МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
**«Южно-Уральский государственный университет
(национальный исследовательский университет)»**
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой, д.ф.-м.н.,
профессор

_____ Л.Б. Соколинский

«___»_____ 2024 г.

Разработка телеграмм-бота для бронирования сеанса игры в компьютерном клубе

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 02.03.02.2024.308-549.ВКР

Научный руководитель,
доцент кафедры СП, к.ф.-м.н.
_____ А.Т. Латипова

Автор работы,
студент группы КЭ-402
_____ А.С. Папулов

Ученый секретарь
(нормоконтролер)
_____ И.Д. Володченко
«___»_____ 2024 г.

Челябинск, 2024 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

УТВЕРЖДАЮ
Зав. кафедрой СП
_____ Л.Б. Соколинский
29.01.2024 г.

ЗАДАНИЕ
на выполнение выпускной квалификационной работы бакалавра
студенту группы КЭ-402
Папулову Андрею Сергеевичу,
обучающемуся по направлению
02.03.02 «Фундаментальная информатика и информационные технологии»

1. **Тема работы** (утверждена приказом ректора от 22.03.2024 г. № 764-13/12)
Разработка телеграмм-бота для бронирования сеанса игры в компьютерном клубе.

2. Срок сдачи студентом законченной работы: 03.06.2024 г.

3. Исходные данные к работе

3.1. Седжвик Р., Уэйн К., Дондеро Р. Программирование на языке Python. // Издательство СПб. Диалектика 2019. – 736 с.

3.2. Мокеев В.В. Web-аналитика на Python. // Издательство Челябинск Издательский Центр ЮУрГУ 2020. – 143 с.

3.3. Лутц М.; Пер. Маккавеева С. Программирование на Python. // Издательство СПб., Символ-Плюс 2002. – 1135 с.

3.4. Рамальо Л.К вершинам мастерства Python. / пер. с англ. Слинкин А.А. – Москва: ДМК-Пресс, 2016. – 767 с.

4. Перечень подлежащих разработке вопросов

4.1. Подготовить материалы по программированию на Python.

4.2. Подготовить материалы по разработке телеграмм-бота.

4.3. Зарегистрировать ID телеграмм-бота**.**

4.4. Разработка телеграмм-бота.

4.5. Тестирование телеграмм-бота.

5. Дата выдачи задания: 29.01.2024 г.

Научный руководитель,
доцент кафедры СП, к.ф.-м.н. А.Т. Латипова

Задание принял к исполнению А.С. Папулов

ГЛОССАРИЙ

1. *Мессенджер* – это приложение или сервис, предназначенный для обмена сообщениями между пользователями через интернет, могут включать в себя текстовые сообщения, аудио- и видеозвонки, передачу файлов, стикеры и другие функции [1].

2. *Автоматизация* – позволяет автоматизировать повторяющиеся или рутинные задачи, упрощая их выполнение и снижая вероятность ошибок [2].

3. *Интеграция* – объединение или соединение различных компонентов, систем или процессов для обеспечения их взаимодействия и совместной работы, может включать в себя различные аспекты, такие как техническое соединение между системами (например, через API или протоколы), согласование данных и форматов, автоматизацию бизнес-процессов и другие [3].

4. *Сервис* – способ предоставления какой-либо услуги или функциональности пользователю [4].

5. *Инновации* – процесс создания или внедрения новых идей, продуктов, услуг или методов, которые привносят значительное изменение или улучшение по сравнению с существующими [5].

6. *Бронирование* – процесс резервирования определенного продукта или услуги на определенное время в будущем [6].

7. *Приватность* – право и возможность человека контролировать доступ к своей личной информации [7].

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность

В наше время телеграмм-боты становятся все более популярными и актуальными. Они представляют собой программы, которые работают в мессенджере [1] Telegram и выполняют различные задачи с автоматизацией [2] задач, удобством использования, интеграцией [3] с другими сервисами [4], развитием и инновациями [5], а также потенциалом для бизнеса и монетизации.

Постановка задачи

Целью выпускной квалификационной работы является разработка телеграмм-бота для бронирования [6] сеанса игры в компьютерном клубе с различными дополнительными функциями, например, авторизация, профиль пользователя, просмотр ПК для сеансов, а также просмотр расписания записавшихся участников. Для достижения поставленной цели необходимо решить следующие задачи:

1) исследовать материалы по телеграмм-ботам;
2) подготовить материалы по разработке телеграмм-ботов;
3) зарегистрировать ID телеграмм-бота;
4) разработать телеграмм-бота;
5) протестировать телеграмм-бота.

Структура и содержание работы

Работа состоит из введения, четырех глав, выводов, заключения и списка литературы. Объем работы составляет 55 страниц, объем списка литературы – 15 источников.

В первой главе описывается процесс определения оптимального набора инструментов для выполнения поставленной задачи, который наиболее полно соответствует установленным требованиям. Этот этап исследования включает в себя анализ различных инструментальных средств и их оценку с точки зрения их соответствия поставленным задачам и критериям эффективности [10, с.172]. В результате выбирается наиболее под-

ходящий набор инструментов, который будет использоваться для дальнейшей реализации проекта.

Вторая глава посвящена тому, как были определены основные функциональные и нефункциональные требования, которые должны соответствовать телеграмм-боту. Также проведен обзор выбранного языка программирования, который будет использоваться в процессе разработки. В рамках этой главы разработан эскизный проект, в котором подробно описаны основные компоненты и функциональные возможности будущего бота . Это позволяет обеспечить ясное понимание того, каким образом будет организована и реализована функциональность бота с учетом установленных требований и выбранных технологий.

В третьей главе в результате проделанной работы была сформирована архитектура телеграмм-бот. В рамках этой главы создается диаграмма компонентов, которая наглядно демонстрирует взаимосвязь между ключевыми элементами системы. Подробно описывается реализация основных классов и методов, применяемых в разработанных компонентах. Для наглядности и анализа представлены скриншоты разработанных функций, позволяющие визуально оценить их внешний вид и интерфейс.

В четвертой главе в результате обеспечения надежной работы телеграмм-бота был проведен обширный процесс тестирования, включающий в себя различные аспекты функциональности и производительности.

В приложении А содержится спецификация вариантов использования для разработанных функций.

В приложении Б содержится код реализации для разработанной функции.

В приложении В содержатся скриншоты разработанных функций из третьей главы.

В приложении Г содержится диаграмма последовательности для некоторых разработанных функций.

1. ОБЗОР АНАЛОГИЧНЫХ ПРОЕКТОВ

1.1. Предметная область и сравнительный анализ аналогов

Телеграмм-боты – это программы, которые работают в мессенджере Telegram и выполняют различные задачи. Они используют Telegram API и Bot API для взаимодействия с пользователями и предоставления различных функций. Для создания телеграмм-ботов разработчики могут использовать различные инструменты и платформы, включая BotFather – официального бота Telegram для создания и настройки ботов.

Тема телеграмм-ботов актуальна и интересна, так как она связана с автоматизацией задач, удобством использования, интеграцией с другими сервисами, развитием и инновациями, а также потенциалом для бизнеса и монетизации. Существуют разные телеграмм-боты с разными назначениями, но все боты начинаются с получения ID телеграмм-бота, который получается в определенном боте «BotFather».

BotFather – официальный бот Telegram для создания и настройки других ботов, он позволяет пользователям создавать свои собственные телеграмм-боты и настраивать их функциональность.

Некоторые основные функции и возможности BotFather представлены ниже.

1. Создание нового бота. Пользователи могут использовать BotFather для создания нового телеграмм-бота. Для этого нужно отправить команду «/newbot» и следовать инструкциям.

2. Настройка имени и описания бота. После создания бота, BotFather позволяет настроить его имя и описание. Эти данные будут отображаться в информации о боте.

3. Получение API-токена. BotFather предоставляет API-токен, который необходим для взаимодействия с Telegram API. API-токен используется для отправки и получения сообщений, управления настройками бота и других операций.

4. Установка команд бота. BotFather позволяет настроить команды, которые будут доступны для пользователя. Это позволяет определить, какие действия может выполнять бот.

5. Управление настройками бота. BotFather предоставляет возможность изменять настройки бота, такие как язык, фотографию профиля, приватность [7] и другие параметры.

Существуют боты для интеграции с другими сервисами, такими как Gmail, Trello и Spotify.

Например, есть боты, которые позволяют получать уведомления о новых письмах в Gmail, управлять задачами в Trello или слушать музыку на Spotify.

Несколько примеров таких ботов представлены ниже.

1. Botmother предоставляет интеграцию с различными сервисами, такими как CMS, QR-коды, платежные системы (например, Robokassa, Kassa), а также интеграцию с Zapier, Albato, ApiX-Drive и Google CRM.

2. Salebot предоставляет интеграцию с различными сервисами для автоматизации продаж и управления клиентскими отношениями.

3. IntellectDialog предлагает интеграцию с Telegram и другими платформами для обработки и анализа сообщений.

4. Flow XO позволяет интегрировать ботов с различными сервисами, такими как Trello, Gmail, Zoho CRM, Basecamp 2 и MailChimp. Он предоставляет возможности автоматизации и управления задачами.

5. Botkit предоставляет инструменты для создания ботов с интеграцией в различные платформы и сервисы. Он поддерживает интеграцию с Microsoft Azure и другими платформами.

Телеграмм-боты также используются в бизнесе, некоторые проекты предлагают ботов для автоматизации задач, управления заказами или предоставления информации о товарах и услугах.

Некоторые особенности и примеры ботов для бизнеса представлены ниже.

1. Botmother предоставляет интеграцию с различными сервисами, такими как CMS, платежные системы (например, Robokassa), а также интеграцию с Zapier, Albato, ApiX-Drive и Google CRM. Он предлагает возможности автоматизации и управления задачами в бизнесе.

2. Salebot предоставляет интеграцию с различными сервисами для автоматизации продаж и управления клиентскими отношениями. Он может помочь в управлении заказами, отслеживании статуса доставки и предоставлении информации о товарах или услугах.

3. Botkit – это инструмент для создания ботов с интеграцией в различные платформы и сервисы. Он поддерживает интеграцию с Microsoft Azure и другими платформами, что позволяет использовать его для разработки ботов для бизнеса.

4. Botlify – это платформа для создания ботов, которая предлагает интеграцию с различными сервисами. Он может быть использован для автоматизации продаж, обработки заказов и предоставления поддержки клиентам.

5. IntellectDialog предоставляет интеграцию с Telegram и другими платформами для обработки и анализа сообщений. Он может быть использован для автоматической обработки запросов клиентов, предоставления информации о продуктах или услугах и других бизнес-процессах [7, с.90].

Вывод по первой главе

В работе рассмотрены такие инструменты, как BotFather, Botmother, Salebot, IntellectDialog, Flow XO, Botkit и Botlify. Каждый из них обладает уникальными особенностями и функциональностью, предлагая интеграцию с разнообразными сервисами для решения различных бизнес-задач. В результате обзора современных телеграмм-ботов был внесен важный вклад в понимание возможностей и применения телеграмм-ботов в современном бизнесе, а также обзор помогает выбрать наиболее подходящий инструмент для конкретных задач и потребностей для разработки.

2. ПРОЕКТИРОВАНИЕ

2.1 Функциональные и нефункциональные требования

Функциональные требования – это перечень требований, которые должна выполнять система, без учета ограничений, связанных с ее реализацией, другими словами требования описывающие поведение системы обрабатывая входные и выходные данные. Можно выделить следующий набор функциональных требований к системе:

1) в боте можно записываться в компьютерный клуб;
2) в боте можно просматривать комплектующие ПК;
3) в боте можно узнать контактную информацию клуба;
4) в боте присутствует профиль пользователя со статистикой использования телеграмм-бота;
5) в боте присутствует чат с администратором;
6) в боте есть функция вывода ключа аутентификации;
7) в боте можно просматривать расписание пользователей записанных на сеанс;
8) в боте присутствуют функции для администратора.

Нефункциональные требования – перечень требований, определяющих качественные характеристики разрабатываемого приложения. Можно выделить следующее функциональное требование – система должна быть написана с помощью Python на ОС Windows.

2.2. Обзор средств для реализации проекта

Преимущества среды программирования Python

Одним из преимуществ является простота и читаемость кода. Python имеет простой и понятный синтаксис, который делает код легким для чтения и написания. Это позволяет разработчикам быстро создавать и поддерживать программы.

Следующим преимуществом будет объектно-ориентированное программирование (ООП). Python – это объектно-ориентированный язык, ко-

торый поддерживает модульный и повторно используемый код. ООП позволяет улучшить организацию кода, упростить обслуживание и повысить возможность повторного использования кода [3, с.127].

Другим преимуществом является большое количество библиотек и фреймворков. Python имеет обширную экосистему библиотек и фреймворков, которые облегчают разработку приложений. Например, библиотека NumPy предоставляет мощные инструменты для работы с числовыми данными, а фреймворк Django упрощает создание веб-приложений.

Последним преимуществом является кросс-платформенность. Python поддерживает различные операционные системы, включая Windows, macOS и Linux. Это позволяет разработчикам создавать приложения, которые могут работать на разных платформах без необходимости переписывать код.

Недостатки среды программирования Python

Одним из недостатков является относительно медленная производительность. Python является интерпретируемым языком программирования, что может приводить к некоторому снижению производительности по сравнению с компилируемыми языками, такими как C++.

Другим недостатком является ограничения GIL. Global Interpreter Lock в Python ограничивает выполнение кода только одним потоком, что может приводить к проблемам с параллельным выполнением и многопоточностью в некоторых случаях.

Последним недостатком является узкая поддержка для мобильной разработки. Python не является первоочередным выбором для разработки мобильных приложений, хотя существуют фреймворки, такие как Kivy и BeeWare, которые позволяют создавать мобильные приложения на Python.

2.3. Диаграмма вариантов использования

Для создания диаграммы вариантов использования был выбран язык графического описания UML. В соответствии с вышеперечисленными тре-

бованиями была составлена UML диаграмма с будущими вариантами взаимодействия в разрабатываемом телеграмм-боте. Одна из диаграмм представлена ниже, которая отражает взаимодействие актера «Пользователь» с ПО.

На рисунке 1 представлена диаграмма вариантов использования для актера «Пользователь».

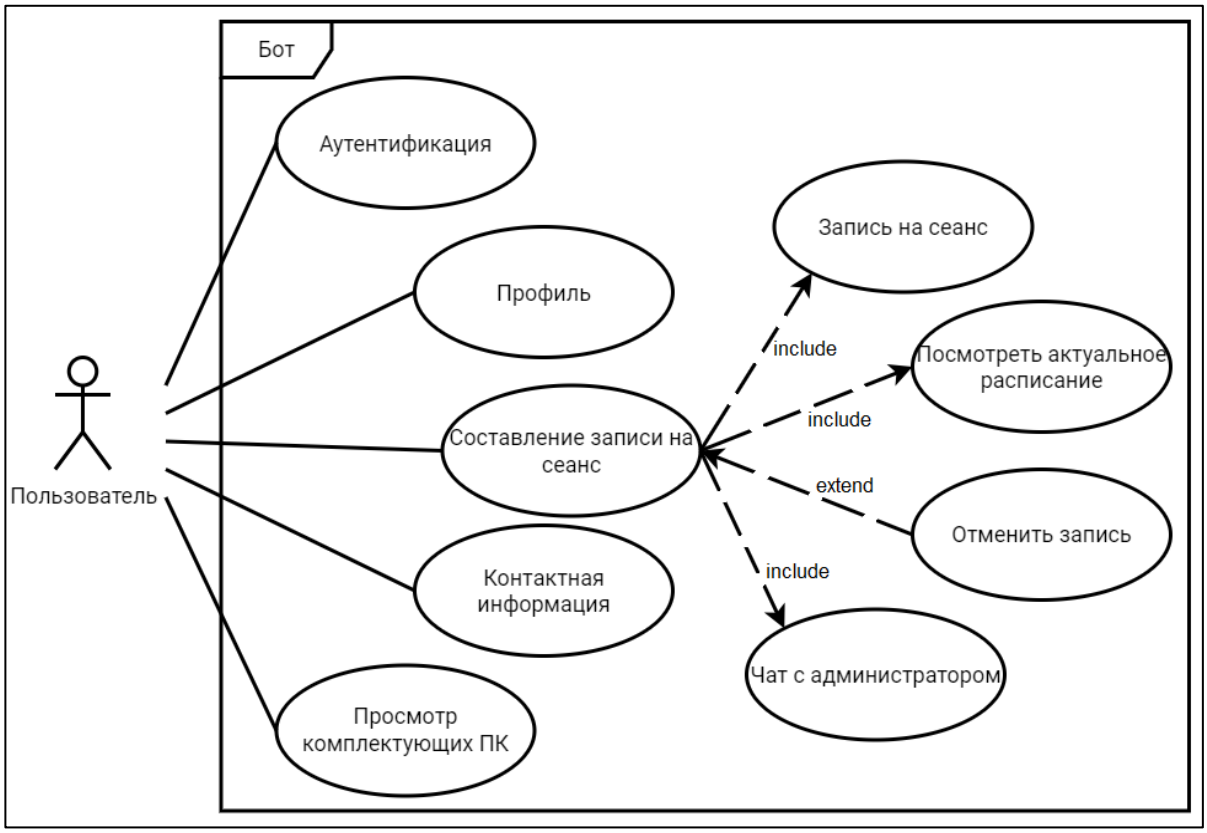

Рисунок 1 – Диаграмма вариантов использования для актера «Пользователь»

Данная архитектура состоит из следующих компонентов:

1) аутентификация;
2) просмотр комплектующих ПК;
3) профиль;
4) просмотр контактной информации клуба;
5) составление записи на сеанс;
6) просмотреть актуальное расписание;
7) запись на сеанс;

8) чат с администратором;

9) отменить запись.

Запустив бота, пользователь начинает взаимодействовать с ботом. В ходе пользования пользователь может просмотреть информацию о клубе, посмотреть комплектующие ПК, узнать свой код аутентификации, просмотреть профиль, составить запись на сеанс. При нажатии кнопки составления записи на сеанс бот предложит выбор функций: запись на сеанс, просмотреть актуальное расписание, отменить запись и чат с администратором. Бот всегда будет открыт при его использовании, закрывать не требуется в связи того, что он находится в мессенджере

Ниже приведена вторая диаграмма, иллюстрирующая взаимодействие актера «Администратор» с ПО.

На рисунке 2 представлена диаграмма вариантов использования для актера «Администратор».

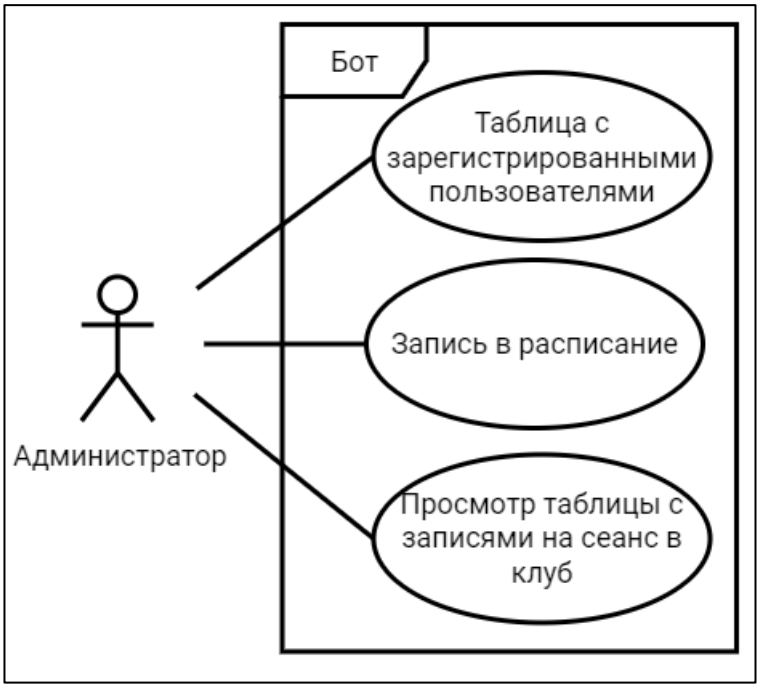

Рисунок 2 – Диаграмма вариантов использования для актера «Администратор»

Данная архитектура состоит из следующих компонентов:

1) таблица с зарегистрированными пользователями;

2) запись пользователей в расписание;

3) просмотр таблицы с записями на сеанс в клуб.

Запустив бота, пользователь, который был назначен администратором и имеющий расширенный доступ к системе, может взаимодействовать с телеграмм-ботом функциями, которые доступны только ему. В ходе использования администратор может просматривать таблицы со всеми зарегистрированными пользователями в боте, может записывать данные о пользователях в общедоступную таблицу с расписанием, а также может просматривать таблицу с записями на сеанс в клуб пользователей. Спецификации вариантов использования расположены в приложении А.

2.4. Эскизный проект разрабатываемой ПО

При создании эскизного проекта для будущего бота следует учитывать несколько ключевых аспектов. Во-первых, необходимо четко определить цель бота и целевую аудиторию, чтобы лучше понять, какие функции будут наиболее полезны для пользователей. Затем следует разработать функционал бота, разделив его на основные и дополнительные возможности, и определить, какие команды или запросы он будет поддерживать. Очень важно также продумать интерфейс бота, будь то текстовый, голосовой или другой вид интеракции.

В процессе создания эскизного проекта необходимо выбрать подходящие технологии для его реализации, определить архитектуру системы и обеспечить безопасность обработки данных пользователей. Также стоит уделить внимание вопросам тестирования и отладки, чтобы гарантировать качество бота перед выпуском в продакшн.

Общие сведения

Телеграмм-бот – это программа, которая автоматизирует задачи и предоставляет услуги через мессенджер Telegram. Он может выполнять различные функции, включая обработку команд, отправку сообщений, работу с базой данных, интеграцию с внешними сервисами и многое другое.

Телеграмм-бот представляет собой программное решение, способное автоматизировать процессы и оказывать услуги через мессенджер Telegram. Его функционал охватывает широкий спектр возможностей, включая обработку команд пользователей, рассылку сообщений, взаимодействие с базами данных, интеграцию с внешними сервисами и многие другие функции. Кроме того, телеграмм-боты могут быть настроены для выполнения специализированных задач, таких как управление расписанием, рассылка новостей, проведение опросов, обработка заказов и даже игровые функции. Их гибкость и масштабируемость делают их незаменимым инструментом для автоматизации и упрощения коммуникации и бизнес-процессов в различных сферах деятельности [9, с. 57].

Интерфейс

Главный экран Telegram состоит из двумерного изображения, на котором отображаются основные компоненты пользовательского интерфейса. Эти элементы включают в себя чат с ботом, кнопки выбора функций, возможность записи на сеанс, контактная информация клуба, просмотр комплектующих ПК, функция чата с администратором, код авторизации для подтверждения пользователя и профиль пользователя со статистикой. Позиция камеры фиксирована и не изменяется, она остается в центре экрана для обеспечения удобства взаимодействия с интерфейсом.

Начало использования

Пользование ботом будет начинаться с входа в окно общения с телеграмм-ботом, после появится текст и начнется общение.

Цель приложения

Целью является запись на сеанс в компьютерный клуб. Функции, предоставляемые в боте, упрощают процесс записи. Среди них выбор комплектующих ПК, определение подходящего стола с установленным ПК, гибкий выбор времени и даты записи, возможность общения с администратором через чат для любых вопросов о записи, а также авторизация для доступа к профилю пользователя и введение необходимой информа-

ции для точной записи на сеанс. После этого данные получает администратор компьютерного клуба в таблицу и записывает пользователя на назначенное время, запись пользователя отображается в расписании, где пользователь может увидеть всех других пользователей, которые записались, таблицу пользователь может увидеть до записи тоже, для того, чтобы более точно установить свое времяпрепровождение в клубе.

Проектирование поля

Поле будет доступно в мессенджере Telegram в виде чата, где пользователь сможет взаимодействовать с различными функциями. Телеграм-бот будет отправлять сообщения с перечислением доступных функций, из которых пользователь сможет выбирать. Некоторые функции будут иметь дополнительные возможности для записи информации.

Вывод по второй главе

После проведенного анализа были выявлены основные функциональные и нефункциональные требования к телеграмм-боту, а также был осуществлен обзор выбранного языка программирования, который будет использоваться в процессе разработки. Этот этап работы позволяет четко определить ожидаемый функционал бота и выбрать наиболее подходящие технологические средства для его реализации. Путем проведения анализа были установлены основные функциональные и нефункциональные требования, которым должен соответствовать телеграмм-бот. Это включает в себя не только возможности бота, но и его производительность, безопасность, масштабируемость и другие аспекты. Это позволяет определить, насколько эффективно выбранные функции соответствуют требованиям проекта и обеспечить успешную реализацию целей.

3. РЕАЛИЗАЦИЯ

3.1. Описание архитектуры приложения

Архитектура системы организована вокруг одной основной сцены - поле, где отображаются сообщения от телеграм-бота. При запуске бота загружается лишь одна сцена в мессенджере, которая представляет собой это поле. Основное меню включает в себя опции, предлагаемые телеграмм-ботом, каждая из которых предоставляет доступ к различным функциям и возможностям системы. Это обеспечивает удобный и интуитивно понятный интерфейс для пользователей, позволяя им легко взаимодействовать с ботом и использовать его функционал (таблица 1).

Таблица 1 – Объяснение функций

Объект	Действие
information	Данные о компьютерном клубе: кто они такие, где находится клуб, вывод контактных данных
komplektPK	Выводится сообщение с комплектующими ПК, которые находятся в клубе
recordinclub	Выводится сообщение с выбором кнопок «Запись на сеанс», «Посмотреть расписание» и «Отменить запись»
record	Запись на сеанс, пишется ФИО и номер телефона для связи
schedule	Просмотр уже составленных и записанных пользователей на определенное время
cancelentry	Отмена отправленной заявки на сеанс
adminchat	Вызов чата с администратором для различных вопросов, которые могу интересовать пользователя
authenticate	Вход в профиль пользователя
profile	Сводка различных данных о пользователя и его статистика о деятельности в клубе
users	Таблица со всеми пользователями, которые пользовались ботом, собраны ID, имена, код аутентификации
adminrecord	Запись в таблицу с расписанием для пользователей, имя, фамилия, номер стола для записи, время начала и конца записи, дата записи
recordusers	Таблица с данными пользователей, которые отправили введенные ими данные для записи в клуб

3.2. Диаграмма компонентов приложения

На рисунке 3 представлена диаграмма компонентов, описывающая архитектурную структуру разрабатываемого приложения. Эта диаграмма позволяет визуализировать различные составляющие приложения и их

взаимосвязи. Стрелки на диаграмме указывают на взаимодействия между компонентами, что помогает понять, как данные компоненты обмениваются информацией и взаимодействуют внутри системы.

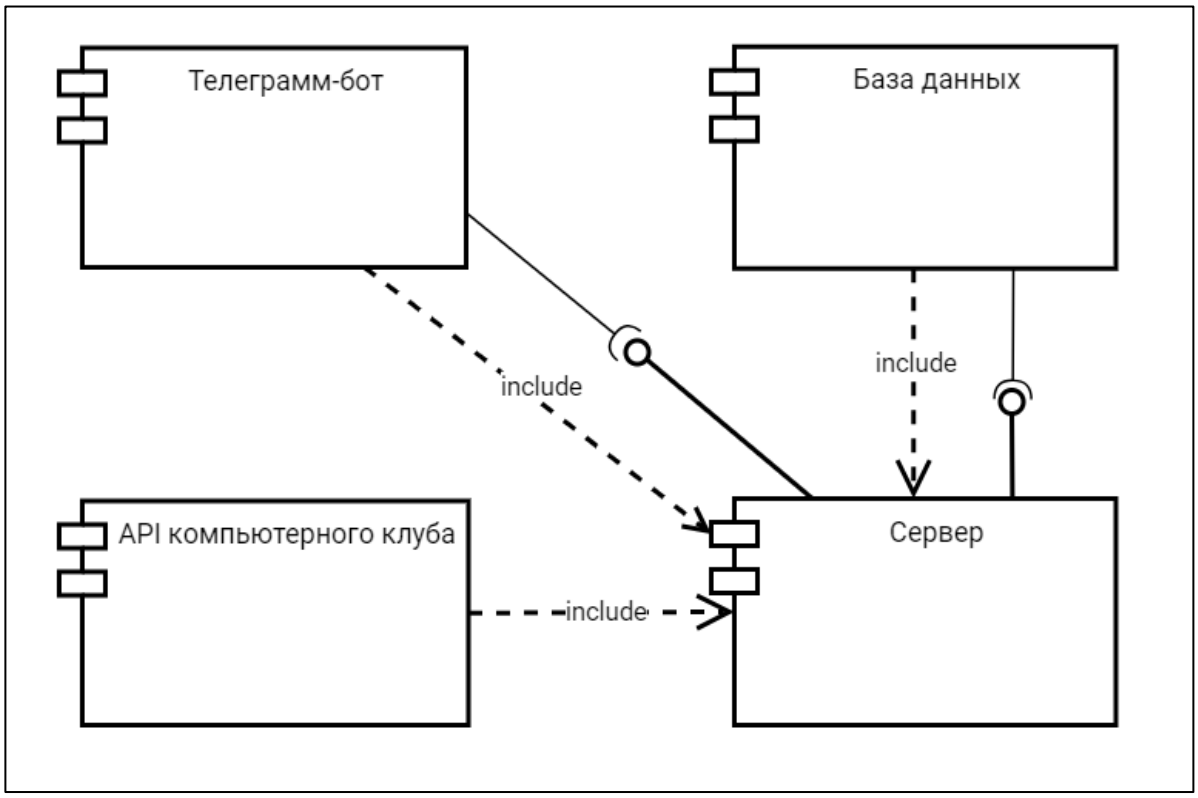

Рисунок 3 – Диаграмма компонентов

Диаграмма компонентов приложения выделяет ключевые модули и компоненты, которые определяют его функциональность. Каждый компонент включает в себя набор модулей, параметров и классов, которые определяют поведение объектов внутри системы. Это позволяет при разработке лучше понять структуру приложения и эффективно управлять его разработкой и сопровождением.

Для эффективной работы системы, связанной с управлением расписанием сеансов и записью на них, необходимо иметь четко структурированные компоненты.

1. Телеграмм-бот – это основной компонент системы, который обрабатывает запросы от пользователей и предоставляет им информацию о сеансах и возможность записаться на них.

2. База данных – это компонент, который хранит информацию о расписании сеансов, зарегистрированных пользователях и их записях на сеансы.

3. API компьютерного клуба – это компонент, который предоставляет доступ к функциональности компьютерного клуба, такой как получение расписания сеансов и регистрация пользователей на сеансы.

4. Сервер – это компонент, который выполняет важную функцию обеспечения взаимодействия между телеграмм-ботом и базой данных, а также между телеграмм-ботом и API компьютерного клуба. Он действует как посредник, принимая запросы от бота и передавая их соответствующим системам для обработки.

Сервер также играет ключевую роль в обеспечении безопасности системы, контролируя доступ к базе данных и API компьютерного клуба, а также обрабатывая аутентификацию пользователей и защищая систему от нежелательных запросов или атак. Он также отвечает за мониторинг и регистрацию журналов активности, что позволяет быстро выявлять и реагировать на любые проблемы или нештатные ситуации.

3.3. Средства реализации и реализация основных компонентов

Для разработки скриптов в приложении был выбран язык программирования Python. Редактирование и компиляция кода осуществлялись в индивидуальной среде разработки PyCharm Community Edition. Это позволило обеспечить удобство и эффективность работы над скриптами, а также обеспечить соответствие кода стандартам и требованиям проекта. Кроме того, использование PyCharm Community Edition обеспечило при разработке широкие возможности по отладке и тестированию созданных скриптов, что способствовало повышению качества и надежности программного обеспечения. В этом пункте будет рассмотрена реализация основных компонентов.

Это реализация телеграмм-бота на Python. В данном компоненте описаны классы, реализующие программу, свойства и функции, отвечающие за работоспособность телеграмм-бота.

Реализация соединения с базой данных пользователей

Функция `create_connection` возвращает соединение с базой данных SQLite, который можно использовать для выполнения различных операций, таких как выполнение SQL-запросов, добавление данных, извлечение информации и т. д. «`return sqlite3.connect('users.db')`»: Эта строка выполняет действие функции и возвращает результат. Метод `connect` используется для установления соединения с базой данных SQLite «`users.db`». Код метода представлен в листинге 1.

Листинг 1 – Функция `create_connection`

```
def create_connection():
    return sqlite3.connect('users.db')
```

Реализация базы данных users

Эта функция предназначена для создания таблицы `users` в базе данных SQLite, если она не существует. Она использует подключение к базе данных, полученное с помощью функции `create_connection`, которая вернет объект соединения. Затем, через этот объект соединения, создается курсор для выполнения операций с базой данных. Далее, с помощью метода `execute`, выполняется SQL-запрос, который создает таблицу `users` с определенными столбцами: `id`: Первичный ключ таблицы, автоматически инкрементируется; `username`: Столбец для хранения имени пользователя; `auth_token`: Столбец для хранения уникального токена аутентификации пользователя; `messages_sent`: Столбец для хранения количества отправленных сообщений (с значением по умолчанию 0); `commands_executed`: Столбец для хранения количества выполненных команд (с значением по умолчанию 0). После выполнения SQL-запроса, изменения сохраняются в

базе данных с помощью метода commit. Код метода представлен в листинге 2.

Листинг 2 – Функция create_table

```
def create_table():
    with create_connection() as conn:
        cursor = conn.cursor()
        cursor.execute('''
            CREATE TABLE IF NOT EXISTS users (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                username TEXT,
                auth_token TEXT UNIQUE,
                messages_sent INTEGER DEFAULT 0,
                commands_executed INTEGER DEFAULT 0
            )
        ''')
        conn.commit()
```

Реализация функции start

Функция start является обработчиком для инициации диалога с пользователем или запуска бота. Она начинает с получения данных о пользователе из объекта update. Затем функция проверяет наличие пользователя в базе данных, используя функцию create_connection для установления соединения с базой данных SQLite. Если пользователь уже существует, извлекаются его данные, включая auth_token, что может быть полезно для аутентификации или использования данных из профиля пользователя. В конце функции отправляется ответное сообщение пользователю, в данном случае, персонализированное приветственное сообщение, если пользователь уже есть в базе данных, а также выводятся доступные команды для пользователя. Эта функция позволяет боту эффективно взаимодействовать с пользователями и управлять их данными в базе данных. Код метода представлен в листинге 3. Скриншот расположен в приложении В на рисунке 1.

Листинг 3 – Функция start

```
def start(update: Update, context: CallbackContext) -> None:
    user_id = update.effective_user.id
    username = update.effective_user.username

    # Проверка, существует ли пользователь с таким ID в базе данных
    with create_connection() as conn:
        cursor = conn.cursor()
```

```
    cursor.execute("SELECT * FROM users WHERE id=?", (user_id,))
    existing_user = cursor.fetchone()

    if existing_user:
        # Если пользователь существует, используем его данные
        auth_token = existing_user[2]  # Индекс 2 - это поле auth_token
        message = update.message
        update.message.reply_text(f'Привет,
{message.from_user.first_name}!\n'
```

Реализация функции authenticate

Функция `authenticate` предназначена для проверки аутентификации пользователя. Она начинает с получения `id` пользователя из объекта `update`. Затем функция использует функцию `create_connection` для установления соединения с базой данных SQLite и создает курсор для выполнения операций. После этого выполняется SQL-запрос для извлечения `auth_token` пользователя из таблицы `users`, используя его `id`. Результат запроса сохраняется в переменную `result`. Если результат запроса не пустой (то есть, пользователь найден в базе данных), то извлекается `auth_token` из результата и отправляется сообщение пользователю с этим токеном для аутентификации. Код метода представлен в листинге 4. Скриншот расположен в приложении В на рисунке 2.

Листинг 4 – Функция `authenticate`

```
def authenticate(update: Update, context: CallbackContext) -> None:
    user_id = update.effective_user.id
    with create_connection() as conn:
        cursor = conn.cursor()
    cursor.execute("SELECT auth_token FROM users WHERE id=?", (user_id,))
    result = cursor.fetchone()

    if result:
        auth_token = result[0]
        update.message.reply_text(f"Аутентификация прошла успешно. Ваш
уникальный идентификатор для аутентификации: {auth_token}")
    else:
        update.message.reply_text("Вы не прошли аутентификацию. Пожалуйста,
используйте команду /start для начала процесса аутентификации.")

    update_commands(update, context)
```

Реализация функции profile

Функция `profile` предназначена для отображения статистики сообщений и выполненных команд пользователя. Сначала она получает `id`

пользователя из объекта update. Затем функция устанавливает соединение с базой данных SQLite с помощью функции create_connection и создает курсор для выполнения операций. Затем функция выполняет SQL-запрос для извлечения статистики сообщений и выполненных команд пользователя из таблицы users, используя его id. Результат запроса сохраняется в переменную result. Если результат запроса не пустой (то есть, пользователь найден в базе данных), то извлекаются данные о количестве отправленных сообщений и выполненных командах, и отправляется сообщение пользователю с этой статистикой. В противном случае, если результат пустой (пользователь не найден в базе данных), отправляется сообщение с просьбой начать с команды /start. В конце функции вызывается функция update_commands, которая обновляет список доступных команд для пользователя. Код метода представлен в листинге 5. Скриншот расположен в приложении В на рисунке 4.

Листинг 5 – Функция profile

```
def profile(update: Update, context: CallbackContext) -> None:
    user_id = update.effective_user.id
    with create_connection() as conn:
        cursor = conn.cursor()
    cursor.execute("SELECT messages_sent, commands_executed FROM users
WHERE id=?", (user_id,))
    result = cursor.fetchone()

    if result:
        messages_sent, commands_executed = result
        update.message.reply_text(f"Статистика:\n"
                                  f"Сообщений отправлено:
{messages_sent}\n"
                                  f"Команд выполнено: {commands_executed}")
    else:
        update.message.reply_text("Профиль не найден. Пожалуйста, начните с
команды /start.")

    update_commands(update, context)
```

Реализация функции update_stats

Эта функция update_stats обновляет статистику сообщений пользователя. Сначала она получает id пользователя из объекта update. Затем функция устанавливает соединение с базой данных SQLite с помощью функции create_connection и создает курсор для выполнения операций.

Далее функция выполняет SQL-запрос для обновления статистики сообщений пользователя в таблице `users`, увеличивая значение `messages_sent` на 1 для соответствующего пользователя. После этого вызывается метод `commit` для сохранения изменений в базе данных. Код метода представлен в листинге 6.

Листинг 6 – Функция `update_stats`

```
def update_stats(update: Update, context: CallbackContext) -> None:
    user_id = update.effective_user.id
    with create_connection() as conn:
        cursor = conn.cursor()
    cursor.execute("UPDATE users SET messages_sent = messages_sent + 1
WHERE id=?", (user_id,))
    conn.commit()
```

Реализация функции update_commands

Эта функция `update_commands` обновляет статистику выполненных команд пользователем. Сначала она получает `id` пользователя из объекта `update`. Затем функция устанавливает соединение с базой данных SQLite с помощью функции `create_connection` и создает курсор для выполнения операций. Далее функция получает объект сообщения и текст сообщения из объекта `update.effective_message`. Она проверяет, является ли текст сообщения командой (например, `/start`, `/authenticate`, `/profile`, `/users`). Если сообщение является командой, то выполняется SQL-запрос для обновления статистики выполненных команд пользователя в таблице `users`, увеличивая значение `commands_executed` на 1 для соответствующего пользователя. Код метода представлен в листинге 7.

Листинг 7 – Функция `update_commands`

```
def update_commands(update: Update, context: CallbackContext) -> None:
    user_id = update.effective_user.id
    with create_connection() as conn:
        cursor = conn.cursor()
        message = update.effective_message
        message_text = message.text
        # Проверяем, является ли сообщение командой
        if message_text in ['/start', '/authenticate', '/profile',
'/users']:
            cursor.execute("UPDATE users SET commands_executed =
commands_executed + 1 WHERE id=?", (user_id,))
            conn.commit()
```

Реализация функции show_users

Функция `show_users` предназначена для отображения информации о пользователях, хранящейся в базе данных. Сначала она устанавливает соединение с базой данных SQLite с помощью функции `create_connection`. Затем создается курсор для выполнения операций. Далее выполняется SQL-запрос для извлечения данных о пользователях из таблицы `users`. Полученные данные сохраняются в переменную `users` с помощью метода `fetchall`. Затем формируется сообщение с информацией о пользователях. Если есть пользователи в базе данных, то создается строка `user_list`, в которой каждый пользователь отображается в отдельной строке с его `id`, `username` и `auth_token`. Эта информация отправляется пользователю в виде текстового сообщения. Если в базе данных нет зарегистрированных пользователей, отправляется сообщение о том, что в базе данных нет пользователей. Код метода представлен в листинге 8. Скриншот расположен в приложении В на рисунке 14.

Листинг 8 – Функция `show_users`

```
def show_users(update: Update, context: CallbackContext, cursor) -> None:
    # Извлечение данных о пользователях из базы данных
    with create_connection() as conn:
        cursor = conn.cursor()
        cursor.execute("SELECT id, username, auth_token FROM users")
        users = cursor.fetchall()

        # Формирование сообщения с информацией о пользователях
        if users:
            user_list = "\n".join([f"ID: {user[0]}, Username: {user[1]}, 
Auth Token: {user[2]}" for user in users])
            update.message.reply_text(f"Пользователи в базе 
данных:\n{user_list}")
        else:
            update.message.reply_text("В базе данных нет зарегистрированных 
пользователей.")
```

Реализация функции information

Функция `information` предназначена для ответа на запрос пользователя о контактной информации клуба. Когда пользователь отправляет сообщение с командой или запросом `information`, бот отправляет текстовое

сообщение с контактной информацией. Код метода представлен в листинге 9. Скриншот расположен в приложении В на рисунке 3.

Листинг 9 – Функция `information`

```
def information(update: Update, context: CallbackContext) -> None:
    update.message.reply_text('Хотите узнать контактную информацию бота? 
Ниже находится информация для связи с нами!\n'
                              'Адрес: просп. Ленина, 
76, г. Челябинск, 454080, Россия\n'
                              'Почта: 
example@mail.ru\n'
                              'Номер телефона: 
+71234567890')
```

Реализация функции komplektPK

Функция `komplektPK` предназначена для ответа на запрос пользователя о комплектующих компьютеров, которые находятся в клубе или доступны для использования. Функция принимает `update` и `context` (объекты из библиотеки `python-telegram-bot`). Внутри функции формируется клавиатура с четырьмя кнопками, каждая из которых представляет один из четырех столов. Каждая кнопка связана с определенным значением (`callback_data`). Затем создается объект `reply_markup` с использованием `InlineKeyboardMarkup`, и этот объект отправляется в ответ на сообщение пользователя с текстом, представленным в `update.message.reply_text`. Код метода представлен в листинге 10. Скриншот расположен в приложении В на рисунке 5.

Листинг 10 – Функция `komplektPK`

```
def komplektPK(update: Update, context: CallbackContext) -> None:
    keyboard = [
        [InlineKeyboardButton("1-ый стол", callback_data='1')],
        [InlineKeyboardButton("2-ой стол", callback_data='2')],
        [InlineKeyboardButton("3-ий стол", callback_data='3')],
        [InlineKeyboardButton("4-ый стол", callback_data='4')]
    ]
    reply_markup = InlineKeyboardMarkup(keyboard)
    update.message.reply_text(
        'Хотите узнать комплектующие ПК, которые находятся у нас в 
клубе?\n'
        'На данный момент имеется 4 места, ниже представлены конфигурации 
наших ПК:',
        reply_markup=reply_markup
    )
```

Реализация функции record_in_club

Функция `record_in_club` предназначена для обработки запроса пользователя о записи в клуб или регистрации. Когда пользователь отправляет сообщение с командой или запросом `record_in_club`, бот отправляет текстовое сообщение с приветственным сообщением и инструкцией о том, что пользователь находится в разделе записи или регистрации. Код метода представлен в листинге 11. Скриншот расположен в приложении В на рисунке 6.

Листинг 11 – Функция `record_in_club`

```
def record_in_club(update: Update, context: CallbackContext) -> None:
    update.message.reply_text('Добро пожаловать в раздел записи!\n'
                              'Здесь происходит составление записи на сеанс
в наш компьютерный клуб!\n'
                              'Ниже представлены функции, которые есть в
этом разделе:\n'
                              "\n"
                              '/record - Запись на сеанс\n'
                              '/schedule - Посмотреть расписание\n'
                              '/cancelrecord - Отменить запись\n'
                              '/adminchat - Чат с администратором[WIP]')
```

Реализация функции create_schedule_connection

Функция `create_schedule_connection` возвращает соединение с базой данных SQLite с именем файла `schedule.db`. Код метода представлен в листинге 12.

Листинг 12 – Функция `create_schedule_connection`

```
def create_schedule_connection():
    return sqlite3.connect('schedule.db')
```

Реализация функции create_schedule_table

Этот код создает таблицу в базе данных SQLite с определенными столбцами и их типами данных для хранения расписания событий или занятий. Функция `create_schedule_table` использует функцию `create_schedule_connection` для установления соединения с базой данных. Затем с помощью курсора выполняется SQL-запрос для создания таблицы `schedule`. Если таблица уже существует, то команда `CREATE TABLE`

IF NOT EXISTS не создаст новую таблицу. Столбцы таблицы включают: first_name: Имя участника события или занятия, хранится в формате текста (TEXT); last_name: Фамилия участника события или занятия, хранится в формате текста (TEXT); table_number: Номер стола или места, связанного с событием или занятием, хранится в формате целого числа (INTEGER); start_time: Время начала события или занятия, хранится в формате текста (TEXT); end_time: Время окончания события или занятия, хранится в формате текста (TEXT); created_at: Время создания записи в таблице, хранится в формате текста (TEXT). После создания таблицы изменения сохраняются в базе данных с помощью метода commit. Код метода представлен в листинге 13.

Листинг 13 – Функция create_schedule_table

```
def create_schedule_table():
    with create_schedule_connection() as conn:
        cursor = conn.cursor()
        cursor.execute('''
            CREATE TABLE IF NOT EXISTS schedule (

                first_name TEXT,
                last_name TEXT,
                table_number INTEGER,
                start_time TEXT,
                end_time TEXT,
                created_at TEXT
            )
        ''')
        conn.commit()
```

Реализация функции admin_record

Функция admin_record предназначена для администратора бота. Она позволяет администратору добавлять записи в расписание. Функция начинает с проверки, является ли пользователь, вызвавший ее, администратором. Если нет, пользователю отправляется сообщение о том, что у него нет прав на добавление записей в расписание. Если пользователь является администратором, начинается процесс разговора с помощью бота. Первым шагом бот отправляет сообщение пользователю с запросом ввода имени. После этого устанавливается состояние FIRST_NAME, что означает

начало процесса ввода информации о записи в расписание, продолжение функционала в листинге 1 приложения Б. Код этого метода представлен в листинге 14. Скриншот расположен в приложении В на рисунке 13.

Листинг 14 – Функция `admin_record`

```
# Определение состояний
FIRST_NAME, LAST_NAME, TABLE_NUMBER, START_TIME, END_TIME, DATE = range(6)

def admin_record(update: Update, context: CallbackContext) -> int:
    print("Функция admin_record вызвана")  # Добавляем эту строку для
отладки
    try:
        # Проверяем, является ли пользователь администратором
        admin_id = "5101058728"  # Замените "your_admin_id" на ID вашего
администратора
        if str(update.effective_user.id) != admin_id:
            update.message.reply_text("Вы не можете добавлять записи в
расписание.")
            return ConversationHandler.END

        # Начинаем разговор и устанавливаем первое состояние
        print("Перед update.message.reply_text('Введите имя:')")  #
Добавляем эту строку для отладки
        update.message.reply_text("Введите имя:")
        print("После update.message.reply_text('Введите имя:')")  #
Добавляем эту строку для отладки
        context.user_data['current_state'] = FIRST_NAME
        print("Переход к FIRST_NAME")  # Добавляем эту строку для отладки
        return FIRST_NAME
    except Exception as e:
        print("Ошибка во время выполнения функции admin_record:", e)
    finally:
        print("Завершение функции admin_record")  # Добавляем эту строку
для отладки
```

Реализация функции record_response

Функция `record_response` обрабатывает ответы пользователя в процессе добавления записей в расписание. Функция начинает с получения текущего состояния разговора из `context.user_data`. Если текущее состояние не установлено (например, если разговор был начат заново), то бот сообщает об ошибке и завершает разговор. Затем функция определяет следующий шаг в зависимости от текущего состояния разговора. Для каждого состояния (имени, фамилии, номера стола, времени начала, времени окончания, даты) бот запрашивает соответствующую информацию у пользователя и сохраняет ее в `context.user_data`. После сохранения данных о дате записи, функция добавляет информацию в базу данных и завершает раз-

говор, отправляя сообщение об успешном добавлении записи в расписание. Код метода представлен в приложении Б в листинге 1.

Реализация функции get_current_schedule

Функция `get_current_schedule` используется для получения текущего расписания. Она начинает с определения текущего времени и времени два дня назад с использованием модуля `datetime`. Затем она устанавливает соединение с базой данных с помощью функции `create_connection`, получает курсор для выполнения запросов к базе данных и выполняет SQL-запрос для выборки записей из таблицы `schedule`, где дата создания (`created_at`) больше или равна времени два дня назад. Результаты запроса сохраняются в переменной `schedule`, которая затем возвращается из функции. Код метода представлен в листинге 15.

Листинг 15 – Функция `get_current_schedule`

```
def get_current_schedule() -> List[Tuple]:
    current_time = datetime.now()
    two_days_ago = current_time - timedelta(days=2)
    with create_connection() as conn:
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM schedule WHERE created_at >= ?",
(two_days_ago.strftime('%Y-%m-%d %H:%M:%S'),))
        schedule = cursor.fetchall()
    return schedule
```

Реализация функции show_current_schedule

Функция `show_current_schedule` используется для отображения текущего расписания в ответ на запрос пользователя. Сначала она вызывает функцию `get_current_schedule`, чтобы получить текущее расписание. Затем она проверяет, существует ли какое-либо расписание (`current_schedule`), и если да, она формирует текст расписания, перебирая каждую запись в расписании и создавая строку с данными о каждой записи. Этот текст расписания затем отправляется пользователю в ответном сообщении с помощью метода `update.message.reply_text`. Если расписание пустое, то пользователю отправляется сообщение о том, что

актуальное расписание пустое. Код метода представлен в листинге 16. Скриншот расположен в приложении В на рисунке 8.

Листинг 16 – Функция `get_current_schedule`

```
def show_current_schedule(update: Update, context: CallbackContext) ->
None:
    current_schedule = get_current_schedule()
    if current_schedule:
        schedule_text = "\n".join([f"Имя: {row[0]}, Фамилия: {row[1]},
Номер стола: {row[2]}, Время начала: {row[3]}, Время окончания: {row[4]},
Дата: {row[5]}" for row in current_schedule])
        update.message.reply_text(f"Актуальное
расписание:\n{schedule_text}")
    else:
        update.message.reply_text("Актуальное расписание пустое.")
```

Реализация функции delete_old_schedule

Функция `delete_old_schedule` используется для удаления старых записей из расписания. Она начинает с определения времени два дня назад с использованием модуля `datetime`. Затем она устанавливает соединение с базой данных с помощью функции `create_connection`, получает курсор для выполнения запросов к базе данных и выполняет SQL-запрос для удаления записей из таблицы `schedule`, где дата создания (`created_at`) меньше, чем время два дня назад. После выполнения запроса изменения фиксируются с помощью метода `conn.commit`. Код метода представлен в листинге 17.

Листинг 17 – Функция `delete_old_schedule`

```
def delete_old_schedule() -> None:
    two_days_ago = datetime.now() - timedelta(days=2)
    with create_connection() as conn:
        cursor = conn.cursor()
        cursor.execute("DELETE FROM schedule WHERE created_at < ?",
(two_days_ago.strftime('%Y-%m-%d %H:%M:%S'),))
        conn.commit()
```

Реализация функции text_message_handler

Функция `text_message_handler` является обработчиком текстовых сообщений, которые могут приходить от пользователя в процессе общения с ботом. Она проверяет текущее состояние разговора пользователя (хранится в `context.user_data`) и в зависимости от этого вызывает соответ-

ствующую функцию для обработки сообщения. Если пользователь находится в процессе добавления новой записи в расписание (т.е., его текущее состояние соответствует одному из состояний FIRST_NAME, LAST_NAME, TABLE_NUMBER, START_TIME, END_TIME, DATE), то вызывается функция record_response для обработки этого сообщения. В противном случае вызывается функция update_stats, которая используется для обновления статистики или выполнения других действий, не связанных с добавлением записей в расписание. Код метода представлен в листинге 18.

Листинг 18 – Функция text_message_handler

```
def text_message_handler(update: Update, context: CallbackContext) -> None:
    current_state = context.user_data.get('current_state')
    if current_state == FIRST_NAME or current_state == LAST_NAME or
current_state == TABLE_NUMBER or current_state == START_TIME or
current_state == END_TIME or current_state == DATE:
        record_response(update, context)
    else:
        update_stats(update, context)
```

Реализация функции main

Код представляет собой главную функцию main, которая запускает бота и определяет его логику работы. Он начинается с инициализации бота, создания соединения с базой данных и определения обработчиков для различных команд и сообщений. Затем функция запускает бота и оставляет его в режиме ожидания, чтобы он мог обрабатывать входящие сообщения и команды от пользователей. Код метода представлен в приложении Б в листинге 2.

Реализация функции button

Функция button принимает два аргумента: update (обновление) и context (контекст), и не возвращает ничего (None). Сначала из обновления update извлекается объект callback_query, который представляет запрос на обратный вызов (callback) пользователя. Затем вызывается метод answer, чтобы подтвердить получение запроса. Затем проверяется значение данных (data) запроса. Если query.data равно '1', отправляется сообщение с информацией о конфигурации первого компьютера и его стои-

мости использования в час, а также отправляется фотография компьютера 'pc1.jpg'.Код метода представлен в листинге 19.

Листинг 19 – Функция button

```
def button(update: Update, context: CallbackContext) -> None:
    query = update.callback_query
    query.answer()
    if query.data == '1':
        query.message.reply_text("Конфигурация ПК первого стола:\n"
                                 "Процессор: Intel Core i5-10600K\n"
                                 "Оперативная память: 16 ГБ DDR4\n"
                                 "Видеокарта: NVIDIA GeForce RTX 3060\n"
                                 "\n"
                                 "Стоимость пользования 100 руб/час")
        context.bot.send_photo(chat_id=query.message.chat_id,
photo=open('pc1.jpg', 'rb'))
    elif query.data == '2':
        query.message.reply_text("Конфигурация ПК второго стола:\n"
                                 "Процессор: AMD Ryzen 9 5900X\n"
                                 "Оперативная память: 32 ГБ DDR4\n"
                                 "Видеокарта: NVIDIA GeForce RTX 3070\n"
                                 "\n"
                                 "Стоимость пользования 200 руб/час")
        context.bot.send_photo(chat_id=query.message.chat_id,
photo=open('pc2.jpg', 'rb'))
    elif query.data == '3':
        query.message.reply_text("Конфигурация ПК третьего стола:\n"
                                 "Процессор: AMD Ryzen 7 5800X\n"
                                 "Оперативная память: 32 ГБ DDR4\n"
                                 "Видеокарта: AMD Radeon RX 6700 XT\n"
                                 "\n"
                                 "Стоимость пользования 300 руб/час")
        context.bot.send_photo(chat_id=query.message.chat_id,
photo=open('pc3.jpg', 'rb'))
    elif query.data == '4':
        query.message.reply_text("Конфигурация ПК четвертого стола:\n"
                                 "Процессор: Intel Core i7-11700K\n"
                                 "Оперативная память: 64 ГБ DDR4\n"
                                 "Видеокарта: NVIDIA GeForce RTX 3080\n"
                                 "\n"
                                 "Стоимость пользования 400 руб/час")
        context.bot.send_photo(chat_id=query.message.chat_id,
photo=open('pc4.jpg', 'rb'))
```

Реализация функции cancelrecord

Этот код проверяет наличие записи пользователя в таблице базы данных record_schedule по его идентификатору. Если запись существует, она удаляется, и пользователю отправляется сообщение об успешной отмене записи. В противном случае пользователю сообщается, что его запись не найдена в таблице. Код метода представлен в листинге 20. Скриншот находится в приложении В на рисунке 9.

Листинг 20 – Функция `button`

```
      def cancel_record(update: Update, context: CallbackContext) -> None:
    user_id = update.effective_user.id
    with create_record_schedule_connection() as conn:
        cursor = conn.cursor()
        cursor.execute(f"SELECT * FROM record_schedule WHERE {user_id} =
?", (user_id,))
        existing_record = cursor.fetchone()
        if existing_record:
            cursor.execute(f"DELETE FROM record_schedule WHERE {user_id} =
?", (user_id,))
            conn.commit()
            update.message.reply_text("Ваша заявка успешно отменена.")
        else:
            update.message.reply_text("Вашей заявки не найдено в таблице
заявок в клуб.")
```

Реализация функции get_record_schedule

Функция `get_record_schedule` вызывает функцию `create_record_schedule_connection`, открывает соединение с базой данных, создает курсор, выполняет SQL-запрос на выбор всех записей из таблицы `record_schedule`, извлекает результат и возвращает его. Код метода представлен в листинге 21.

Листинг 21 – Функция `get_record_schedule`

```
      def get_record_schedule():
    with create_record_schedule_connection() as conn:
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM record_schedule")
        return cursor.fetchall()
```

Вывод по третьей главе

По завершении работы была разработана архитектура телеграмм-бота, а также представлены инструменты, примененные в его реализации. Была составлена диаграмма компонентов, подробно описывающая взаимосвязь ключевых элементов системы. Кроме того, была представлена детальная спецификация основных функций и методов, примененных в реализованных компонентах, что обеспечило более полное понимание структуры и функциональности разработанного бота.

4. ТЕСТИРОВАНИЕ

4.1. Функциональное и юзабилити тестирование

Функциональное тестирование

В ходе данного тестирования проверялось реализованное приложение на соответствие предъявленным функциональным и нефункциональным требованиям, а также особенностям, описанным на этапе проектирования и эскизного проекта. В таблице 2 представлены результаты функционального тестирования реализованного телеграмм-бота для записи в компьютерный клуб.

Таблица 2 – Функциональное тестирование

№	Название теста	Действия	Ожидаемый результат	Тест пройден?
1	Проверка команды «Комплекты ПК»	1. Зайти в бота. 2. Нажать на команду в меню «start» или написать требуемую команду.	Выводится сообщение с описанием имеющихся ПК.	Да
2	Проверка команды «Контактная информация»	1. Зайти в бота. 2. Нажать на команду в меню «start» или написать требуемую команду.	Выводится сообщение с контактной информацией клуба.	Да
3	Проверка команды «Профиль»	1. Зайти в бота. 2. Нажать на команду в меню «start» или написать требуемую команду.	Выводится сообщение со статистикой, команда включается в статистику и пополняет статистику.	Да
4	Проверка генерации кода для аутентификации	1. Зайти в бота. 2. Нажать на команду в меню «start» или написать требуемую команду.	Для пользователя выводится свой собственный код для аутентификации.	Да
5	Проверка вывода таблицы с пользователя для администратора	1. Зайти в бота. 2. Написать команду «users».	Выводится таблица с зарегистрированными пользователями.	Да
6	В меню «recordinclub» проверка на вывод актуального расписания	1. Зайти в бота. 2. Нажать или написать команду «recordinclub». 3. Нажать или написать команду «schedule».	Выводится таблица с данными пользователей, которые удаляются по прошествию двух дней.	Да

Продолжение таблицы 2

№	Название теста	Действия	Ожидаемый результат	Тест пройден ?
7	В меню «recordinclub» проверка на удаление заявки из актуального расписания	1. Зайти в бота. 2. Нажать или написать команду «recordinclub». 3. При помощи администратора вписать предоставленные данные из записи заявки в актуальное расписание. 4. Нажать или написать команду «cancelrecord».	Выводится таблица, где у отсутствует заявка пользователя, либо вывод текста «Актуальное расписание отсутствует».	Да
8	Нажатие на кнопку в меню «komplektPK»	1. Зайти в бота. 2. Нажать на команду в меню «start» или написать требуемую команду. 3. В меню «komplektPK» нажать на кнопку одного из 4-ех столов.	Выводятся данные нажатого стола, а именно определенная комплектация ПК с картинкой.	Да
9	Написание или нажатие на команду «show_schedule»	1. Зайти в бота. 2. Написать требуемую команду.	Выводится таблица со всеми пользователями, которые подали заявку, либо выводится, что заявок нет на данный момент.	Да
10	Успешная запись в актуальное расписание	1. Зайти в бота. 2. Написать требуемую команду для записи в актуальное расписание. 3. Поочередно записывать требуемые данные в таблицу.	Выводится таблица с всеми пользователями, которые подали заявку и там находится написанная заявка.	Да
11	Написание или нажатие на команду «users»	1. Зайти в бота. 2. Написать требуемую команду.	Выводится таблица со всеми зарегистрированными пользователями.	Да
12	Проверка команды «record»	1. Зайти в бота. 2. Написать или нажать на требуемую команду и постепенно вписывать требуемые данные.	После записи появляется текст «Заявка успешно отправлена на обработку».	Да

Окончание таблицы 2

№	Название теста	Действия	Ожидаемый результат	Тест пройден ?
12	Проверка команды «record»	3. Зайти в бота. 4. Написать или нажать на требуемую команду и постепенно вписывать требуемые данные.	После записи появляется текст «Заявка успешно отправлена на обработку».	Да
13	Проверка команды «start»	1. Зайти в бота. 2. Нажать на требуемую команду.	После нажатия на команду, которая возникает при первом входе в бота, должно отобразиться стартовое меню с командами	Да
14	Проверка команды «adminhelp»	1. Зайти в бота. 2. Написать требуемую команду.	После прописывания команды в чат бота появляется меню с командами для администратора,	Да
15	Проверка регистрации пользователей в таблице «users»	1. Найти тестировщика. 2. Требуется, чтобы тестировщик запустил бота.	При запуске и входе в телеграмм-бот другого пользователя его краткие данные записываются в таблицу «users»	Да
16	Проверка команды «adminchat»	1. Запуск бота. 2. Написать или нажать на требуемую команду.	После написания команды вызывается чат с администратором.	Да
17	Проверка команды «endchat»	1. Запуск бота. 2. Написать или нажать на команду «adminchat». 3. Написать команду «endchat».	После написания команды «adminchat» и написания «endchat» чат с администратором заканчивает свою работу.	Да
18	Проверка отправки сообщений при выполнении команды «adminchat»	1. Запуск бота. 2. Написать или нажать на команду «adminchat». 3. При отправке сообщения пользователь или администратор получают такое же сообщение.	После написания команды «adminchat» сообщения отправляются корректно как от пользователя или администратора.	Да

Юзабилити тестирование

Для оценки удобства использования приложения были приглашены на тестирование несколько человек из моего круга общения не причастных к разработке. Эксперимент включал проверку приложения двумя пользователями. В результате тестирования были выявлены некоторые проблемы, касающиеся функциональности приложения и механики игры.

1. Некоторые команды, введенные пользователями, не учитывались в общей статистике профиля. Это могло привести к недостоверным данным о достижениях игрока.

2. Не всегда после нажатия на определенную команду появлялось ожидаемое сообщение от бота. Это создавало путаницу у пользователей и могло мешать нормальному взаимодействию с приложением.

Также планируется исследование на период с 15 по 22 июня в помещениях компьютерного клуба, находящегося недалеко от ЮУрГу. Участники и тестировщики будут приглашены по расписанию на удобное для них время.

Вывод по четвертой главе

В ходе проделанной работы, все функциональные тесты были успешно завершены, что является положительным результатом. Однако, благодаря юзабилити тестированию обнаружины и устранены обнаруженные ошибки, что позволило улучшить общий опыт пользователей от использования приложения.

ЗАКЛЮЧЕНИЕ

В заключении данной работы следует отметить важность и актуальность разработки телеграмм-ботов в современном мире информационных технологий. Создание ботов становится все более востребованным в различных сферах, включая развлекательную индустрию, образование, бизнес и многое другое.

Изучение методов программирования ботов на языке Python и их реализация в данной работе позволили получить ценный опыт и практические навыки, которые могут быть применены в различных проектах и задачах.

А также в данной работе была проведена разработка телеграмм-бота для бронирования сеанса игры в компьютерном клубе для ОС Windows на языке Python. При этом были решены следующие задачи.

1. Изучены методы программирования ботов и телеграмм-ботов на Python.
2. Обзоры аналогов.
3. Спроектировано приложение.
4. Реализовано приложение.
5. Протестировано разработанное приложение, сделав несколько заданий для тестирования.

В результате данной работы были получены как практические, так и теоретические знания в области создания ботов и телеграмм-ботов на языке Python. Полученный опыт и навыки могут быть успешно применены в дальнейших проектах и исследованиях в области информационных технологий.

ЛИТЕРАТУРА

1. Седжвик Р., Уэйн К., Дондеро Р. Программирование на языке Python. // СПб. Диалектика 2019 – 736 с.

2. Мокеев В.В. Web-аналитика на Python. // Челябинск Издательский Центр ЮУрГУ 2020 – 143 с.

3. Лутц М. Программирование на Python. / Пер. Маккавеева С. – СПб. Символ-Плюс 2002 – 1135 с.

4. Рамальо Л.К вершинам мастерства Python. / пер. с англ. Слинкин А. А. – Москва: ДМК-Пресс, 2016. – 767 с.

5. Доусон М. Программируем на Python. // Санкт-Петербург и др.: Питер, 2022. – 416 с.

6. Васильев А.Н. Программирование на Python в примерах и задачах. // Москва: Эксмо, 2022. – 614 с.

7. МакГрат М. Программирование на Python для начинающих / пер. с англ. Райтмана М. А. – Москва: Эксмо, 2022. – 192 с.

8. И. ван Лейнингем Освой самостоятельно Python за 24 часа. / Пер. с англ. Лузина В.Д., Ревы О.Н. – М. и др.: Вильямс, 2001. – 443 с.

9. Прохоренок Н.А. Python. Самое необходимое. // СПб.: БХВ-Петербург, 2011. – 408 с.

10. Слаткин Б. Секреты Python: 59 рекомендаций по написанию эффективного кода. / пер. с англ. Гузикевича А.Г. – М.; СПб.: Диалектика, 2019. – 270 с.

11. Доусон М. Программируем на Python. / пер. с англ. Порицкого В. – СПб. и др.: Питер, 2020. – 414 с.

12. Бэрри П. Изучаем программирование на Python. / пер. с англ. Райтман М.А. – М.: ЭКСМО, 2019. – 618 с.

13. Златопольский Д.М. Основы программирования на языке Python: учебник. // М.: ДМК ПРЕСС, 2018. – 394 с.

14. Сузи Р.А. Язык программирования Python: Учеб. Пособие. // М.: Интернет-Университет Информационных Технологий: БИНОМ. Лаборатория знаний, 2006. – 326 с.

15. Бизли Д.М. Язык программирования Python: Справ.: Пер. с англ. // Киев: ДиаСофт, 2000. – 326 с.

ПРИЛОЖЕНИЯ

Приложение А. Спецификация вариантов использования

В данном приложении приведена спецификация вариантов использования (ВИ) расположена в таблицах 1–9.

Таблица 1 – Спецификация ВИ «Контактная информация»

Прецедент: Контактная информация
ID: 1
Краткое описание: Пользователь может получить контактную информацию о клубе.
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловия: Пользователь нажимает или вводит команду «Контактная информация»
Основной поток: 1. Пользователь запустил бота. 2. Пользователь нажимает или вводит команду «Контактная информация». 3. Выводится сообщение с контактной информацией клуба.
Постусловия: Сообщение выводится и бот ожидает дальнейших действий со стороны пользователя.
Альтернативные потоки: Приложение выводит сообщение об ошибке.

Таблица 2 – Спецификация ВИ «Комплектующие ПК»

Прецедент: Комплектующие ПК
ID: 2
Краткое описание: Пользователь может посмотреть комплектующие ПК, которые доступны в клубе.
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловия: Пользователь нажимает или вводит команду «Комплектующие ПК»
Основной поток: 1. Пользователь запустил бота. 2. Пользователь нажал на кнопку «Комплектующие ПК». 3. Бот выводит сообщение ввиде списка о имеющихся ПК и комплектовке.
Постусловия: Сообщение выводится и бот ожидает дальнейших действий со стороны пользователя.
Альтернативные потоки: Приложение выводит сообщение об ошибке.

Таблица 3 – Спецификация ВИ «Аутентификация»

Прецедент: Аутентификация
ID: 3
Краткое описание: Пользователь может посмотреть на код аутентификации который может понадобиться для подтверждения личности.
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловия: Пользователь нажимает или вводит команду «Аутентификация»
Основной поток: 1. Пользователь запустил бота. 2. Пользователь нажал на кнопку «Аутентификация». 3. Бот выводит сообщение, где выводится код аутентификации.
Постусловия: Сообщение выводится и бот ожидает дальнейших действий со стороны пользователя.
Альтернативные потоки: Приложение выводит сообщение об ошибке.

Таблица 4 – Спецификация ВИ «Профиль»

Прецедент: Профиль
ID: 4
Краткое описание: Пользователь может посмотреть на профиль, где собрана статистика некоторых действий пользователя.
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловия: Пользователь нажимает или вводит команду «Профиль»
Основной поток: 1. Пользователь запустил бота. 2. Пользователь нажал на кнопку «Профиль». 3. Бот выводит сообщение, где собрана статистика пользователя.
Постусловия: Сообщение выводится и бот ожидает дальнейших действий со стороны пользователя.
Альтернативные потоки: Приложение выводит сообщение об ошибке.

Таблица 5 – Спецификация ВИ «Составление записи на сеанс»

Прецедент: Аутентификация
ID: 5
Краткое описание: Пользователь, нажимая или написав команду, видит меню с разделом записи на сеанс телеграмм-бота.
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловия: Пользователь нажимает или вводит команду «Составление записи на сеанс»
Основной поток: 1. Пользователь запустил бота. 2. Пользователь нажал на кнопку «Составление записи на сеанс». 3. Бот выводит сообщение, где выводится меню раздела записи на сеанс с командами телеграмм-бота и кратким их описанием.
Постусловия: Сообщение выводится и бот ожидает дальнейших действий со стороны пользователя.
Альтернативные потоки: Приложение выводит сообщение об ошибке.

Таблица 6 – Спецификация ВИ «Посмотреть актуальное расписание»

Прецедент: Посмотреть актуальное расписание
ID: 6
Краткое описание: Пользователь может посмотреть на актуальное расписание, выводится сообщение с таблицей расписания или сообщение о том, что расписание пустое.
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловия: Пользователь нажимает или вводит команду «Посмотреть актуальное расписание»
Основной поток: 1. Пользователь запустил бота. 2. Пользователь нажимает или вводит команду «Посмотреть актуальное расписание». 3. Бот выводит сообщение, где находится таблица с актуальным расписанием.
Постусловия: Сообщение выводится и бот ожидает дальнейших действий со стороны пользователя.
Альтернативные потоки: Приложение выводит сообщение о том, что таблица пуста.

Таблица 7 – Спецификация ВИ «Чат с администратором»

Прецедент: Чат с администратором
ID: 7
Краткое описание: Пользователь может открыть чат с администратором для каких-либо вопросов по поводу бота.
Главные актеры: Пользователь
Второстепенные актеры: Администратор
Предусловия: Пользователь нажимает или вводит команду «Чат с администратором»
Основной поток: 1. Пользователь запустил бота. 2. Пользователь нажимает или вводит команду «Чат с администратором». 3. Бот выводит сообщение о том, что требуется подождать, пока администратор не примет запрос.
Постусловия: Сообщение выводится и бот ожидает дальнейших действий со стороны пользователя, а также администратора.
Альтернативные потоки: Приложение выводит сообщение об ошибке.

Таблица 8 – Спецификация ВИ «Запись на сеанс»

Прецедент: Запись на сеанс
ID: 8
Краткое описание: Пользователь может отправить запись на сеанс посредством ввода некоторых данных отвечая на появляющиеся вопросы, которые задает телеграмм-бот.
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловия: Пользователь нажимает или вводит команду «Запись на сеанс»
Основной поток: 1. Пользователь запустил бота. 2. Пользователь нажимает или вводит команду «Чат с администратором». 3. Бот выводит сообщения с вопросами о некоторых данных, которые вводит пользователь, после ответа на вопросы бот отправляет данные в таблицу с заявкам на запись в клуб, а пользователю выводится сообщение «Заявка отправлена на рассмотрение».
Постусловия: Сообщение выводится и бот ожидает дальнейших действий со стороны пользователя.
Альтернативные потоки: Приложение выводит сообщение об ошибке.

Таблица 9 – Спецификация ВИ «Отменить запись»

Прецедент: Отменить запись
ID: 9
Краткое описание: Пользователь может отменить заявку на запись на сеанс в клуб, если он отправлял заявку.
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловия: Пользователь нажимает или вводит команду «Отменить запись»
Основной поток: 1. Пользователь запустил бота. 2. Пользователь нажимает или вводит команду «Отменить запись». 3. Заявка удаляется из таблицы с заявками и выводится сообщение «Заявка отменена».
Постусловия: Сообщение выводится и бот ожидает дальнейших действий со стороны пользователя.
Альтернативные потоки: Приложение выводит сообщение об ошибке.

Приложение Б. Код реализации компонентов

В данном приложении приведен код, реализующий массивы `record_response` (листинг 1) и `main` (листинг 2).

Листинг 1 – Функция `record_response`

```
def record_response(update: Update, context: CallbackContext) -> int:
    print("Функция record_response вызвана")  # Добавляем эту строку для
отладки
    current_state = context.user_data.get('current_state', None)
    if current_state is None:
        # Если текущее состояние не установлено, завершаем разговор
        update.message.reply_text("Что-то пошло не так. Попробуйте снова.")
        return ConversationHandler.END

    # Определяем следующий шаг в зависимости от текущего состояния
    if current_state == FIRST_NAME:
        context.user_data['first_name'] = update.message.text
        update.message.reply_text("Введите фамилию:")
        context.user_data['current_state'] = LAST_NAME
        print("Переход к LAST_NAME")
        return LAST_NAME
    elif current_state == LAST_NAME:
        context.user_data['last_name'] = update.message.text
        update.message.reply_text("Введите номер стола:")
        context.user_data['current_state'] = TABLE_NUMBER
        print("Переход к TABLE_NUMBER")
        return TABLE_NUMBER
    elif current_state == TABLE_NUMBER:
        context.user_data['table_number'] = update.message.text
        update.message.reply_text("Введите стартовое время записи (в
формате ЧЧ:ММ):")
        context.user_data['current_state'] = START_TIME
        print("Переход к START_TIME")
        return START_TIME
    elif current_state == START_TIME:
        context.user_data['start_time'] = update.message.text
        update.message.reply_text("Введите конечное время записи (в формате
ЧЧ:ММ):")
        context.user_data['current_state'] = END_TIME
        print("Переход к END_TIME")
        return END_TIME
    elif current_state == END_TIME:
        context.user_data['end_time'] = update.message.text
        update.message.reply_text("Введите дату записи (в формате ГГГГ-ММ-
ДД):")
        context.user_data['current_state'] = DATE
        print("Переход к DATE")
        return DATE
    elif current_state == DATE:
        context.user_data['date'] = update.message.text
        # Сохраняем данные в базу данных
        with create_connection() as conn:
            cursor = conn.cursor()
            print("Создано соединение с базой данных")
            print("Перед выполнением запроса SQL")
            cursor.execute(
                "INSERT INTO schedule (first_name, last_name, table_number,
start_time, end_time, created_at) VALUES (?, ?, ?, ?, ?, ?)",
                (context.user_data['first_name'],
context.user_data['last_name'],
```

```
                context.user_data['table_number'],
context.user_data['start_time'], context.user_data['end_time'],
                context.user_data['date']))
            print("Выполнен запрос SQL")
            conn.commit()
            print("Изменения зафиксированы в базе данных")

        update.message.reply_text("Запись успешно добавлена в расписание.")
        # Сбрасываем текущее состояние
        context.user_data['current_state'] = None
        return ConversationHandler.END
```

Листинг 2 – Функция `main`

```
def main() -> None:
    # Инициализация бота
    updater = Updater("6456170853:AAHKtXIuk66vW_HaUAzmh2wEar7eYp0c-ik")
    dispatcher = updater.dispatcher

    # Создание таблицы в базе данных
    create_table()

    # Создание таблицы для записей в расписании
    create_schedule_table()
    delete_old_schedule()

    # Сохраняем объект курсора в bot_data
    cursor = create_connection().cursor()
    dispatcher.bot_data['cursor'] = cursor

    # Добавление обработчиков команд
    dispatcher.add_handler(CommandHandler("start", lambda update, context:
start(update, context)))
    dispatcher.add_handler(CommandHandler("authenticate", lambda update,
context: authenticate(update, context)))
    dispatcher.add_handler(CommandHandler("profile", lambda update,
context: profile(update, context)))
    dispatcher.add_handler(CommandHandler("users", lambda update, context:
show_users(update, context, cursor)))
    dispatcher.add_handler(CommandHandler("adminchat", admin_chat))
    dispatcher.add_handler(CallbackQueryHandler(confirm_admin))
    dispatcher.add_handler(CommandHandler("information", information))
    dispatcher.add_handler(CommandHandler("komplektPK", komplektPK))
    dispatcher.add_handler(CommandHandler("recordinclub", record_in_club))
    dispatcher.add_handler(CommandHandler("adminrecord", admin_record))
    dispatcher.add_handler(CommandHandler("shedule", lambda update,
context: show_users(update, context, cursor)))
    dispatcher.add_handler(CommandHandler("schedule",
show_current_schedule))
    dispatcher.add_handler(CommandHandler("record", record))

    # Добавление обработчика ошибок
    dispatcher.add_error_handler(error)
    # Регистрация обработчиков для статистики
    dispatcher.add_handler(MessageHandler(Filters.text & ~Filters.command,
text_message_handler))
    dispatcher.add_handler(MessageHandler(Filters.command,
update_commands))
    dispatcher.add_handler(MessageHandler(Filters.text & ~Filters.command,
handle_text))
```

Окончание листинга 2 приложения Б

```
    # Запуск бота
    updater.start_polling()
    updater.idle()

if __name__ == '__main__':
    main()
```

Приложение В. Скриншоты приложения

Скриншоты разработанных функций приложения приведены на рисунках 1–14.

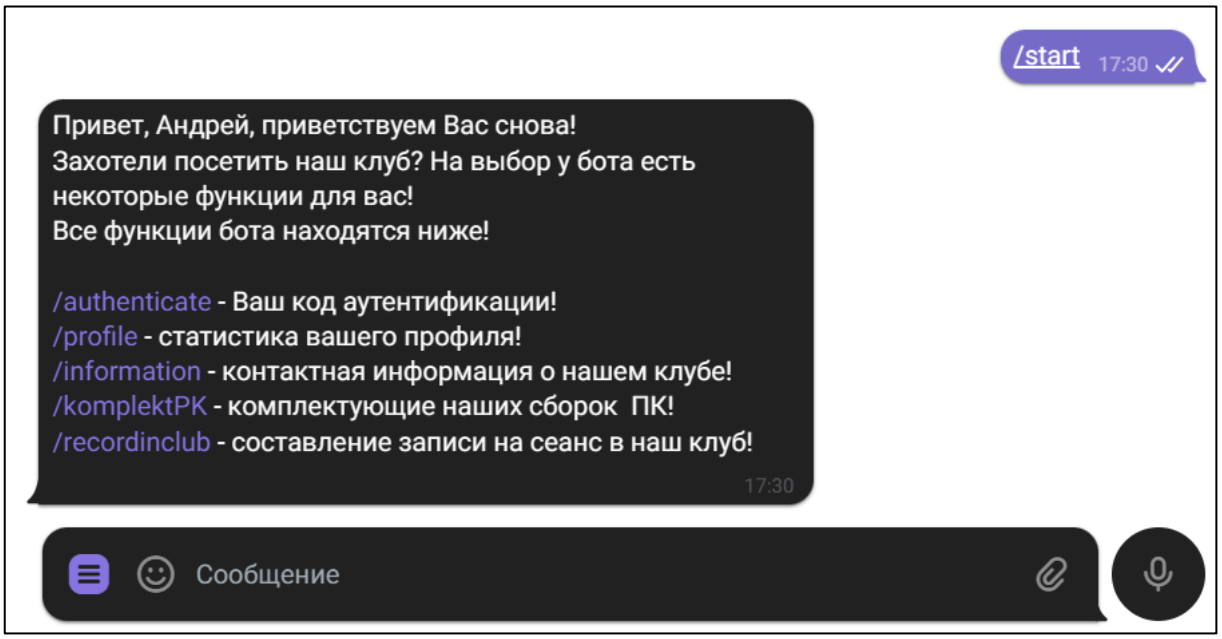

Рисунок 1 – Функция start

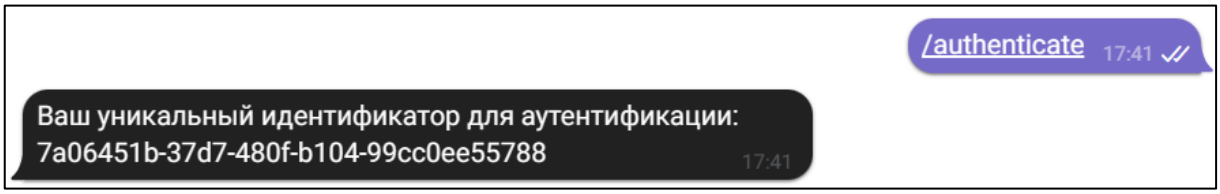

Рисунок 2 – Функция authenticate

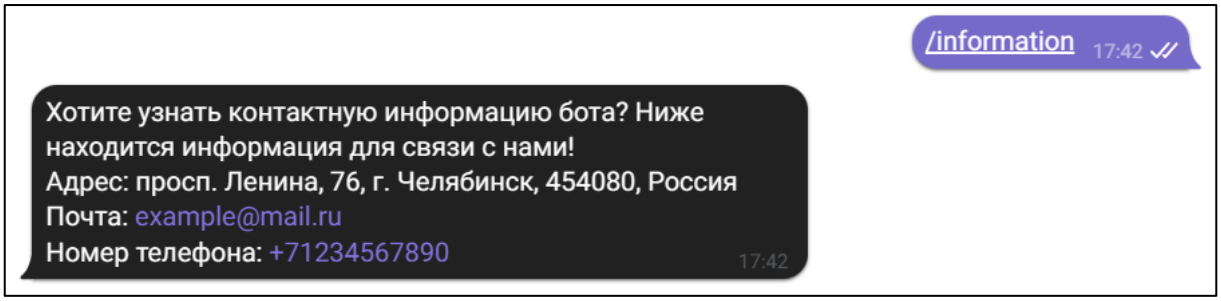

Рисунок 3 – Функция information

Рисунок 4 – Функция profile

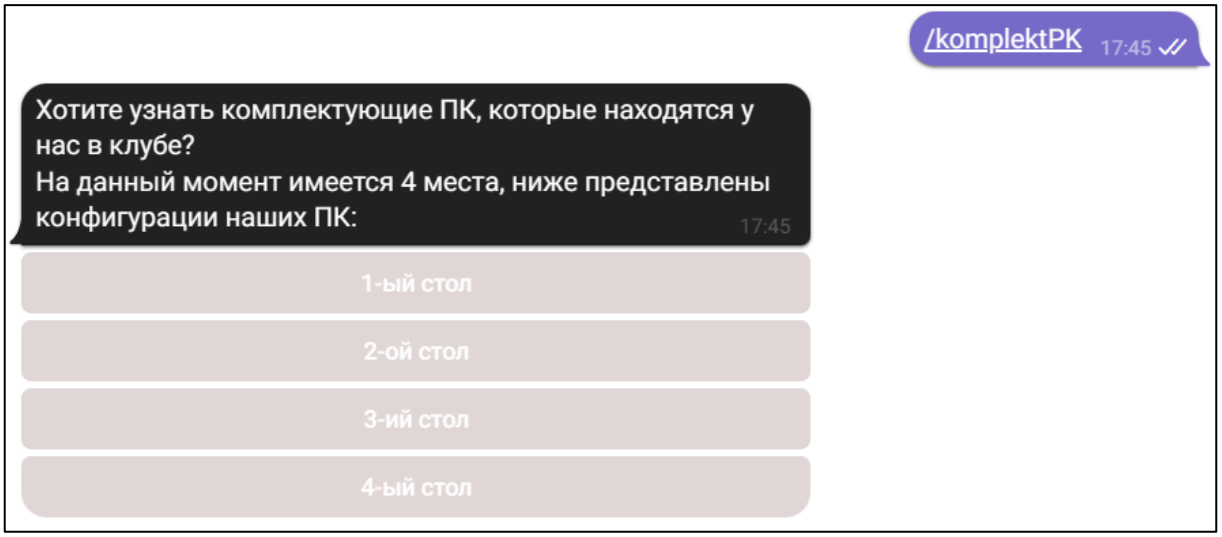

Рисунок 5 – Функция komplektPK

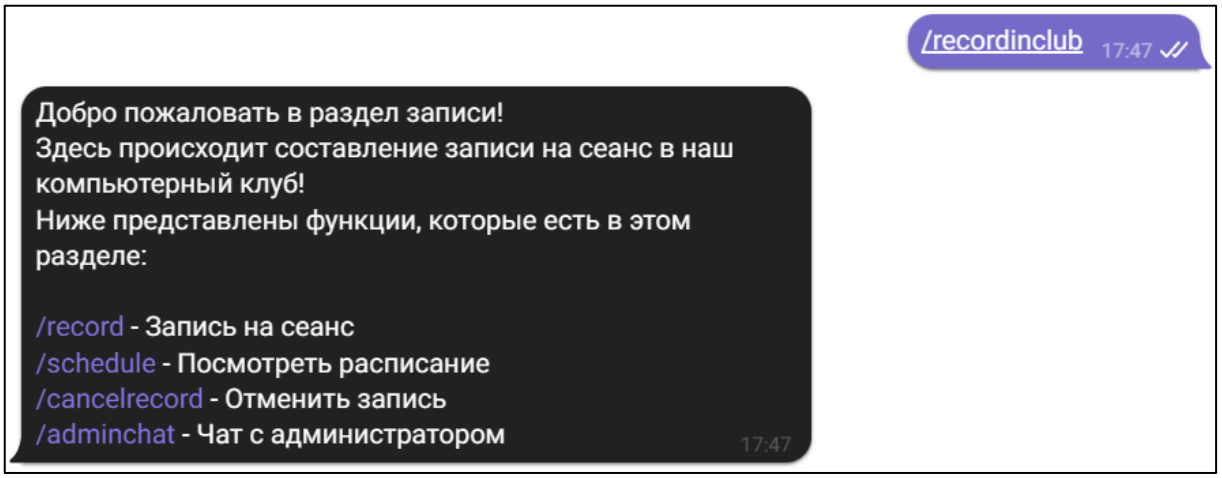

Рисунок 6 – Функция recordinclub

Рисунок 7 – Функция record

Рисунок 8 – Функция schedule

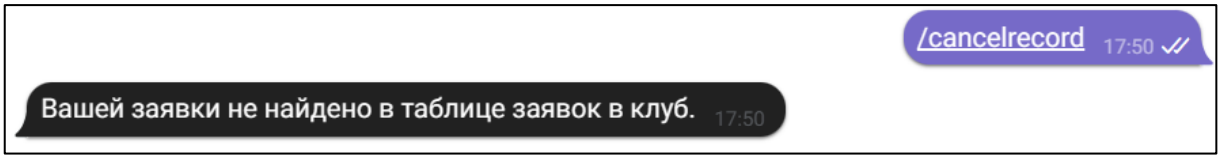

Рисунок 9 – Функция cancelrecord

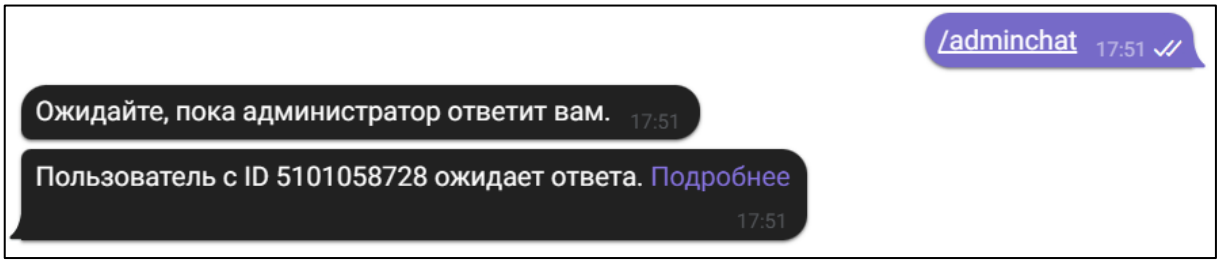

Рисунок 10 – Функция adminchat

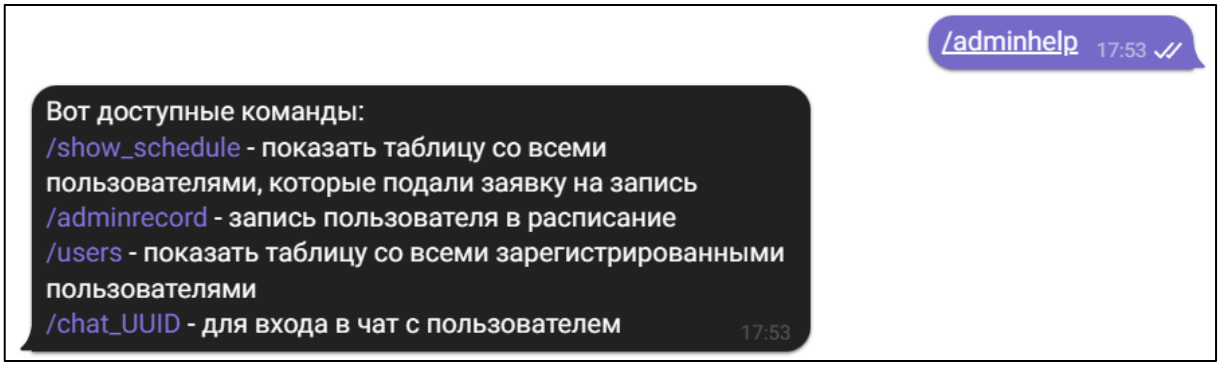

Рисунок 11 – Функция adminhelp

Рисунок 12 – Функция show_schedule

Рисунок 13 – Функция adminrecord

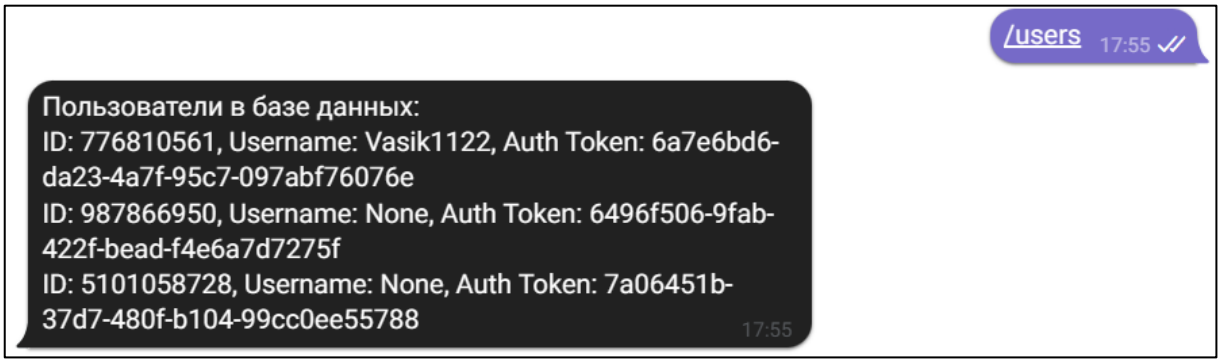

Рисунок 14 – Функция show_users

Приложение Г. Диаграмма последовательности

В данном приложении приведена диаграмма последовательности расположенная на рисунке 15.

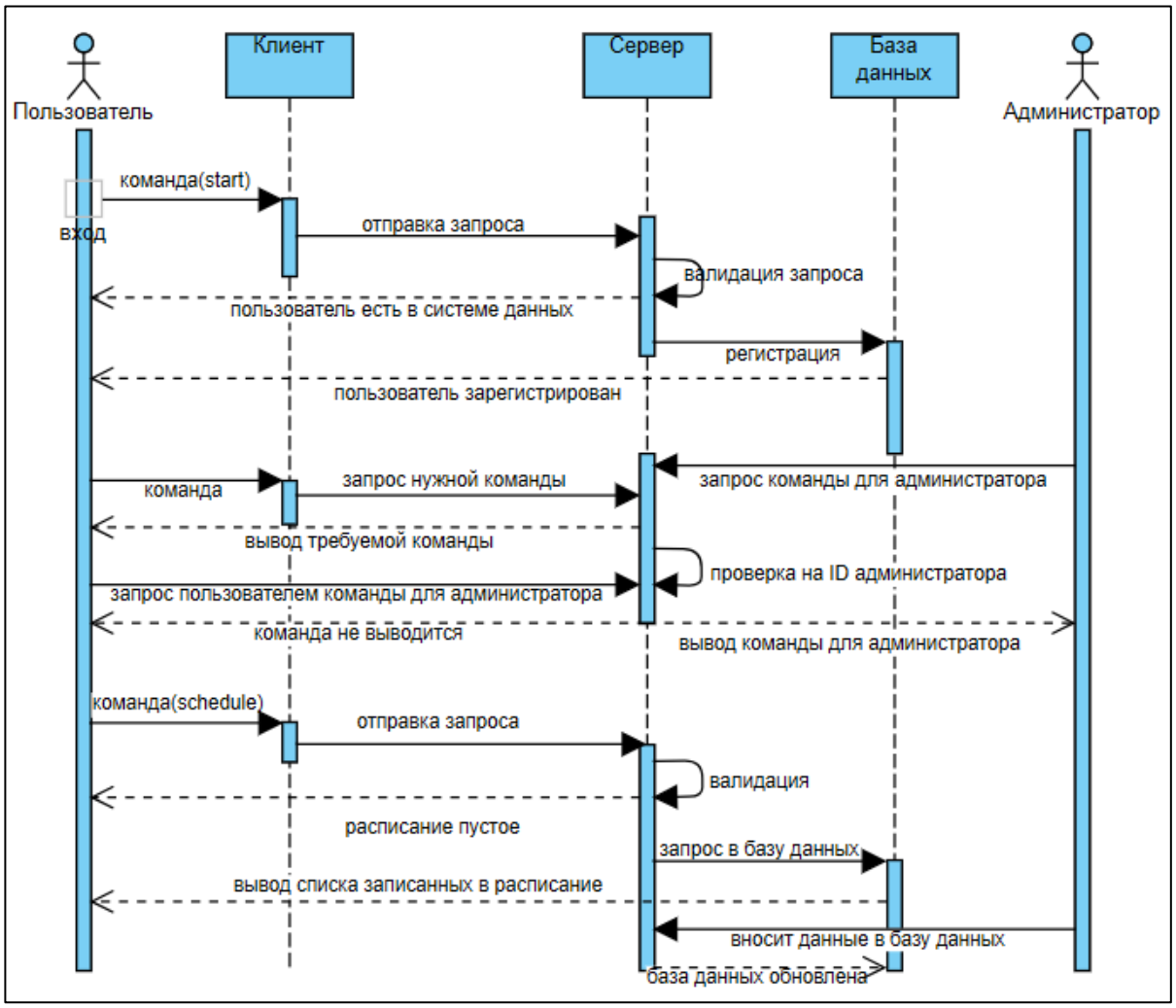

Рисунок 15 – Диаграмма последовательности