующие функциональные требования к чат-боту:

Обработка входных данных.

```
normalize_text(s: str) → str
```

Задача: приведение пользовательского текста к стандартному виду для корректного сравнения и поиска. Требования:

- убрать начальные и конечные пробелы;
- привести весь текст к нижнему регистру;
- удалить все знаки препинания и специальные символы.

Формирование тренировочной программы.

```
get_training_program(height_cm: float, weight_kg: float, goal: str) → str
```

Задача: на основе введённых пользователем параметров (рост, вес, цель тренировок) формировать индивидуальную программу, извлекая данные из базы данных. Требования:

- определение типа телосложения (экторморф, мезоморф, эндоморф) на основе расчетного индекса массы тела (BMI).
- корректный поиск программы тренировок, соответствующей заданной цели (например, похудение, набор массы, рекомпозиция и т.д.), с использованием параметров запроса.
- обработка ситуации, когда программа не найдена – возврат информативного сообщения об отсутствии подходящей программы.

Поиск изображений для упражнений.

```
get_local_image(exercise_name: str) → str
```

Задача: поиск локального изображения, соответствующего названию упражнения. Требования:

- нормализовать входное название с помощью `normalize_text`;

- использовать словарь локальных изображений для поиска точного совпадения или наиболее подходящего варианта;

- проверить наличие файла по заданному пути (с использованием, например, `os.path.exists`).

```
get_exercise_images(exercise_name: str) → list
```

Задача: обеспечить пользователю изображение с техникой выполнения упражнения. Требования:

- сначала вызвать `get_local_image` для проверки наличия локального ресурса;

- если локальное изображение найдено, вернуть его в виде списка;

- если нет – выполнить запрос к внешнему API (например, Google Custom Search API) для получения списка URL-адресов изображений;

- обрабатывать возможные исключения и возвращать сообщение об ошибке в случае неудачи.

Обработка текста программы тренировок.

```
extract_exercises(text: str) → list
```

Задача: извлечение списка упражнений из форматированного текста программы. Требования:

- использовать регулярное выражение для поиска строк, начинающихся с порядкового номера;

- возвращать список названий упражнений без лишних символов.

```
generate_callback_data(exercise: str) → str
```

Задача: формирование уникального идентификатора для inline-кнопок Telegram, связанного с конкретным упражнением. Требования:

- преобразовать строку с названием упражнения, заменяя специальные символы;

- обеспечить, чтобы размер итоговых данных не превышал ограничение Telegram (не более 64 байт);

- итоговый идентификатор должен начинаться с префикса (например, «tech_ex_»).

Организация интерактивного взаимодействия через Telegram.

```
list_exercises_callback(update, context)
```

Задача: при запросе пользователем списка упражнений – извлечь текст программы, выделить список упражнений, сгенерировать для каждого уникальный идентификатор (с помощью `generate_callback_data`) и сформировать интерактивное меню с `inline`-кнопками. Требования:

- сохранить сопоставление уникальных идентификаторов с полными названиями упражнений в пользовательских данных для последующего восстановления;

- добавить кнопки навигации, например, «Вернуться к плану», «Завершить тренировку», «В главное меню».

```
tech_exercise_callback(update, context)
```

Задача: обработка выбора конкретного упражнения пользователем для получения изображений с техникой выполнения. Требования:

- определить полное название выбранного упражнения с использованием сохраненного сопоставления или извлечения части `callback_data`;

- вызвать функцию `get_exercise_images` для получения соответствующих изображений;

- отправить пользователю медиagруппу с изображениями и, при необходимости, сопровождающей подписью.

```
close_tech_callback(update, context)
```

Задача: Очистка временных сообщений, связанных с просмотром техники выполнения упражнения. Требования:

- удалить сообщения с изображениями и кнопками, используя соответствующие API Telegram;

- очистить данные, связанные с временным хранением идентификаторов сообщений, в пользовательском контексте.

`main()`

Задача: Инициализация и запуск Telegram-бота. Требования:

- зарегистрировать все обработчики команд и callback-запросов;
- запустить основной цикл (например, с использованием `asyncio.run(main())`) для обеспечения работы бота 24/7.

Все вышеописанные функциональные требования отображены на UML-диаграмме деятельности (рисунок 1.7):

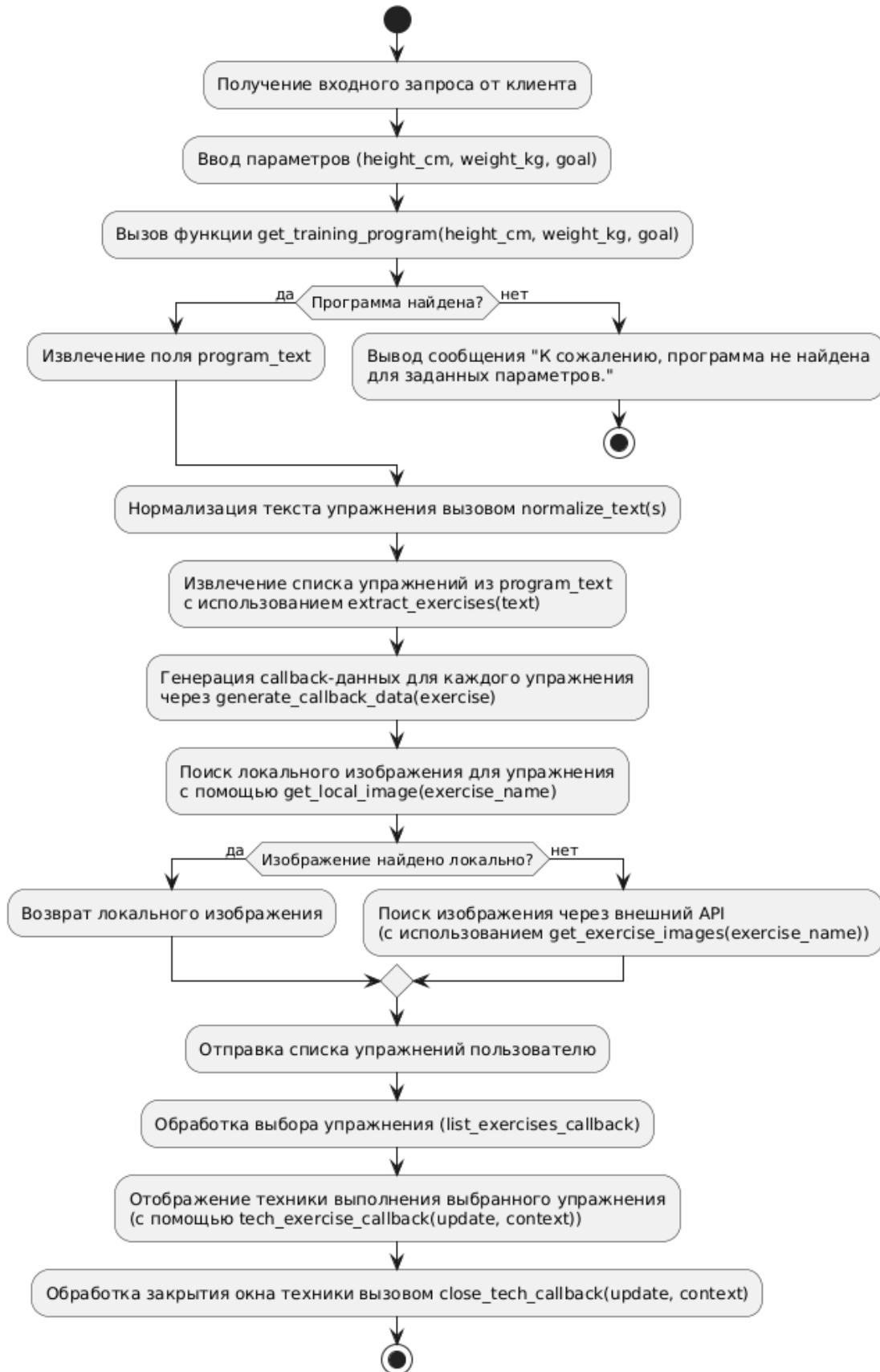


Рисунок 1.7 — UML-диаграмма деятельности

1.4.3 Требования к СУБД

Для работы нашего фитнес-бота необходимо создать базу данных, которая будет основой для формирования индивидуальных тренировочных программ и хранения информации об упражнениях. Важно заранее продумать ее структуру, чтобы позже легко интегрировать её с логикой бота. В нашем случае база должна позволять:

- хранить тренировочные программы. Например, программа для похудения или набора массы для определённого типа телосложения (экторморф, мезоморф или эндоморф);
- хранить связанные с программами упражнения. Каждая программа состоит из списка упражнений, и для каждого упражнения важно указать его название, а также опционально — путь к изображению, демонстрирующему правильную технику выполнения.

Такая архитектура позволит по запросу пользователя подбирать нужную программу и, при необходимости, отображать иллюстрации для корректного выполнения упражнений.

Проект должен содержать две основные таблицы: Programs и Exercises. Ниже приведена детальная спецификация каждой из них.

Таблица 1 — Атрибуты таблицы «Programs»

Название столбца	Тип данных	Краткое описание	Пример данных
id	INTEGER	Идентификатор программы	1
body_type	VARCHAR(50)	Тип телосложения (например, ectomorph)	«ectomorph»
goal	VARCHAR(50)	Цель тренировки	«loss»

program_text	TEXT	Текст программы	«Программа похудения для эктоморфа: ...»
--------------	------	-----------------	--

Таблица Programs предназначена для хранения информации о тренировочных программах. Каждая программа описывается типом телосложения, целью тренировки и полным текстовым описанием программы.

Эта таблица будет основной для хранения уникальных программ. Мы сможем быстро искать нужную программу по типу телосложения и цели тренировки, что окажется полезным при формировании индивидуальных планов.

Таблица Exercises будет содержать перечень упражнений для каждой программы. Здесь важно сохранить информацию о том, к какой программе относится упражнение, а также имя самого упражнения и, если есть, ссылку или путь к изображению, демонстрирующему технику выполнения.

Эта таблица агрегирует данные об отдельных упражнениях, входящих в программу. Именно через внешний ключ program_id мы будем связывать каждое упражнение с конкретной программой, что позволит удобно обновлять или редактировать содержание программы.

Таблица 2 — Атрибуты таблицы «Exercises»

Название столбца	Тип данных	Краткое описание	Пример данных
id	INTEGER	Идентификатор упражнения	1
program_id	INTEGER	Связь с программой из таблицы Programs	1 (ссылается на запись с id=1 в таблице Programs)
exercise_name	VARCHAR(100)	Название упражнения	«Приседания со штангой»
technique_image_url	VARCHAR(255)	Ссылка или путь на изображение упражнения	«http://example.com/bench_press.jpg» или локальный путь

Наглядное представление о том, как будет устроена база данных нашего фитнес-бота представлена на рисунке 1.8.

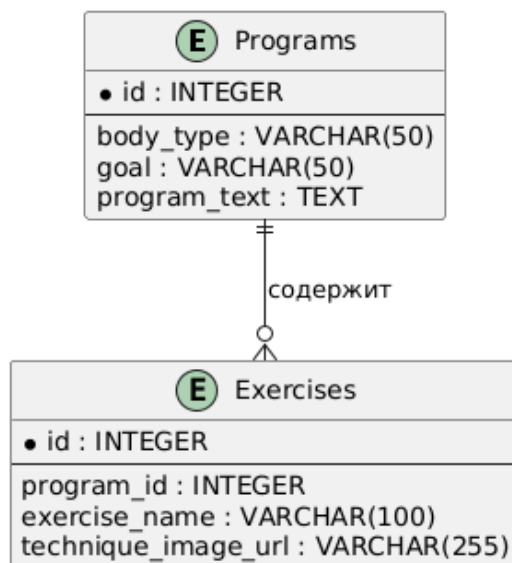


Рисунок 1.8 — Физическая схема БД

Схема демонстрирует, что одна программа может включать несколько упражнений, так как связи между таблицами реализованы по принципу «один-ко-многим». Это сделано для того, чтобы при запросе, к примеру, программы для похудения, быстро получить весь список упражнений, связанных с этой программой.

1.5 Выбор инструментов разработки

Грамотный подбор инструментов для разработки — это основа успешного создания информационной системы. Правильное решение позволяет оптимизировать весь процесс, значительно сократить время и затраты на разработку, а также обеспечить необходимое качество и безопасность конечного продукта. Необдуманный выбор технологий может обернуться проблемами как в

ходе создания, так и в эксплуатации системы, поэтому важно тщательно анализировать требования проекта перед принятием решения.

1.5.1 Выбор языка программирования

Каждый язык программирования обладает своим набором особенностей, существенно влияющих на итоговую эффективность реализации проекта. От тщательного подбора подходящего языка часто зависит успех всей системы.

Одним из наиболее удобных инструментов для разработки телеграм-ботов является Python. Этот язык отличается своим понятным и лаконичным синтаксисом, что упрощает и ускоряет процесс программирования.

Кроме того, для создания телеграм-ботов можно рассмотреть другие языки. Например, JavaScript широко применяется в веб-разработке, обладает динамической типизацией и синтаксисом, похожим на C. Однако, несмотря на обширный набор библиотек для работы с веб-интерфейсами, ему не хватает встроенных средств для научных вычислений и работы с матрицами, что может стать ограничением при реализации определённых функциональностей.

RНР, изначально разработанный для динамичных веб-приложений, также может применяться для создания ботов. Он удобен в использовании благодаря динамической типизации и схожести синтаксиса с C, но уступает Python в количестве специализированных библиотек и универсальности для разработки сложных алгоритмов.

Ещё один вариант — Java, широко используемая платформа для мобильных приложений, особенно на Android. Благодаря строгой типизации и компиляции кода, Java обеспечивает высокую производительность, хотя из-за объёма требуемого кода процесс разработки может оказаться более трудоёмким по сравнению с Python.

С учётом всех этих факторов, мой выбор пал на Python. Его простота, универсальность и большое количество библиотек делают его оптимальным для разработки телеграм-бота.

Кроме того, Python предлагает большой набор готовых инструментов разработки, библиотек и фреймворков, существенно ускоряющих написание кода. Например, библиотека `python-telegram-bot` упрощает интеграцию с Telegram API, что позволяет сосредоточиться на логике приложения, не тратя лишнего времени на рутинные задачи.

Ещё одним аргументом в пользу Python является его способность обрабатывать большие объёмы данных и выполнять сложные вычисления. Это особенно актуально для информационных систем, где требуется оперативно анализировать данные из баз, использовать аналитические модули или реализовывать алгоритмы машинного обучения. Оптимизированные инструменты языка в сочетании с высокой производительностью помогают существенно ускорить вычислительные процессы.

Именно по этим причинам я остановился на Python при создании информационной системы для фитнес-тренера, реализованной на базе телеграм-бота.

1.5.2 Выбор СУБД

Выбор системы управления базами данных (СУБД) является критически важным этапом при разработке программного обеспечения, поскольку от него зависит то, насколько эффективно можно структурировать и обрабатывать большие массивы информации, используемой в приложении. СУБД представляют собой специализированные программные системы, предоставляющие интерфейсы для создания, модификации и извлечения данных.

Главным фактором при выборе СУБД является её способность удовлетворять специфическим требованиям проекта. Если приложение

рассчитано на работу с масштабными данными, то необходимо отдавать предпочтение решениям, которые обеспечивают высокую производительность и возможность гибкого масштабирования. Также важны вопросы безопасности, доступности, надежности и удобства администрирования, а также совместимость с языками программирования и технологиями, которые применяются в проекте.

Среди реляционных баз данных одним из часто упоминаемых решений является Oracle. Эта система востребована в крупных корпоративных приложениях благодаря своим возможностям в управлении транзакциями, обеспечении высокого уровня безопасности и регулярном резервном копировании данных. Но, несмотря на это, высокая стоимость лицензирования и поддержки делает Oracle не самым оптимальным выбором для небольших проектов или стартапов.

В качестве альтернативы многие разработчики обращаются к MySQL. Это свободная и открытая СУБД, которая широко применяется в веб-разработке. MySQL обладает высокой производительностью, но её возможности в категории масштабирования и поддержке сложных транзакций могут оказаться менее развитыми по сравнению с другими системами.

Следующим аналогом СУБД для телеграм-бота является MongoDB. В отличие от традиционных реляционных СУБД, MongoDB использует документо-ориентированный подход, где данные хранятся в виде JSON-подобных документов, а не строго организованных таблиц. Такая схема обеспечивает динамичность и лёгкость масштабирования, особенно при работе с неструктурированными данными большого объёма. Дополнительными преимуществами MongoDB являются высокая доступность и устойчивость к отказам, что делает её привлекательной для приложений, зависящих от скорости доступа к данным.

Однако, для нашей информационной системы, реализующей функционал телеграм-бота, выбор пал на SQLite. Эта легковесная, встроенная СУБД, не требующая отдельного серверного процесса, идеально подходит для небольших и средних проектов благодаря своей простоте развертывания и минимальным требованиям к инфраструктуре. Надежное соблюдение принципов ACID, высокая скорость операций ввода-вывода и возможность быстрого прототипирования позволяют сосредоточиться на разработке функционала бота, а также в случае необходимости в будущем легко перейти на более масштабируемое решение. Простота сопровождения, стабильная производительность и отсутствие необходимости в сложной серверной настройке делают SQLite оптимальным выбором для нашего проекта.

1.5.3 Выбранные технологии

По результатам комплексного анализа мы определили технологические решения.

Платформа для бота: использование Telegram в качестве основы для работы чат-бота.

Среда разработки и инструментарий: Visual Studio Code, язык программирования Python, а также выбранные библиотеки:

- python-telegram-bot – основная библиотека для работы с Telegram API. Обеспечивает построение диалогов, обработку команд, inline-клавиатуры и асинхронное выполнение;
- SQLAlchemy – ORM для работы с базой данных. Используется для определения моделей, создания сессий и выполнения запросов;
- FastAPI – фреймворк для создания API-сервера, который объединяет бизнес-логику и работу с данными;
- Requests – библиотека для выполнения HTTP-запросов. Применяется для обращения к Google Custom Search API (при поиске изображений в интернете);

- Pillow – библиотека для обработки изображений. Используется для открытия, конвертации и подготовки изображений перед отправкой в Telegram.
- asyncio – стандартный модуль Python для организации асинхронного программирования. Используется для асинхронного выполнения операций;
- СУБД – SQLite как производительная платформа для хранения и обработки данных.

1.5.4 Выбор IDE

В качестве среды разработки для нашего телеграм-бота «Индивидуальный фитнес-тренер» выбран Visual Studio Code. Это легковесный, кросс-платформенный редактор от Microsoft, который отлично подходит для создания Python-проектов и обладает широкими возможностями для настройки и расширения функциональности.

Преимущества Visual Studio Code для нашего проекта:

- официальная поддержка и активное сообщество. VS Code регулярно обновляется, поддерживается Microsoft и огромным сообществом разработчиков, что обеспечивает стабильность и своевременное устранение ошибок. Благодаря этому мы можем быть уверены, что инструменты, необходимые для создания нашего телеграм-бота, всегда актуальны;
- интеграция с Python и системой контроля версий. С помощью официального расширения Python VS Code обеспечивает автодополнение кода, интеллектуальную подсветку синтаксиса, отладку и поддержку виртуальных окружений. Встроенная поддержка Git позволяет легко вести контроль версий, что особенно важно для командной работы и поддержки чистоты кода;
- удобство использования и гибкость настроек. Интерфейс VS Code интуитивно понятен, а многочисленные настройки позволяют адаптировать среду

под личные предпочтения разработчика. Это включает настройку клавиатурных сокращений, тем оформления, а также конфигурацию сборки проекта и отладку;

– расширяемость. Возможность установки дополнительных расширений позволяет интегрировать любые инструменты, которые могут понадобиться для разработки телеграм-бота — от инструментов для форматирования кода и линтеров до плагинов для работы с Docker и удалёнными серверами. Это даёт невероятную гибкость и позволяет подстраивать рабочее пространство под конкретные задачи проекта;

– встроенные инструменты отладки и профилирования. Интегрированная система отладки в VS Code позволяет легко устанавливать точки останова, шаг за шагом проходить по коду и анализировать переменные. Это существенно упрощает поиск и исправление ошибок, а также помогает оптимизировать производительность кода бота;

– кросс-платформенность. Visual Studio Code работает на Windows, Linux и macOS, что позволяет разработчику использовать привычную среду в любой операционной системе. Это особенно удобно при тестировании и развертывании бота на различных платформах.

1.6 Установка и настройка инструментов разработки

1.6.1 Установка Visual Studio Code

Для начала работы над проектом скачана и установлена интегрированная среда с официального сайта (рисунок 1.9).

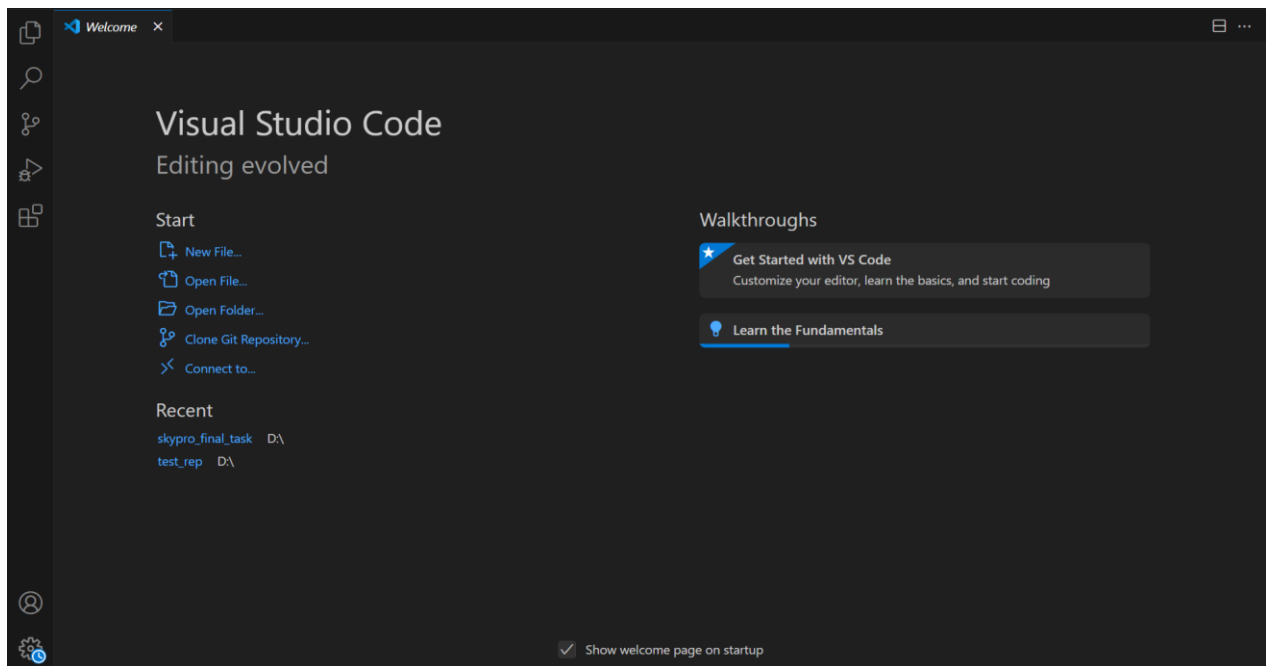


Рисунок 1.9 — Среда Visual Studio Code

После скачивания и установки .exe файла запускается сама программа.

1.6.2 Настройка IDE

Устанавливаем необходимые расширения для работы с Python [1] (рисунок 1.10).

1.6.3 Создание проекта

После установки расширений нам будет предложено создать пустой файл Python (рисунок 1.11).

Затем создаем дерево проекта и добавляем необходимые папки (рисунок 1.12).

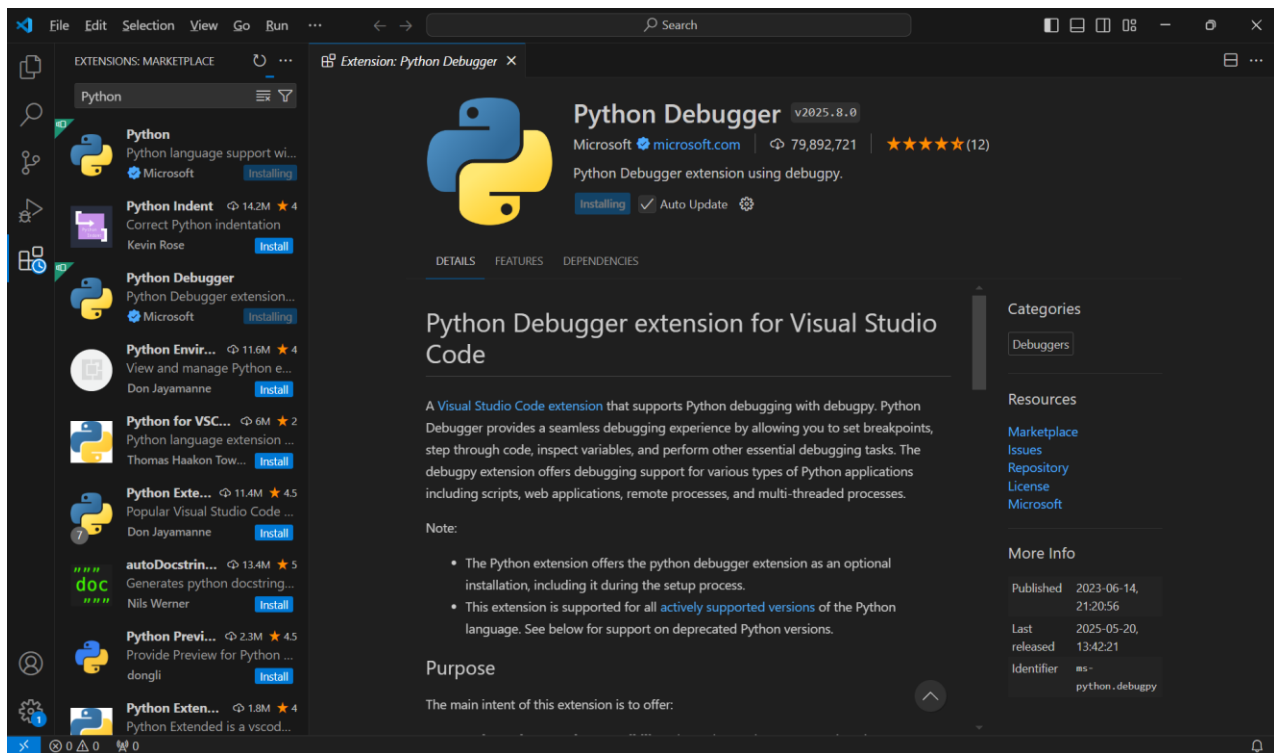


Рисунок 1.10 — Установка расширений

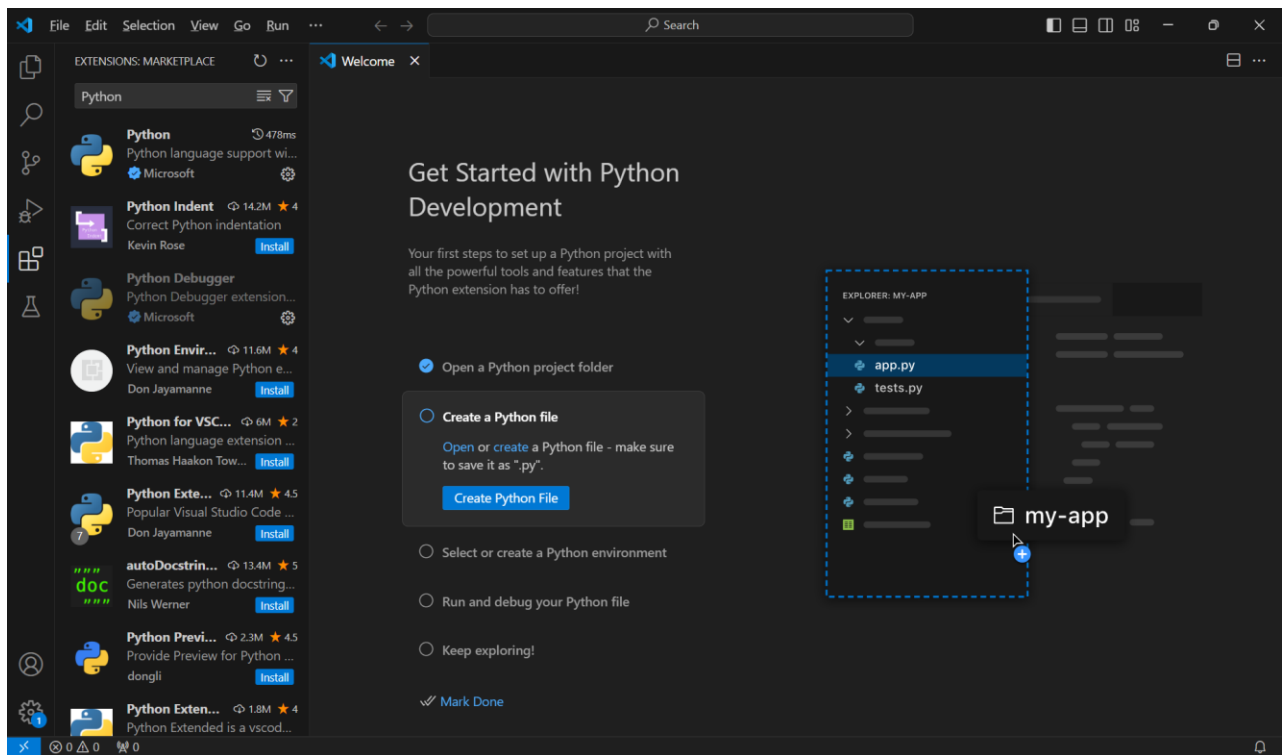


Рисунок 1.11 — Создание проекта

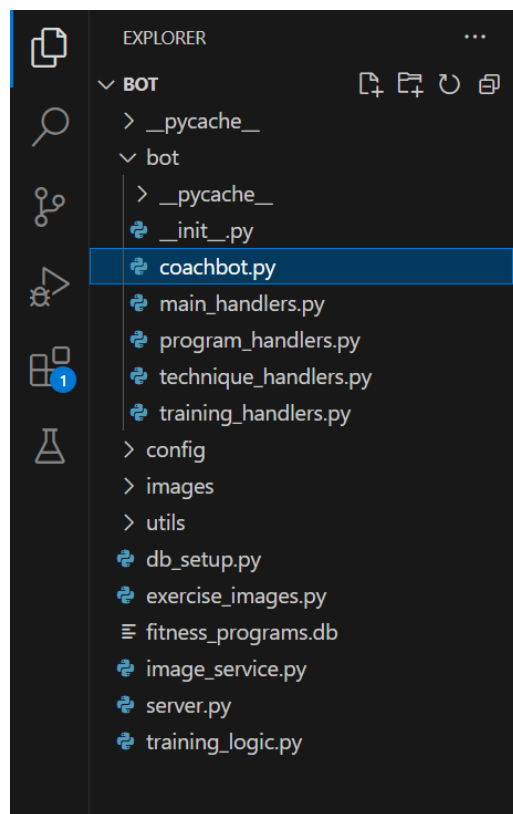


Рисунок 1.12 — Добавление файлов и папок

После этого наш проект готов к использованию и созданию приложения.

1.6.4 Регистрация бота

Первым шагом для создания чат-бота является его регистрация в Telegram. В этом разделе будет описана процедура первоначального создания бота и описание его пользовательского интерфейса.

Для начала разработки бота необходимо зарегистрироваться в Telegram или войти в личный аккаунт, если таковой имеется. Затем в поиске надо найти чат «BotFather» (рисунок 1.15). Это официальный бот Telegram, который как раз поможет пройти процесс регистрации нашего чат-бота в сервисе [2].

Следующим шагом необходимо написать в чат-бот команду «/start» для его запуска, после чего мы получим список необходимых нам команд для создания нового бота (рисунок 1.16).

Далее, вводим команду «/newbot» для начала регистрации нашего бота, указываем желаемое название и логин, который будет записан в ссылке (рисунок 1.17).

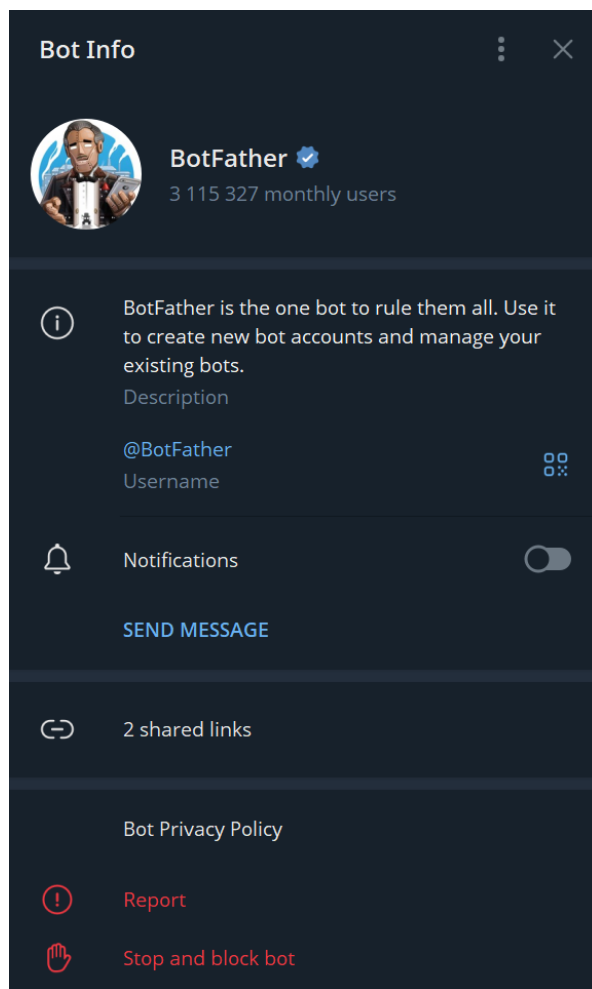


Рисунок 1.15 — Бот «BotFather»

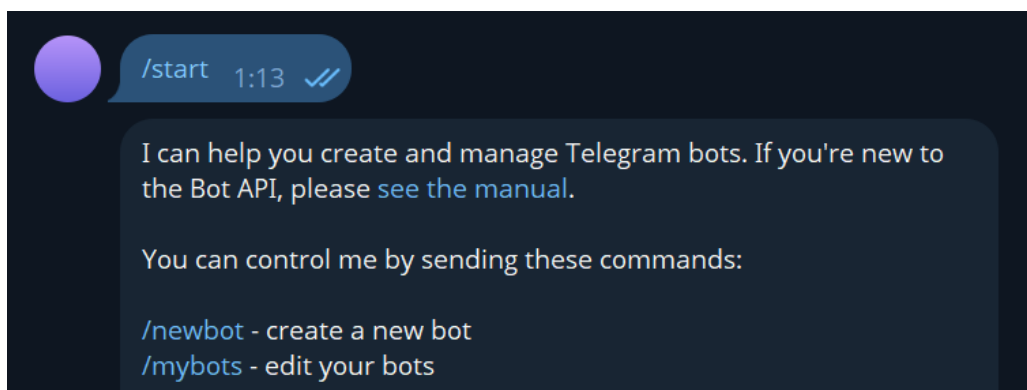


Рисунок 1.16 — Список команд

После проделанной процедуры «BotFather» присваивает нашему боту токен (рисунок 1.17), который является ключом доступа к нашему боту и далее будет использован при написании программы. Хранить токен необходимо в безопасном месте, чтобы злоумышленники не могли завладеть им. Этот токен будет использоваться для аутентификации нашего бота при взаимодействии с Telegram API. После этого шага можно приступить к созданию пользовательского интерфейса и дальнейшей разработке нашего бота.

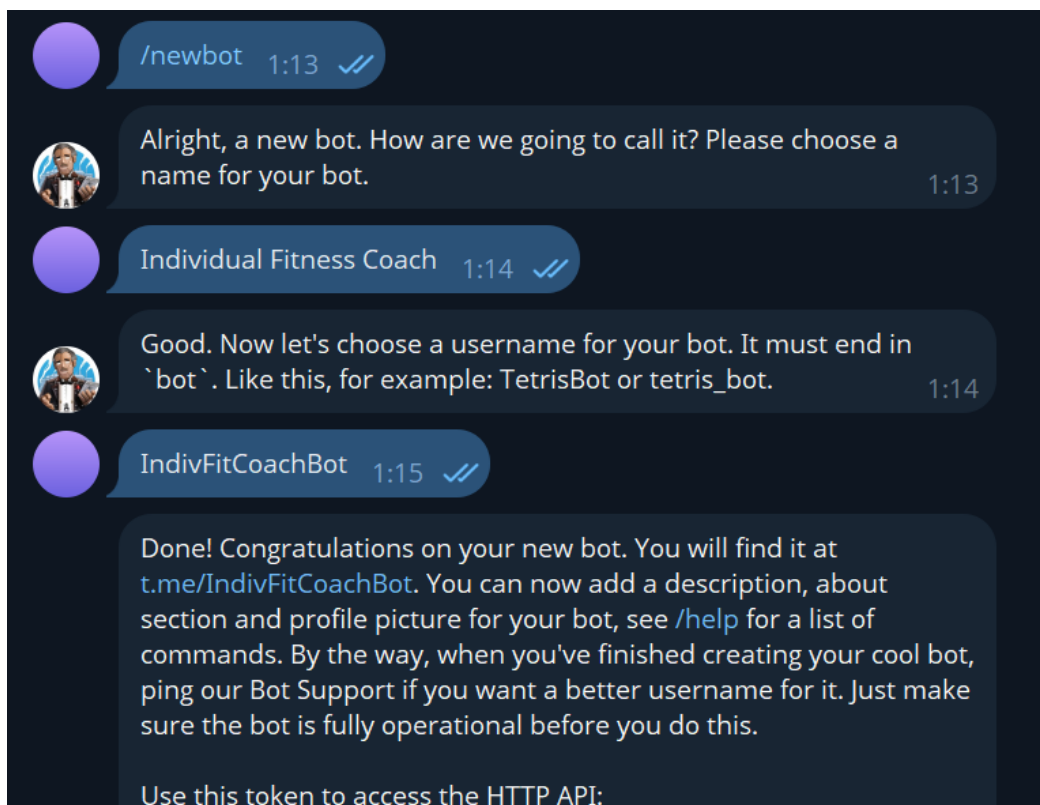


Рисунок 1.17 — Заполнение названия бота.

1.7 Реализация бота

1.7.1 Создание БД

Для хранения информации о пользователях, тренировках и упражнениях в нашем фитнес-боте разработана СУБД, состоящая из двух взаимосвязанных таблиц [3]. Такая модель позволяет гибко организовать данные, обеспечивая возможность составления индивидуальных тренировочных программ и хранения детальной информации о ходе тренировок. В основу физической модели легли две ключевые сущности: Program и Exercise.

Класс Program расположен в `db_setup.py` и используется также в `training_logic.py`. Этот класс хранит информацию о тренировочных программах. Он применяется для сохранения разных вариантов программ в зависимости от

типа телосложения (например, «ectomorph», «mesomorph», «endomorph») и цели тренировок (например, «loss», «gain», «recomp», «strength», «tonus»).

Атрибуты:

- id – уникальный идентификатор записи;
- body_type – строковое значение, определяющее тип телосложения;
- goal – строковое значение, определяющее цель тренировок;
- program_text – текстовое описание тренировочной программы,

включающее перечень упражнений и рекомендации.

Пример кода:

```
class Program(Base):
    __tablename__ = "programs"
    id = Column(Integer, primary_key=True)
    body_type = Column(String(50), nullable=False) #
    "ectomorph", "mesomorph", "endomorph"
    goal = Column(String(50), nullable=False) # "loss",
    "gain", "recomp", "strength", "tonus"
    program_text = Column(Text, nullable=False)
```

Класс Exercise расположен в db_setup.py и предназначен для хранения информации о конкретных упражнениях, связанных с программой тренировок. Эта сущность позволяет детализировать программу, разбив её на отдельные упражнения с указанием техники выполнения (например, ссылкой на изображение).

Атрибуты:

- id – уникальный идентификатор упражнения;
- program_id – внешний ключ, указывающий на связь с конкретной программой из таблицы Program;
- exercise_name – название упражнения;

– `technique_image_url` – строка, содержащая URL изображения техники выполнения, если оно задано.

Пример кода:

```
class Exercise(Base):
    __tablename__ = "exercises"
    id = Column(Integer, primary_key=True)
    program_id = Column(Integer, ForeignKey("programs.id"),
nullable=False)
    exercise_name = Column(String(100), nullable=False)
    technique_image_url = Column(String(255)) # URL на
картинку с техникой выполнения
```

Сущность `Exercise` позволяет при расширении функционала иметь более детализированную информацию об упражнениях, входящих в программу. Особенность модели заключается в том, что она реализует связь «один ко многим»: одному объекту `Program` может соответствовать несколько записей `Exercise`. Связь реализована посредством внешнего ключа `program_id` в классе `Exercise`. Это позволяет связывать несколько упражнений с одной программой. Отношение таблиц `Program` и `Exercise` отображает диаграмма классов (рисунок 1.18).

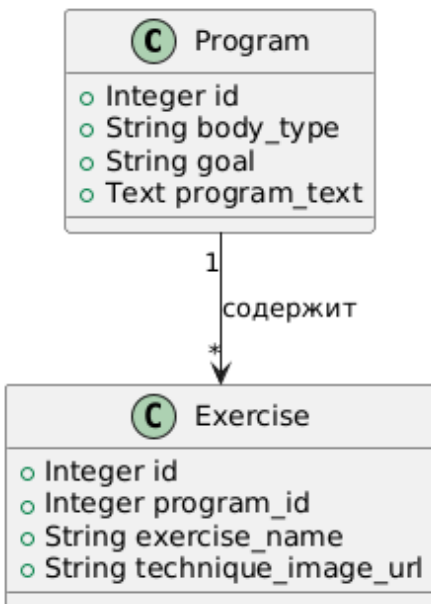


Рисунок 1.18 — Диаграмма классов

1.7.2 Описание функций приложения

В данном разделе описываются ключевые функции, реализованные в рамках проекта фитнес-бота. Эти функции отвечают за нормализацию входящих данных, поиск иллюстраций упражнений, анализ текста тренировочной программы и организацию интерактивного взаимодействия с пользователем через Telegram. Рассмотрим подробнее каждую функцию и её роль в системе.

Функция `normalize_text(s: str)` предназначена для приведения вводимого пользователем текста к единому виду. Это достигается за счёт удаления лишних пробелов, перевода всего текста в нижний регистр и удаления знаков препинания. Такая обработка обеспечивает корректное сравнение строк, например, при сопоставлении названий упражнений. Пример реализации:

```
def normalize_text(s: str) -> str:
    s = s.strip().lower() # Убираем пробелы и приводим к
                           # нижнему регистру
```

```

s = re.sub(r'[\w\s]', '', s) # Удаляем символы, не
являющиеся буквенно-цифровыми или пробелами
return s

```

Эта функция позволяет, например, корректно сопоставлять ввод фразы «Приседания, со штангой!» с записью «приседания со штангой» в базе данных. Функция `get_local_image(exercise_name: str)` осуществляет поиск изображения для упражнения в локальном хранилище. Сначала название упражнения нормализуется с помощью `normalize_text()`, затем производится попытка найти точное совпадение или наиболее подходящий вариант в словаре локальных изображений. Если нужный файл существует по указанному пути, функция возвращает путь к изображению; иначе — возвращается значение, указывающее на отсутствие локального ресурса [4]. Пример кода:

```

def get_local_image(exercise_name: str) -> str:
    normalized = normalize_text(exercise_name)
    # 1. Поиск точного совпадения
    for key, path in LOCAL_EXERCISE_IMAGE_MAP.items():
        norm_key = normalize_text(key)
        if normalized == norm_key and os.path.exists(path):
            return path
    # 2. Поиск кандидатов по подстроке
    candidates = []
    for key, path in LOCAL_EXERCISE_IMAGE_MAP.items():
        norm_key = normalize_text(key)
        if norm_key in normalized and os.path.exists(path):
            candidates.append((norm_key, path))
    if candidates:
        # Выбирается кандидат с максимальной длиной ключа
        (более специфичный)
        candidates.sort(key=lambda x: len(x[0]), reverse=True)
        return candidates[0][1]

```

```
return None
```

Благодаря такому подходу система надёжно определяет наличие локального изображения, даже если название упражнения задано с незначительными ошибками.

Функция `get_exercise_images(exercise_name: str)` отвечает за обеспечение пользователя изображениями техники выполнения упражнения. Сначала она пытается найти локальное изображение с помощью `get_local_image()`. Если локальное изображение найдено, оно возвращается в виде списка (так как логика дальнейшей обработки ожидает список изображений). Если же изображение отсутствует в локальной базе, выполняется запрос к внешнему API (например, Google Custom Search API) для поиска изображений в интернете. Пример реализации:

```
def get_exercise_images(exercise_name: str) -> list:
    local_image = get_local_image(exercise_name)
    if local_image:
        return [local_image]

    # Если локальное изображение не найдено, выполняется запрос
    к внешнему API.

    query = f"{exercise_name} техника выполнения"
    encoded_query = quote_plus(query)
    url = (
        f"https://www.googleapis.com/customsearch/v1?key=YOUR_API_KEY"
        f"&cx=YOUR_CSE_ID&q={encoded_query}&searchType=image"
        f"&imgSize=huge&imgType=photo&fileType=jpg&safe=active&num=10"
    )
    try:
        response = requests.get(url)
        response.raise_for_status()
        results = response.json()
```

```

        images = [item.get("link") for item in
results.get("items", []) if item.get("link")]
        return images if images else ["Изображение не найдено."]
except Exception as e:
    return [f"Ошибка при запросе: {e}"]

```

Таким образом, пользователь получает возможность увидеть картинку даже в случае отсутствия локального изображения.

Для дальнейшей работы с программой тренировок необходимо выделить из текста список упражнений. Функция `extract_exercises(text: str)` анализирует текст и при помощи регулярных выражений извлекает названия упражнений:

```

def extract_exercises(text: str) -> list:
    pattern = re.compile(r'^\d+\.\s*(.*)', re.MULTILINE)
    exercises = [match.strip() for match in
pattern.findall(text)]
    logging.info("Extracted exercises: %s", exercises)
    return exercises

```

Помимо этого, функция `generate_callback_data(exercise: str)` создаёт уникальные идентификаторы для кнопок, отображаемых в интерфейсе бота [5]. Эти идентификаторы помогают однозначно определить, какое упражнение выбрал пользователь, и гарантируют, что размер данных не превышает установленных ограничений:

```

def generate_callback_data(exercise: str) -> str:
    base = "tech_ex_"
    safe = re.sub(r'\W+', '_', exercise)
    callback_data = base + safe
    while len(callback_data.encode('utf-8')) > 64:
        callback_data = callback_data[:-1]
    return callback_data

```

Когда пользователь запрашивает список упражнений из тренировочной программы, срабатывает функция `list_exercises_callback(update, context)`. Она извлекает текст программы, анализирует его с помощью `extract_exercises()`, а затем для каждого упражнения создаёт кнопку с уникальным идентификатором. Таким образом, пользователь может выбрать интересующее его упражнение для получения дополнительных сведений о технике выполнения:

```
async def list_exercises_callback(update: Update, context:
ContextTypes.DEFAULT_TYPE) -> None:
    training_program_text =
context.user_data.get("training_program_text", "")
    exercises = extract_exercises(training_program_text)
    exercise_buttons = []
    callback_mapping = {}
    for ex in exercises:
        cb_data = generate_callback_data(ex)
        callback_mapping[cb_data] = ex
        exercise_buttons.append([InlineKeyboardButton(ex,
callback_data=cb_data)])
    # Добавление кнопок для навигации
    exercise_buttons.append(
        [InlineKeyboardButton("Вернуться к плану",
callback_data="return_plan"),
        InlineKeyboardButton("Завершить тренировку",
callback_data="finish_training"),
        InlineKeyboardButton("В главное меню",
callback_data="start")]
    )
    context.user_data["callback_mapping"] = callback_mapping
    reply_markup = InlineKeyboardMarkup(exercise_buttons)
```

```
await update.callback_query.message.reply_text("Выберите  
упражнение для просмотра техники:", reply_markup=reply_markup)
```

При выборе конкретного упражнения запускается функция `tech_exercise_callback(update, context)`, которая получает полное название упражнения, запрашивает соответствующие изображения через `get_exercise_images()` и отправляет их пользователю в виде медиагруппы. Это позволяет пользователю увидеть визуальное руководство по выполнению упражнения:

```
async def tech_exercise_callback(update: Update, context:
ContextTypes.DEFAULT_TYPE) -> None:
    query = update.callback_query
    await query.answer()
    data = query.data
    callback_mapping = context.user_data.get("callback_mapping",
    {})
    # Получение полного имени упражнения
    exercise_name = callback_mapping.get(data,
data[len("tech_ex_"):])
    loop = asyncio.get_running_loop()
    images = await loop.run_in_executor(None,
get_exercise_images, exercise_name)
    # Логика отправки изображений (упрощённая для примера)
    if images:
        await query.message.reply_text(f"Техника выполнения:
{exercise_name}\nИзображения: {images}")
    else:
        await query.message.reply_text("Изображения не найдены")
```

Для поддержания порядка в чате предусмотрена функция `close_tech_callback(update, context)`, которая удаляет временные сообщения (например, изображения и кнопки), связанные с просмотром техники.

Это помогает вернуть пользователя к основному меню без накопления ненужных данных:

```
async def close_tech_callback(update: Update, context:
ContextTypes.DEFAULT_TYPE) -> None:
    query = update.callback_query
    await query.delete_message() # Удаляем сообщение с кнопкой
закрытия техники

    chat_id = query.message.chat_id
    bot = context.bot
    tech_media_ids = context.user_data.get("tech_media_ids", [])
    for message_id in tech_media_ids:
        await bot.delete_message(chat_id=chat_id,
message_id=message_id)

    # Очистка данных о временных сообщениях
    context.user_data.pop("tech_media_ids", None)
```

Функция `generate_key()` отвечает за создание симметричного ключа, необходимого для шифрования и дешифрования конфигурационных файлов [6]. В реализации используется метод `Fernet.generate_key()` из библиотеки `cryptography`, что гарантирует использование современного алгоритма симметричного шифрования. Сгенерированный ключ сохраняется в отдельный файл `key.key`, чтобы его можно было использовать повторно при последующих операциях с конфигурационными данными. Такой подход позволяет обеспечить защиту токена бота от несанкционированного доступа. Пример реализации:

```
def generate_key():
    key = Fernet.generate_key() # Генерируем симметричный ключ
    with open("key.key", "wb") as key_file:
        key_file.write(key)      # Сохраняем ключ в файл
    print("Ключ сгенерирован и сохранён в key.key")

if __name__ == "__main__":
    generate_key()
```

Функция `encrypt_file()` предназначена для шифрования исходного конфигурационного файла `config.json`. Вначале функция считывает симметричный ключ из файла, затем инициализирует объект шифрования Fernet с этим ключом. Далее происходит чтение исходного файла и шифрование его содержимого, после чего результат записывается в новый зашифрованный файл `config.enc`. Такой механизм позволяет защитить конфигурацию от несанкционированного прочтения, сохраняя все необходимые параметры в безопасном виде. Пример реализации:

```
def encrypt_file(input_file: str, output_file: str, key_file:
str):
    # Читаем ключ
    with open(key_file, "rb") as kf:
        key = kf.read()
    fernet = Fernet(key)
    # Читаем исходный конфигурационный файл
    with open(input_file, "rb") as file:
        original = file.read()

    # Шифруем данные
    encrypted = fernet.encrypt(original)
    # Сохраняем зашифрованные данные
    with open(output_file, "wb") as enc_file:
        enc_file.write(encrypted)
    print(f"Файл {input_file} успешно зашифрован в
{output_file}")
if __name__ == "__main__":
    encrypt_file("config.json", "config.enc", "key.key")
```

Функция `load_config()`. Для работы приложения необходимо расположить конфигурационные данные в открытом виде, поэтому разработан механизм их расшифровки. Функция `load_config()` выполняет обратную

операцию: она считывает ранее сохранённый симметричный ключ, затем открывает зашифрованный файл конфигурации и используя объект Fernet, расшифровывает его содержимое. Полученные данные преобразуются из формата JSON в словарь Python, что позволяет легко использовать конфигурационные настройки в коде приложения. Пример реализации:

```
def load_config(config_enc_path: str = "config.enc",
key_file_path: str = "key.key") -> dict:
    # Читаем ключ из файла
    with open(key_file_path, "rb") as kf:
        key = kf.read()
    fernet = Fernet(key)
    # Читаем зашифрованный файл конфигурации
    with open(config_enc_path, "rb") as enc_file:
        encrypted = enc_file.read()
    # Расшифровываем данные
    decrypted = fernet.decrypt(encrypted)
    config = json.loads(decrypted)
    return config

if __name__ == "__main__":
    config = load_config()
    print("Загруженная конфигурация:", config)
```

Наконец, функция `main()` отвечает за инициализацию Telegram-бота, регистрацию всех обработчиков команд и callback-запросов, а затем за запуск основного цикла, который обеспечивает непрерывную работу бота. Пример основного блока:

```
if __name__ == '__main__':
    asyncio.run(main())
```

Запуск этой функции позволяет боту работать 24/7, обрабатывая запросы пользователей и предоставляя актуальную информацию о тренировочных программах и технике выполнения упражнений [7].

Для лучшего понимания внутренней логики работы нашего фитнес-бота была создана диаграмма деятельности, демонстрирующая взаимодействие основных функций приложения (рисунок 1.19). Эта диаграмма помогает наглядно продемонстрировать, как функции взаимодействуют друг с другом, какие данные передаются между модулями, а также как реализована обработка альтернативных сценариев (например, когда локальное изображение отсутствует и требуется запрос к внешнему API). Это позволяет не только лучше понять архитектуру приложения, но и выявить возможные точки для оптимизации и отладки.

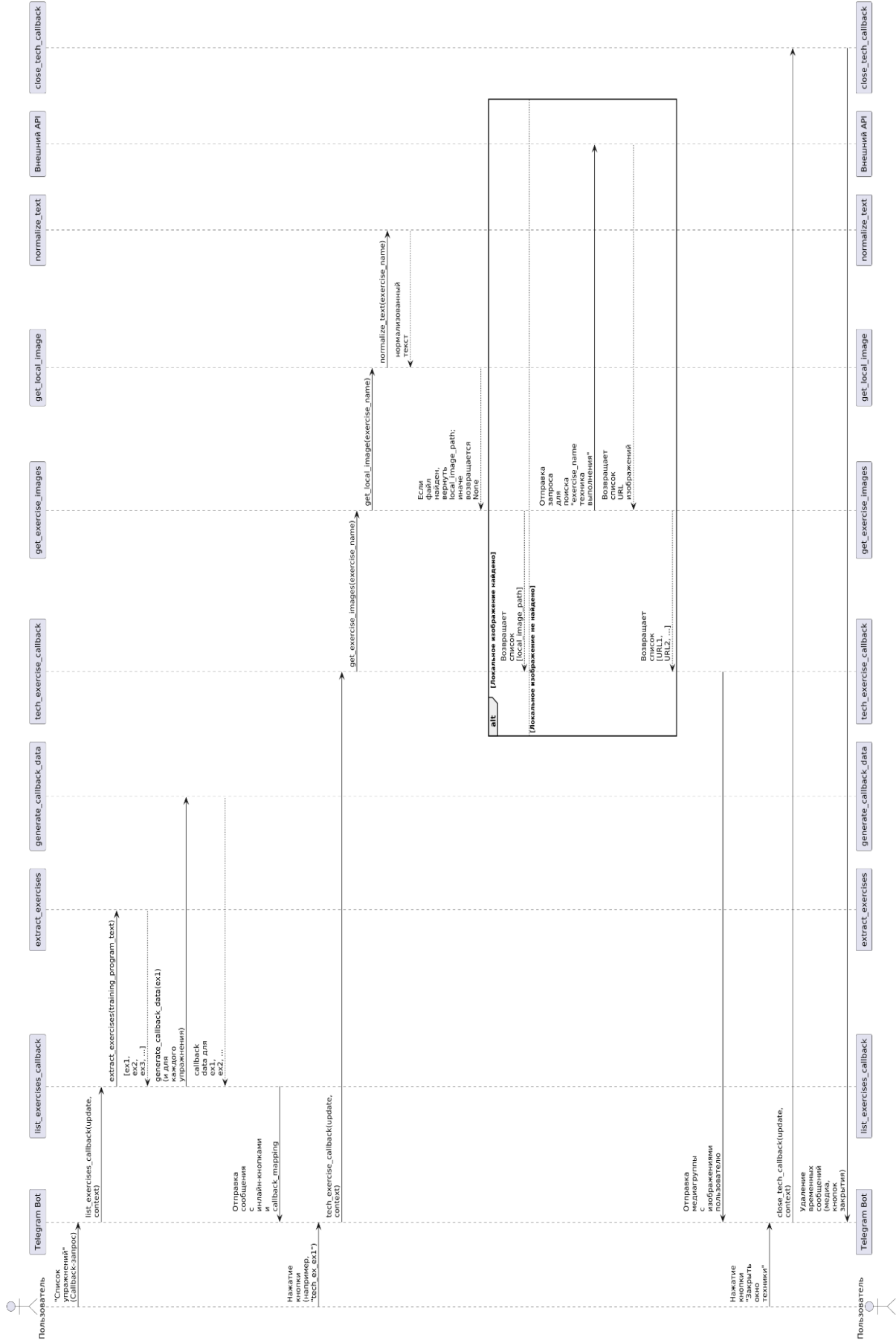


Рисунок 1.19 — UML-диаграмма деятельности

1.8 Перенос проекта на виртуальный сервер

После того, как разработана наша информационная система, её необходимо перенести на виртуальный сервер для автономной работы, удобства использования, стабильности работы и повышения безопасности. Перед началом выгрузки системы необходимо пройти процесс настройки виртуального сервера, который будет использоваться для работы.

Первоочередно необходимо зарегистрироваться на хостинге, получить IP-адрес, логин, и пароль (или ключ для SSH). Затем можно приступить непосредственно к настройке нашего сервера. Для работы будет использована операционная система Ubuntu ver. 22.04. После получения IP-адреса и пароля приступаем к подключению. Для этого в командную строку вводим наши полученные данные. После входа запускаем процесс обновления пакетов командой `apt update && apt upgrade -y` (рисунок 1.20).

```
root@4896841-jw28837:~# apt update && apt upgrade -y
Hit:1 http://zabbix.repo.timeweb.ru/ubuntu focal InRelease
Hit:2 http://archive.ubuntu.com/ubuntu jammy InRelease
Get:3 http://archive.ubuntu.com/ubuntu jammy-updates InRelease [128 kB]
Get:4 http://archive.ubuntu.com/ubuntu jammy-backports InRelease [127 kB]
Get:5 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 Packages [2,598 kB]
Get:6 http://archive.ubuntu.com/ubuntu jammy-updates/universe amd64 Packages [1,207 kB]
Ign:7 http://security.ubuntu.com/ubuntu jammy-security InRelease
Get:7 http://security.ubuntu.com/ubuntu jammy-security InRelease [129 kB]
Get:8 http://security.ubuntu.com/ubuntu jammy-security/main amd64 Packages [2,352 kB]
Get:9 http://security.ubuntu.com/ubuntu jammy-security/universe amd64 Packages [974 kB]
Fetched 7,516 kB in 1min 54s (66.0 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
12 packages can be upgraded. Run 'apt list --upgradable' to see them.
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Calculating upgrade... Done
The following packages will be upgraded:
  apt apt-transport-https apt-utils libapt-pkg6.0 libc-bin libc6 libgssapi-krb5-2 libk5crypto3 libkrb5-3
  libkrb5support0 locales tzdata
12 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
8 standard LTS security updates
Need to get 11.6 MB of archives.
```

Рисунок 1.20 — Обновление пакетов

Следующим шагом загружаем файлы проекта на сервер с помощью WinSCP (рисунок 1.21).

Дальнейшим действием после загрузки файлов на сервер необходимо создать виртуальное окружение командой `python3 -m venv venv`, активировать его и произвести установку зависимостей. Затем создается `systemd`-сервис для автономной работы бота и его автоматического перезапуска командой: `папо /etc/systemd/system/coachbot.service`. После этого обновляем список служб, запускаем сервер и проверяем статус (рисунок 1.22).

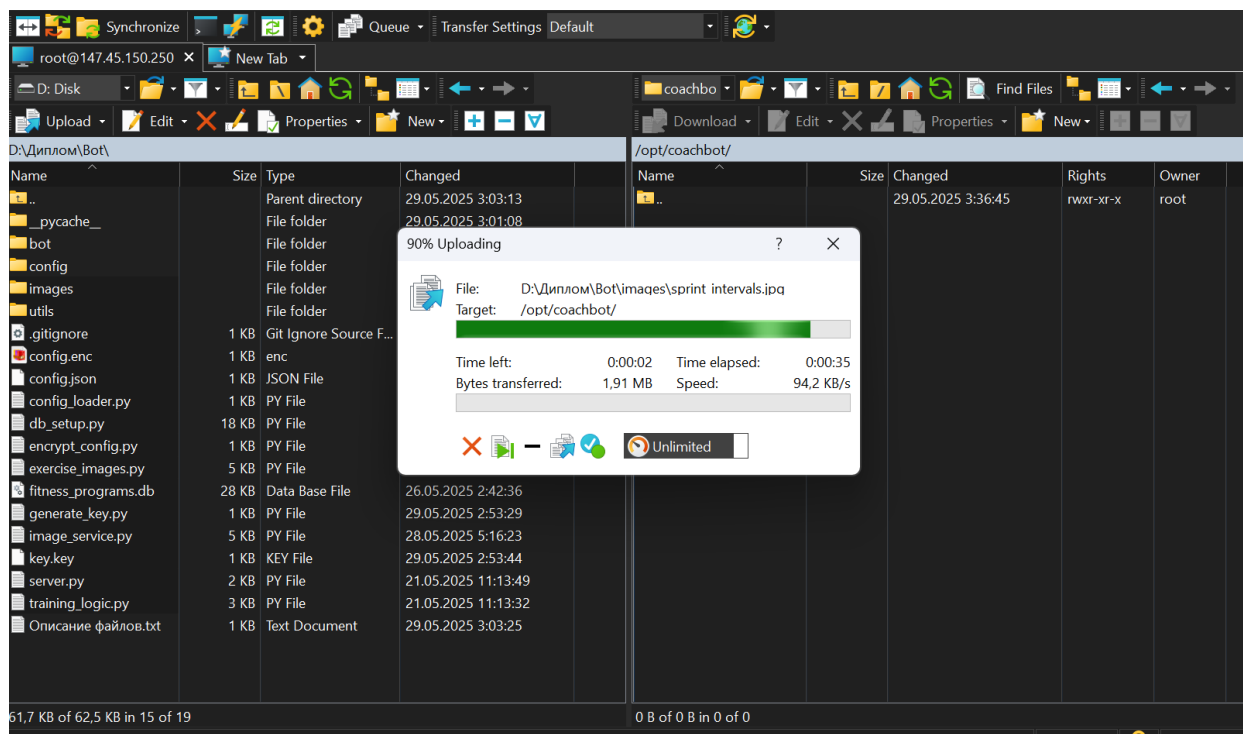


Рисунок 1.21 — Загрузка файлов на сервер

```
Last login: Thu May 29 03:58:51 2025 from 77.91.101.220
root@4896841-jw28837:~# nano /opt/coachbot/exercise_images.py
root@4896841-jw28837:~# systemctl restart coachbot.service
root@4896841-jw28837:~# systemctl status coachbot.service
● coachbot.service - Telegram Bot CoachBot Service
   Loaded: loaded (/etc/systemd/system/coachbot.service; enabled; vendor preset: enabled)
   Active: active (running) since Thu 2025-05-29 04:10:25 MSK; 6s ago
     Main PID: 4712 (python)
       Tasks: 2 (limit: 2275)
      Memory: 35.0M
         CPU: 586ms
    CGroup: /system.slice/coachbot.service
            └─4712 /opt/coachbot/venv/bin/python /opt/coachbot/bot/coachbot.py

May 29 04:10:25 4896841-jw28837 systemd[1]: Started Telegram Bot CoachBot Service.
May 29 04:10:25 4896841-jw28837 python[4712]: 2025-05-29 04:10:25,771 - httpx - INFO - HTTP Request: POST https://api.t>
May 29 04:10:26 4896841-jw28837 python[4712]: 2025-05-29 04:10:26,048 - httpx - INFO - HTTP Request: POST https://api.t>
May 29 04:10:26 4896841-jw28837 python[4712]: 2025-05-29 04:10:26,099 - httpx - INFO - HTTP Request: POST https://api.t>
May 29 04:10:26 4896841-jw28837 python[4712]: 2025-05-29 04:10:26,100 - telegram.ext.Application - INFO - Application s>
lines 1-15/15 (END)
```

Рисунок 1.22 — Проверка статуса службы

Служба запущена и работает без ошибок, значит бот готов к работе и работает в автономном режиме.

1.9 Описание финального архитектурного решения

После выполнения разработки всех компонентов нашего бота и размещения готового приложения на сервер, следует выполнить презентацию финального архитектурного решения информационной системы. Для визуализации размещения компонентов системы и их взаимодействия между собой на уровне аппаратного и программного обеспечения разработана диаграмма развертывания. В данной диаграмме отражены ключевые компоненты системы, их функциональное назначение и взаимосвязь между ними, что позволяет получить целостное представление о том, как будет работать приложение (рисунок 1.23).

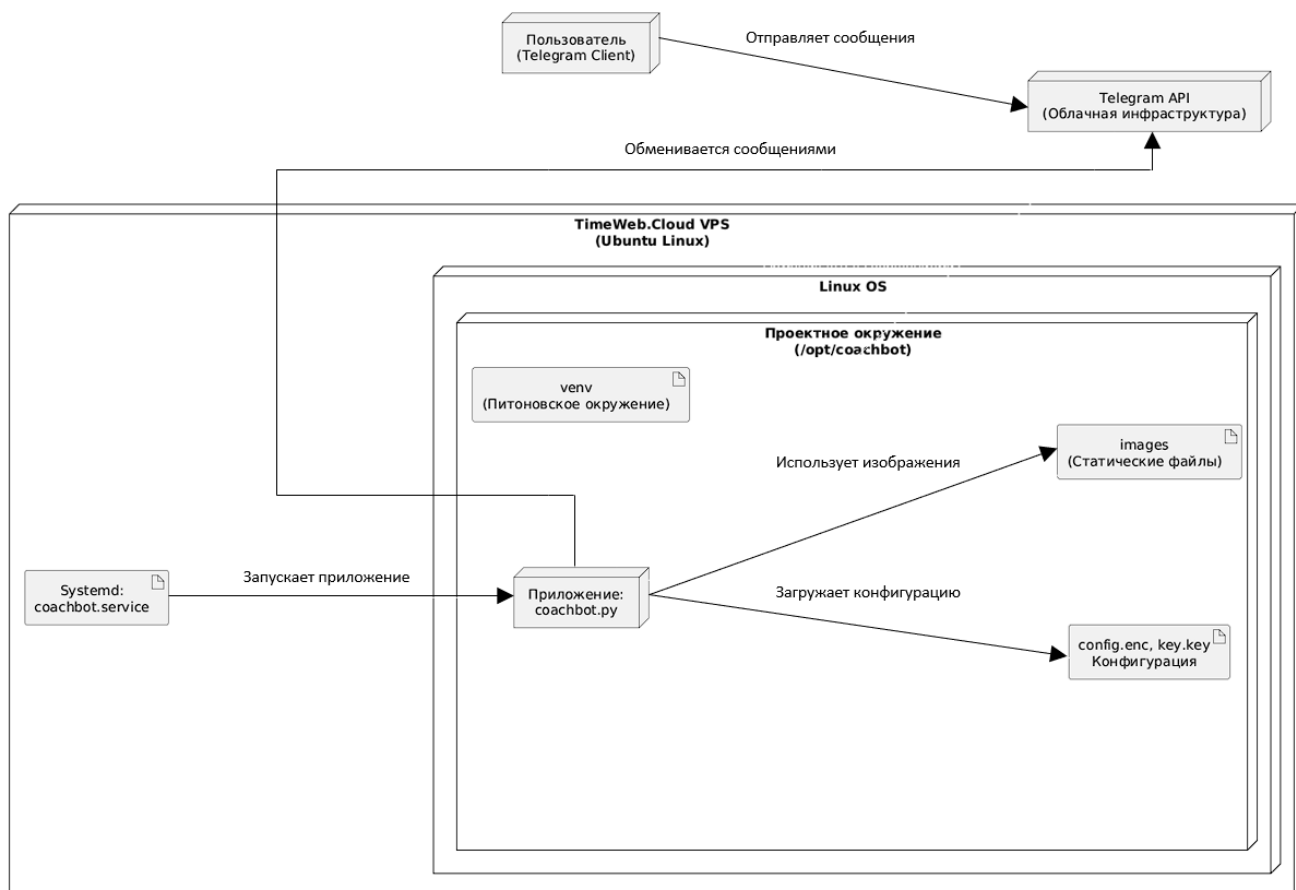


Рисунок 1.23 – UML-диаграмма развертывания

Основным элементом диаграммы является сервер, который развернут на виртуальном сервере TimeWeb.Cloud под управлением Ubuntu. Здесь размещается весь исходный код проекта: API-сервер, логика работы с базой данных, обработка запросов от Telegram и взаимодействие с внешними сервисами. Приложение работает в изолированном Python-окружении (venv), что обеспечивает контроль над зависимостями и стабильность работы. Отдельным элементом выделена папка с изображениями, которая хранит статические файлы, используемые ботом для демонстрации техники выполнения упражнений.

Диаграмма также отражает взаимодействие пользователей с системой через Telegram-клиент и API Telegram, через которое сообщения и запросы передаются на серверное приложение. Особое внимание уделено безопасности: конфигурационные данные шифруются и хранятся в виде защищённых файлов (config.enc, key.key), что позволяет предотвратить доступ к настройкам проекта злоумышленникам.

1.10 Описание интерфейса приложения

При первом взаимодействии с «@IndivFitCoachBot» пользователь начинает с нажатия команды «/start» в Telegram-клиенте. После ввода этой команды бот мгновенно отправляет приветственное сообщение, оформленное в дружелюбном и понятном стиле. В сообщении присутствует краткое описание возможностей бота и перечень предоставляемых опций, таких как «Инфо» и «Составить программу тренировок» (рисунок 1.24).

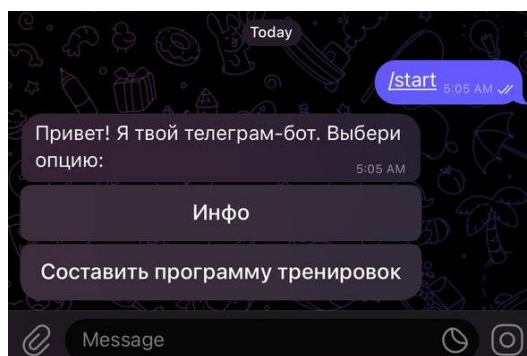


Рисунок 1.24 — Запуск бота

После приветственного сообщения пользователь видит главное меню, представленное с помощью inline-клавиатуры. Кнопки меню интуитивно понятны и оформлены единообразно. Каждый элемент кнопочной панели сопровождается коротким пояснением его назначения, что позволяет пользователю без труда ориентироваться в функционале бота. Например, кнопка «Инфо» открывает раздел с информационными сообщениями о боте и методиках тренировок, а кнопка «Составить программу» переводит к процессу индивидуального подбора тренировки (рисунок 1.25).

При выборе опции «Составить программу» пользователь направляется в серию сообщений, где бот запрашивает необходимые данные для расчёта индивидуальной программы. Здесь происходит последовательный сбор параметров — сначала бот спрашивает рост пользователя, затем вес, а в некоторых случаях может потребоваться дополнительная информация (например, уровень физической подготовки). Каждый запрос сопровождается кратким пояснением, зачем нужны эти данные, что снижает риск недопонимания.

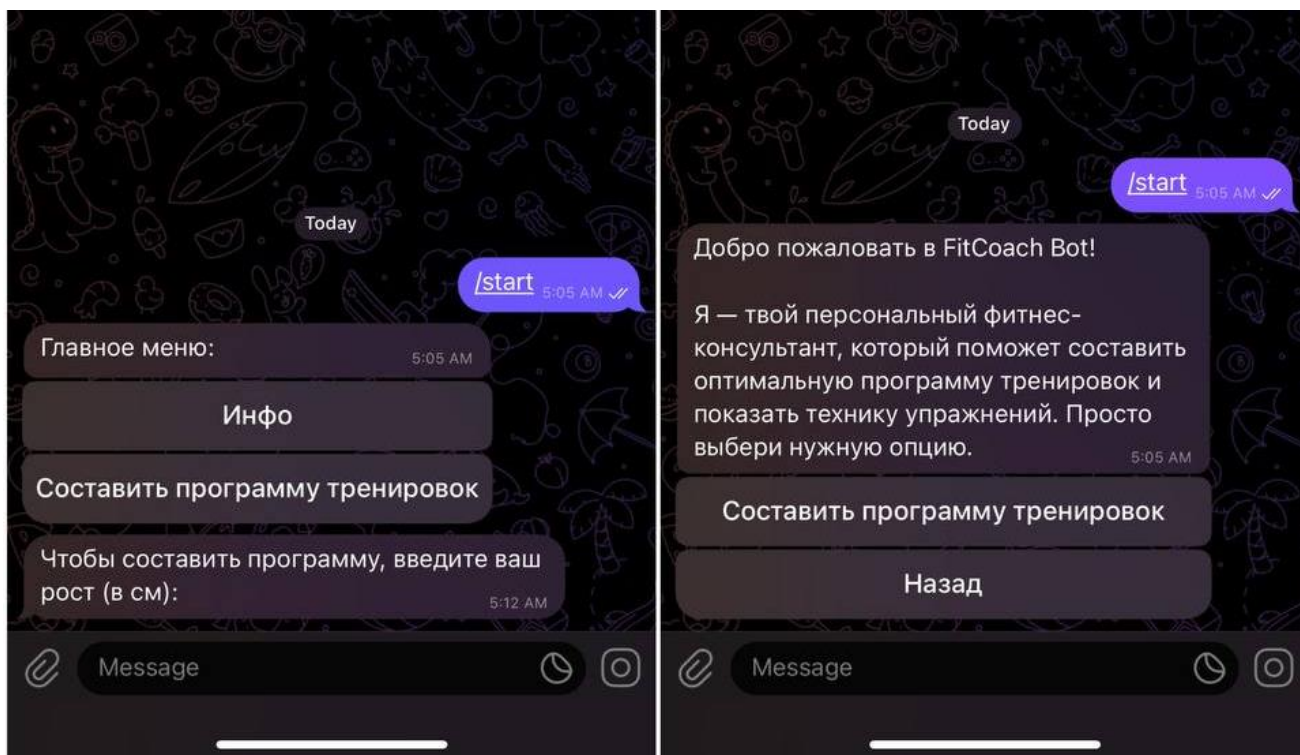


Рисунок 1.25 — Кнопки «Инфо» и «Составить программу тренировок»

В этом блоке интерфейс характеризуется пошаговым диалогом. Пользователь вводит свои данные в текстовом формате, а бот, принимая их, обновляет состояние сеанса и предоставляет обратную связь. Продуманная логика взаимодействия позволяет пользователю не запутаться и корректно ввести информацию (рисунок 1.26).

После получения необходимых данных бот начинает обработку запроса: вычисляет индивидуальную тренировочную программу, используя встроенную бизнес-логику, и, при необходимости, подготавливает данные из базы. Программа выводится в виде многострочного текстового сообщения, где каждая строка описывает упражнение с указанием количества подходов и повторений. Дополнительно рядом с текстом могут располагаться кнопки для выбора дополнительного режима, например, для просмотра техники выполнения конкретного упражнения.

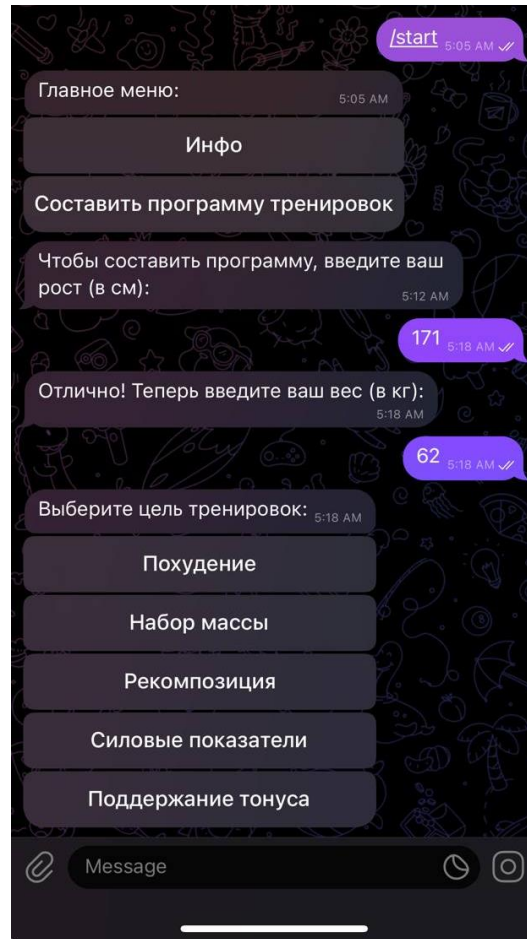


Рисунок 1.26 — Ввод параметров пользователя

Интерфейс данного этапа продуман таким образом, что пользователь сразу видит важную информацию, структурированную по пунктам. Наличие кнопок позволяет моментально перейти к просмотру изображений с техникой выполнения (рисунок 1.27).

Дальнейшая функциональность бота позволяет подробно ознакомиться с техникой выполнения определённых упражнений. При нажатии кнопки «Посмотреть технику» бот отображает список упражнений из сформированной тренировочной программы. Пользователь выбирает интересующее его упражнение, после чего бот инициирует процесс вызова функции, отвечающей за получение изображений.

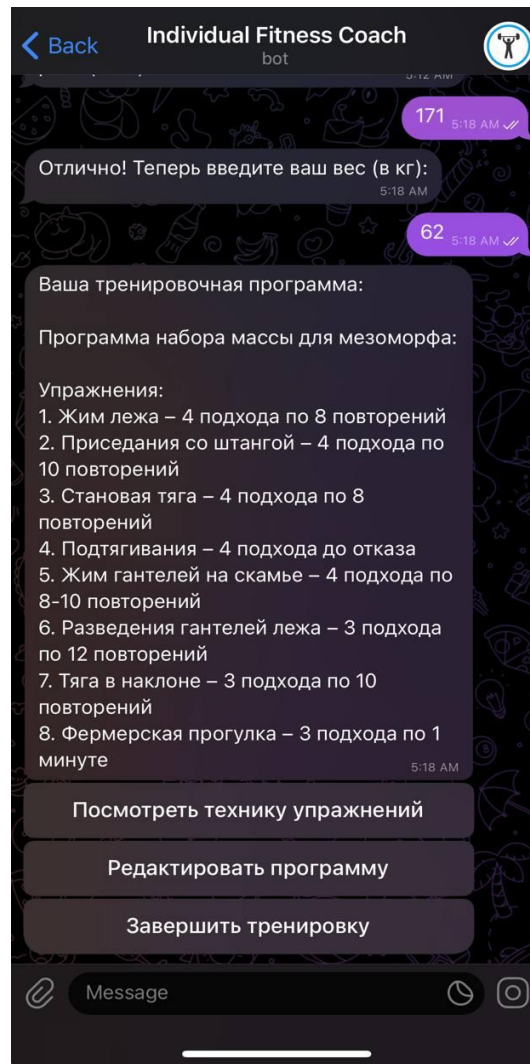


Рисунок 1.27 — Вывод готовой программы

Дальнейшая функциональность бота позволяет подробно ознакомиться с техникой выполнения определённых упражнений. При нажатии кнопки «Посмотреть технику» бот отображает список упражнений из сформированной тренировочной программы. Пользователь выбирает интересующее его упражнение, после чего бот инициирует процесс вызова функции, отвечающей за получение изображений.

В зависимости от наличия локальных файлов (например, изображения правильной техники выполнения всегда проверяются в каталоге с фото), бот либо

сразу отправляет необходимые изображения, либо выполняет обратный вызов к внешним API для получения изображений из интернета. Для пользователя разница прозрачна: он видит последовательное появление медиа-сообщений с поясняющими подписями. Интерфейс этого блока также разработан с учётом удобства: предусмотрены кнопки для перехода обратно к программе или завершения просматриваемого раздела (рисунок 1.28).

Помимо основных функций, бот также предлагает дополнительные опции, такие как завершение тренировки или возврат к предыдущим пунктам меню. Завершение сеанса сопровождается коротким отчетным сообщением, в котором подводится итог выполненной тренировки (рисунок 1.29).



Рисунок 1.28 — Просмотр техники выполнения упражнений

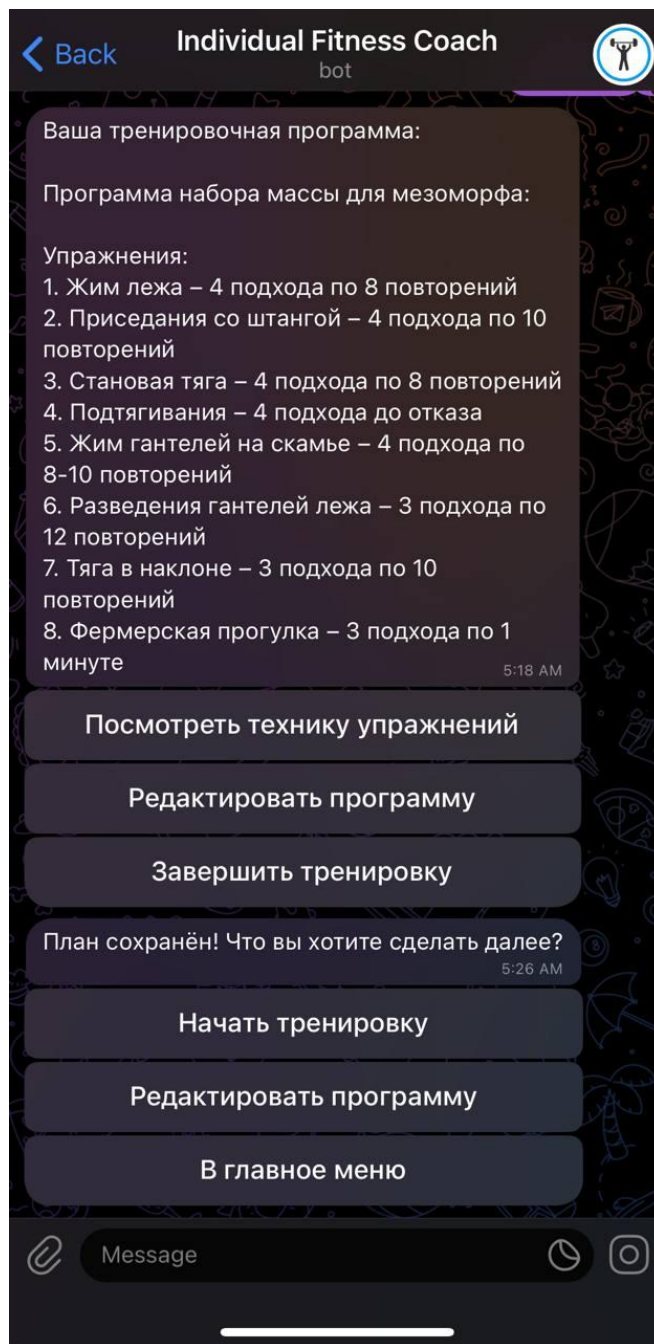


Рисунок 1.29 — Завершение тренировки

2 Безопасность жизнедеятельности

2.1 Правила безопасной работы с персональным компьютером

В настоящем разделе описаны основные меры и рекомендации, направленные на обеспечение безопасности при эксплуатации персонального компьютера, с учетом требований действующих стандартов безопасности труда:

- профилактика зрительного утомления. Длительное нахождение за компьютером без регулярных перерывов может негативно сказываться на работе глаз. Необходимо организовывать обязательные короткие паузы, а также следить за оптимальным уровнем освещенности и правильной настройкой монитора, чтобы уменьшить риск развития головной боли и ухудшения зрения;

- эргономика рабочего места. Неправильная организация рабочего пространства, неэргономичная мебель и неадаптированная поза могут привести к проблемам с позвоночником, мышечному напряжению и хроническим болям в спине и шее. Рекомендуется использовать регулируемые кресла, правильно располагать стол и монитор, а также периодически выполнять гимнастику для спины;

- электромагнитная безопасность. Настоятельно рекомендуется использовать исправное и современное оборудование, поскольку устаревшая техника может генерировать избыточное электромагнитное излучение. Обеспечение безопасного расстояния между ПК и рабочим местом способствует снижению потенциального риска для здоровья;

- пожароопасность и электрическая безопасность. Нарушения правил подключения электрических устройств, использование дефектных кабелей или перегруженные розетки могут привести к короткому замыканию, возгораниям и

поражению электрическим током. Следует строго соблюдать инструкции по эксплуатации электрооборудования, регулярно проверять его исправность и избегать размещения легковоспламеняющихся материалов рядом с источниками теплоты;

– защита информационных ресурсов. Использование простых паролей и неаккуратное обращение с конфиденциальной информацией может привести к утечке данных. Рекомендуется применять надежные средства аутентификации и поддерживать актуальность программного обеспечения для защиты от несанкционированного доступа;

– контроль уровня шума. При работе с компьютером в условиях повышенного фонового шума возникает дополнительное напряжение, что может сказываться на работоспособности и общем самочувствии. Обеспечение акустического комфорта рабочего места является важной составляющей безопасности труда.

2.2 Правила безопасного использования смартфона

При эксплуатации мобильных устройств также необходимо уделять внимание ряду специфических мер предосторожности:

– профилактика негативного влияния на здоровье. Длительное использование смартфона, особенно в вечернее время, может нарушать природный ритм организма, ухудшать качество сна и приводить к повышенной утомляемости. Рекомендуется ограничивать время активного использования устройства и применять функции защиты экрана, снижающие яркость и синюю подсветку;

– защита личных данных. Смартфон является портативным носителем значительного объема личной информации и данных. Потеря или кража такого

устройства может привести к несанкционированному доступу к конфиденциальной информации. Для предотвращения подобных ситуаций рекомендуется применять сложные пароли, биометрическую аутентификацию и регулярно проводить резервное копирование данных;

– минимизация воздействия радиочастотного излучения. Хотя современные исследования не выявляют серьезных опасностей, длительные телефонные разговоры или интенсивное использование устройства в зонах слабого сигнала могут способствовать увеличенному излучению. Использование гарнитур и уменьшение длительности вызовов позволят снизить потенциальное воздействие;

– обеспечение физической безопасности. При нахождении в общественных местах следует соблюдать осторожность, чтобы избежать кражи или утраты смартфона. Рекомендуется не оставлять устройство без присмотра, использовать защитные чехлы, а также применять функции удаленного блокирования в случае его потери.

Соблюдение представленных рекомендаций позволяет снизить риски для здоровья и информационной безопасности, обеспечивая надежную и безопасную эксплуатацию как персонального компьютера, так и мобильных устройств.

Заключение

Выполненная выпускная квалификационная работа, посвящённая созданию чат-бота «Индивидуальный фитнес-тренер» для предоставления персонализированных фитнес-программ через мессенджер Telegram, продемонстрировала успешную реализацию всех поставленных целей. Разработка системы позволила объединить ряд функциональных компонентов, обеспечивающих оперативное взаимодействие с пользователями, корректное формирование тренировочных программ и поддержку безопасности при хранении конфиденциальных данных.

В ходе работы были решены следующие задачи:

- анализ потребностей целевой аудитории. Проведён системный анализ запросов потенциальных пользователей в сфере фитнеса, что позволило выявить основные проблемы и ожидания в вопросах подбора индивидуальных тренировок. Это послужило основой для формулирования требований к функциональным возможностям системы;

- выбор платформы и инструментов разработки. Благодаря широкому функционалу и популярности, в качестве платформы для взаимодействия с пользователями выбран мессенджер Telegram. Использование языка Python обеспечило скорость разработки, гибкость архитектуры и доступ к богатому набору библиотек, что значительно упростило реализацию ключевых функций;

- создание эффективной базы данных. Для хранения информации о тренировочных программах и упражнениях использована база данных SQLite. Это позволило обеспечить быстрый доступ к данным и надёжное их хранение при относительно небольших объёмах информации;

- реализация функционала подбора фитнес-программ. Были разработаны модули, осуществляющие анализ входных параметров (рост, вес, цель

тренировок) и выбор соответствующей программы. Интеграция функции `get_training_program(height_cm:float,weight_kg:float,goal:str) → str` позволила обеспечить персонализацию предлагаемого плана, адаптированного к индивидуальным характеристикам пользователя;

- обработка визуальных материалов. Созданы механизмы для поиска и отображения изображений упражнений. Система сначала пытается получить локальное изображение, а при отсутствии его – обращается к внешнему API, что обеспечивает полноту и актуальность предоставляемых наглядных материалов;

- обеспечение безопасности конфигурации. Особое внимание уделено защите конфиденциальных данных. Реализованы функции шифрования и последующей расшифровки конфигурационных файлов с использованием современной криптографической библиотеки, что гарантирует сохранность токена бота и других критически важных параметров;

- интеграция и тестирование. Все модули систематизированы и объединены в единое программное решение, прошедшее комплексное тестирование на корректность работы и устойчивость к внешним воздействиям. Это позволило обеспечить стабильность системы при непосредственном взаимодействии с пользователями.

В итоговом варианте разработанная информационная система «Индивидуальный фитнес-тренер» предоставляет пользователям возможность быстро получить адаптированную программу тренировок, а также наглядно ознакомиться с техникой выполнения упражнений. Полученное решение обладает потенциалом для дальнейшего развития — его можно расширить дополнительными функциями, такими как мониторинг прогресса, анализ тренировочных данных или интеграция с носимыми устройствами. Таким образом, разработанная система способна внести значительный вклад в

повышение качества фитнес-консультирования и мотивации пользователей к здоровому образу жизни.

Список использованных источников

1. Thecode.media. «Прокачиваем телеграм-бота: пишем полноценный бэкенд»: [Электронный ресурс]. URL: <https://thecode.media/front-back-2/>.
2. python-telegram-bot. Официальная документация: [Электронный ресурс]. URL: <https://python-telegram-bot.readthedocs.io> (дата обращения: 06.05.2025)
3. Habr.com. «Как создать бота в Telegram на Python»: [Электронный ресурс]. URL: <https://habr.com/ru/articles/700000> (дата обращения: 15.05.2025)
4. GitHub: «Создаем чат-бота на python. Полное руководство» [Электронный ресурс]. URL: <https://mastergroosha.github.io/telegram-tutorial/> (дата обращения: 30.05.2025)
5. Stepik.org. «Телеграм-боты на python и aiogram»: [Электронный ресурс]. URL: <https://stepik.org/course/120924/promo>
6. Habr.com. «Полный гайд по разработке telegram-ботов на python и aiogram»: [Электронный ресурс]. URL: <https://habr.com/ru/articles/732136/>
7. КонсультантПлюс.: [Электронный ресурс]. URL: <https://www.consultant.ru> (дата обращения: 30.05.2025)

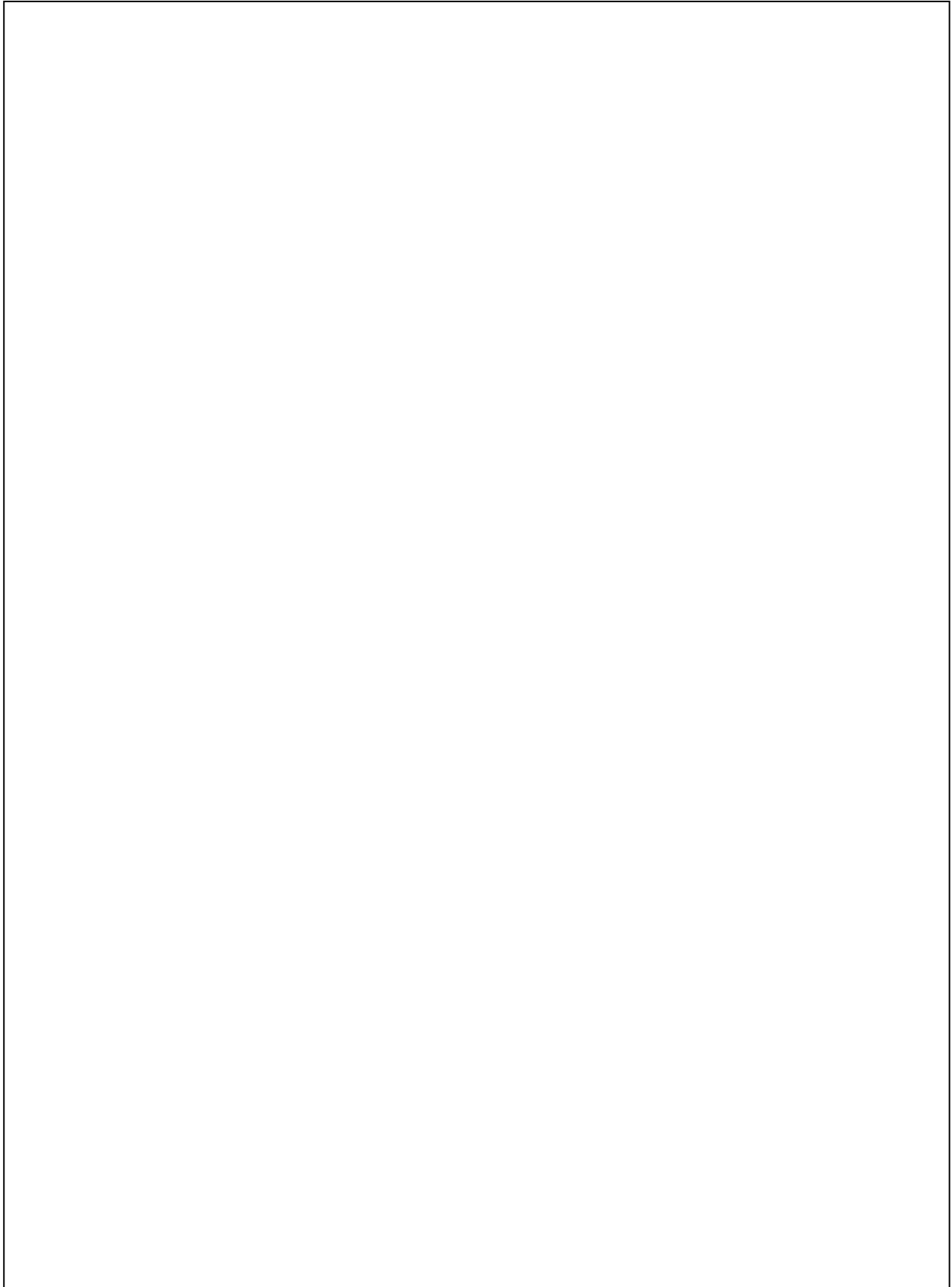
Приложение А

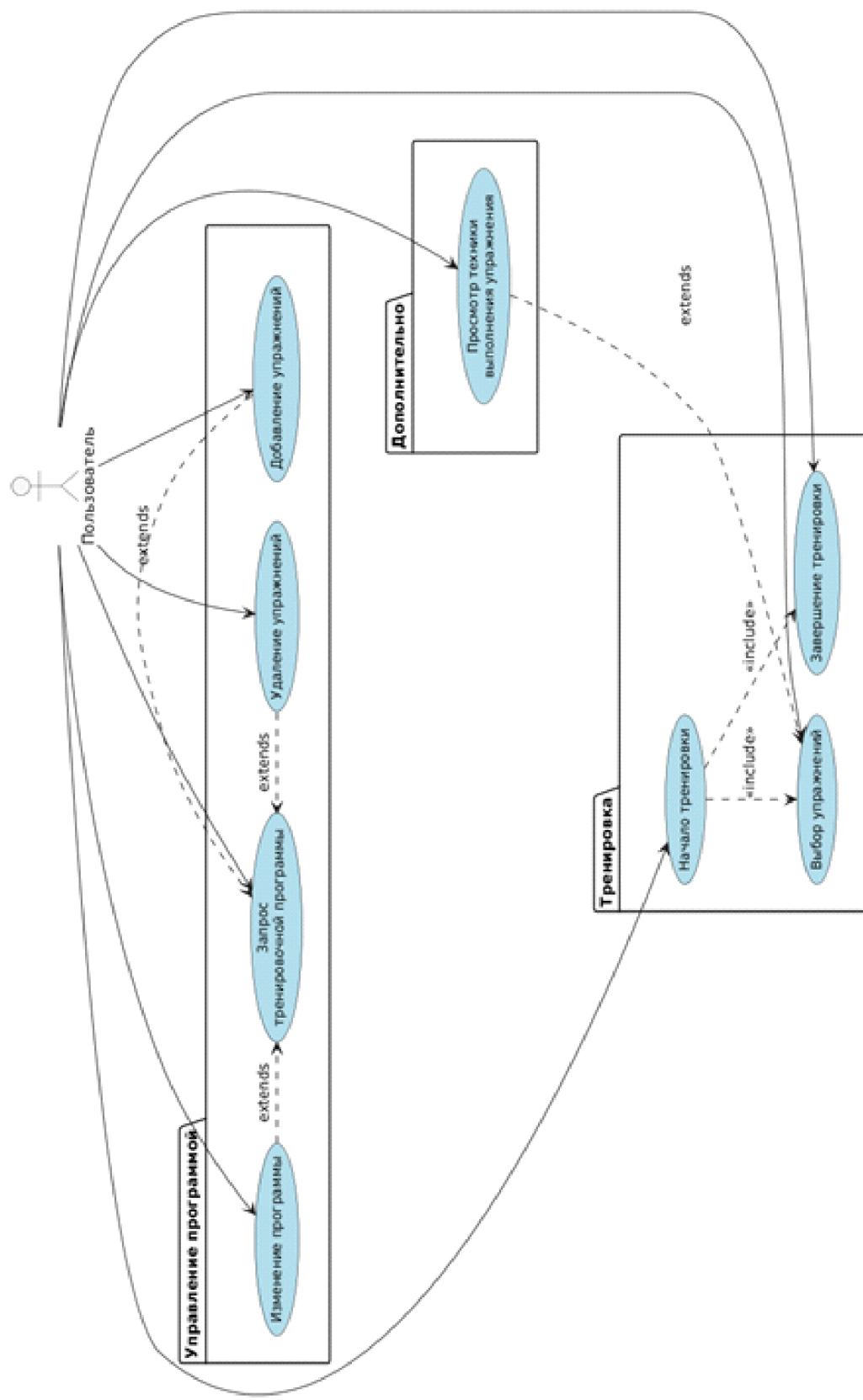
(Обязательное)

Телеграмм-бот «Индивидуальный фитнес-тренер»

UML-диаграммы

Приложение А.1 – UML – диаграмма прецедентов

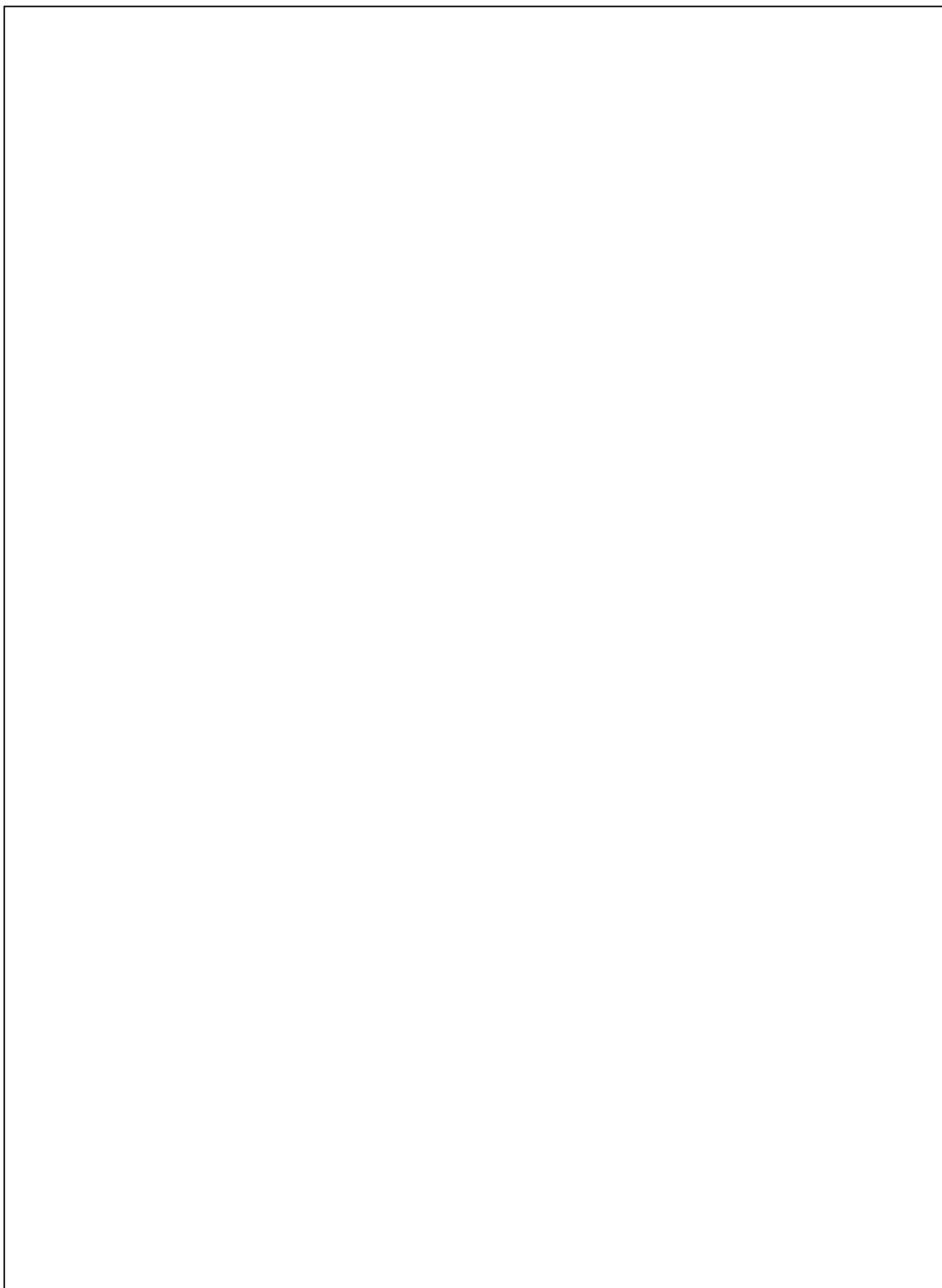




Приложение А.2 – UML – диаграмма последовательности

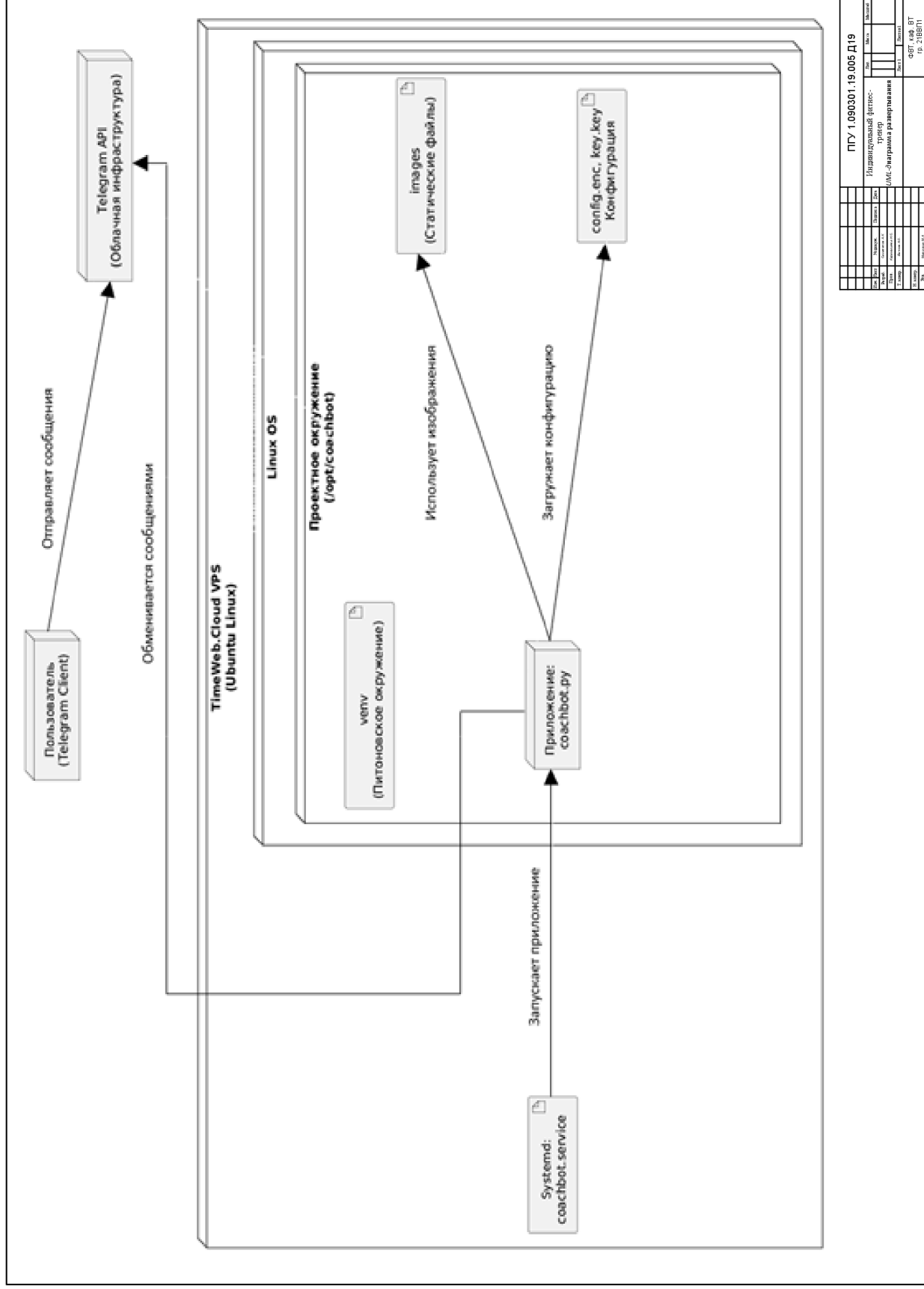


Приложение А.3 – UML – диаграмма деятельности



Приложение А.4 – UML – диаграмма развертывания





Приложение Б

(Обязательное)

Telegramм-бот «Индивидуальный фитнес-тренер»

Листинг программы

Файл coachbot.py:

```
import asyncio
import logging
import nest_asyncio
import warnings
from telegram import Bot
from telegram.error import Conflict
from telegram.ext import (
    ApplicationBuilder,
    CommandHandler,
    CallbackQueryHandler,
    ConversationHandler,
    MessageHandler,
    filters
)

# Импорт загрузчика конфигурации для получения секретного токена
from config_loader import load_config
from config.constants import BOT_TOKEN, TRAINING_HEIGHT,
TRAINING_WEIGHT, TRAINING_GOAL, EDIT_PROGRAM, ADD_EXERCISE

# Загружаем конфигурацию и обновляем значение BOT_TOKEN
config = load_config() # Получаем словарь с настройками
BOT_TOKEN = config.get("bot_token")
if not BOT_TOKEN:
    raise Exception("Не удалось загрузить токен бота из
зашифрованного файла.")

# Импорт обработчиков
from bot.training_handlers import training_start,
process_height, process_weight, process_goal, cancel
from bot.program_handlers import start_edit_program,
edit_program_callback, process_add_exercise,
return_plan_callback
from bot.technique_handlers import list_exercises_callback,
tech_exercise_callback, close_tech_callback
from bot.main_handlers import start, info_callback,
main_menu_callback, start_training_callback
```

```

# Отключаем стандартные предупреждения по per_message
warnings.filterwarnings(
    "ignore",
    message="If 'per_message=True', all entry points, state
handlers, and fallbacks must be 'CallbackQueryHandler'"
)
warnings.filterwarnings(
    "ignore",
    message="If 'per_message=False', 'CallbackQueryHandler' will
not be tracked for every message"
)

# Применяем nest_asyncio для избежания конфликтов с асинхронным
циклом
nest_asyncio.apply()

logging.basicConfig(
    format="%(asctime)s - %(name)s - %(levelname)s -
%(message)s",
    level=logging.INFO
)

async def main():
    # Создаем объект Bot и удаляем вебхук с очисткой ожидающих
    обновлений.
    bot = Bot(BOT_TOKEN)
    await bot.delete_webhook(drop_pending_updates=True)

    # Создаем экземпляр приложения с безопасным токеном
    app = ApplicationBuilder().token(BOT_TOKEN).build()

    # ConversationHandler для создания тренировочной программы.
    training_conv_handler = ConversationHandler(
        entry_points=[CallbackQueryHandler(training_start,
pattern="^training$")],
        states={
            TRAINING_HEIGHT: [MessageHandler(filters.TEXT &
~filters.COMMAND, process_height)],
            TRAINING_WEIGHT: [MessageHandler(filters.TEXT &
~filters.COMMAND, process_weight)],
            TRAINING_GOAL: [CallbackQueryHandler(process_goal,
pattern="^goal_")]
        },
        fallbacks=[CommandHandler("cancel", cancel)]
    )

```

```

    # ConversationHandler для редактирования программы.
    program_conv_handler = ConversationHandler(
        entry_points=[CallbackQueryHandler(start_edit_program,
pattern="^edit_program$")],
        states={
            EDIT_PROGRAM:
[CallbackQueryHandler(edit_program_callback,
pattern="^(del_.*|add|save|cancel_add|finish_training)$")],
            ADD_EXERCISE: [MessageHandler(filters.TEXT &
~filters.COMMAND, process_add_exercise)]
        },
        fallbacks=[CommandHandler("cancel", cancel)]
    )

    # Регистрируем обработчики.
    app.add_handler(CommandHandler("start", start))
    app.add_handler(CallbackQueryHandler(info_callback,
pattern="^info$"))
    app.add_handler(training_conv_handler)
    app.add_handler(program_conv_handler)

app.add_handler(CallbackQueryHandler(start_training_callback,
pattern="^start_training$"))
    app.add_handler(CallbackQueryHandler(main_menu_callback,
pattern="^start$"))

app.add_handler(CallbackQueryHandler(list_exercises_callback,
pattern="^list_exercises$"))
    app.add_handler(CallbackQueryHandler(tech_exercise_callback,
pattern="^tech_ex_"))
    app.add_handler(CallbackQueryHandler(return_plan_callback,
pattern="^return_plan$"))
    app.add_handler(CallbackQueryHandler(close_tech_callback,
pattern="^close_tech$"))
    # Глобальный обработчик для 'finish_training'
    app.add_handler(CallbackQueryHandler(edit_program_callback,
pattern="^finish_training$"))

    try:
        await app.run_polling()
    except Conflict as e:
        logging.error(f"Conflict error occurred: {e}")
        # Если возникает конфликт — убедитесь, что бот не
запущен параллельно.
        return

if __name__ == '__main__':

```

```
asyncio.run(main())
```

Файл server.py:

```
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
from training_logic import get_training_program # логика работы
с базой
from image_service import get_exercise_images # сервис для
получения изображений

app = FastAPI(
    title="FitCoach Service API",
    description="API для подбора программы тренировок и
получения изображений упражнений",
    version="1.0"
)

class ProgramRequest(BaseModel):
    height_cm: float
    weight_kg: float
    goal: str

@app.post("/get_program")
async def get_program(request: ProgramRequest):
    training_program = get_training_program(request.height_cm,
request.weight_kg, request.goal)
    if training_program:
        return {"program": training_program}
    else:
        raise HTTPException(status_code=404, detail="Программа
не найдена для заданных параметров.")

@app.get("/get_exercise_images")
async def get_exercise_images_endpoint(exercise: str):
    images = get_exercise_images(exercise)
    if images and images[0].startswith("http"):
        return {"exercise": exercise, "images": images}
    else:
        raise HTTPException(status_code=404, detail="Изображения
не найдены.")

if __name__ == '__main__':
    import uvicorn
    uvicorn.run("server:app", host="0.0.0.0", port=8000,
reload=True)
```

Файл training_logic.py:

```
from sqlalchemy import create_engine, Column, Integer, String, Text
from sqlalchemy.orm import declarative_base, sessionmaker

# Создание базового класса для моделей
Base = declarative_base()

# Модель таблицы программ тренировок
class Program(Base):
    __tablename__ = 'programs'
    id = Column(Integer, primary_key=True)
    body_type = Column(String(50), nullable=False) # Например,
    "ectomorph", "mesomorph", "endomorph"
    goal = Column(String(50), nullable=False) # Например,
    "loss", "gain", "recomp", "strength", "tonus"
    program_text = Column(Text, nullable=False)

# Создание движка для подключения к базе данных (SQLite)
engine = create_engine('sqlite:///fitness_programs.db')
Session = sessionmaker(bind=engine)

def get_body_type(height_cm: float, weight_kg: float) -> str:
    """
    Определяет тип телосложения на основе индекса массы тела
    (BMI) .

    Возвращает:
    "ectomorph" для BMI < 18.5;
    "mesomorph" для BMI от 18.5 до 25;
    "endomorph" для BMI >= 25.
    """
    height_m = height_cm / 100
    bmi = weight_kg / (height_m ** 2)
    if bmi < 18.5:
        return "ectomorph"
    elif bmi < 25:
        return "mesomorph"
    else:
        return "endomorph"

def get_training_program(height_cm: float, weight_kg: float,
goal: str) -> str:
    """
    Находит и возвращает тренировочную программу из базы данных
    по параметрам пользователя:
```

рост, вес и цель (например, похудение, набор массы и др.).

Если программа не найдена - возвращает сообщение об отсутствии программы.

```
"""
body_type = get_body_type(height_cm, weight_kg)
session = Session()
try:
    program =
session.query(Program).filter_by(body_type=body_type,
goal=goal).first()
    if program:
        return program.program_text
    else:
        return "К сожалению, программа не найдена для
заданных параметров."
finally:
    session.close()
```

Файл image_service.py:

```
import os
import requests
from urllib.parse import quote_plus
import logging
import re

from exercise_images import LOCAL_EXERCISE_IMAGE_MAP

def normalize_text(s: str) -> str:
    """
    Приводит текст к стандартному виду: обрезает пробелы,
    переводит в нижний регистр,
    удаляет знаки препинания.
    """
    s = s.strip().lower()
    s = re.sub(r'[^\\w\\s]', '', s) # удаляем символы, не
являющиеся буквенно-цифровыми или пробелами
    return s

def get_local_image(exercise_name: str) -> str:
    """
    Пытается найти локальное изображение для данного названия
    упражнения.
```

1. Сначала выполняется точное совпадение после нормализации.

2. Если точного совпадения нет, выбираются все кандидаты, у которых нормализованный ключ содержится в нормализованном названии упражнения. Из них выбирается тот, у которого длина ключа максимальна (более специфичное совпадение).

Возвращает путь к изображению, если такой найден, иначе - None.

```
"""
normalized = normalize_text(exercise_name)
logging.debug("Normalized exercise name: '%s'", normalized)

# 1. Точное совпадение для каждого ключа
for key, path in LOCAL_EXERCISE_IMAGE_MAP.items():
    norm_key = normalize_text(key)
    if normalized == norm_key and os.path.exists(path):
        logging.info("Exact match found for '%s': %s",
exercise_name, path)
        return path

# 2. Поиск кандидатов по подстроке
candidates = []
for key, path in LOCAL_EXERCISE_IMAGE_MAP.items():
    norm_key = normalize_text(key)
    if norm_key in normalized and os.path.exists(path):
        logging.debug("Candidate found for key '%s': %s",
key, path)
        candidates.append((norm_key, path))
if candidates:
    # Выбираем кандидата с максимальной длиной ключа (более
специфичный)
    candidates.sort(key=lambda x: len(x[0]), reverse=True)
    chosen = candidates[0][1]
    logging.info("Returning candidate for '%s': %s",
exercise_name, chosen)
    return chosen
return None

def get_exercise_images(exercise_name: str) -> list:
    """
    Возвращает список изображений для упражнения.

    Если для упражнения найдено локальное изображение (через
get_local_image),
    функция сразу возвращает путь к файлу. Если локального файла
нет - выполняется
    запрос через Google Custom Search API.
```

```

"""
local_image = get_local_image(exercise_name)
if local_image:
    logging.info("Returning local image for '%s': %s",
exercise_name, local_image)
    return [local_image]

# Если локальное изображение не найдено, выполняется поиск
через Google Custom Search API.
api_key = "AIzaSyBK8rwN0ha8rrXlw_1tUE5_qP0wIOX0PN8"
cse_id = "35e19009f61534992"

query = f"{exercise_name} техника выполнения"
encoded_query = quote_plus(query)

url = (
    f"https://www.googleapis.com/customsearch/v1?"
    f"key={api_key}"
    f"&cx={cse_id}"
    f"&q={encoded_query}"
    f"&searchType=image"
    f"&imgSize=huge"
    f"&imgType=photo"
    f"&fileType=jpg"
    f"&safe=active"
    f"&num=10"
)

try:
    response = requests.get(url)
    response.raise_for_status()
    results = response.json()
    images = []
    if "items" in results:
        for item in results["items"]:
            image_url = item.get("link")
            if image_url:
                images.append(image_url)
        if images:
            return images
        else:
            return ["Изображение не найдено."]
    else:
        return ["Изображение не найдено."]
except Exception as e:
    return [f"Ошибка при запросе: {e}"]

```

Файл main_handlers.py:

```
import logging
from telegram import Update, InlineKeyboardButton,
InlineKeyboardMarkup
from telegram.ext import ContextTypes

async def start(update: Update, context:
ContextTypes.DEFAULT_TYPE) -> None:
    """
    Отправляет стартовое меню с опциями "Инфо" и "Составить
программу тренировок".
    Если существует сохранённая программа или сгенерированная
программа, добавляет кнопку "Начать тренировку".
    """
    buttons = [
        [InlineKeyboardButton("Инфо", callback_data="info")],
        [InlineKeyboardButton("Составить программу тренировок",
callback_data="training")]
    ]
    if ((context.user_data.get("saved_plan") and
context.user_data.get("saved_plan").strip()) or
        (context.user_data.get("training_program_text") and
context.user_data.get("training_program_text").strip())):
        buttons.insert(1, [InlineKeyboardButton("Начать
тренировку", callback_data="start_training")])
    reply_markup = InlineKeyboardMarkup(buttons)
    if update.message:
        await update.message.reply_text("Привет! Я твой
телеграм-бот. Выбери опцию:", reply_markup=reply_markup)
    elif update.callback_query:
        await update.callback_query.edit_message_text("Привет! Я
твой телеграм-бот. Выбери опцию:", reply_markup=reply_markup)

async def info_callback(update: Update, context:
ContextTypes.DEFAULT_TYPE) -> None:
    """
    Выводит информацию о боте, добавляя кнопку "Назад" для
возврата в главное меню.
    """
    query = update.callback_query
    await query.answer()
    info_text = (
        "Добро пожаловать в FitCoach Bot!\n\n"
        "Я – твой персональный фитнес-консультант, который
поможет составить оптимальную программу тренировок "
```

```

        "и показать технику упражнений. Просто выбери нужную
опцию."
    )
    keyboard = [
        [InlineKeyboardButton("Составить программу тренировок",
callback_data="training")],
        [InlineKeyboardButton("Назад", callback_data="start")]
    ]
    reply_markup = InlineKeyboardMarkup(keyboard)
    await query.edit_message_text(text=info_text,
reply_markup=reply_markup)

```

```

async def main_menu_callback(update: Update, context:
ContextTypes.DEFAULT_TYPE) -> None:
    """

```

```

        Возвращает пользователя в главное меню. Если существует
сохранённая программа или сгенерированный текст,
добавляет кнопку "Начать тренировку".
    """

```

```

    query = update.callback_query
    await query.answer()
    buttons = [
        [InlineKeyboardButton("Инфо", callback_data="info")],
        [InlineKeyboardButton("Составить программу тренировок",
callback_data="training")]
    ]
    if ((context.user_data.get("saved_plan") and
context.user_data.get("saved_plan").strip()) or
        (context.user_data.get("training_program_text") and
context.user_data.get("training_program_text").strip())):
        buttons.insert(1, [InlineKeyboardButton("Начать
тренировку", callback_data="start_training")])
    reply_markup = InlineKeyboardMarkup(buttons)
    await query.edit_message_text("Главное меню:",
reply_markup=reply_markup)

```

```

async def start_training_callback(update: Update, context:
ContextTypes.DEFAULT_TYPE) -> int:
    """

```

```

        Показывает сохранённую тренировочную программу, а также
добавляет кнопку "Завершить тренировку".
    """

```

```

    query = update.callback_query
    await query.answer()
    saved_plan = ""
    if context.user_data.get("saved_plan"):
        saved_plan = context.user_data.get("saved_plan")

```

```

        elif context.user_data.get("training_program_text"):
            saved_plan =
context.user_data.get("training_program_text")
            saved_plan = saved_plan.strip() if saved_plan else ""
            if saved_plan:
                keyboard = [
                    [InlineKeyboardButton("Посмотреть технику
упражнений", callback_data="list_exercises")],
                    [InlineKeyboardButton("Редактировать программу",
callback_data="edit_program")],
                    [InlineKeyboardButton("Завершить тренировку",
callback_data="finish_training")],
                    [InlineKeyboardButton("В главное меню",
callback_data="start")]
                ]
                reply_markup = InlineKeyboardMarkup(keyboard)
                await query.edit_message_text(
                    text=f"Ваша программа тренировки:\n\n{saved_plan}",
                    reply_markup=reply_markup
                )
            else:
                await query.edit_message_text("План не найден.
Пожалуйста, составьте программу тренировок!")
                return 0

```

Файл `technique_handlers.py`:

```

import asyncio
import io
import logging
import requests

from telegram import Update, InlineKeyboardButton,
InlineKeyboardMarkup, InputMediaPhoto
from telegram.ext import ContextTypes
from telegram.error import BadRequest
from utils.helpers import extract_exercises,
generate_callback_data
from image_service import get_exercise_images # импорт для
получения изображений

async def list_exercises_callback(update: Update, context:
ContextTypes.DEFAULT_TYPE) -> None:
    """
    Отображает меню упражнений для запроса техники.
    Если не удаётся извлечь список упражнений из
    сгенерированного текста,

```

используется список, сохранённый в пользовательских данных.

```
При формировании кнопок сохраняем сопоставление
callback_data с полным названием упражнения,
чтобы потом можно было восстановить оригинальное имя.
"""

query = update.callback_query
await query.answer("Обработка запроса...")
training_program_text =
context.user_data.get("training_program_text")
if not training_program_text:
    await query.message.reply_text("Программа не найдена.
Повторите запрос.")
    return

logging.info("Training program text: %s",
training_program_text)
exercises = extract_exercises(training_program_text)
if not exercises:
    logging.warning("Regex extraction не дала результата.
Используем сохранённые упражнения.")
    exercises = context.user_data.get("program_exercises",
[])

if not exercises:
    await query.message.reply_text("Не удалось извлечь
список упражнений. Проверьте формат программы.")
    return

callback_mapping = {}
exercise_buttons = []
for ex in exercises:
    cb_data = generate_callback_data(ex)
    callback_mapping[cb_data] = ex # сохраняем полное имя
упражнения
    exercise_buttons.append([InlineKeyboardButton(ex,
callback_data=cb_data)])
context.user_data["callback_mapping"] = callback_mapping

extra_buttons = [
    InlineKeyboardButton("Вернуться к плану",
callback_data="return_plan"),
    InlineKeyboardButton("Завершить тренировку",
callback_data="finish_training"),
    InlineKeyboardButton("В главное меню",
callback_data="start")
]
exercise_buttons.append(extra_buttons)
```

```
    reply_markup = InlineKeyboardMarkup(exercise_buttons)
    await query.message.reply_text("Выберите упражнение для
просмотра техники:", reply_markup=reply_markup)
```

```
async def tech_exercise_callback(update: Update, context:
ContextTypes.DEFAULT_TYPE) -> None:
    """
```

Получает и отправляет пользователю изображения с техникой выполнения выбранного упражнения.

Для определения полного названия упражнения используется сохранённый mapping.

При отправке медиа-группы, если изображение локальное - открывается напрямую, иначе загружается из интернета.

```
    """
    query = update.callback_query
    await query.answer()
    data = query.data
    prefix = "tech_ex_"
    callback_mapping = context.user_data.get("callback_mapping",
    {})
    # Получаем полное название упражнения из mapping, если оно
    сохранено
    exercise_name = callback_mapping.get(data,
    data[len(prefix):])
    formatted_exercise = exercise_name # предполагаем, что
    exercise_name уже отформатировано

    # Получаем список изображений (локальный путь или URL)
    loop = asyncio.get_running_loop()
    images = await loop.run_in_executor(None,
    get_exercise_images, exercise_name)

    media = []
    headers = {
        "User-Agent": (
            "Mozilla/5.0 (Windows NT 10.0; Win64; x64) "
            "AppleWebKit/537.36 (KHTML, like Gecko) "
            "Chrome/115.0.0.0 Safari/537.36"
        )
    }
    count = 0
    max_images = 2
    for image_source in images:
        if count >= max_images:
            break
        try:
```

```

        if image_source.startswith("http"):
            # Загрузка изображения через интернет
            response = await asyncio.to_thread(requests.get,
image_source, headers=headers, timeout=10)
            response.raise_for_status()
            image_content = response.content
        else:
            # Локальный файл: открываем его напрямую
            with open(image_source, "rb") as f:
                image_content = f.read()

        image_bytes = io.BytesIO(image_content)
        from PIL import Image
        try:
            im = Image.open(image_bytes)
            im = im.convert("RGB")
            new_image_bytes = io.BytesIO()
            im.save(new_image_bytes, format="JPEG")
            new_image_bytes.seek(0)
            new_image_bytes.name = f"image_{count}.jpg"
        except Exception as e:
            logging.error("Ошибка обработки изображения от
%s: %s", image_source, e)
            continue

        caption = f"Техника выполнения:
{formatted_exercise}" if count == 0 else None
        media.append(InputMediaPhoto(media=new_image_bytes,
caption=caption))
        count += 1
    except Exception as e:
        logging.error("Ошибка загрузки изображения %s: %s",
image_source, e)

    if media:
        try:
            sent_media = await
query.message.reply_media_group(media=media)
            media_message_ids = [msg.message_id for msg in
sent_media]
            context.user_data["tech_media_ids"] =
media_message_ids
        except Exception as e:
            logging.error("Ошибка отправки медиagrппы: %s", e)
        try:

```

```

        sent_msg = await
query.message.reply_photo(photo=media[0].media,
caption=f"Техника выполнения: {formatted_exercise}")
        media_message_ids = [sent_msg.message_id]
        context.user_data["tech_media_ids"] =
media_message_ids
        except Exception as e2:
            logging.error("Ошибка отправки фото: %s", e2)
            keyboard = [[InlineKeyboardButton("Продолжить",
callback_data="return_plan")]]
            reply_markup = InlineKeyboardMarkup(keyboard)
            await query.message.reply_text(
                "Извините, не удалось загрузить изображение.
Нажмите 'Продолжить' для возврата к плану.",
                reply_markup=reply_markup
            )
            return
        sent_close_msg = await query.message.reply_text(
            "Нажмите 'Закрыть окно техники' для закрытия окна
техники.",

reply_markup=InlineKeyboardMarkup([[InlineKeyboardButton("Закрыт
ь окно техники", callback_data="close_tech")]])
        )
        context.user_data["tech_close_msg_id"] =
sent_close_msg.message_id
        else:
            keyboard = [[InlineKeyboardButton("Продолжить",
callback_data="return_plan")]]
            reply_markup = InlineKeyboardMarkup(keyboard)
            await query.message.reply_text(
                "Извините, не удалось загрузить ни одно изображение.
Нажмите 'Продолжить' для возврата к плану.",
                reply_markup=reply_markup
            )

async def close_tech_callback(update: Update, context:
ContextTypes.DEFAULT_TYPE) -> None:
    """
    Удаляет все сообщения, связанные с отображением техники:
    - Сообщение с кнопкой "Закрыть окно техники" (текущее).
    - Сообщения с медиа (фотографиями), идентификаторы которых
были сохранены.
    """
    query = update.callback_query
    await query.answer()

```

```

try:
    await query.delete_message()
except BadRequest as e:
    if "message to delete not found" in str(e).lower():
        logging.info("Сообщение с кнопкой закрытия уже
удалено.")
    else:
        logging.error("Ошибка удаления сообщения с кнопкой
закрытия: %s", e)

    chat_id = query.message.chat_id
    bot = context.bot
    tech_media_ids = context.user_data.get("tech_media_ids", [])
    for message_id in tech_media_ids:
        try:
            await bot.delete_message(chat_id=chat_id,
message_id=message_id)
        except BadRequest as e:
            if "message to delete not found" in str(e).lower():
                logging.info("Сообщение с медиа (ID: %s) уже
удалено.", message_id)
            else:
                logging.error("Ошибка удаления сообщения с
медиа: %s", e)

    tech_close_msg_id =
context.user_data.get("tech_close_msg_id")
    if tech_close_msg_id:
        try:
            await bot.delete_message(chat_id=chat_id,
message_id=tech_close_msg_id)
        except BadRequest as e:
            if "message to delete not found" in str(e).lower():
                logging.info("Сообщение с кнопкой закрытия (ID:
%s) уже удалено.", tech_close_msg_id)
            else:
                logging.error("Ошибка удаления сообщения с
кнопкой закрытия: %s", e)

    context.user_data.pop("tech_media_ids", None)
    context.user_data.pop("tech_close_msg_id", None)

```

Файл training_handlers.py:

```

import logging
from telegram import Update, InlineKeyboardButton,
InlineKeyboardMarkup, ReplyKeyboardRemove

```

```

from telegram.ext import ContextTypes, ConversationHandler
from config.constants import TRAINING_HEIGHT, TRAINING_WEIGHT,
TRAINING_GOAL

async def training_start(update: Update, context:
ContextTypes.DEFAULT_TYPE) -> int:
    """
    Запускает диалог создания программы: запрашивает рост у
пользователя.
    Вместо редактирования исходного сообщения отправляет новое,
чтобы сохранить приветственное/информационное сообщение.
    """
    query = update.callback_query
    await query.answer()
    await query.message.reply_text("Чтобы составить программу,
введите ваш рост (в см):")
    return TRAINING_HEIGHT

async def process_height(update: Update, context:
ContextTypes.DEFAULT_TYPE) -> int:
    """
    Обрабатывает ввод роста. Если ввод корректен, запрашивает
вес.
    """
    try:
        height = float(update.message.text)
        context.user_data["height"] = height
        await update.message.reply_text("Отлично! Теперь введите
ваш вес (в кг):")
        return TRAINING_WEIGHT
    except ValueError:
        await update.message.reply_text("Введите корректное
числовое значение для роста.")
        return TRAINING_HEIGHT

async def process_weight(update: Update, context:
ContextTypes.DEFAULT_TYPE) -> int:
    """
    Обрабатывает ввод веса. Если ввод корректен, предлагает
выбрать цель тренировок.
    """
    try:
        weight = float(update.message.text)
        context.user_data["weight"] = weight
        keyboard = [
            [InlineKeyboardButton("Похудение",
callback_data="goal_loss")],

```

```

        [InlineKeyboardButton("Набор массы",
callback_data="goal_gain")],
        [InlineKeyboardButton("Рекомпозиция",
callback_data="goal_recomp")],
        [InlineKeyboardButton("Силовые показатели",
callback_data="goal_strength")],
        [InlineKeyboardButton("Поддержание тонуса",
callback_data="goal_tonus")]
    ]
    reply_markup = InlineKeyboardMarkup(keyboard)
    await update.message.reply_text("Выберите цель
тренировок:", reply_markup=reply_markup)
    return TRAINING_GOAL
except ValueError:
    await update.message.reply_text("Введите корректное
числовое значение для веса.")
    return TRAINING_WEIGHT

async def process_goal(update: Update, context:
ContextTypes.DEFAULT_TYPE) -> int:
    """
    Обработывает выбор цели, получает тренировочную программу и
предлагает дальнейшие действия.
    Вместо отправки двух сообщений редактирует существующее,
чтобы избежать дублирования.
    """
    query = update.callback_query
    await query.answer()
    goal = query.data.replace("goal_", "")
    context.user_data["goal"] = goal

    from training_logic import get_training_program
    training_program_text = get_training_program(
        context.user_data["height"],
        context.user_data["weight"],
        goal
    )
    context.user_data["training_program_text"] =
training_program_text

    keyboard = [
        [InlineKeyboardButton("Посмотреть технику упражнений",
callback_data="list_exercises")],
        [InlineKeyboardButton("Редактировать программу",
callback_data="edit_program")],
        [InlineKeyboardButton("Завершить тренировку",
callback_data="finish_training")]
    ]

```

```

    ]
    reply_markup = InlineKeyboardMarkup(keyboard)
    await query.edit_message_text(
        text=f"Ваша тренировочная
программа:\n\n{training_program_text}",
        reply_markup=reply_markup
    )
    return ConversationHandler.END

async def cancel(update: Update, context:
ContextTypes.DEFAULT_TYPE) -> int:
    """
    Завершает диалог создания программы.
    """
    await update.message.reply_text("Диалог отменен.",
reply_markup=ReplyKeyboardRemove())
    return ConversationHandler.END

```

Файл program_handlers.py:

```

import logging
from telegram import Update, InlineKeyboardButton,
InlineKeyboardMarkup
from telegram.ext import ContextTypes, ConversationHandler
from config.constants import EDIT_PROGRAM, ADD_EXERCISE
from utils.helpers import extract_exercises

async def start_edit_program(update: Update, context:
ContextTypes.DEFAULT_TYPE) -> int:
    """
    Запускает режим редактирования программы. Если список
упражнений ещё не заполнен,
    пытается извлечь упражнения из ранее сгенерированного текста
программы.
    """
    if not context.user_data.get("program_exercises"):
        training_text =
context.user_data.get("training_program_text", "")
        extracted = extract_exercises(training_text)
        context.user_data["program_exercises"] = extracted if
extracted else []
    return await show_program(update, context)

async def show_program(update: Update, context:
ContextTypes.DEFAULT_TYPE) -> int:
    """
    Формирует и отображает меню редактирования программы.

```

Выводит текущую программу и генерирует кнопки для редактирования.

```
"""
    exercises = context.user_data.get("program_exercises", [])
    text = (
        "Текущая программа тренировок:\n\n" +
        "\n".join(f"{i + 1}. {ex}" for i, ex in
enumerate(exercises))
        if exercises else "Программа пуста."
    )
    keyboard = [[InlineKeyboardButton(f"Удалить «{ex}»",
callback_data=f"del_{i}")]
                for i, ex in enumerate(exercises)]
    keyboard.append([InlineKeyboardButton("Добавить упражнение",
callback_data="add")])
    keyboard.append([InlineKeyboardButton("Сохранить программу",
callback_data="save")])
    reply_markup = InlineKeyboardMarkup(keyboard)
    if update.callback_query:
        await update.callback_query.edit_message_text(text=text,
reply_markup=reply_markup)
    else:
        await update.message.reply_text(text,
reply_markup=reply_markup)
    return EDIT_PROGRAM
```

```
async def edit_program_callback(update: Update, context:
ContextTypes.DEFAULT_TYPE) -> int:
```

```
"""
    Обрабатывает действия в режиме редактирования программы и
устраняет дублирование сообщений.
```

```
"""
    query = update.callback_query
    await query.answer()
    data = query.data
    exercises = context.user_data.get("program_exercises", [])

    if data.startswith("del_"):
        index = int(data.split("_")[1])
        if 0 <= index < len(exercises):
            removed = exercises.pop(index)
            await query.answer(text=f"Удалено: {removed}")
    elif data == "add":
        await query.edit_message_text("Введите название
упражнения, которое хотите добавить:")
        return ADD_EXERCISE
    elif data == "save":
```

```

        final_text = "Ваш окончательный план тренировок:\n\n" +
"\n".join(
    f"{i + 1}. {ex}" for i, ex in enumerate(exercises)
)
await query.edit_message_text(final_text)
keyboard = [
    [InlineKeyboardButton("Посмотреть технику
упражнений", callback_data="list_exercises")],
    [InlineKeyboardButton("Редактировать программу",
callback_data="edit_program")],
    [InlineKeyboardButton("Завершить тренировку",
callback_data="finish_training")],
    [InlineKeyboardButton("В главное меню",
callback_data="start")]
]
reply_markup = InlineKeyboardMarkup(keyboard)
await query.message.reply_text("Выберите дальнейшее
действие:", reply_markup=reply_markup)
return EDIT_PROGRAM
elif data == "finish_training":
    final_text = (
        "Ваш окончательный план тренировок:\n\n" +
        "\n".join(f"{i + 1}. {ex}" for i, ex in
enumerate(exercises))
    ) if exercises else
context.user_data.get("training_program_text", "").strip()
    if not final_text:
        final_text = "Программа тренировок не сформирована."
    context.user_data["saved_plan"] = final_text

    # Здесь мы больше не удаляем сообщение,
    # чтобы сохранить историю сообщений с момента запуска
бота до выдачи готовой программы.
    keyboard = [
        [InlineKeyboardButton("Начать тренировку",
callback_data="start_training")],
        [InlineKeyboardButton("Редактировать программу",
callback_data="edit_program")],
        [InlineKeyboardButton("В главное меню",
callback_data="start")]
    ]
    reply_markup = InlineKeyboardMarkup(keyboard)
    await query.edit_message_text("План сохранён! Что вы
хотите сделать далее?", reply_markup=reply_markup)
    return ConversationHandler.END
elif data == "cancel_add":
    return await show_program(update, context)

```

```

        elif data.startswith("add_exercise_"):
            exercise_to_add = data[len("add_exercise_"):]
            exercises.append(exercise_to_add)
            await query.answer(text=f"Добавлено: {exercise_to_add}")

    return await show_program(update, context)

async def process_add_exercise(update: Update, context:
ContextTypes.DEFAULT_TYPE) -> int:
    """
    Обработывает ввод нового упражнения для добавления в
    программу.
    """
    exercise_name = update.message.text.strip()
    if exercise_name:
        context.user_data.setdefault("program_exercises",
[[]]).append(exercise_name)
        await update.message.reply_text(f"Упражнение
'{exercise_name}' добавлено.")
    else:
        await update.message.reply_text("Название не должно быть
пустым. Попробуйте ещё раз.")
    return await show_program(update, context)

async def return_plan_callback(update: Update, context:
ContextTypes.DEFAULT_TYPE) -> None:
    """
    Обработывает действие "Вернуться к плану": возвращает
    пользователя к просмотру текущего плана тренировок.
    """
    query = update.callback_query
    await query.answer()
    plan = context.user_data.get("saved_plan") or
context.user_data.get("training_program_text", "План
отсутствует.")
    keyboard = [
        [InlineKeyboardButton("Посмотреть технику упражнений",
callback_data="list_exercises")],
        [InlineKeyboardButton("Редактировать программу",
callback_data="edit_program")],
        [InlineKeyboardButton("Завершить тренировку",
callback_data="finish_training")],
        [InlineKeyboardButton("В главное меню",
callback_data="start")]
    ]
    reply_markup = InlineKeyboardMarkup(keyboard)

```

```
await query.edit_message_text(text=f"Ваш план
тренировки:\n\n{plan}", reply_markup=reply_markup)
```

Файл helpers.py:

```
import re
import logging

def extract_exercises(text: str) -> list:
    """
    Извлекает упражнения из тренировочной программы.
    Пример: "1. Приседания - 3 подхода, 12 повторений" →
    "Приседания - 3 подхода, 12 повторений"
    """
    pattern = re.compile(r'^\d+\.\s*(.*)', re.MULTILINE)
    matches = pattern.findall(text)
    exercises = [match.strip() for match in matches if
    match.strip()]
    logging.info("Extracted exercises: %s", exercises)
    return exercises

def generate_callback_data(exercise: str) -> str:
    """
    Генерирует callback_data для кнопки на основе текста
    упражнения.
    Обеспечивает, что размер данных не превышает 64 байта.
    """
    base = "tech_ex_"
    safe = re.sub(r'\W+', '_', exercise)
    callback_data = base + safe
    while len(callback_data.encode('utf-8')) > 64:
        callback_data = callback_data[:-1]
    return callback_data
```

Файл db_setup.py:

```
from sqlalchemy import Column, Integer, String, Text,
ForeignKey, create_engine
from sqlalchemy.orm import declarative_base, sessionmaker
from sqlalchemy.exc import SQLAlchemyError

# Создаем базовый класс для наших моделей
Base = declarative_base()

# Таблица с программами тренировок
class Program(Base):
```

```

    __tablename__ = "programs"
    id = Column(Integer, primary_key=True)
    body_type = Column(String(50), nullable=False) #
    "ectomorph", "mesomorph", "endomorph"
    goal = Column(String(50), nullable=False) # "loss",
    "gain", "recomp", "strength", "tonus"
    program_text = Column(Text, nullable=False)

# (Опционально) Таблица с упражнениями для каждой программы
class Exercise(Base):
    __tablename__ = "exercises"
    id = Column(Integer, primary_key=True)
    program_id = Column(Integer, ForeignKey("programs.id"),
    nullable=False)
    exercise_name = Column(String(100), nullable=False)
    technique_image_url = Column(String(255)) # URL на картинку
с техникой выполнения

# Создаем движок для подключения к SQLite базе
engine = create_engine("sqlite:///fitness_programs.db",
echo=True)

# Удаляем старые таблицы (осторожно - данные будут потеряны) и
создаем новые
Base.metadata.drop_all(engine)
Base.metadata.create_all(engine)

# Создаем сессию для работы с базой данных
Session = sessionmaker(bind=engine)
session = Session()

# Обновленные данные программ тренировок с большим количеством
упражнений
programs_data = [
    # Экторморф
    {
        "body_type": "ectomorph",
        "goal": "loss",
        "program_text": (
            "Программа похудения для эктоморфа:\n\n"
            "Упражнения:\n"
            "1. Легкий бег или кардио - 30 минут\n"
            "2. Приседания с собственным весом - 3 подхода по 15
повторений\n"
            "3. Отжимания - 3 подхода по 12 повторений\n"
            "4. Планка - 3 подхода по 60 секунд\n"

```

```

        "5. Выпады с собственным весом - 3 подхода по 12
повторений\n"
        "6. Скручивания для пресса - 3 подхода по 20
повторений\n"
        "7. Махи ногами - 3 подхода по 15 повторений\n"
        "8. Заключительная растяжка - 5-10 минут"
    )
},
{
    "body_type": "ectomorph",
    "goal": "gain",
    "program_text": (
        "Программа набора массы для эктоморфа:\n\n"
        "Упражнения:\n"
        "1. Жим лежа - 4 подхода по 8-10 повторений\n"
        "2. Приседания со штангой - 4 подхода по 10
повторений\n"
        "3. Становая тяга - 4 подхода по 8 повторений\n"
        "4. Подтягивания - 4 подхода до отказа\n"
        "5. Подъем гантелей на бицепс - 3 подхода по 10
повторений\n"
        "6. Разведение гантелей на горизонтальной скамье - 3
подхода по 12 повторений\n"
        "7. Фермерская прогулка - 3 подхода по 1 минуте\n"
        "8. Завершающая растяжка - 5-10 минут"
    )
},
{
    "body_type": "ectomorph",
    "goal": "recomp",
    "program_text": (
        "Программа рекомпозиции для эктоморфа:\n\n"
        "Упражнения:\n"
        "1. Интервальный бег - 20 минут (чередование высоких
и низких темпов)\n"
        "2. Жим гантелей - 4 подхода по 10 повторений\n"
        "3. Тяга верхнего блока - 4 подхода по 12
повторений\n"
        "4. Выпады с гантелями - 3 подхода по 12 повторений
на каждую ногу\n"
        "5. Планка - 3 подхода по 60 секунд\n"
        "6. Бёрпи - 3 подхода по 10 повторений\n"
        "7. Скручивания корпуса - 3 подхода по 20
повторений\n"
        "8. Заключительная растяжка - 5-10 минут"
    )
},

```

```

{
  "body_type": "ectomorph",
  "goal": "strength",
  "program_text": (
    "Программа для увеличения силовых показателей для
эктоморфа:\n\n"
    "Упражнения:\n"
    "1. Жим штанги лежа - 5 подходов по 5 повторений\n"
    "2. Приседания со штангой - 5 подходов по 5
повторений\n"
    "3. Становая тяга - 5 подходов по 3-5 повторений\n"
    "4. Армейский жим - 4 подхода по 6 повторений\n"
    "5. Подтягивания - 4 подхода до отказа\n"
    "6. Тяга гантелей в наклоне - 4 подхода по 8
повторений\n"
    "7. Отжимания на брусьях - 3 подхода до отказа\n"
    "8. Плиометрические прыжки - 3 подхода по 10
повторений"
  )
},
{
  "body_type": "ectomorph",
  "goal": "tonus",
  "program_text": (
    "Программа для поддержания тонуса для
эктоморфа:\n\n"
    "Упражнения:\n"
    "1. Кардио на эллиптическом тренажере - 30 минут\n"
    "2. Отжимания - 3 подхода по 15 повторений\n"
    "3. Приседания с собственным весом - 3 подхода по 20
повторений\n"
    "4. Планка - 3 подхода по 60 секунд\n"
    "5. Бёрпи - 3 подхода по 12 повторений\n"
    "6. Прыжки через скакалку - 3 подхода по 2 минуты\n"
    "7. Йога-поза 'собака мордой вниз' - 3 подхода по 30
секунд\n"
    "8. Растяжка - 5-10 минут"
  )
},
# Мезоморф
{
  "body_type": "mesomorph",
  "goal": "loss",
  "program_text": (
    "Программа похудения для мезоморфа:\n\n"
    "Упражнения:\n"

```

```

        "1. Высокoинтенсивный интервальный тренинг (HIIT) -
20-30 минут\n"
        "2. Приседания с собственным весом - 4 подхода по 15
повторений\n"
        "3. Отжимания - 4 подхода по 15 повторений\n"
        "4. Скакалка - 3 подхода по 2 минуты\n"
        "5. Планка - 3 подхода по 60 секунд\n"
        "6. Бег с ускорениями - 10 интервалов по 30
секунд\n"
        "7. Бёрпи - 3 подхода по 10 повторений\n"
        "8. Скручивания для пресса - 3 подхода по 20
повторений"
    )
},
{
    "body_type": "mesomorph",
    "goal": "gain",
    "program_text": (
        "Программа набора массы для мезоморфа:\n\n"
        "Упражнения:\n"
        "1. Жим лежа - 4 подхода по 8 повторений\n"
        "2. Приседания со штангой - 4 подхода по 10
повторений\n"
        "3. Становая тяга - 4 подхода по 8 повторений\n"
        "4. Подтягивания - 4 подхода до отказа\n"
        "5. Жим гантелей на скамье - 4 подхода по 8-10
повторений\n"
        "6. Разведения гантелей лежа - 3 подхода по 12
повторений\n"
        "7. Тяга в наклоне - 3 подхода по 10 повторений\n"
        "8. Фермерская прогулка - 3 подхода по 1 минуте"
    )
},
{
    "body_type": "mesomorph",
    "goal": "recomp",
    "program_text": (
        "Программа рекомпозиции для мезоморфа:\n\n"
        "Упражнения:\n"
        "1. Жим штанги лежа - 4 подхода по 10 повторений\n"
        "2. Тяга верхнего блока - 4 подхода по 12
повторений\n"
        "3. Выпады с гантелями - 3 подхода по 12 повторений
на каждую ногу\n"
        "4. Планка - 3 подхода по 60 секунд\n"
        "5. Бёрпи - 3 подхода по 10 повторений\n"
        "6. Махи ногами - 3 подхода по 15 повторений\n"

```

```

        "7. Скручивания корпуса - 3 подхода по 20
повторений"
    )
    },
    {
        "body_type": "mesomorph",
        "goal": "strength",
        "program_text": (
            "Программа для увеличения силовых показателей для
мезоморфа:\n\n"
            "Упражнения:\n"
            "1. Жим штанги лежа - 5 подходов по 5 повторений\n"
            "2. Приседания со штангой - 5 подходов по 5
повторений\n"
            "3. Становая тяга - 5 подходов по 3-5 повторений\n"
            "4. Армейский жим - 4 подхода по 6 повторений\n"
            "5. Тяга в наклоне - 4 подхода по 8 повторений\n"
            "6. Подтягивания - 4 подхода до отказа\n"
            "7. Отжимания на брусьях - 3 подхода до отказа\n"
            "8. Разрывные прыжки - 3 подхода по 10 повторений"
        )
    },
    {
        "body_type": "mesomorph",
        "goal": "tonus",
        "program_text": (
            "Программа для поддержания тонуса для
мезоморфа:\n\n"
            "Упражнения:\n"
            "1. Легкое кардио (бег или эллиптический тренажёр) -
30 минут\n"
            "2. Отжимания - 3 подхода по 15 повторений\n"
            "3. Приседания с собственным весом - 3 подхода по 20
повторений\n"
            "4. Планка - 3 подхода по 60 секунд\n"
            "5. Растяжка - 5-10 минут\n"
            "6. Упражнения на трицепс (отжимания узким хватом) -
3 подхода по 12 повторений\n"
            "7. Йога-поза 'собака мордой вниз' - 3 подхода по 30
секунд\n"
            "8. Мягкий бег с переменной скоростью - 10 минут"
        )
    },
    # Эндоморф
    {
        "body_type": "endomorph",
        "goal": "loss",

```

```

    "program_text": (
        "Программа похудения для эндоморфа:\n\n"
        "Упражнения:\n"
        "1. Высокоинтенсивное кардио (бег или велотренажёр)
- 30-40 минут\n"
        "2. Приседания с собственным весом - 4 подхода по 15
повторений\n"
        "3. Отжимания - 4 подхода по 12 повторений\n"
        "4. Упражнения на пресс (скручивания) - 4 подхода по
20 повторений\n"
        "5. Планка - 3 подхода по 60 секунд\n"
        "6. Скакалка - 3 подхода по 2 минуты\n"
        "7. Интервальный бег - 10 интервалов по 30 секунд\n"
        "8. Бёрпи - 3 подхода по 12 повторений"
    )
},
{
    "body_type": "endomorph",
    "goal": "gain",
    "program_text": (
        "Программа набора массы для эндоморфа:\n\n"
        "Упражнения:\n"
        "1. Силовой тренинг с умеренными весами (жим лежа,
приседания, становая тяга) - 4 подхода по 8 повторений\n"
        "2. Гребной тренажёр - 3 подхода по 10 повторений\n"
        "3. Жим сидя - 3 подхода по 10 повторений\n"
        "4. Отжимания на брусьях - 3 подхода до отказа\n"
        "5. Подъёмы гантелей на бицепс - 3 подхода по 10
повторений\n"
        "6. Тяга верхнего блока - 3 подхода по 10
повторений\n"
        "7. Приседания с гантелями - 3 подхода по 12
повторений\n"
        "8. Выпады - 3 подхода по 12 повторений на каждую
ногу"
    )
},
{
    "body_type": "endomorph",
    "goal": "recomp",
    "program_text": (
        "Программа рекомпозиции для эндоморфа:\n\n"
        "Упражнения:\n"
        "1. Жим штанги лежа - 4 подхода по 10 повторений\n"
        "2. Приседания - 4 подхода по 12 повторений\n"
        "3. Тяга верхнего блока - 4 подхода по 12
повторений\n"

```

```

        "4. Планка - 3 подхода по 60 секунд\n"
        "5. Отжимания - 3 подхода по 15 повторений\n"
        "6. Упражнения на пресс (скручивания) - 3 подхода по
20 повторений\n"
        "7. Скакалка - 3 подхода по 2 минуты"
    )
},
{
    "body_type": "endomorph",
    "goal": "strength",
    "program_text": (
        "Программа для увеличения силовых показателей для
эндоморфа:\n\n"
        "Упражнения:\n"
        "1. Жим штанги лежа - 5 подходов по 5 повторений\n"
        "2. Приседания со штангой - 5 подходов по 5
повторений\n"
        "3. Становая тяга - 5 подходов по 3-5 повторений\n"
        "4. Жим ногами - 4 подхода по 8 повторений\n"
        "5. Подтягивания - 4 подхода до отказа\n"
        "6. Отжимания на брусьях - 3 подхода до отказа\n"
        "7. Тяга в наклоне - 3 подхода по 8 повторений\n"
        "8. Плиометрические прыжки - 3 подхода по 10
повторений"
    )
},
{
    "body_type": "endomorph",
    "goal": "tonus",
    "program_text": (
        "Программа для поддержания тонуса для
эндоморфа:\n\n"
        "Упражнения:\n"
        "1. Легкое кардио (бег или эллиптический тренажёр) -
30 минут\n"
        "2. Отжимания - 3 подхода по 15 повторений\n"
        "3. Приседания с собственным весом - 3 подхода по 20
повторений\n"
        "4. Планка - 3 подхода по 60 секунд\n"
        "5. Легкая растяжка - 5-10 минут\n"
        "6. Йога-поза 'собака мордой вниз' - 3 подхода по 30
секунд\n"
        "7. Упражнения на гибкость (наклоны, скручивания) -
3 подхода по 15 повторений\n"
        "8. Ходьба в быстром темпе - 20 минут"
    )
}

```

```

]

# Загрузка данных в базу
try:
    for prog in programs_data:
        program = Program(
            body_type=prog["body_type"],
            goal=prog["goal"],
            program_text=prog["program_text"]
        )
        session.add(program)
    session.commit()
    print("Данные успешно добавлены в базу!")
except SQLAlchemyError as e:
    session.rollback()
    print("Ошибка при работе с базой:", e)
finally:
    session.close()

```

Файл exercise_images.py:

```

# Словарь локальных изображений для упражнений.
# Путь к папке с изображениями на сервере: /opt/coachbot/images

LOCAL_EXERCISE_IMAGE_MAP = {
    "жим штанги лежа": "/opt/coachbot/images/bench_press.jpg",
    "тяга верхнего блока": "/opt/coachbot/images/lat_pull.jpg",
    "выпады с гантелями":
"/opt/coachbot/images/lunges_dumbbells.jpg",
    "планка": "/opt/coachbot/images/plank.jpg",
    "бёрпи": "/opt/coachbot/images/burpee.jpg",
    "махи ногами": "/opt/coachbot/images/leg_swings.jpg",
    "скручивания корпуса": "/opt/coachbot/images/crunches.jpg",
    "приседания со штангой":
"/opt/coachbot/images/squats_barbell.jpg",
    "становая тяга": "/opt/coachbot/images/deadlift.jpg",
    "армейский жим": "/opt/coachbot/images/military_press.jpg",
    "тяга в наклоне": "/opt/coachbot/images/bent_over_row.jpg",
    "подтягивания": "/opt/coachbot/images/pull_ups.jpg",
    "отжимания на брусьях": "/opt/coachbot/images/dips.jpg",
    "разрывные прыжки":
"/opt/coachbot/images/plyometric_jumps.jpg",
    "легкое кардио (бег или эллиптический тренажёр)":
"/opt/coachbot/images/light_cardio.jpg",
    "отжимания": "/opt/coachbot/images/pushups.jpg",
    "приседания с собственным весом":
"/opt/coachbot/images/bodyweight_squats.jpg",

```

"растяжка": "/opt/coachbot/images/stretch.jpg",
"упражнения на трицепс (отжимания узким хватом)":
"/opt/coachbot/images/triceps_pushups.jpg",
"йога-поза 'собака мордой вниз'":
"/opt/coachbot/images/downward_dog.jpg",
"мягкий бег с переменной скоростью":
"/opt/coachbot/images/jog_variant.jpg",
"высокоинтенсивное кардио (бег или велотренажёр)":
"/opt/coachbot/images/hiit_cardio.jpg",
"упражнения на пресс (скручивания)":
"/opt/coachbot/images/crunches_abs.jpg",
"скакалка": "/opt/coachbot/images/skipping_rope.jpg",
"интервальный бег":
"/opt/coachbot/images/interval_running.jpg",
"силовой тренинг с умеренными весами (жим лежа, приседания, становая тяга)":
"/opt/coachbot/images/moderate_strength_training.jpg",
"гребной тренажёр":
"/opt/coachbot/images/rowing_machine.jpg",
"жим сидя": "/opt/coachbot/images/seated_press.jpg",
"подъёмы гантелей на бицепс":
"/opt/coachbot/images/bicep_curls.jpg",
"приседания с гантелями":
"/opt/coachbot/images/dumbbell_squats.jpg",
"выпады": "/opt/coachbot/images/lunges.jpg",
"жим ногами": "/opt/coachbot/images/leg_press.jpg",
"приседания": "/opt/coachbot/images/squats.jpg",
"легкая растяжка": "/opt/coachbot/images/light_stretch.jpg",
"упражнения на гибкость (наклоны, скручивания)":
"/opt/coachbot/images/flexibility_ex.jpg",
"ходьба в быстром темпе":
"/opt/coachbot/images/brisk_walking.jpg",
"плиометрические прыжки":
"/opt/coachbot/images/plyometric_jumps.jpg",

Дополнительно добавленные упражнения:
"легкий бег или кардио":
"/opt/coachbot/images/light_run_or_cardio.jpg",
"выпады с собственным весом":
"/opt/coachbot/images/bodyweight_lunges.jpg",
"скручивания для пресса":
"/opt/coachbot/images/abs_crunches.jpg",
"жим лежа": "/opt/coachbot/images/bench_press_simple.jpg",
"разведения гантелей лежа":
"/opt/coachbot/images/dumbbell_fly.jpg",
"фермерская прогулка":
"/opt/coachbot/images/farmer_walk.jpg",

```

        "жим гантелей": "/opt/coachbot/images/dumbbell_press.jpg",
        "кардио на эллиптическом тренажере":
"/opt/coachbot/images/elliptical_cardio.jpg",
        "прыжки через скакалку":
"/opt/coachbot/images/jump_rope_jumps.jpg",
        "высокоинтенсивный интервальный тренинг (hiit)":
"/opt/coachbot/images/hiit_training.jpg",
        "бег с ускорениями":
"/opt/coachbot/images/sprint_intervals.jpg",
        "жим гантелей на скамье":
"/opt/coachbot/images/bench_dumbbell_press.jpg"
    }

```

Файл constants.py:

```

# Состояния для разговоров с ботом
TRAINING_HEIGHT = 0
TRAINING_WEIGHT = 1
TRAINING_GOAL = 2
EDIT_PROGRAM = 3
ADD_EXERCISE = 4

# Токен бота будет загружаться из зашифрованного файла
BOT_TOKEN = None

```

Файл generate_key.py:

```

# generate_key.py
from cryptography.fernet import Fernet

def generate_key():
    key = Fernet.generate_key() # генерируем симметричный ключ
    with open("key.key", "wb") as key_file:
        key_file.write(key)
    print("Ключ сгенерирован и сохранён в key.key")

if __name__ == "__main__":
    generate_key()

```

Файл encrypt_config.py:

```

from cryptography.fernet import Fernet

def encrypt_file(input_file: str, output_file: str, key_file:
str):
    # Читаем ключ

```

```

with open(key_file, "rb") as kf:
    key = kf.read()
fernet = Fernet(key)

# Читаем исходный конфигурационный файл
with open(input_file, "rb") as file:
    original = file.read()

# Шифруем данные
encrypted = fernet.encrypt(original)

# Сохраняем зашифрованные данные
with open(output_file, "wb") as enc_file:
    enc_file.write(encrypted)

print(f"Файл {input_file} успешно зашифрован в
{output_file}")

if __name__ == "__main__":
    encrypt_file("config.json", "config.enc", "key.key")

```

Файл config_handlers.py:

```

import json
from cryptography.fernet import Fernet

def load_config(config_enc_path: str = "config.enc",
key_file_path: str = "key.key") -> dict:
    # Читаем ключ из файла
    with open(key_file_path, "rb") as kf:
        key = kf.read()
    fernet = Fernet(key)

    # Читаем зашифрованный файл конфигурации
    with open(config_enc_path, "rb") as enc_file:
        encrypted = enc_file.read()

    # Расшифровываем данные
    decrypted = fernet.decrypt(encrypted)
    config = json.loads(decrypted)
    return config

if __name__ == "__main__":
    config = load_config()
    print("Загруженная конфигурация:", config)

```

Приложение В

(Обязательное)

Telegramм-бот «Индивидуальный фитнес-тренер»

Презентация

Telegramм-бот "Индивидуальный фитнес-тренер"

- Автор работы:
студент гр.21ВВП1 Сагатов А.К.
- Научный руководитель:
д.т.н., профессор Зинкин С.А.

Актуальность темы

Telegram-бот — это удобный инструмент, которым ежедневно пользуется большое количество людей

Формат

Telegram-бот удобен в использовании

Развитие

Растущая аудитория пользователей телеграм

Популярность

Миллионы людей интегрируют спорт в свой образ жизни

Выбранные технологии

VS Code

Среда разработки

Python

Язык программирования

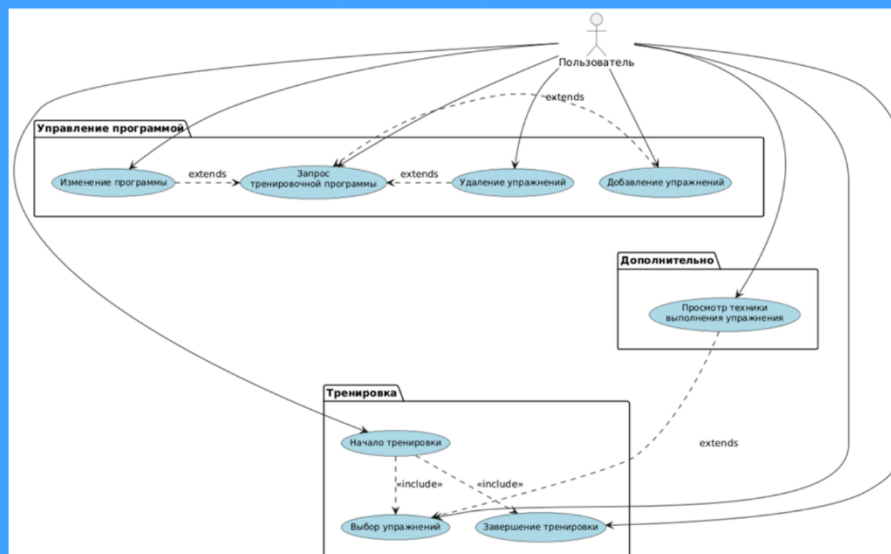
SQLite

СУБД

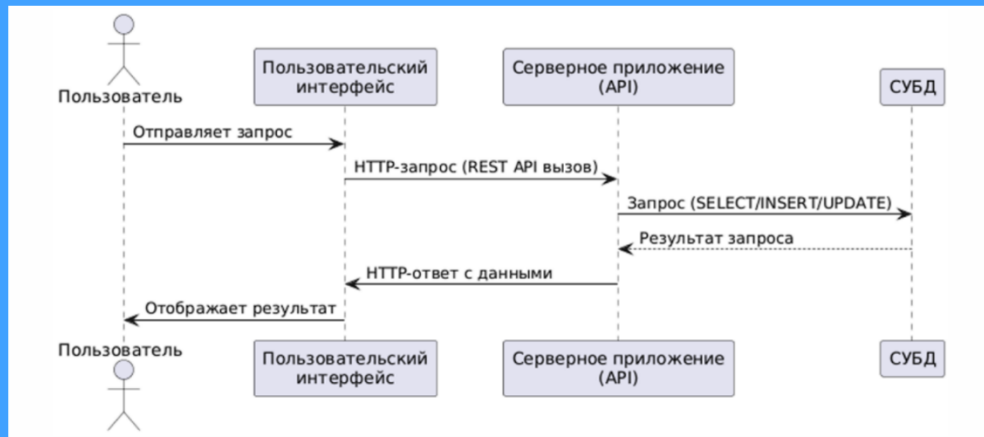
TimeWeb.cloud

Виртуальный сервер

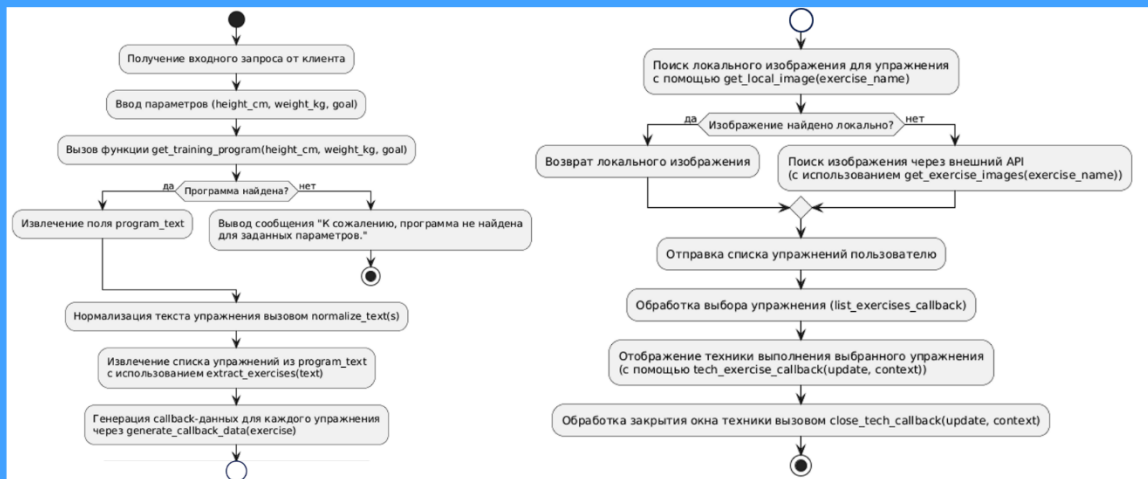
UML-диаграмма прецедентов



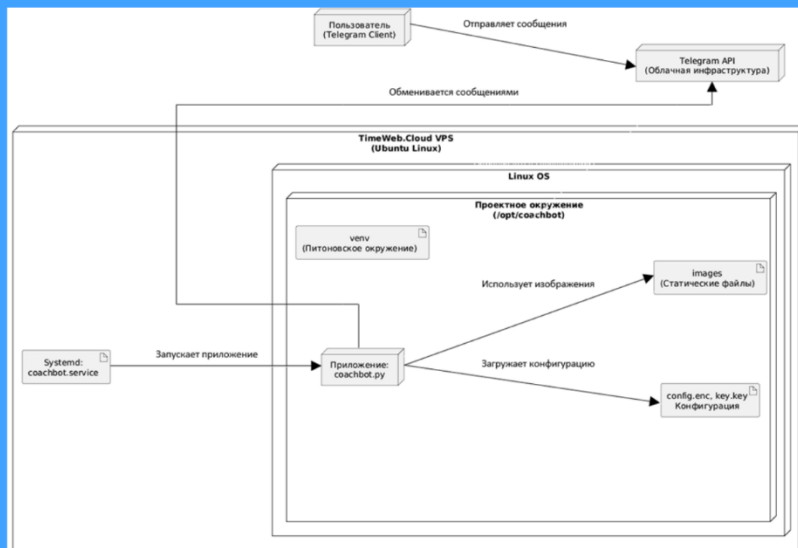
UML-диаграмма последовательности



UML-диаграмма деятельности



UML-диаграмма развертывания



Спасибо за
внимание!