

Разработка чат-бота телеграмм для работы со спортивным сайтом

Программа:

«Искусственный интеллект. Цифровые Профессии»

Специализация:

«Инженер искусственный интеллект»

Студент:

Максимов Павел Сергеевич

Дипломный проект: «Разработка чат-бота телеграмм для работы со спортивным сайтом»

Содержание:

Введение	3
Глава 1. Теоретическая часть	6
1.2. Выбор инструментов.....	6
1.1.1. MS Visual Studio Code	6
1.1.2. Telegram	8
1.1.3. Microsoft Office 365 Word	10
1.2. Источник информации	11
Глава 2. Практическая часть	12
2.1. Среда разработки	12
2.2. Установка интерпретатора Python 3	15
2.3. Установка необходимых python-пакетов.	18
2.3.1. Pandas	18
2.3.2. Python-dotenv.....	18
2.3.3. Selenium	19
2.3.4. pyTelegramBotAPI.....	20
2.4. Начало работы. Создание телеграм-бота и получение токена.....	21
2.5. Написание кода	23
2.5.1. Заполнение переменных окружения.	23
2.5.2. Собираем информацию о турнирах на странице сайта.	24
2.5.3. Обработка вводимых данных	33
2.5.4. Загрузка истории личных встреч.	36
2.5.5. Основной код телеграмм-бота.	41
Заключение.....	70
Список литературы.....	72
Приложение	74

Введение

В современном мире цифровые технологии проникают во все сферы жизни, включая спорт. Интернет стал неотъемлемой частью спортивного сообщества, предоставляя доступ к новостям, результатам, статистике и аналитике. Однако, несмотря на обилие информации, иногда возникает необходимость в быстром и удобном получении ответов на конкретные вопросы, а также в персональной поддержке и консультации. В этом контексте чат-боты, основанные на искусственном интеллекте, представляют собой перспективный инструмент для улучшения пользовательского опыта и повышения уровня интерактивности спортивных сайтов.

Темой проекта является разработка чат-бота в телеграмм для работы со спортивным сайтом.

Актуальность темы данной дипломной работы обусловлена следующими факторами:

- Повышение востребованности чат-ботов: Чат-боты становятся все более популярными в различных отраслях, включая спорт. Их способность предоставлять информацию в реальном времени, автоматизировать рутинные задачи и обеспечивать персональный подход делает их незаменимым инструментом для улучшения пользовательского опыта.
- Развитие технологий искусственного интеллекта: Прогресс в области искусственного интеллекта (ИИ) позволяет разрабатывать более сложные и умные чат-боты, способные вести более естественные диалоги и давать более точные ответы.

- Рост популярности мессенджера Telegram: Telegram является одним из самых популярных мессенджеров в мире, предлагая широкий набор функций для разработки чат-ботов и интеграции с другими сервисами.

Целью дипломной работы является разработка чат-бота Telegram для спортивного сайта, способного предоставлять пользователям следующие возможности:

1. Получение актуальной информации о спортивных событиях: результаты матчей, расписание, таблицы турниров.
2. Поиск информации о спортсменах: статистика и даты следующих выступлений

В рамках дипломной работы **будут решены следующие задачи**:

- Выбор платформы и инструментов разработки: анализ доступных платформ и библиотек для разработки чат-ботов Telegram.
- Разработка архитектуры и функциональности чат-бота: определение ключевых функций и структуры разработки.
- Разработка интерфейса чат-бота: создание удобного и интуитивно понятного интерфейса для взаимодействия пользователей с чат-ботом.
- Тестирование и отладка чат-бота: проведение тестирования и отладки разработанного чат-бота для обеспечения его правильной работы и достижения заданных целей.

Ожидается, что разработанный чат-бот Telegram станет ценным инструментом для пользователей спортивного сайта, предоставляя им удобный и эффективный доступ к необходимой информации и поддержке.

Результаты дипломной работы могут быть применены в практике для улучшения пользовательского опыта на спортивных сайтах и в других отраслях, где используются чат-боты.

Научная новизна дипломной работы заключается в разработке и внедрении инновационного решения для повышения уровня интерактивности и удобства пользовательского опыта спортивного сайта с помощью чат-бота Telegram, обученного на основе методов машинного обучения.

Практическая значимость работы заключается в создании рабочего прототипа чат-бота Telegram, который может быть использован в реальных условиях для улучшения пользовательского опыта и повышения эффективности работы спортивного сайта.

Инструменты:

Visual Studio Code, Python, Selenium, MS Office 365 ProPlus, Telegram, Amvera Cloud

Глава 1. Теоретическая часть

1.2. Выбор инструментов

В качестве инструментов, необходимых для написания дипломной работы будем использовать бесплатные программы, находящиеся в свободном доступе, такие как MS Visual Studio Code, Telegram, MS Office

1.1.1. MS Visual Studio Code

Visual Studio Code - это популярный, бесплатный и мощный редактор кода, который идеально подходит для написания работы, особенно если она включает в себя программирование[1].

Почему именно Visual Studio Code:

- **Универсальность.** Visual Studio Code поддерживает множество языков программирования, включая Python, Java, C++, JavaScript, HTML, CSS, и т.д., что делает его идеальным инструментом для работы с различными типами проектов. Возможность использования Visual Studio Code для написания текста делает его универсальным инструментом для создания документации, отчетов и самой дипломной работы.
- **Функциональность.**
 - Интеллектуальное автодополнение кода: Visual Studio Code предлагает умные подсказки и автодополнение кода, что ускоряет процесс написания и снижает количество ошибок.
 - Отладка: Встроенный отладчик позволяет эффективно находить и исправлять ошибки в коде.

- Встроенный терминал: Visual Studio Code имеет интегрированный терминал, который позволяет запускать команды и работать с Git-репозиториями.
- Расширения: Visual Studio Code обладает богатым набором расширений, которые добавляют новые функции, такие как поддержка различных языков, темы оформления, инструменты для работы с базами данных и многое другое.
- **Бесплатность и открытый код**. Visual Studio Code бесплатный и доступен для всех платформ (Windows, macOS, Linux). Открытый исходный код позволяет разработчикам вносить свой вклад в развитие и улучшение редактора.
- **Интеграция с Git**. Visual Studio Code имеет встроенную поддержку Git, что позволяет легко управлять версиями кода и работать с репозиториями.
- **Простота использования**. Visual Studio Code имеет интуитивно понятный интерфейс, что делает его доступным даже для новичков. Богатая документация и множество обучающих материалов онлайн помогают быстро освоить редактор.

Visual Studio Code предлагает мощные возможности для написания дипломной работы, сочетая в себе функциональность редактора кода, универсальность и простоту использования[2].

1.1.2. Telegram

Telegram – это приложение для обмена сообщениями, ориентированное на скорость и безопасность, оно супербыстрое, простое и бесплатное[12].

Телеграмм-бот – это автоматизированная программа, функционирующая в мессенджере Telegram, способная выполнять определенные задачи по запросу пользователя [9]. Проще говоря, это как виртуальный помощник, который всегда доступен и готов помочь вам.

Преимущества использования телеграмм-ботов:

- **Удобство.** Все взаимодействие происходит в привычной среде мессенджера, без необходимости установки дополнительных приложений.
- **Доступность.** Бот всегда онлайн и готов к работе 24/7.
- **Автоматизация.** Освобождает вас от рутинных задач.
- **Персонализация.** Бот может быть настроен под ваши индивидуальные потребности.
- **Интеграция.** Боты могут интегрироваться с другими сервисами, расширяя свои возможности.

Причины выбора Telegram в качестве платформы для создания чат-бота.

- **Открытый API.** Telegram предоставляет разработчикам простой и понятный интерфейс для создания ботов.

- **Большая аудитория.** Telegram имеет более 500 миллионов активных пользователей, что обеспечивает широкую базу для потенциальных пользователей бота.
- **Безопасность.** Telegram известен своей безопасностью и конфиденциальностью данных.
- **Разнообразие инструментов.** Telegram предлагает широкий спектр инструментов для создания ботов, от простых до сложных, с возможностью интеграции с внешними сервисами.
- **Гибкость.** Telegram позволяет создавать боты для самых разных целей, от простого информационного помощника до сложного электронного магазина.

Телеграмм-бот – это мощный инструмент, способный значительно упростить жизнь и оптимизировать работу. Telegram предлагает удобную и доступную платформу для создания ботов с широкими возможностями для персонализации и интеграции. Независимо от ваших потребностей, телеграмм-бот может стать вашим надежным помощником [7, 10, 11].

1.1.3. Microsoft Office 365 Word

- Microsoft Word является самым популярным и удобным текстовым редактором.
- Он доступен на большинстве компьютеров и легко устанавливается.
- Интерфейс программы интуитивно понятен и не требует специальных знаний.
- Простой доступ к различным функциям форматирования текста, таблиц, списков, изображений и других элементов делает процесс написания дипломной работы более комфортным.
- Microsoft Word является универсальным форматом, который совместим с различными операционными системами и устройствами.
- Возможность совместной работы над документами в реальном времени позволяет участвовать в написании диплома нескольким людям, что особенно актуально при работе в группах.
- Microsoft Word имеет обширную документацию и множество онлайн-ресурсов, которые помогают разобраться с различными функциями программы[8].

1.2. Источник информации

Основным и единственным источником информации для работы чат-бота является сайт <https://www.sport-liga.pro/ru> [13].

Liga Pro — это организация, которая проводит регулярные турниры по настольному теннису, волейболу, баскетболу 3x3, киберболу, микрофутзалу и интерактивным дисциплинам: FIFA, NHL и NBA. Также на регулярной основе в Liga Pro проводятся турниры по настольному теннису для слабовзржих спортсменов. За трансляциями матчей Liga Pro следят зрители из 150 стран мира.

Liga Pro — организатор регулярных турниров по настольному теннису

История Liga Pro начинается с турниров по настольному теннису среди профессионалов и любителей в 2018 году. Сейчас на спортивных площадках лиги играют 400 разрядников и высококвалифицированных профессионалов в настольном теннисе. Онлайн-трансляции противостояний набирают около 35 миллионов просмотров за год.

В настольном теннисе Liga Pro самая полноценная статистика: у каждого игрока есть карточка с данными о количестве игр, соперников и набранному рейтингу. Регулярные соревнования, у которых есть собственный сайт, механизм расчета рейтинга и разделения участников по уровню мастерства на лиги — когда-то это было пределом мечтаний для участников любительских турниров как в Москве, так и в других регионах России.

Этот вид спорта стал особенно популярным с началом пандемии коронавируса в 2020 году. Тогда турниры Liga Pro стали известными не только в России, но и во всем мире.

Глава 2. Практическая часть

2.1. Среда разработки

Среда разработки (IDE) — это комплекс программных средств, используемый программистами для разработки программного обеспечения (ПО).

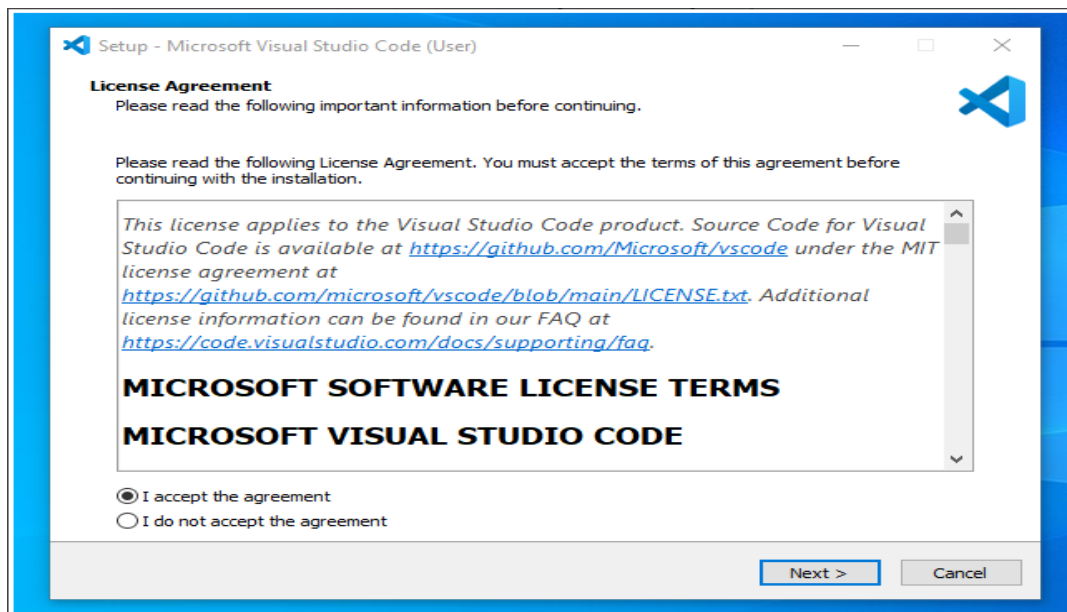
Среда разработки включает в себя:

- текстовый редактор;
- транслятор (компилятор и/или интерпретатор);
- средства автоматизации сборки.

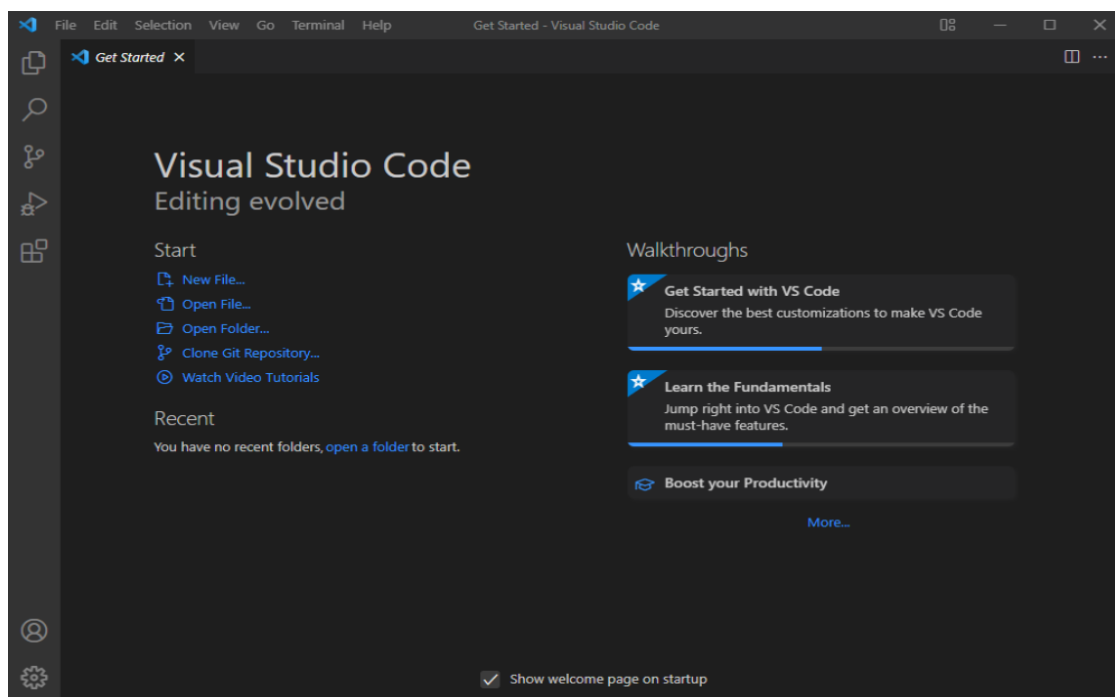
В качестве среды разработки был использован MS Visual Studio Code, он более удобен, чем Visual Studio и более понятен, чем PyCharm.

Для установки MS Visual Studio Code нужно:

1. Перейти на официальный сайт <https://code.visualstudio.com/Download> , используя любой веб-браузер[3]
2. Скачать установочный файл для своей операционной системы
3. Запустить установку приложения
4. После открытия окна установки будет предложено принять условия использования.



5. Далее нужно будет выбрать папку установки и данные местоположения файла для запуска. Жмите кнопки **NEXT**
6. Затем будет предложено начать установку. Нажмите на кнопку **INSTALL**
7. После нажатия на кнопку **INSTALL** будет произведена установка, она займет не более 1 минуты
8. После завершения установки установите флажок на **LAUNCH Visual Studio Code** и нажмите **ДАЛЕЕ**
9. Откроется окно, где вы сможете выбрать папку для начала работы.



10. По умолчанию программа использует английский язык. Чтобы перевести интерфейс на русский необходимо открыть палитру команд сочетанием клавиш **Ctrl + Shift + P**. Затем в появившемся поле ввести **Configure Display Language** и нажать **Enter**. В списке найти и выбрать русский язык и перезапустить приложение.

11. Для удобства работы с кодом можно установить автосохранение. Для этого надо перейти во вкладку **Файл** и поставить галочку напротив **Автосохранение**.

Более подробную информацию про MS Visual Studio Code можно получить, прочитав официальную документацию [2].

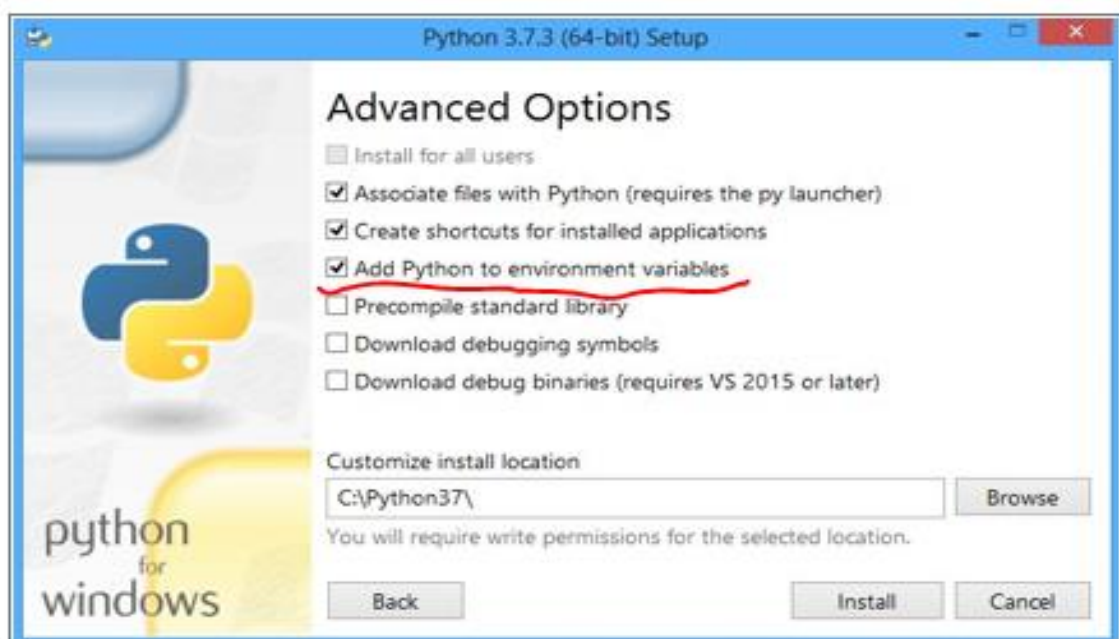
2.2. Установка интерпретатора Python 3

Для написания чат-бота мною был выбран язык программирования Python. Это интерпретируемый, объектно-ориентированный высокоуровневый язык программирования с динамической типизацией[5].

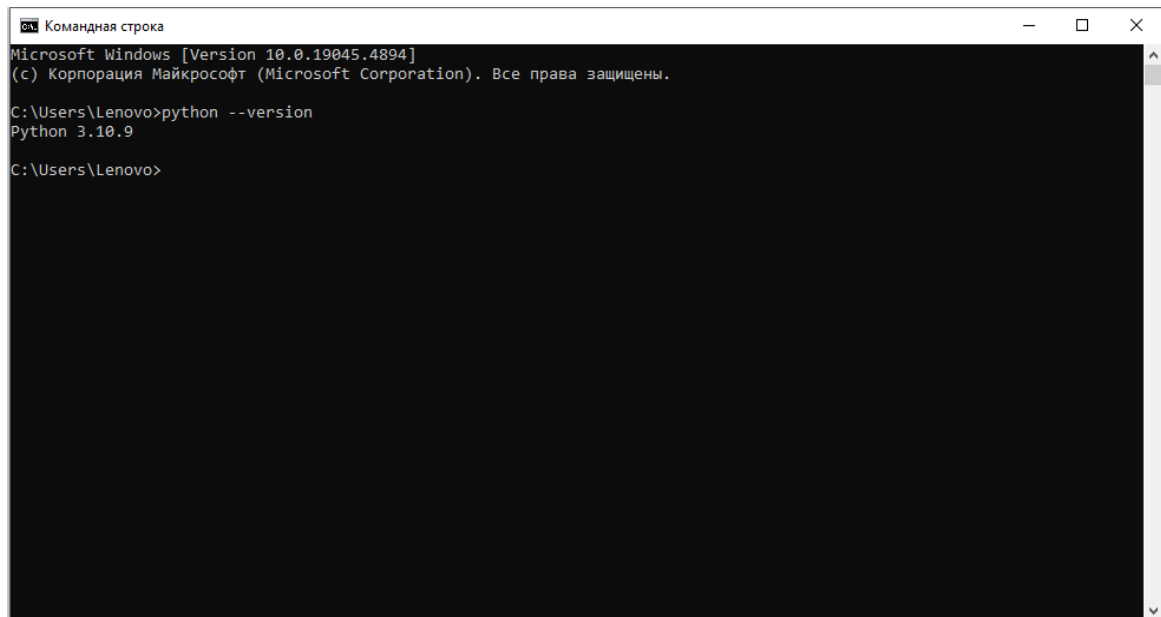
Преимущества языка Python является то, что он имеет простой понятный синтаксис и большую стандартную библиотеку переносимых функций. Также Python хорошо подходит для новичков.

Для того, чтобы установить Python 3 на свой компьютер необходимо:

1. Пройти по ссылке на официальный сайт и скачать нужную версию. Рекомендуется не использовать последнюю версию. Лучше всего подойдет версия 3.8.10 или 3.10.9[4].
2. Запустите установочный файл и следуйте инструкциям
3. Не забудьте отменить пункт **Add Python to environment variables**, чтобы не устанавливать переменные окружения вручную.



4. Версию интерпретатора можно проверить в командной строке **cmd** введя команду **python --version**



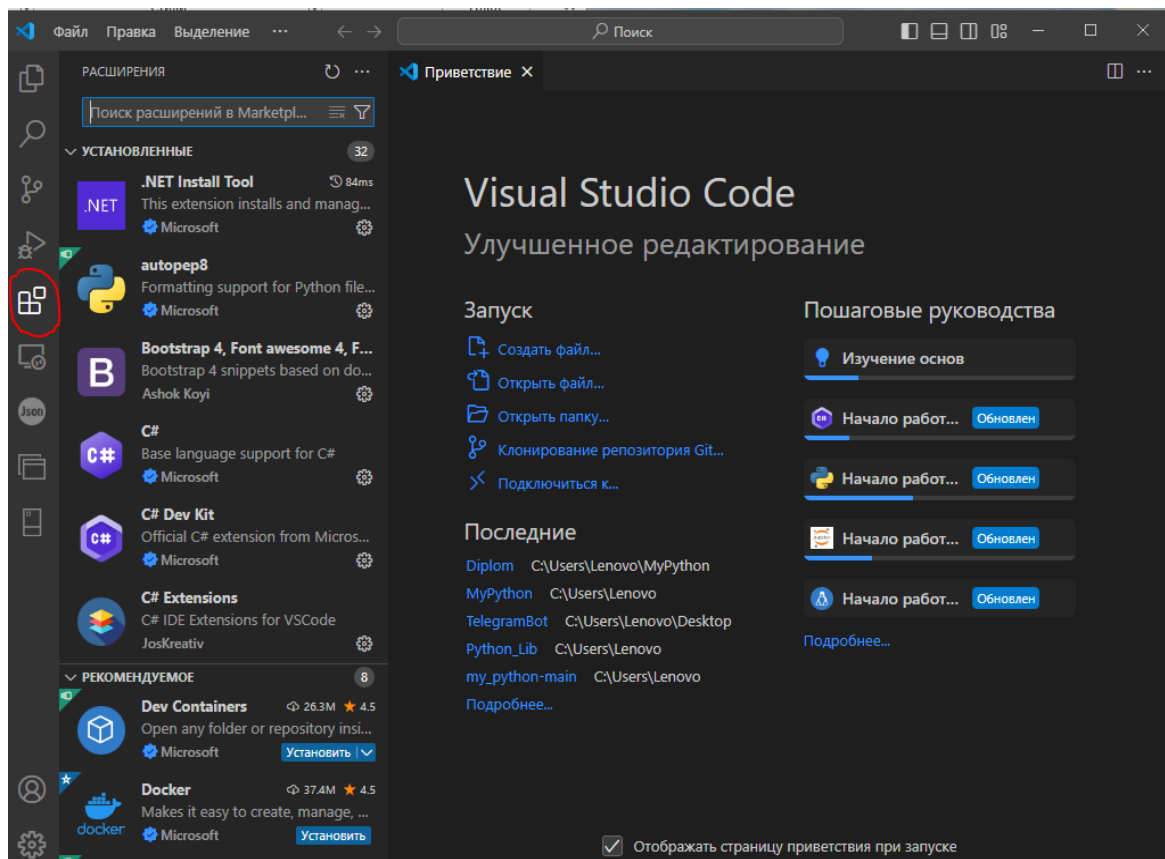
```
Командная строка
Microsoft Windows [Version 10.0.19045.4894]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\Users\Lenovo>python --version
Python 3.10.9

C:\Users\Lenovo>
```

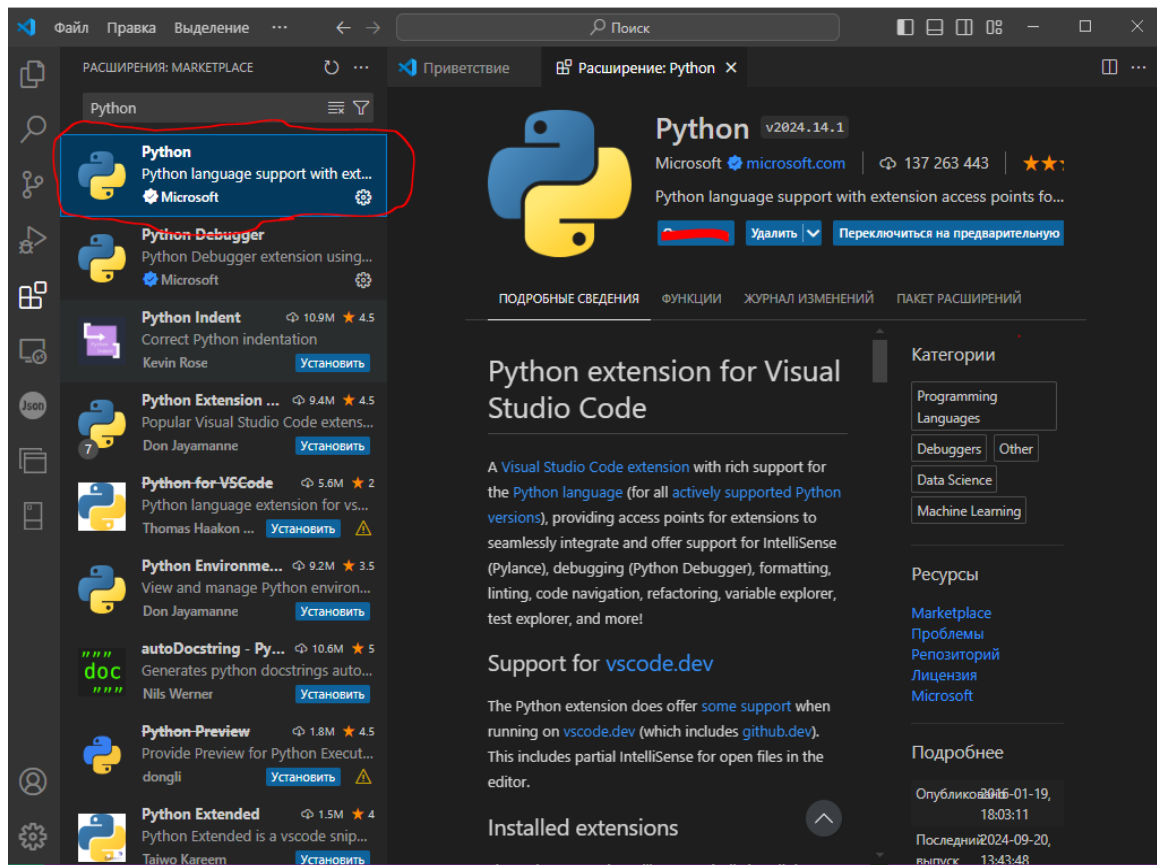
Чтобы установить расширение Python в Visual Studio Code необходимо:

1. Перейти во вкладку расширения[6].



2. В поле ввести Python и нажать **Enter**

3. Выбрать первый предложенный вариант (Он официальный от Microsoft) и нажать кнопку **Установить**



2.3. Установка необходимых python-пакетов.

Для успешной работы с текущей программой необходимо установить следующие библиотеки:

2.3.1. Pandas

Pandas — это программная библиотека на языке Python для обработки и анализа данных. Она предоставляет специальные структуры данных и операции для манипулирования числовыми таблицами и временными рядами.

Из этой библиотеки нам пригодится объект `DataFrame` для управления массивами двухмерных данных (проще говоря, таблицами)

```
pip install pandas
```

Подробнее про библиотеку Pandas можно прочитать в ее документации [14].

2.3.2. Python-dotenv

Python-dotenv считывает пары ключ-значение из `.env` файла и может устанавливать их как переменные окружения.

В своей программе я использую эту библиотеку для хранения токена на чат-бота и логина и пароля для авторизации на сайте.

Установить ее можно следующей командой в терминале:

```
pip install python-dotenv
```

2.3.3. Selenium

Selenium — это набор инструментов с открытым исходным кодом для тестирования веб-приложений, автоматизации работы браузеров и администрирования сайтов [15].

Он предоставляет расширения для эмуляции взаимодействия пользователя с браузерами, сервер распределения для масштабирования распределения браузеров и инфраструктуру для реализации спецификации W3C WebDriver , которая позволяет писать взаимозаменяемый код для всех основных веб-браузеров.

В основе Selenium лежит **WebDriver** — интерфейс для написания наборов инструкций, которые можно запускать попеременно во многих браузерах. После того, как вы все установили, всего несколько строк кода позволят вам войти в браузер.

Установить ее можно следующей командой в терминале

```
pip install selenium
```

2.3.4. pyTelegramBotAPI

Telebot – это популярная библиотека Python для создания ботов для мессенджера Telegram. Она предоставляет удобный и интуитивный API для взаимодействия с Telegram Bot API [16].

Основные возможности:

Простота в использовании: Telebot обеспечивает простой и лаконичный синтаксис, делая создание ботов доступным даже для новичков в программировании.

Поддержка всех типов сообщений: Обработка текстовых, голосовых, видео, фотографий, документов, стикеров, анимации и других типов сообщений.

Встроенные методы для отправки сообщений: Отправка текста, файлов, стикеров, клавиатур, кнопок и других элементов интерфейса.

Работа с группами и каналами: Возможность создавать ботов для групп и каналов, отправлять сообщения и получать информацию о пользователях.

Поддержка inline-режима: Создание интерактивных кнопок и форм в сообщениях.

Устанавливает также через терминал командой:

```
pip install pyTelegramBotAPI
```

2.4. Начало работы. Создание телеграм-бота и получение токена.

1. Первое, что нужно сделать – это найти в поиске Telegram BotFather (<http://t.me/BOTSpplus/288>) – это такой помощник, с помощью, которого можно создать своего бота [17].

2. Отправим команду /start и получим в ответ:

«I can help you create and manage Telegram bots»:

(Я могу помочь вам создавать и управлять ботами Telegram.)

«You can control me by sending these commands»:

(Вы можете управлять мной, написав эти команды):

/newbot - создать нового Бота.

/mybots - выбрать бота.

/setname - изменить имя бота.

/setdescription - изменить описание бота.

/setabouttext - изменить информацию о боте.

/setuserpic - изменить аватар бота.

/setcommands - изменить список команд бота.

/deletebot - удалить бота.

/token - сгенерировать новый токен.

/revoke - посмотреть старый токен.

/setinline - включить inline режим

/setinlinegeo - разрешение на определение местоположения GPS у

пользователей

/setinlinefeedback - изменить настройки inline

/setjoingroups - определяет, можно ли добавлять вашего бота в группы.

/setprivacy - определяет, все ли сообщения видит ваш бот в группах.

/mygames - ваши игры

/newgame - создать новую игру

/listgames - получить список ваших игр.

/editgame - редактировать игру.

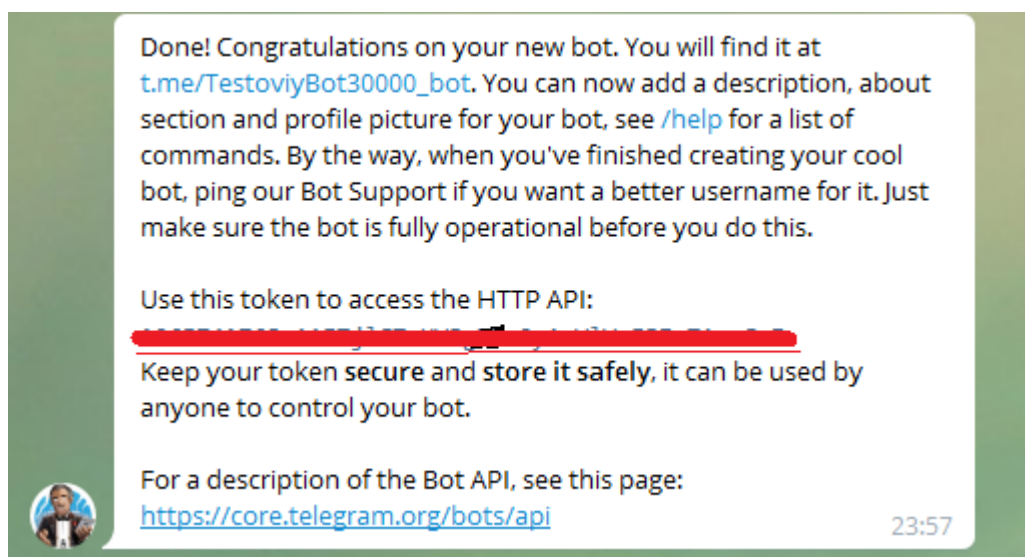
/deletegame - удалить вашу игру.

3. Выбираем **/newbot** и будем создавать новый бот

4. Далее задаем своему боту имя

5. Затем бот просит ввести имя пользователя (адрес, по которому ваше бота можно будет найти через поиск). Оно обязательно должно оканчиваться на **bot**

6. Ответом от бота будет сообщение:



Код под красной линией нужно сохранить, это и есть токен от бота.

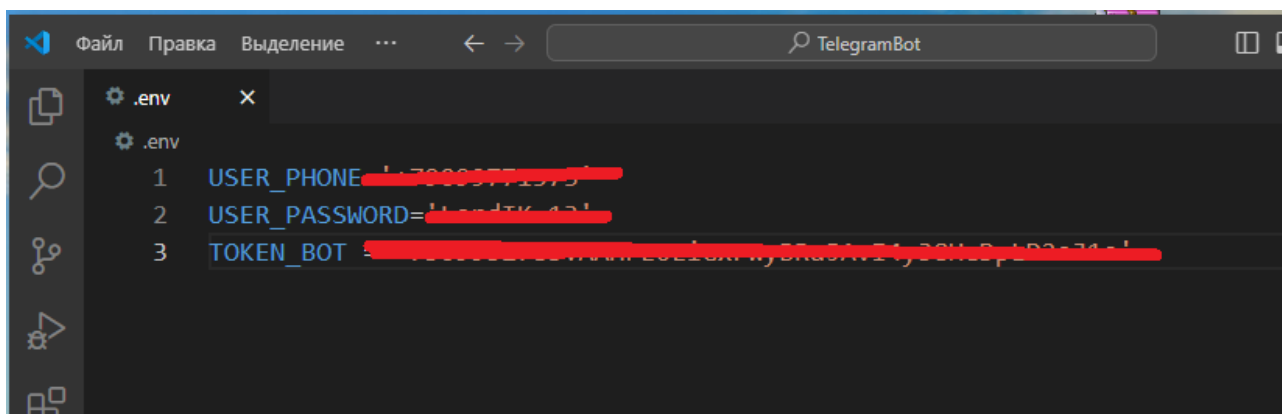
2.5. Написание кода

После установки среды разработки, окружения и необходимых библиотек мы создали образ телеграм бота и сейчас начнем наполнять его функционалом.

Для это в Visual Studio Code будем создавать необходимые файлы и наполнять их функционалом.

2.5.1. Заполнение переменных окружения.

Откроем окно VS Code. Во вкладке Проводник создадим файл “.env”



Используя переменные окружения укажите свой номер телефона **USER_PHONE** и пароль **PASSWORD** для доступа на сайт Liga Pro. А также укажите токен **TOKEN_BOT** от телеграм-бота.

2.5.2. Собираем информацию о турнирах на странице сайта.

Создаем файл **parser_tournament.py**, в котором будем собирать функции, нужные для получения списка турниров.

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
import time
import pandas as pd
import os
from dotenv import load_dotenv
```

Импортируем нужные библиотеки.

``from selenium import webdriver`` – Импортирует Selenium WebDriver, который позволяет программно управлять веб-браузером.

``from selenium.webdriver.common.by import By`` – импортирует методы для идентификации веб-элементов с помощью различных селекторов

``from selenium.webdriver.support.ui import WebDriverWait`` – импортирует класс ``WebDriverWait``, который позволяет вам ждать выполнения определенных условий на веб-странице перед взаимодействием с элементами.

``from selenium.webdriver.support import expected_conditions as EC`` – импортирует ожидаемые условия, которые можно использовать с ``WebDriverWait``, чтобы указать, чего вы ждете (например, элемент должен быть видимым, кликабельным и т. д.).

``import time`` – импортирует встроенный модуль ``time``, который позволяет добавлять задержки и контролировать время выполнения операций скрапинга.

``import pandas as pd`` – импортирует библиотеку Pandas.

``import os`` – Импортирует модуль ``os`` для взаимодействия с операционной системой

``from dotenv import load_dotenv`` – Импортирует функцию ``load_dotenv`` из библиотеки ``dotenv``, которая позволяет загружать переменные окружения из файла ``.env`` для хранения конфиденциальной информации вроде ключей API или паролей. системой, что полезно для операций с файловой системой.

```
URL='https://www.sport-liga.pro/ru/table-tennis/tournaments'
```

```
dotenv_path = os.path.join(os.path.dirname(__file__), '.env')
# загрузка нужных переменных с телефоном и паролем
if os.path.exists(dotenv_path):
    load_dotenv(dotenv_path)
    USER_PHONE = os.getenv("USER_PHONE")
    USER_PASSWORD = os.getenv("USER_PASSWORD")
```

Переменная URL содержит основной адрес сайта, с которым будем работать.

``dotenv_path = os.path.join(os.path.dirname(file), '.env')`` – Эта строка определяет путь к файлу ``.env``.

``os.path.join()`` складывает эту директорию с именем файла ``.env``, формируя полный путь.

``if os.path.exists(dotenv_path):`` – Проверяет, существует ли файл ``.env`` по указанному пути.

`load_dotenv(dotenv_path)`` – Если файл ``.env`` существует, эта строка загружает переменные окружения из него.

``USER_PHONE = os.getenv("USER_PHONE")`` – Используется функция ``os.getenv()`` для получения значения переменной ``USER_PHONE`` из загруженного файла ``.env``.

``USER_PASSWORD = os.getenv("USER_PASSWORD")`` – Аналогично, получает значение переменной ``USER_PASSWORD`` из файла ``.env``.

```
# авторизация и загрузка страницы
def authorisation(url):

    print('Идет загрузка данных')
    driver = webdriver.Chrome() # Используйте драйвер для вашего браузера

    driver.get(url)
    driver.maximize_window()

    login_button = WebDriverWait(driver, 10).until(
        EC.element_to_be_clickable((By.XPATH, '//header/div[2]/p')))
    login_button.click()

    # Переключаемся на вкладку "Телефон"
    phone_tab = WebDriverWait(driver, 10).until(
        EC.element_to_be_clickable((By.XPATH, '//form/div[1]/div[2]')))
    phone_tab.click()

    # Заполняем поле "Номер телефона"
    phone_field = driver.find_element(By.XPATH, '//form/div[2]/label/input')
    phone_field.send_keys(USER_PHONE)

    # Заполняем поле "Пароль"
    password_field = driver.find_element(By.XPATH, '//form/div[3]/label/input')
    password_field.send_keys(USER_PASSWORD)

    # Нажимаем кнопку "Войти"
    login_button = driver.find_element(By.XPATH, '//form/button[2]')
    login_button.click()

    # ожидаем загрузки страницы
    time.sleep(3)
    print('Страница загружена')
    return driver
```

Функция **authorisation()** выполняет авторизацию на сайте.

``driver = webdriver.Chrome()`` – создает объект driver для Google Chrome

``driver.get()`` – загружает рабочую страницу в браузер

``driver.maximize_window()`` – максимизирует окно браузера (разворачивает на полный экран)

``WebDriverWait(driver, 10).until(EC.element_to_be_clickable((By.XPATH, '//header/div[2]/p')))`` – ожидает появление кнопки Войти, чтобы авторизоваться на сайте.

``login_button.click()`` – и нажимает ее

Главной причиной выбора библиотеки Selenium для скрапинга данных с этого сайта стало то, что окно авторизации представляет собой всплывающее окно, и другие инструменты скрапинга просто не умеют с ним работать.

Далее ищем вкладку "Телефон" и ждем на нее

```
`phone_tab = WebDriverWait(driver, 10).until(
    EC.element_to_be_clickable((By.XPATH, '//form/div[1]/div[2]'))
phone_tab.click()
phone_field = driver.find_element(By.XPATH, '//form/div[2]/label/input')
```

`phone_field.send_keys(USER_PHONE)`` – подставляем свой номер телефона

Аналогичные манипуляции проводим с паролем.

```
`login_button = driver.find_element(By.XPATH, '//form/button[2]')
login_button.click()`
```

– снова находим кнопку Войти и кликаем на нее.

``time.sleep(3)`` – ожидаем 3 секунды для полной загрузки страницы

``return driver`` – функция возвращает объект драйвер, который по сути является открытой страницей сайта в браузере.

```
# получение списка всех турниров
def get_more_results(driver):
    more_results = driver.find_element(By.XPATH, '//div[1]/button[2]/div')
    while True:
        try:
            more_results.click()
            time.sleep(0.5)
            driver.execute_script('window.scrollTo(0,1000)')
            time.sleep(0.5)
            more_results = driver.find_element(By.XPATH, '//div[1]/button[2]/div')
            time.sleep(0.5)
        except Exception as ex:
            print(ex)
            break
```

Функция `get_more_results()` принимает на вход объект `driver`

`more_results = driver.find_element(By.XPATH, '//div[1]/button[2]/div')` – ищет элемент на странице с помощью Xpath который представляет кнопку "Показать еще".

`while True:` – создает цикл `while`, который будет продолжаться, пока не произойдет ошибка.

`try:` – начинает блок `try`, чтобы обрабатывать возможные исключения (например, элемент не найден).

`more_results.click()` – кликает на кнопку "Показать еще".

`time.sleep(0.5)` – делает паузу на 0.5 секунды, чтобы дать странице время загрузиться.

`driver.execute_script('window.scrollTo(0,1000)')` – использует JavaScript для прокрутки страницы вниз на 1000 пикселей.

``more_results = driver.find_element(By.XPATH, '//div[1]/button[2]/div')`` – повторно ищет кнопку "Показать еще", предполагая, что она может быть обновлена после загрузки новых результатов.

``except Exception as ex`` – этот блок кода выполняется, если произошла ошибка в блоке ``try``, то есть кнопка “Показать еще” не найдена. Цикл прерывается. Функция отработала.

```
# получение ссылок на турниры
def get_tournament_url_list(driver):
    results = driver.find_elements(By.XPATH, '//table/tbody/tr/td[2]/a')
    lst_url = [res.get_attribute("href") for res in results]
    return lst_url
```

Функция `get_tournament_url_list()` принимает на вход объект `driver` и возвращает список ссылок на турниры `lst_url`.

``driver.find_elements(By.XPATH, '//table/tbody/tr/td[2]/a')`` – ищет все элементы на странице которые соответствуют XPath выражению ``//table/tbody/tr/td[2]/a`` – все ссылки **a** внутри ячеек **td** с индексом 2 (**td[2]**), которые находятся в строках **tr**, Внутри тела таблицы **tbody** любой из таблиц на странице **table**.

Подробнее про XPath можно прочитать в документации [18, 20]

``lst_url = [res.get_attribute("href") for res in results]`` – эта строка собирает ссылки из найденных элементов и формирует список.

``res.get_attribute("href")`` – из каждого найденного элемента (`res`) извлекает атрибут "href", который содержит ссылку.

``[... for res in results]`` – генератор списков – создает список из элементов, полученных в результате цикла.

```

# получение списка участников турниров
def get_list_of_players(driver):
    button = driver.find_elements(By.XPATH, '//table/tbody/tr/td[4]')

    lst_players = []
    for i, but in enumerate(button):
        but.click()
        time.sleep(1.25)
        family = driver.find_elements(
            By.XPATH,
            f'//table/tbody/tr[{i+1}]/td[4]//table/tbody/tr/td[2]/a/div/p')

        lst_players.append([el.text for el in family])
    driver.find_element(By.XPATH, f'//table/tbody/tr[{i+1}]/td[4]').click()
    time.sleep(0.5)
    return lst_players

```

Функция `get_list_of_players()` получает на вход объект `driver` и возвращает список участников турнира `lst_players`

`'button = driver.find_elements(By.XPATH, '//table/tbody/tr/td[4]')` – эта строка находит все элементы HTML, соответствующие XPath-выражению `'//table/tbody/tr/td[4]'`. Эти элементы являются кнопками, которые раскрывают информацию о каждом турнире.

`'lst_players=[]'` – инициализируем пустой список

`'for i, but in enumerate(button)'` – цикл перебирает найденные кнопки (`button`) с помощью `enumerate()`, получая индекс каждую кнопку и саму кнопку.

`'but.click()'` – кликает на кнопку и открывает всплывающее окно с участниками турнира

```

'family = driver.find_elements(By.XPATH, f'//table/tbody/tr[{i+1}]/td[4]//table/tbody/tr/td[2]/a/div/p')

```

– ищет элементы (Фамилии участников), содержащиеся в динамически генерирующемся (с помощью индексов) XPath-выражении.

`'lst_players.append([el.text for el in family])'` – добавляем Фамилии игроков в список.

`'driver.find_element(By.XPATH, f'//table/tbody/tr[{i+1}]/td[4]').click()'` – повторно нажимает последнюю кнопку, чтобы всплывающее окно исчезло.

```
def start_parser_tournament(url, dat):

    dri = authorisation(url + f'?date_from={dat}')
    get_more_results(dri)
    lst_url = get_tournament_url_list(dri)
    lst_players = get_list_of_players(dri)

    data_res = pd.DataFrame()
    tours = dri.find_elements(By.XPATH, '//table/tbody/tr/td[2]/a')
    date = dri.find_elements(By.XPATH, '//table/tbody/tr/td[1]/a')
    status = dri.find_elements(By.XPATH, '//table/tbody/tr/td[5]')

    data_res['date'] = [el.get_attribute("text").split('.')[0] for el in date]
    data_res['time'] = [el.get_attribute("text").split(' ')[-1] for el in date]
    data_res['tables'] = [el.get_attribute("text").split('.')[0].split(' ')[1] for el in tours]
    data_res['liga'] = [el.get_attribute("text").split('.')[1] for el in tours]
    data_res['players'] = lst_players
    data_res['url'] = lst_url
    data_res['status'] = [el.text for el in status]

    print('Данные о турнирах загружены')

    data_res.to_csv(f'tours_in_{dat}.csv', index=False)
    print('Данные о турнирах записаны')
```

Функция **start_parser_tournament()** принимает на вход два аргумента **url** – адрес и **dat** – дату.

Функция обобщает работу всех предыдущих функций:

1. `'dri = authorisation(url + f'?date_from={dat}')` – получает объект driver по полному URL-адресу, которые включает в себя дату.
2. `'get_more_results(dri)'` – обрабатывает страницу, чтобы получить больше результатов
3. `'get_tournament_url_list(dri)'` – извлекает список URL-адресов всех турниров с веб-страницы.
4. `'get_list_of_players(dri)'` – извлекает список игроков, которые участвуют в этих турнирах.

`'data_res = pd.DataFrame()'` – создается пустой DataFrame, который будет хранить информацию о турнирах.

``tours = dri.find_elements(By.XPATH, '//table/tbody/tr/td[2]/a')`` – ищет и получает элементы с Названием турнира и Рейтинге участников

``date = dri.find_elements(By.XPATH, '//table/tbody/tr/td[1]/a')`` – ищет и получает элементы, которые соответствуют Дате и Времени проведения турнира

``status = dri.find_elements(By.XPATH, '//table/tbody/tr/td[5]')`` – получает элементы со Статусом турнира

Далее в `DataFrame()` добавляются столбцы **date, time, tables, liga, players, url и status** (Дата, Время, Название стола турнира, Рейтинг лиги, Список участников, Ссылка на турнир, Статус турнира)

``data_res.to_csv(f'tours_in_{dat}.csv', index=False)`` – данные записываются в CSV-файл с именем 'tours_in_{dat}.csv', где `dat` – выбранная дата.

Результатом работы этой функции и всего скрипта является датафрейм со списком турниров на выбранную дату.

2.5.3. Обработка вводимых данных

Создаем файл **processing.py**, в котором будем собирать функции, нужные для обработки вводимой информации.

Скрипт выполняет функции обработки данных в датафрейме и простые проверки на правильность ввода, поэтому особых дополнительных библиотек для его работы не нужно.

``import pandas as pd`` – импортируем только библиотеку Pandas

```
def processing_players(df):
# ОБРАБОТКА СТОЛБЦА С УЧАСТНИКАМИ
    players = []

    for i in range(len(df['players'])):
        full_name = []
        for j in range(len(df['players'][i].split(','))):
            full_name.append(df['players'][i].split(',')[j].strip(r"[/ '"] )
        players.append(full_name)

    players = [[fio.split(' ')[0] + ' ' + fio.split(' ')[1][0] + '.'
                + fio.split(' ')[2][0] + '.' for fio in player] for player in players]

    df['players'] = players
    return df
```

Функция `processing_players()` на выход получает датафрейм, проводит с ним манипуляции и возвращает все тот же датафрейм. Цель функции сократить полное имя игрока (Фамилия, имя, Отчество) до сокращенного (Фамилия И.О).

``players = []`` – инициализируется пустой список, куда будут добавляться игроки.

Через первый цикл **for** перебираем строки датафрейма, то есть каждый турнир. Второй цикл **for** проходится по списку участников турнира (в зависимости от даты и времени начала их может быть 4 или 5). Метод ``split(',')`` разделяет строку по запятой (отделяет полное имя каждого участника), а метод ``strip(r"[/ '"]`` через регулярное выражение [21, 22] убирает лишние пробелы, кавычки и скобки из каждого имени. Полученные данные записываются в список **full_name**.

```
`players = [[fio.split(' ')[0] + ' ' + fio.split(' ')[1][0] + ' ' + fio.split(' ')[2][0] + '.' for fio in player] for player in players]`
```

– используется list comprehension для перебора полных имен в списке. Из каждого имени извлекается фамилия, первая буква имени и первая буква отчества, и они соединяются в новое имя в формате «Фамилия И.О.»

``df['players'] = players`` – полученный список добавляется в датафрейм в столбец **players**.

```
def search_player(name, df):  
    ~  
    tours = pd.DataFrame()  
  
    for i in range(len(df['players'])):  
        for full_name in df['players'][i]:  
            if name.lower() == full_name.split(' ')[0].lower():  
                tours = tours.append(df.iloc[i], ignore_index=True)  
    return tours
```

Функция **search_player()** получает на вход фамилию игрока и датафрейм.

``tours = pd.DataFrame()`` – создается пустой датафрейм, в который будут добавляться данные о турнирах, где игрок принимает участие.

``for i in range(len(df['players'])):`` – цикл, проходящий по всем строкам в **df**, по колонке **players**.

``for full_name in df['players'][i]:`` – вложенный цикл, проходящий по всем именам игроков в строке **i** датафрейма.

``if name.lower() == full_name.split(' ')[0].lower():`` – условие, проверяющее, совпадает ли фамилия (преобразованное в нижний регистр) с первой частью имени из списка имен **full_name** (также преобразованное в нижний регистр).

``full_name.split(' ')[0]`` – извлекает первую фамилию из полного имени.

``tours = tours.append(df.iloc[i], ignore_index=True)`` - если условие истинно (имя игрока совпадает), то строка **i** из **df** добавляется в DataFrame **tours**.

```
def check_date(date_str):
    # Разделяем строку на год, месяц и день
    year, month, day = map(int, date_str.split('-'))
    # устанавливаем флаг корректности даты
    flag = True
    # Проверяем корректность года, месяца и дня
    if not (1 <= month <= 12):
        flag = False
    if not (1 <= day <= 31):
        flag = False
    # Проверяем корректность дня для месяцев с 30 днями
    if month in [4, 6, 9, 11] and day == 31:
        flag = False
    # Проверяем корректность дня для февраля
    if month == 2:
        if (year % 4 == 0 and (year % 100 != 0 or year % 400 == 0)) and day > 29:
            flag = False
        elif day > 28:
            flag = False
    return flag
```

Функция **check_date()** принимает на вход строку даты и возвращает флаг корректности проверки.

Дата разделяется на части (год, месяц, день), которые проверяются отдельно. В случае, если дата не соответствует одному из условий флаг корректности меняется на False (проверка не пройдена). Если же все условия были выполнены, флаг остается True (проверка пройдена).

2.5.4. Загрузка истории личных встреч.

Для работы создаем файл **parser_games.py**, в котором находят функции, необходимые для получения информации о личных встречах игроков

```
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
import time
import pandas as pd
import os
from dotenv import load_dotenv
from parser_tournaments import authorisation

# загрузка нужных переменных с телефоном и паролем
dotenv_path = os.path.join(os.path.dirname(__file__), '.env')
if os.path.exists(dotenv_path):
    load_dotenv(dotenv_path)
    USER_PHONE = os.getenv("USER_PHONE")
    USER_PASSWORD = os.getenv("USER_PASSWORD")
```

Для работы импортируем нужные библиотеки и получаем переменные окружения для работы.

`from parser_tournaments import authorisation` – импортируем ранее разобрannую функцию **authorisation()** для авторизации на сайте.

```
# получение url списка игр
def get_match_list_url(driver):
    match_list = WebDriverWait(driver, 10).until(
        EC.presence_of_all_elements_located((By.XPATH, '//div[2]/div[3]/div/div/div/div[1]/a')))
    match_list = match_list[:len(match_list)//2]
    return match_list
```

Функция **get_match_list_url()** аналогична ранее разобрannой функции **get_tournament_list_url()** и возвращает список элементов, содержащих ссылки на игры в турнире.

Полученные по XPath значения дублируются, поэтому обрезаем полученный список пополам `match_list[:len(match_list)//2]`.

``dri.execute_script("window.open(");")`` – используется JavaScript для открытия новой вкладке в браузере

``dri.switch_to.window(dri.window_handles[-1])`` - переключение на новую вкладку.

``dri.get(match_url)`` – переход на страницу по извлеченной ссылке.

В блоке try используется WebDriverWait с ожиданием в 10 секунд для поиска элемента с историями личных встреч и клик по этому элементу.

Открываемая страница содержит в себе строку вида **«Игрок1 счет_личных_встреч Игрок2»**, ее мы и ищем в следующем блоке

``WebDriverWait(dri, 3).until(EC.presence_of_element_located((By.XPATH, '//section/div[2]'))).find_element(By.XPATH, '//section/div[2]').text``.

Обработка исключений except. Если элемент с историей личных встреч не найден, данные заполняются вручную, игрокам присваивается счет 0:0.

Полученные результаты добавляются в список game.

``dri.close()`` –закрывает текущую вкладку

``dri.switch_to.window(dri.window_handles[0])`` – возвращает на изначальную вкладку.

Функция возвращает список game, содержащих историю личных встреч.

```

# # обработка информации по матчам
def processing_match_information(game):
    player_1 = []
    player_2 = []
    games = []
    sets = []

    df = pd.DataFrame()

    for el in game:
        pl_1 = el.split(' ')[3][0] + ' ' + el.split(' ')[3][1][:1] + '.' + \
            el.split(' ')[3][2][:1] + '.'
        player_1.append(pl_1)
        pl_2 = el.split(' ')[-3][0] + ' ' + el.split(' ')[-3][1][:1] + '.' + \
            el.split(' ')[-3][2][:1] + '.'
        player_2.append(pl_2)
        games.append(''.join(el.split(' ')[3:6]))
        sets.append(el.split(' ')[6])

    df['player_1'] = player_1
    df['player_2'] = player_2
    df['games'] = games
    df['sets'] = sets

    return df

```

Функция **processing_match_information()** принимает на вход список истории личных встреч и преобразует ее в датафрейм.

Создаются пустые списки **player_1**, **player_2**, **games**, **sets** и пустой датафрейм **df**.

Цикл **for** поэлементно проходит по каждому значению из входящего списка **game**. Разделяя его на части методами **split()** и используя срезы строк. Каждая часть добавляется в соответствующий список.

Затем созданный датафрейм заполняется столбцами и возвращается в результате работы функции.

```
def my_func(url):  
    # проходим авторизацию на сайте  
    driver = authorisation(url)  
    # получаем элементы с ссылками на игры турнира  
    h2h_games = get_match_list_url(driver)  
    # получаем историю личных встреч  
    games_info = get_match_infomation(h2h_games, driver)  
    # преобразуем данные в датафрейм  
    df = processing_match_information(games_info)  
  
    return df
```

Вспомогательная функция **my_func()** объединяет все функции для работы.

2.5.5. Основной код телеграмм-бота.

```
import re
import os
import telebot
from telebot import types
import pandas as pd
from datetime import datetime, timedelta
from dotenv import load_dotenv

from processing import search_player, processing_players, check_date

from parser_tournaments import start_parser_tournament

from parser_games import my_func
```

'import telebot' — библиотека инструментов для работы с API Telegram.

'from telebot import types' — импортирует класс **types** из библиотеки **telebot**.

'import re' — модуль **re** для работы с регулярными выражениями.

'from datetime import datetime, timedelta' — импортирует классы **datetime** и **timedelta** из модуля **datetime**, используется для работы с датами и временем.

'import os' — импортирует модуль **os** для работы с файловой системой.

'from dotenv import load_dotenv' — функция **load_dotenv** из библиотеки **dotenv** позволяет загружать переменные окружения из файла **.env**.

```
'from processing import search_player, processing_players, check_date'
```

```
'from parser_tournaments import start_parser_tournament'
```

```
'from parser_games import my_func'
```

— импортируем функции из вспомогательных модулей, которые разобрали выше

```
pd.options.display.max_colwidth = 100

URL = 'https://www.sport-liga.pro/ru/table-tennis/tournaments'
```

`pd.options.display.max_colwidth = 100` — настраивает вывод в консоль, а именно определяет ширину столбцов на 100 символов

`URL = 'https://www.sport-liga.pro/ru/table-tennis/tournaments'` — задаём основной URL-адрес для работы

```
# кнопки 1 этапа
def stage_1(message):
    markup = telebot.types.ReplyKeyboardMarkup(resize_keyboard=True)
    markup.row("Турниры сегодня", "Турниры завтра")
    markup.row("Выбрать дату")
    bot.send_message(message.chat.id,
                      "Привет! 🤖 Хотите посмотреть расписание турниров?",
                      reply_markup=markup)

# кнопки 2 этапа
def stage_2(message):
    markup = telebot.types.ReplyKeyboardMarkup(resize_keyboard=True)
    markup.add('Поиск по игроку', 'Поиск по параметрам', 'Завершить работу')
    bot.send_message(message.chat.id, "Выберите дальнейшее действие:",
                      reply_markup=markup)
```

Функция **stage_1(message)** отвечает за первый этап диалога.

Она создает клавиатуру с помощью

```
telebot.types.ReplyKeyboardMarkup(resize_keyboard=True)
```

`resize_keyboard=True` означает, что кнопки будут занимать всю ширину экрана.

Функция добавляет три кнопки в два ряда:

``markup.row("Турниры сегодня", "Турниры завтра")`` – добавляет первый ряд с кнопками "Турниры сегодня" и "Турниры завтра".

``markup.row("Выбрать дату")`` – добавляет второй ряд с кнопкой "Выбрать дату".

Затем функция отправляет приветственное сообщение пользователю с помощью ``bot.send_message`` вместе с созданной клавиатурой ``reply_markup=markup``.

Функция **stage_2(message)** отвечает за второй этап диалога. Она создает аналогичную клавиатуру только с тремя кнопками в один ряд:

``markup.add('Поиск по игроку', 'Поиск по параметрам', 'Завершить работу')``

– добавляет ряд с кнопками "Поиск по игроку", "Поиск по параметрам" и "Завершить работу".

```
# функция получения датафрейма по дате
def get_df(message, dat):
    global data
    filename = f'tours_in_{dat}.csv'
    try:
        data = pd.read_csv(filename)
    ~
    except Exception as ex:
        bot.send_message(message.chat.id,
                          "Выполняется загрузка данных с сайта")
        start_parser_tournament(URL, dat)
        data = pd.read_csv(filename)
        bot.send_message(message.chat.id, "Турниры загружены")

    bot.send_message(message.chat.id, "Обрабатываю данные")
    data = processing_players(data)
    stage_2(message)
```

Функция `get_df(message, dat)` принимает на вход входящее сообщение и дату.

``global data`` — объявляет, что переменная **data** является глобальной, то есть доступна из других частей кода (это главный датафрейм, с которым работает бот. Он представляет собой список турниров, проходящих в заданную дату)

``filename = f'tours_in_{dat}.csv`` — создает имя файла, которое будет использоваться для хранения информации о турнирах. (пока это будет csv-файл, в дальнейшем буду пробовать подключить базу данных).

``try`` — блок кода, который пытается выполнить чтение файла ``data = pd.read_csv(filename)``

``except`` — если файл не найден, срабатывает исключение, в чат бота отправляется сообщение о загрузке данных с сайта.

``start_parser_tournament(URL, dat)`` — запускает разобранную ранее функцию, которая получает данные о турнирах с указанного URL и сохраняет их в CSV-файл.

``data = pd.read_csv(filename)`` — снова происходит попытка прочесть данные из CSV-файла (теперь он точно есть) .

``bot.send_message(message.chat.id, "Турниры загружены")`` — отправляет сообщение в чат бота о завершении загрузки.

``bot.send_message(message.chat.id, "Обрабатываю данные")`` — отправляет сообщение в чат о начале обработки данных.

``data = processing_players(data)`` — происходит обработка информации о игрока (сокращение полного имени до фамилии и инициалов),

``stage_2(message)`` — запускает функцию **stage_2**, которая переводит бота на 2 этап работы.

```

# функция обработки ввода даты
def check_date_input(message):
    dat = message.text
    global date
    try:
        dat = re.split(r'[.,/]', message.text)
        if len(dat[2]) == 2:
            dat[2] = '20' + dat[2]
        if len(dat[1]) == 1:
            dat[1] = '0' + dat[1]
        if len(dat[0]) == 1:
            dat[0] = '0' + dat[0]
        selected_date = f'{dat[2]}-{dat[1]}-{dat[0]}'
        flag = check_date(selected_date)
        if flag:
            date = selected_date
            bot.send_message(message.chat.id, f"Дата '{date}' успешно сохранена")
            get_df(message, selected_date)

    except Exception:
        bot.send_message(message.chat.id,
                          'Неверный формат даты. \
                          Пожалуйста, введите дату в формате ДД-ММ-ГГГГ:')
        bot.register_next_step_handler(message, check_date_input)

```

Функция **check_date_input(message)** — проверяет вводимую дату и приводит её к нужному виду

`dat = message.text` — получает текст сообщения от пользователя.

`global date` — объявляет, что переменная **date** является глобальной (она тоже нужна для работы всего бота).

`try...except` — блок кода, который обрабатывает возможные ошибки ввода даты.

`dat = re.split(r'[.,/]', message.text)` — используя регулярное выражение, разбивает введенный текст на части, используя разделители ",", ".", "/".

При введении данных может не использоваться 0 перед числом, если значение даты и месяца меньше 10. Дальнейший код проверяет такую ситуацию и добавляет 0 в начало. Также он дополняет значение года до полного, если в нем 2 цифры

``selected_date = f'{dat[2]}-{dat[1]}-{dat[0]}'`` – форматирует полученные данные в дату в формате "ГГГГ-ММ-ДД". Именно в таком порядке дата используется в ссылках на сайте. Напомню, пользователь вводит дату в привычном для него виде ДД-ММ-ГГГГ.

``check_date(selected_date)`` — функция проверяет на правильно значений в дате и возвращает флаг True или False

``date = selected_date`` — если все проверки пройдены, вводимое значение даты сохраняется в глобальную переменную date.

```
`bot.send_message(message.chat.id, f'Дата '{date}' успешно сохранена")`
```

— бот отправляет сообщение в чат о успешном сохранении даты.

``get_df(message, selected_date)`` — вызывается функция **get_df** для получения данных о турнирах за выбранную дату.

``except Exception as ex`` — обрабатывает ошибки ввода. Отправляет сообщение в чат о неверном формате даты и просит повторить ввод.

``bot.register_next_step_handler(message, check_date_input)`` — бот регистрирует шаг, который будет вызываться при следующем сообщении пользователя для повторного ввода даты.

```

# загрузка токена для бота
dotenv_path = os.path.join(os.path.dirname(__file__), '.env')
if os.path.exists(dotenv_path):
    load_dotenv(dotenv_path)
    TOKEN_BOT = os.getenv("TOKEN_BOT")

# Токен бота
bot = telebot.TeleBot(TOKEN_BOT)

# НАЧАЛО РАБОТЫ БОТА. обработка /start
@bot.message_handler(commands=['start'])
def start(message):
    stage_1(message)

```

```

`dotenv_path = os.path.join(os.path.dirname(file), '.env')
if os.path.exists(dotenv_path):
    load_dotenv(dotenv_path)
    TOKEN_BOT = os.getenv("TOKEN_BOT")`

```

— уже знакомая загрузка токена бота

`bot = telebot.TeleBot(TOKEN_BOT)` — создание объекта бота и передача в него нашего токена

`@bot.message_handler(commands=['start'])` — обработка команды **/start**

— представляет собой декоратор, который указывает, что функция **start()** должна быть вызвана при получении сообщения с командой **/start**.

`def start(message)` — сама функция **start** вызывает другую функцию **stage_1**, которая отправляет приветственное сообщение пользователю и запускает 1 этап общения

```

# Обработка ответов из 1 этапа
@bot.message_handler(
    func=lambda message: message.text in ["Турниры сегодня", "Турниры завтра",
                                           "Выбрать дату"])
def tournaments_today(message):
    global date

    if message.text == "Турниры сегодня":
        today = datetime.now().strftime('%Y-%m-%d')
        date = today
        bot.send_message(message.chat.id, f"Дата '{date}' успешно сохранена",
                           get_df(message, today))

    elif message.text == "Турниры завтра":
        tomorrow = (datetime.now() + timedelta(days=1)).strftime("%Y-%m-%d")
        date = tomorrow
        bot.send_message(message.chat.id,
                           f"Дата '{tomorrow}' успешно сохранена")
        bot.send_message(message.chat.id, get_df(message, tomorrow))

    elif message.text == "Выбрать дату":
        bot.send_message(message.chat.id, "Введите дату в формате ДД-ММ-ГГГГ:")
        bot.register_next_step_handler(message, check_date_input)
    else:
        bot.send_message(message.chat.id, "Неверный выбор.")

# обработка кнопки "Поиск по игроку"
@bot.message_handler(func=lambda message: message.text == 'Поиск по игроку')
def search_by_player(message):
    # Получение имени игрока от пользователя
    bot.send_message(message.chat.id, "Введите фамилию игрока:")
    bot.register_next_step_handler(message, process_player_search)

```

Этот код реализует обработку ответов на 1 этапе в боте, связанном с турнирами. Давайте разберемся, как он работает:

`@bot.message_handler(func=lambda message: message.text in ["Турниры сегодня", "Турниры завтра", "Выбрать дату"])` — этот декоратор определяет функцию **tournaments_today** как обработчик для сообщений, содержащих три указанных варианта ответа.

``func=lambda message: message.text in ...`` — это лямбда-функция [ссылка] , которая проверяет, находится ли текст сообщения в списке вариантов.
``global date`` — здесь тоже используется глобальная переменная даты

Теперь разберём каждый ответ отдельно:

``if message.text == "Турниры сегодня"`` — если выбрано это значение, переменная **date** получает текущую дату в формате YYYY-MM-DD с помощью ``datetime.now().strftime('%Y-%m-%d')`` и отправляет сообщение, подтверждающее выбранную дату, и вызывает функцию ``get_df(message, today)`` для получения и отправки информации о турнирах.

``elif message.text == "Турниры завтра"`` — если выбран этот вариант. В переменную **date** записывается завтрашнее число с помощью

```
`datetime.now() + timedelta(days=1)`.
```

Также отправляется сообщение с подтверждением выбора даты и вызывает функция ``get_df(message, tomorrow)``.

``elif message.text == "Выбрать дату"`` — если выбран этот вариант, пользователю отправляется сообщение с просьбой ввести дату в формате ДД-ММ-ГГГГ. Используется ``bot.register_next_step_handler(message, check_date_input)`` для регистрации функции **check_date_input** как обработчика для следующего сообщения от пользователя.

``else`` — если же пользователь ввёл другое сообщение, ему будет отправлено сообщение об ошибке.

На втором этапе пользователю предлагается выбрать удобный для него поиск.

```

# обработка кнопки "Поиск по игроку"
@bot.message_handler(func=lambda message: message.text == 'Поиск по игроку')
def search_by_player(message):
    # Получение имени игрока от пользователя
    bot.send_message(message.chat.id, "Введите фамилию игрока:")
    bot.register_next_step_handler(message, process_player_search)

# функция запуска поиска по фамилии
def process_player_search(message):
    global data
    # Получение имени игрока из сообщения
    player_name = message.text

    tours = search_player(player_name, data)
    if not tours.empty:
        for i, row in tours.iterrows():
            url_tours = row['url']
            keyboard = [[types.InlineKeyboardButton(
                "Перейти на страницу турнира", url=url_tours)]]
            reply_markup = types.InlineKeyboardMarkup(keyboard)

            row.drop('url', inplace=True)
            bot.send_message(message.chat.id, row.to_string(),
                             reply_markup=reply_markup)
    else:
        bot.send_message(message.chat.id, 'Такой игрок в эту дату не играет')

    stage_2(message)

```

Обработка команды "Поиск по игроку"

```
`@bot.message_handler(func=lambda message: message.text == 'Поиск по
игроку')`
```

– декоратор, который определяет, что функция **search_by_player** будет вызвана, когда пользователь вводит команду "Поиск по игроку".

``bot.send_message(message.chat.id, "Введите фамилию игрока:")`` – отправляет пользователю сообщение с просьбой ввести фамилию игрока.

``bot.register_next_step_handler(message, process_player_search)`` – регистрирует следующую функцию **process_player_search**, которая будет вызвана после получения фамилии игрока.

Функция поиска по фамилии **process_player_search(message)**

``global data`` – объявляет, что функция использует глобальную переменную **data**, которая хранит информацию о турнирах

``player_name = message.text`` – получает введенную пользователем фамилию игрока.

``tours = search_player(player_name, data)`` – вызывается функция **search_player**, которая должна найти информацию о турнирах, в которых участвует этот игрок.

``if not tours.empty:`` – проверяет, найдены ли турниры.

Если турниры найдены:

``for i, row in tours.iterrows():`` – перебирает найденные строки данных.

``url_tours = row['url']`` – получает URL турнира из строки данных.

Создается кнопка "Перейти на страницу турнира" с ссылкой на URL турнира.

``row.drop('url', inplace=True)`` – удаляет столбец **'url'** из строки данных, чтобы не отображать его в сообщении.

``bot.send_message(message.chat.id, row.to_string(), reply_markup=reply_markup)`` – отправляет сообщение с информацией о турнире и кнопкой.

Если турниры не найдены:

``bot.send_message(message.chat.id, 'Такой игрок в эту дату не играет')``

– отправляет сообщение, что игрок не найден.

``stage_2(message)`` – вызывается функция **stage_2**, которая, вероятно, является частью второго этапа работы бота.

Переходим к 3 этапу. Разбираем кнопки

```
# кнопки 3 этапа
def stage_3(message):
    markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
    itembtn1 = types.KeyboardButton('Выбрать стол')
    itembtn2 = types.KeyboardButton('Выбрать время')
    itembtn3 = types.KeyboardButton('Начать поиск')
    itembtn4 = types.KeyboardButton('Вернуться назад')
    markup.add(itembtn1, itembtn2, itembtn3)
    markup.add(itembtn4)

    bot.send_message(message.chat.id, "Что вы хотите сделать?",
                      reply_markup=markup)
```

`def stage_3(message):` — определяет функцию `stage_3`, которая принимает объект `message` (сообщение от пользователя).

`markup = types.ReplyKeyboardMarkup(resize_keyboard=True)` — создает объект **markup** типа **types.ReplyKeyboardMarkup**. Это объект, который представляет собой клавиатуру с кнопками, которая будет отправлена пользователю.

Аргумент **resize_keyboard=True** указывает, что клавиатура должна автоматически подстраиваться под размер экрана.

`itembtn1 = types.KeyboardButton('Выбрать стол')` — создает кнопку с текстом "Выбрать стол".

`itembtn2 = types.KeyboardButton('Выбрать время')` — создает кнопку с текстом "Выбрать время".

`itembtn3 = types.KeyboardButton('Начать поиск')` — создает кнопку с текстом "Начать поиск".

`itembtn4 = types.KeyboardButton('Вернуться назад')` — создает кнопку с текстом "Вернуться назад".

``markup.add(itembtn1, itembtn2, itembtn3)`` — добавляет первые три кнопки на клавиатуру.

``markup.add(itembtn4)`` — добавляет кнопку "Вернуться назад" на отдельную строку клавиатуры.

``bot.send_message(message.chat.id, "Что вы хотите сделать?", reply_markup=markup)`` — отправляет сообщение пользователю с текстом "Что вы хотите сделать?" и прикрепляет к нему созданную клавиатуру с четырьмя кнопками.

```
# Данные для кнопок
tables = ['A3', 'A4', 'A5', 'A6', 'A9', 'A15']
times = ["07:30", "07:45", "08:00", "09:30", "11:30", "11:45", "12:00",
         "14:00",
         "15:30", "15:45", "16:00", "16:45", "18:30", "19:30", "19:45",
         "20:00",
         "20:30", "21:00", "23:30", "23:45"]

# Хранение выбранных опций
selected_tables = {}
selected_time = {}
value_tables = []
value_time = []

# обработка кнопки "Поиск по параметрам"
@bot.message_handler(
    func=lambda message: message.text == 'Поиск по параметрам')
def search_parameters(message):
    stage_3(message)
```

Список **tables** содержит в себе данные для кнопок поиска по названию стола в турнире.

Список **time** — содержит возможное время начала турниров.

Словари **selected_tables** и **selected_time** будут использоваться для временного хранения активированных кнопок. Ключи словаря ссылаются на кнопки, а значения словаря представляют собой значения списков.

Список **value_tables** нужен для окончательной фиксации выбранных значений названий турниров (фиксация происходит кнопкой "ОК" на выпадающей клавиатуре).

value_time – аналогичный список, только для фиксации времени

`@bot.message_handler(func=lambda message: message.text == 'Поиск по параметрам')` – декоратор, регистрирует ответ пользователя «Поиск по параметрам», оборачивает функцию **search_parameters(message)**, которая запускает клавиатуру для 3 этапа общения с ботом

```
# обработка ответа "Выбрать стол"
@bot.message_handler(func=lambda message: message.text == 'Выбрать стол')
def select_table(message):
    markup = types.InlineKeyboardMarkup()
    buttons = []
    for table in tables:
        buttons.append(types.InlineKeyboardButton(table, callback_data=table))
    markup.add(*buttons[:3])
    markup.add(*buttons[3:])

    markup.add(types.InlineKeyboardButton("ОК", callback_data="ok_tables"))
    bot.send_message(message.chat.id, "Выберите кнопки:", reply_markup=markup)

# обработка ответа "Выбрать время"
@bot.message_handler(func=lambda message: message.text == 'Выбрать время')
def select_time(message):
    markup = types.InlineKeyboardMarkup(row_width=3)

    for i in range(0, len(times), 3):
        row = []
        for j in range(i, min(i + 3, len(times))):
            row.append(
                types.InlineKeyboardButton(times[j], callback_data=times[j])
            )
        markup.row(*row)

    markup.add(types.InlineKeyboardButton("ОК", callback_data="ok_times"))
    bot.send_message(message.chat.id, "Выберите кнопки:", reply_markup=markup)
```

Первая функция обрабатывает ответ «Выбрать стол»

```
`@bot.message_handler(func=lambda message: message.text == 'Выбрать стол')`
```

– декоратор, который обрабатывает сообщения, содержащие текст "Выбрать стол".

``select_table(message)`` – функция, которая будет вызвана при обработке сообщения.

В теле функции создается объект выпадающей клавиатуры Inline [23]

```
`markup = types.InlineKeyboardMarkup()`
```

``buttons = []`` – создается пустой список, в который будут добавляться кнопки.

``for table in tables:`` – цикл, который добавляет кнопку для каждого стола из списка ``tables``.

``buttons.append(types.InlineKeyboardButton(table, callback_data=table))`` – создается кнопка с текстом ``table`` и данными **table**, которые будут переданы при нажатии на кнопку.

``markup.add(buttons[:3])`` – добавляются первые 3 кнопки на клавиатуру.

``markup.add(buttons[3:])`` – добавляются оставшиеся кнопки на клавиатуру.

```
`markup.add(types.InlineKeyboardButton("OK", callback_data="ok_tables"))`
```

– добавляется кнопка "OK" с данными **ok_tables**, которая будет использоваться для подтверждения выбора.

```
`bot.send_message(message.chat.id, "Выберите кнопки:", reply_markup=markup)`
```

– отправляется сообщение с клавиатурой.

Обработка команды "Выбрать время":

```
@bot.message_handler(func=lambda message: message.text == 'Выбрать время')
```

– декоратор, который обрабатывает сообщения, содержащие текст "Выбрать время".

```
select_time(message) – функция, которая будет вызвана при обработке сообщения.
```

```
markup = types.InlineKeyboardMarkup(row_width=3) – создается объект inline-клавиатуры с 3 кнопками в строке.
```

```
for i in range(0, len(times), 3): – цикл, который итерирует по списку times с шагом 3.
```

```
row = [] – создается список для хранения кнопок в строке.
```

```
for j in range(i, min(i + 3, len(times))): – внутренний цикл, который добавляет 3 кнопки в строку.
```

```
row.append(types.InlineKeyboardButton(times[j], callback_data=times[j]))
```

– создается кнопка с текстом `times[j]` и данными `times[j]`.

```
markup.row(row) – добавляется строка кнопок на клавиатуру.
```

```
markup.add(types.InlineKeyboardButton("OK", callback_data="ok_times")) – добавляется кнопка "OK" с данными ok_times, которая будет использоваться для подтверждения выбора.
```

```
bot.send_message(message.chat.id, "Выберите кнопки:", reply_markup=markup)
```

– отправляется сообщение с клавиатурой.


```

# обработка нажатия на кнопки в выборе стола
@bot.callback_query_handler(func=lambda call: call.data in tables)
def table_callback(call):
    global selected_tables
    if call.data in selected_tables:
        selected_tables[call.data] = False
    else:
        selected_tables[call.data] = True

    # Обновляем текст кнопки
    button_text = call.data
    if selected_tables[call.data]:
        button_text += "  "
    else:
        button_text = call.data

    # Обновляем инлайн клавиатуру
    markup = types.InlineKeyboardMarkup()
    buttons = []
    for el in tables:
        button = types.InlineKeyboardButton(el, callback_data=el)
        if el in selected_tables:
            button.text += "  "
        buttons.append(button)

    markup.add(*buttons[:3])
    markup.add(*buttons[3:])

    markup.add(types.InlineKeyboardButton("ОК", callback_data="ok_tables"))

    # Обновляем сообщение с инлайн клавиатурой
    bot.edit_message_text(
        chat_id=call.message.chat.id,
        message_id=call.message.message_id,
        text="Выберите кнопки:",
        reply_markup=markup)

```

Этот код реализует обработку нажатия кнопок в инлайн клавиатуре для выбора стола в боте Telegram.

```
`@bot.callback_query_handler(func=lambda call: call.data in tables)`
```


– декоратор, позволяющий боту реагировать на действия пользователя с инлайн-клавиатурой. В данном случае обрабатываются только те события, где

значение **call.data** (данные, которые были переданы при нажатии кнопки) присутствует в списке **tables**.

``def table_callback(call):`` – эта функция вызывается при нажатии на кнопку в инлайн клавиатуре.

``global selected_tables`` – получаем доступ к глобальной переменной **selected_tables**, которая, хранит информацию о нажатых кнопках.

``if call.data in selected_tables:`` – проверяет, была ли кнопка, на которую нажали, уже выбрана. Если кнопка была ранее выбрана и уже содержится в списке, то ей устанавливает значение False. Если же кнопки в списке нет, то ей устанавливается значение True.

Дальше обновляется текст кнопки. Когда кнопка была выбрана рядом с названием появляется символ "  ".

``button_text = call.data`` –сохраняет текст кнопки в переменную **button_text**
``if selected_tables[call.data]:`` – если кнопка выбрана, к её тексту добавляется галочка. В противном случае текст кнопки остается без изменений.

Затем клавиатуру надо обновить.

``markup = types.InlineKeyboardMarkup()`` – создает объект

InlineKeyboardMarkup, который будет использоваться для построения новой инлайн клавиатуры.

``buttons = []`` – создает пустой список, в котором будут храниться кнопки для новой клавиатуры.

``for el in tables:`` – перебирает все элементы из списка **tables**.

``button = types.InlineKeyboardButton(el, callback_data=el)`` – создает кнопку с текстом **el** и данными **el**, которые будут переданы при нажатии на кнопку.

``if el in selected_tables:`` – проверяет, была ли кнопка выбрана. Если да, то к её тексту добавляется символ "✓".

``buttons.append(button)`` – добавляет созданную кнопку в список **buttons**.

``markup.add(buttons[:3])`` – добавляет первые три кнопки из списка **buttons** в первую строку клавиатуры.

``markup.add(buttons[3:])`` – добавляет остальные кнопки из списка **buttons** во вторую строку клавиатуры.

``markup.add(types.InlineKeyboardButton("OK", callback_data="ok_tables"))`` – добавляет кнопку "OK" с данными **ok_tables**, которая, вероятно, будет использоваться для подтверждения выбора столов.

``bot.edit_message_text(chat_id=call.message.chat.id, message_id=call.message.message_id, text="Выберите кнопки:", reply_markup=markup)`` – обновляет сообщение, отправленное ботом ранее, с новой инлайн клавиатурой, где:

``chat_id`` – идентификатор чата, в котором находится сообщение.

``message_id`` – идентификатор сообщения, которое нужно обновить.

``text`` – новый текст сообщения.

``reply_markup`` – обновленная инлайн клавиатура.

```
# обработка нажатия ОК в выборе стола
@bot.callback_query_handler(func=lambda call: call.data == "ok_tables")
def ok_table_callback(call):
    global selected_tables
    global value_tables

    selected_buttons = []
    for button, state in selected_tables.items():
        if state:
            selected_buttons.append(button)
    value_tables = list(selected_tables.keys())
    bot.send_message(call.message.chat.id,
                     f"Выбраны кнопки: {' '.join(selected_buttons)}")
    selected_tables = {}

    stage_3(call.message)
```

Этот код представляет собой обработчик для нажатия кнопки "ОК" в контексте выбора стола.

`@bot.callback_query_handler` – этот декоратор указывает, что функция **ok_table_callback** будет вызываться при получении ответа "ok_tables" от бота

Функция `ok_table_callback(call)`:

`global selected_tables` и `global value_tables` – эти строки объявляют, что функция будет работать с глобальными переменными **selected_tables** и **value_tables**.

`selected_buttons = []` – создается пустой список, в который будут добавлены названия выбранных столов.

`for button, state in selected_tables.items():` – цикл перебирает элементы словаря **selected_tables**, где **button** - ключ словаря (название стола), а **state** - значение словаря (булево значение, показывающее, выбран стол или нет).

``if state:`` – если стол выбран (значение **state** равно ``True``), то:

``selected_buttons.append(button)`` – добавляется название стола в список **selected_buttons**.

``value_tables = list(selected_tables.keys())`` – в список **value_tables** сохраняются все ключи (названия столов) из словаря **selected_tables**.

``bot.send_message(call.message.chat.id, f'Выбраны кнопки: {'`
``.join(selected_buttons)}')`` – отправляется сообщение пользователю в чат с указанием выбранных столов.

``selected_tables = {}`` – очищается словарь **selected_tables**.

``stage_3(call.message)`` – вызывается функция **stage_3**.

Следующая часть кода реализует обработку нажатия кнопок выбора времени. Она аналогична функции по выбору стола, поэтому разъяснять ее еще раз не буду. Подробнее остановлюсь только на обновлении инлайн клавиатуры

```
markup = types.InlineKeyboardMarkup()

for i in range(0, len(times), 3):
    row = []
    for j in range(i, min(i + 3, len(times))):
        button = types.InlineKeyboardButton(times[j],
                                              callback_data=times[j])
        if times[j] in selected_time and selected_time[times[j]]:
            button.text += " ✓"
        row.append(button)
    markup.add(*row)

markup.add(types.InlineKeyboardButton("OK", callback_data="ok_times"))
```


Цикл ``for i in range(0, len(times), 3):`` – перебирает список **times** с шагом 3.

Это нужно для того, чтобы записать ряды с кнопками по 3 штуки в каждом.

`row = []` – создает пустой список для хранения кнопок в строке.

Внутренний цикл `for j in range(i, min(i + 3, len(times))):` создает 3 кнопки в каждой строке клавиатуры.

`types.InlineKeyboardButton(times[j], callback_data=times[j])` – создает кнопку с текстом **times[j]** (время начала турнира) и данными обратного вызова (**callback_data**), также равными **times[j]**.

`if times[j] in selected_time and selected_time[times[j]]:` – проверяет, был ли выбрана кнопка с временем. Если да, то к тексту кнопки добавляется символ "  ".
".

`row.append(button)` – добавляет созданную кнопку в список **row**.

`markup.add(row)` – эта строка добавляет все кнопки из списка **row** на клавиатуру.

`markup.add(types.InlineKeyboardButton("OK", callback_data="ok_times"))` – та строка добавляет кнопку "OK" с данными обратного вызова **ok_times** на клавиатуру.

```

# обработка ответа "Начать поиск"
@bot.message_handler(func=lambda message: message.text == 'Начать поиск')
def search_by_parameters(message):
    """ """

    global value_tables
    global value_time
    global data
    global ut
    ut = []
    bot.send_message(
        message.chat.id,
        f"Вы выбрали параметры: {value_time} и {value_tables}"
    )

    filename = f'tours_in_{date}.csv'
    data = pd.read_csv(filename)
    processing_players(data)
    df = data.loc[
        data['tables'].isin(value_tables) & data['time'].isin(value_time)]
    if not df.empty:
        num = 1
        for i, row in df.iterrows():
            url_tours = row['url']
            ut.append(url_tours)

            keyboard = [[types.InlineKeyboardButton(
                "Перейти на страницу турнира", url=url_tours)],
                [types.InlineKeyboardButton(
                    "Посмотреть статистику игроков",
                    callback_data=f"stats_players_{num}")]]

            reply_markup = types.InlineKeyboardMarkup(keyboard)

            row.drop('url', inplace=True)
            bot.send_message(message.chat.id, f'{row.to_string()}',
                             reply_markup=reply_markup)
            num += 1
        else:
            bot.send_message(message.chat.id,
                             'С такими параметрами турниры не найдены')
    stage_2(message)

```

Этот код является частью бота Telegram, который помогает искать турниры по определенным параметрам.

`@bot.message_handler(func=lambda message: message.text == 'Начать поиск')` – эта строка определяет обработчик сообщений, который срабатывает только тогда, когда пользователь отправляет сообщение "Начать поиск".

`def search_by_parameters(message):` – эта строка определяет функцию **search_by_parameters**, которая будет вызвана, когда пользователь отправляет сообщение "Начать поиск".

`global value_tables, value_time, data, ut` – объявляются глобальные переменные, доступные из других частей кода.

`ut = []` – эта строка инициализирует пустой список.

`bot.send_message(message.chat.id, f'Вы выбрали параметры: {value_time} и {value_tables}')` – отправляется пользователю сообщение с подтверждением выбранных параметров.

`filename = f'tours_in_{date}.csv'` – создание имени файла, содержащего информацию о турнирах.

`data = pd.read_csv(filename)` – чтение информации о турнирах из файла CSV с помощью библиотеки **pandas**.

`processing_players(data)` – вызов функции **processing_players** с аргументом **data**.

`df = data.loc[data['tables'].isin(value_tables) & data['time'].isin(value_time)]`

– фильтрация датафрейма по выбранным параметрам **value_tables** и **value_time**.

`data.loc` – метод для фильтрации DataFrame по условиям.

``data['tables'].isin(value_tables)`` – условие, проверяющее, присутствует ли стол турнира в списке выбранных столов.

``data['time'].isin(value_time)`` – условие, проверяющее, присутствует ли время проведения турнира в списке выбранных времен.

``if not df.empty:`` – эта строка проверяет, есть ли результат фильтрации датафрейма.

``num = 1`` – объявляется счетчик для нумерации турниров

``for i, row in df.iterrows():`` – цикл проходит по каждой строке **row** в отфильтрованном датафрейме **df**.

``url_tours = row['url']`` – извлечение URL турнира из строки **row**.

``ut.append(url_tours)`` – добавляет URL турнира в список **ut**.

``keyboard = [[types.InlineKeyboardButton("Перейти на страницу турнира", url=url_tours)], [types.InlineKeyboardButton("Посмотреть статистику игроков", callback_data=f"stats_players_{num}")]]`` – эта строка кода создает инлайн клавиатуру с двумя кнопками:

"Перейти на страницу турнира": переводит пользователя на страницу турнира по полученному URL-адресу.

"Посмотреть статистику игроков": вызывает обработчик **stats_players** с аргументом **num**.

``reply_markup = types.InlineKeyboardMarkup(keyboard)`` – сохраняет созданную клавиатуру в переменную ``reply_markup``.

``row.drop('url', inplace=True)`` – удаляет столбец **url** из текущей строки датафрейма.

``bot.send_message(message.chat.id, f'{row.to_string()}',
reply_markup=reply_markup)`` – отправляет в чат сообщение с содержимым строки датафрейма **row.to_string()**.

К сообщению прикрепляет **reply_markup**, т.е. созданную клавиатуру с кнопками.

``num += 1`` –увеличивает счетчик ``num`` на 1, чтобы в следующей итерации цикла нумерация кнопок была корректной.

``else:`` – выполняется, если ``df`` пустой, т.е. турниры не найдены. Отправляет сообщение в чат, что турниры не найдены.

``stage_2(message)`` – вызывает функцию **stage_2**, бот возвращается на второй этап работы

```

@bot.callback_query_handler(
    func=lambda call: call.data.startswith("stats_players_"))
def stat_players(call):
    index = int(call.data.split('_')[2]) - 1
    bot.send_message(call.message.chat.id, 'Ожидайте загрузки')

    name = f'{ut[index].split("/")[-1]}'
    try:
        data1 = pd.read_csv(f'{name}.csv')

    except Exception:

        data1 = my_func(ut[index])
        data1.to_csv(f'{name}.csv', index=False)

    bot.send_message(call.message.chat.id,
        f'{data1.to_string(index=False, header=True, col_space=15)}')

```

Этот код обрабатывает статистику игроков.

`@bot.callback_query_handler(func=lambda call: call.data.startswith("stats_players_"))` – декоратор указывает, что функция **stat_players** будет вызвана при получении ответа, начинающегося с "stats_players_".

Функция **stat_players(call)** получает статистику личных встреч игр в выбранном турнире.

`index = int(call.data.split('_')[2]) - 1` – эта строка извлекает индекс игрока из ответа пользователя. Он определяет для какого именно турнира пользователь хочет получить статистику игр.

`bot.send_message(call.message.chat.id, 'Ожидайте загрузки')` – отправляет сообщение в чат пользователя, уведомляя его о том, что данные загружаются.

``name = f {ut[index].split("/")[-1]}`` – формирует имя файла, в котором будет храниться история личных встреч. Название представляет собой порядковый номер турнира, взятый из ссылки на этот турнир.

``data1 = pd.read_csv(f {name}.csv)`` – пытается прочитать данные из CSV-файла, соответствующего имени игрока.

``except`` – обрабатывает исключение, если файл не найден. Изначально программа должна идти по этому пути.

В этом случае функция вызывается **`my_func(ut[index])`**, которая принимает на выход ссылку из списка под нужным индексом. Эту функцию мы разбирали выше, она нужна чтобы получить данные игрока и записать их в CSV-файл.

``bot.send_message(call.message.chat.id, f {data1.to_string(index=False, header=True, col_space=15)})`` – отправляет в чат пользователя отформатированную таблицу с данными игрока.

```
# обработка кнопки "Вернуться назад"
@bot.message_handler(func=lambda message: message.text == 'Вернуться назад')
def go_back(message):
    stage_2(message)

# обработка кнопки "Завершить работу"
@bot.message_handler(func=lambda message: message.text == 'Завершить работу')
def end_session(message):
    bot.send_message(message.chat.id, "Работа завершена!")
    stage_1(message)

bot.polling(none_stop=True)
```

Заключительная часть кода, которая обрабатывает ответы «Вернуться назад» и «Завершить работу».

```
`@bot.message_handler(func=lambda message: message.text == 'Вернуться назад')`
```

– эта строка является декоратором, который определяет обработчик для сообщений, содержащих текст "Вернуться назад".

```
`def go_back(message):`
```

– функция, возвращает бота ко второму этапу общения с пользователем. На этом этапе пользователю предлагается выбрать варианты поиска.

```
`@bot.message_handler(func=lambda message: message.text == 'Завершить работу')`
```

– аналогичный декоратор, как в предыдущем блоке, который определяет обработчик для сообщений с текстом "Завершить работу".

В функции **end_session()** содержит в себе действия при нажатии пользователем кнопки «Завершить работу»:

```
`bot.send_message(message.chat.id, "Работа завершена!")`
```

– отправляет сообщение с текстом "Работа завершена!" пользователю.

```
`stage_1(message):`
```

– переход бота на 1 этап работы, а именно на приветствие и предложения выбора даты.

```
`bot.polling(none_stop=True)`
```

– этот метод запускает бота в режиме постоянного опроса Telegram API на предмет новых сообщений.

`none_stop=True` – параметр, указывающий, что бот должен продолжать работу даже при возникновении ошибок.

Заключение

Разработанный в рамках дипломной работы чат-бот Telegram для спортивного сайта успешно реализовал поставленную цель – предоставить пользователям удобный инструмент для получения актуальной спортивной информации.

Чат-бот предоставляет пользователям возможность оперативно получать данные о результатах матчей, расписании спортивных событий, турнирных таблицах, а также искать информацию о спортсменах, включая их статистику и даты предстоящих выступлений.

Разработка бота с использованием Telegram API позволила интегрировать его в экосистему популярного мессенджера, что обеспечивает максимальную доступность и удобство использования.

В процессе разработки были реализованы следующие ключевые функции:

- **Автоматический поиск и обработка данных.** Чат-бот способен самостоятельно собирать информацию из различных источников, включая официальные сайты спортивных организаций, новостные ресурсы и специализированные сервисы.
- **Интуитивный интерфейс.** Чат-бот взаимодействует с пользователями через простые и понятные команды, что делает его доступным для широкого круга пользователей, независимо от их технических навыков.

В будущем планируется дальнейшее развитие чат-бота, включая:

- **Расширение функциональности** – будет добавлена более расширенная статистика по играм, что даст возможность сделать прогноз на предстоящий матч. Также будут добавлены карточки игрока, что даст пользователю возможность посмотреть информацию о нем.
- **Улучшение точности и скорости работы** – код бота будет улучшаться. Некоторые медленные функции делаться быстрее.
- **Размещение чат-бота на удаленном сервере Amvera** – даст возможность использовать чат-бот постоянно, не зависимо от запуска кода на личном десктопном устройстве.

Разработка чат-бота Telegram для спортивного сайта является актуальной задачей, поскольку современные пользователи все чаще предпочитают получать информацию в удобном и доступном формате. Данный проект продемонстрировал потенциал использования чат-ботов для эффективного распространения спортивной информации и повышения вовлеченности пользователей в мир спорта.

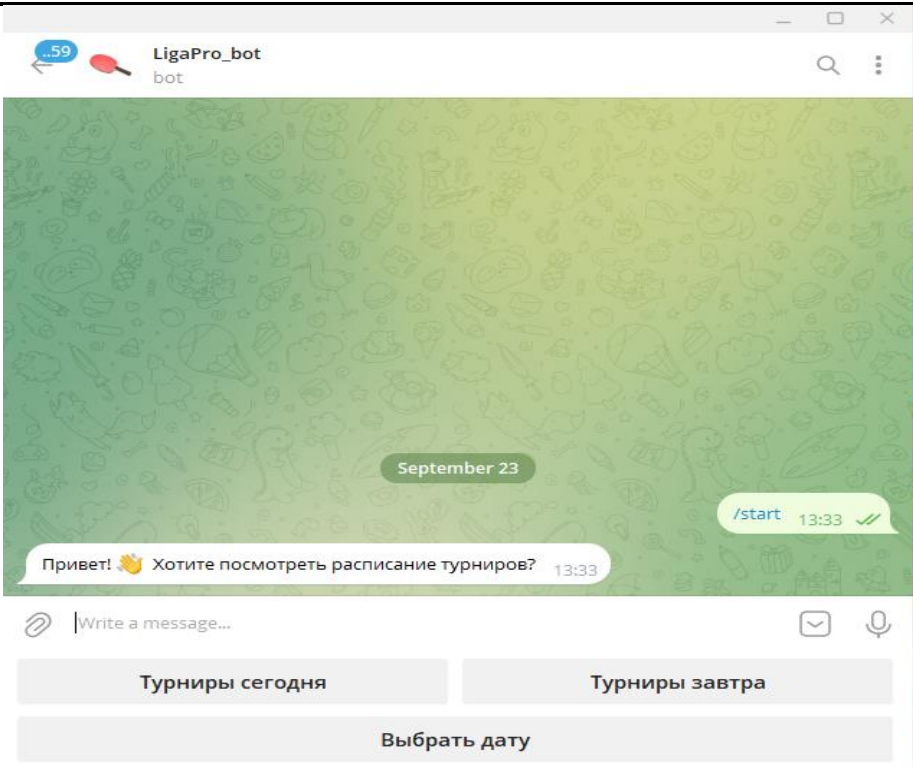
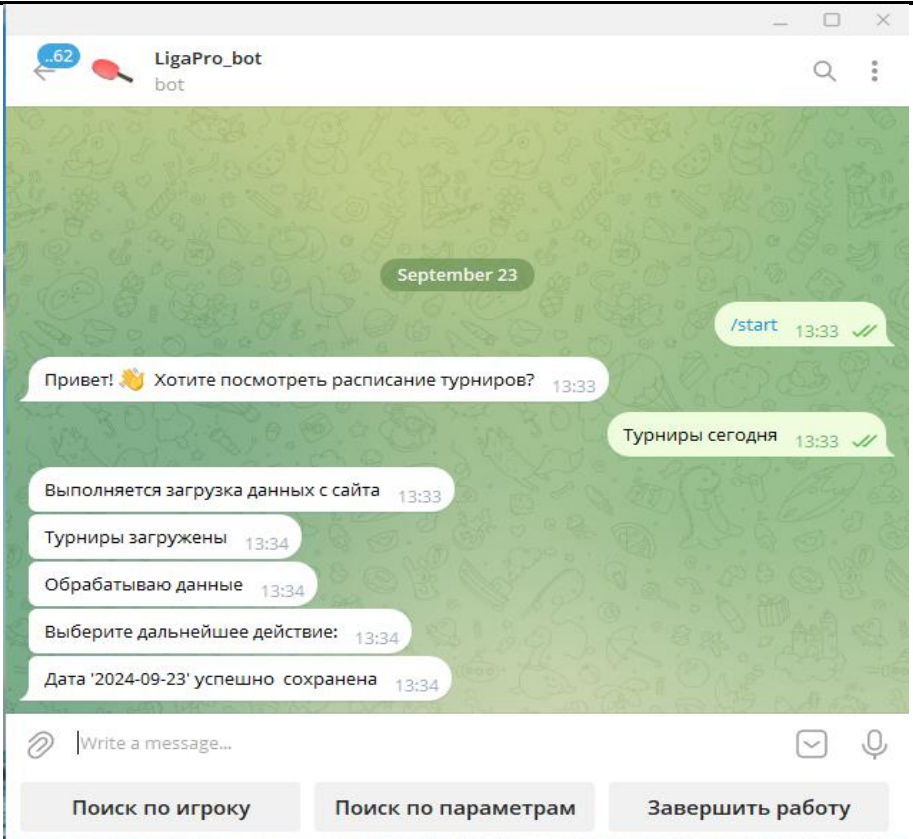
Список литературы

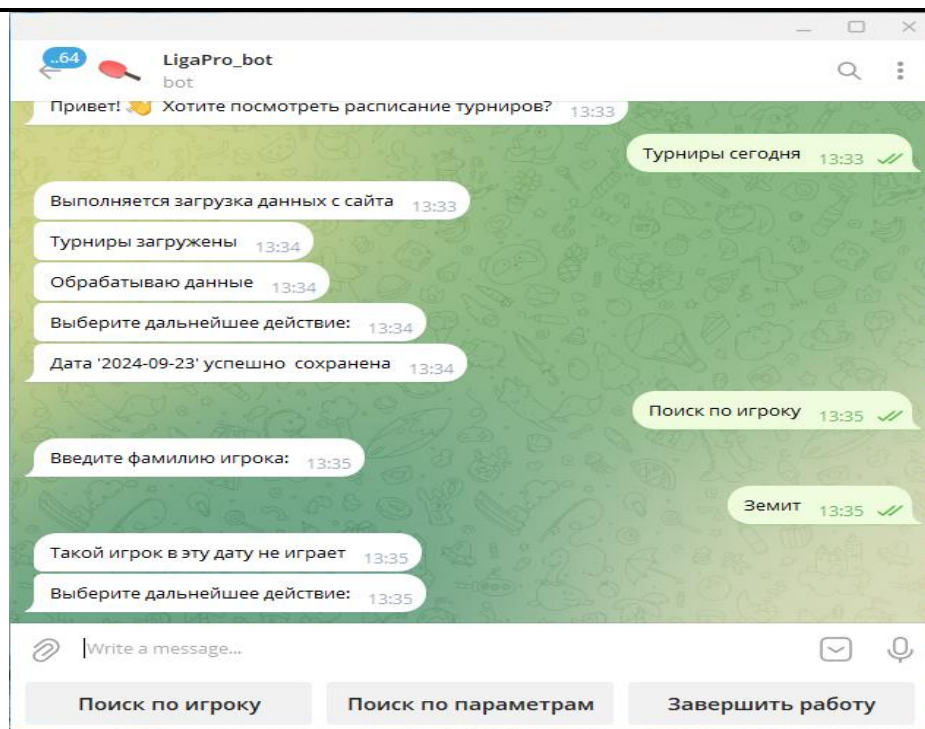
1. https://ru.wikipedia.org/wiki/Visual_Studio_Code «Википедия. Visual Studio Code»
2. <https://code.visualstudio.com/docs> «Документация Visual Studio Code»
3. <https://code.visualstudio.com/download/> «Сайт загрузки Visual Studio Code»
4. <https://www.python.org/downloads/> «Сайт загрузки интерпретатора Python»
5. <https://ru.wikipedia.org/wiki/Python> «Википедия. Язык программирования Python»
6. <https://habr.com/ru/articles/490754/> «Редактор кода Visual Studio Code. Самый подробный гайд по настройке и установке плагинов для начинающих»
7. <https://vc.ru/social/818851-7-prichin-sozdat-telegram-bota-dlya-vashei-kompanii> «7 причин создать Telegram-бота для вашей компании»
8. <https://www.microsoft.com/ru-ru/microsoft-365/word?market=ru> «Все об MS Office»
9. <https://botcreators.ru/blog/telegram-chat-bota/> «Telegram — как платформа для чат бота. Стоит ли его использовать?»
10. <https://habr.com/ru/companies/otus/articles/759368/> «Взгляд на телеграм-ботов изнутри»
11. <https://umnico.com/ru/blog/chatbot-for-telegram/> «Чат-бот в Telegram: что важно знать и как создать»

12. <https://telegram.org/> «Официальный сайт мессенджера Telegram»
13. <https://www.sports.ru/football/blogs/3204841.html> «Блог о компании Лига Про»
14. <https://pandas.pydata.org/docs/> «Документация библиотеки Pandas»
15. <https://www.selenium.dev/documentation/> «Документация библиотеки Selenium»
16. <https://pytba.readthedocs.io/ru/latest/index.html> «Документация библиотеки pyTelegramBotAPI»
17. <http://t.me/BOTSplus/288> «Ссылка на BotFather»
18. <https://ru.wikipedia.org/wiki/XPath> «Википедия. Язык разметки XPath»
19. https://www.geeksforgeeks.org/find_element_by_xpath-driver-method-selenium-python/ «find_element(By.XPATH) driver method – Selenium Python»
20. <https://habr.com/ru/companies/otus/articles/533354/> «Функции XPath»
21. https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B3%D1%83%D0%B%D1%8F%D1%80%D0%BD%D1%8B%D0%B5_%D0%B2%D1%8B%D1%80%D0%B0%D0%B6%D0%B5%D0%BD%D0%B8%D1%8F
«Регулярные выражения»
22. <https://devanych.ru/technologies/shpargalka-po-regulyarnym-vyrazheniyam>
«Шпаргалка по регулярным выражениям»
23. <https://habr.com/ru/articles/775630/> «Клавиатуры в pyTelegramBotAPI»

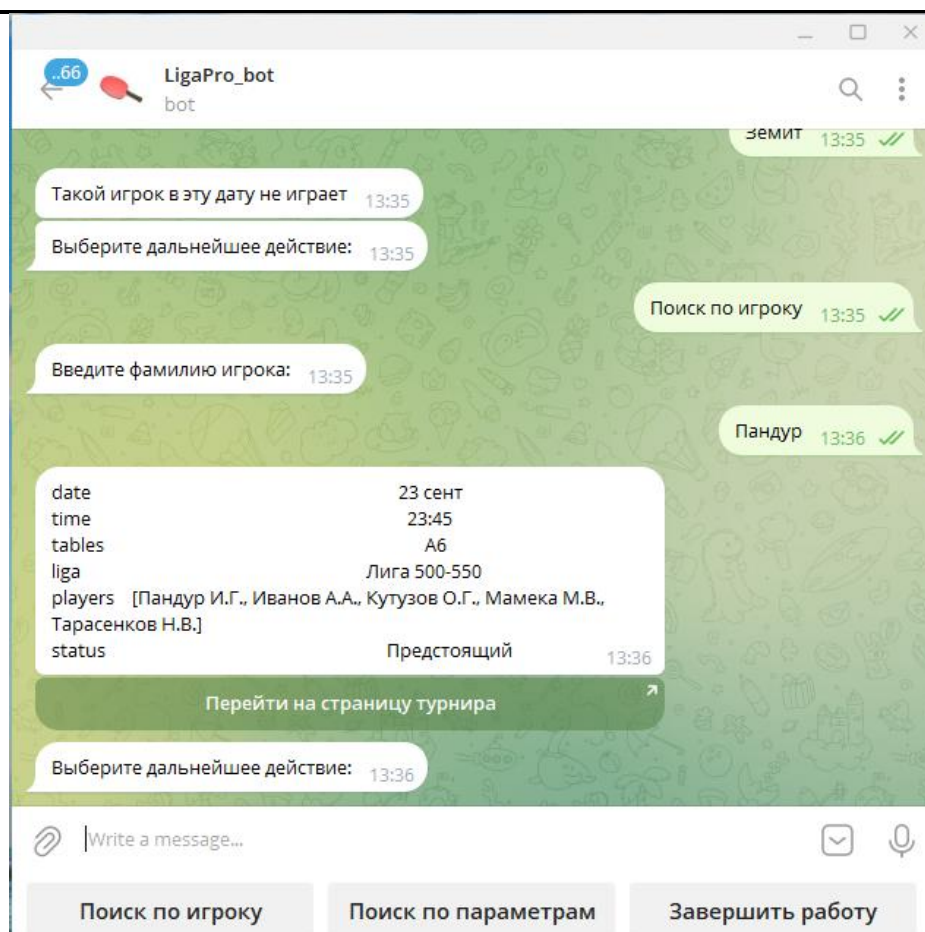
Приложение

Примеры использования чат-бота.

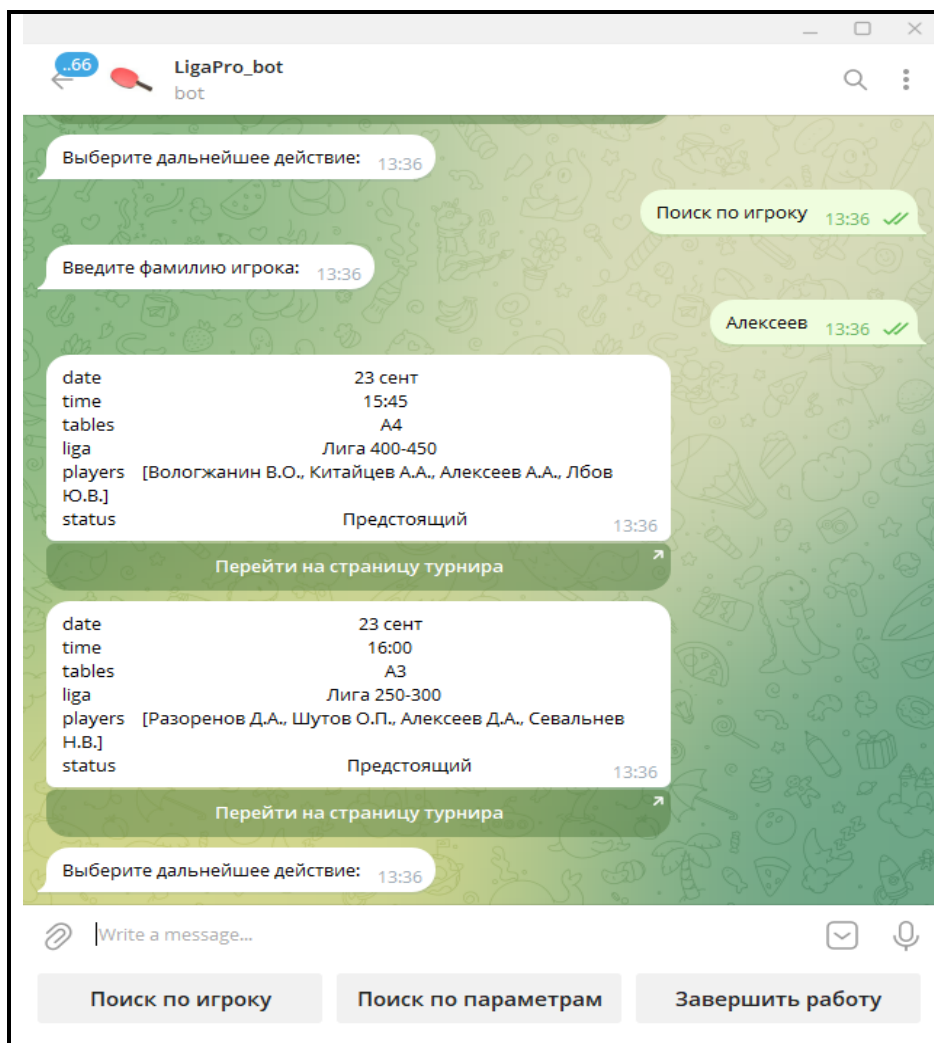
	<p>Начало работы.</p> <p>Приветствие пользователя и первый выбор кнопок</p>
	<p>Нажатие кнопки «Турниры сегодня».</p> <p>Получение ответа от бота.</p> <p>Загрузка данных с сайта занимает около 50 секунд.</p> <p>Переход ко второму этапу общения:</p> <p>предложение пользователю совершить поиск</p>



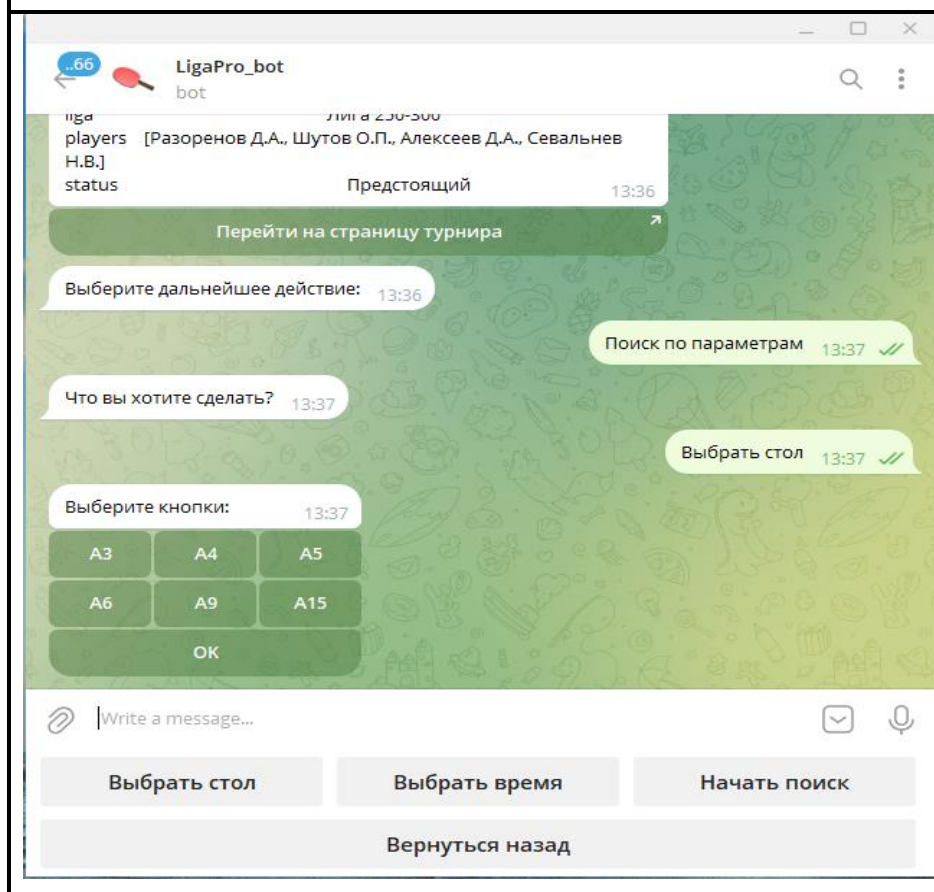
Производится
поиск по игроку:
пользователь
вводит фамилию и
получает ответ, что
«Игрок в эту дату
не играет»



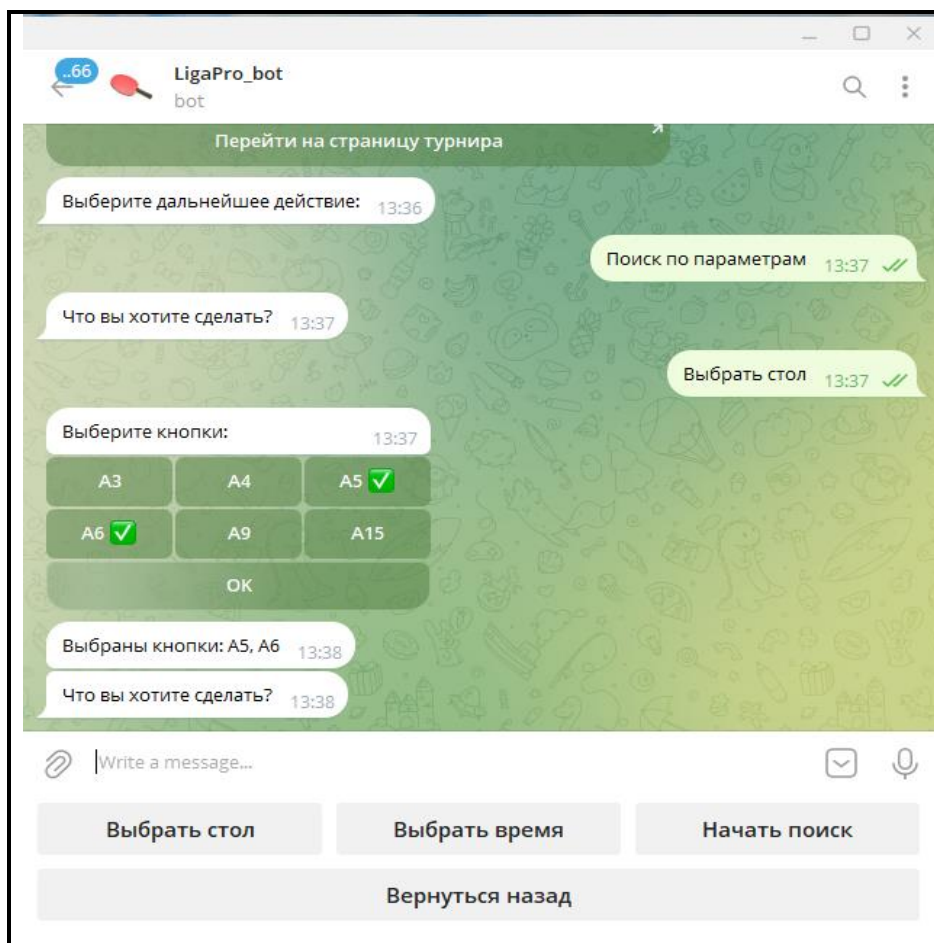
Производится
поиск по игроку:
пользователь
вводит фамилию и
получает в ответ
один турнир, в
котором играет
указанный игрок



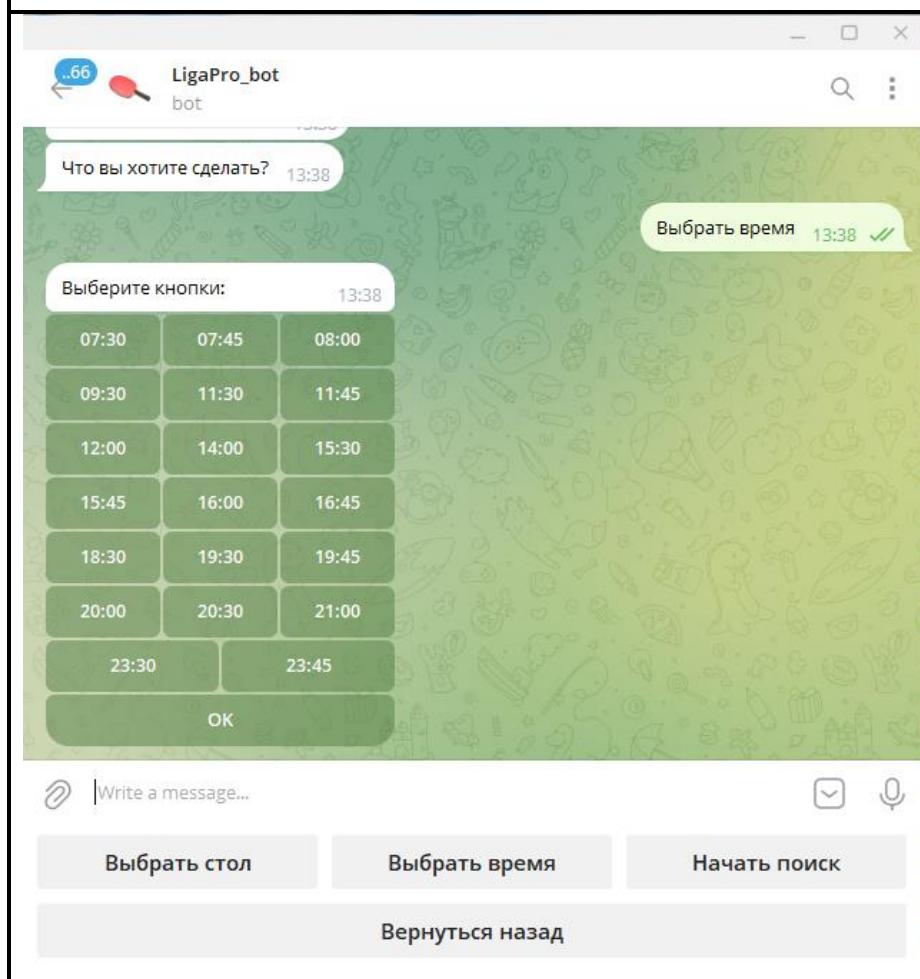
Производится
поиск по игроку:
пользователь
вводит фамилию и
получает в ответ
список турниров,
где участвует этот
игрок



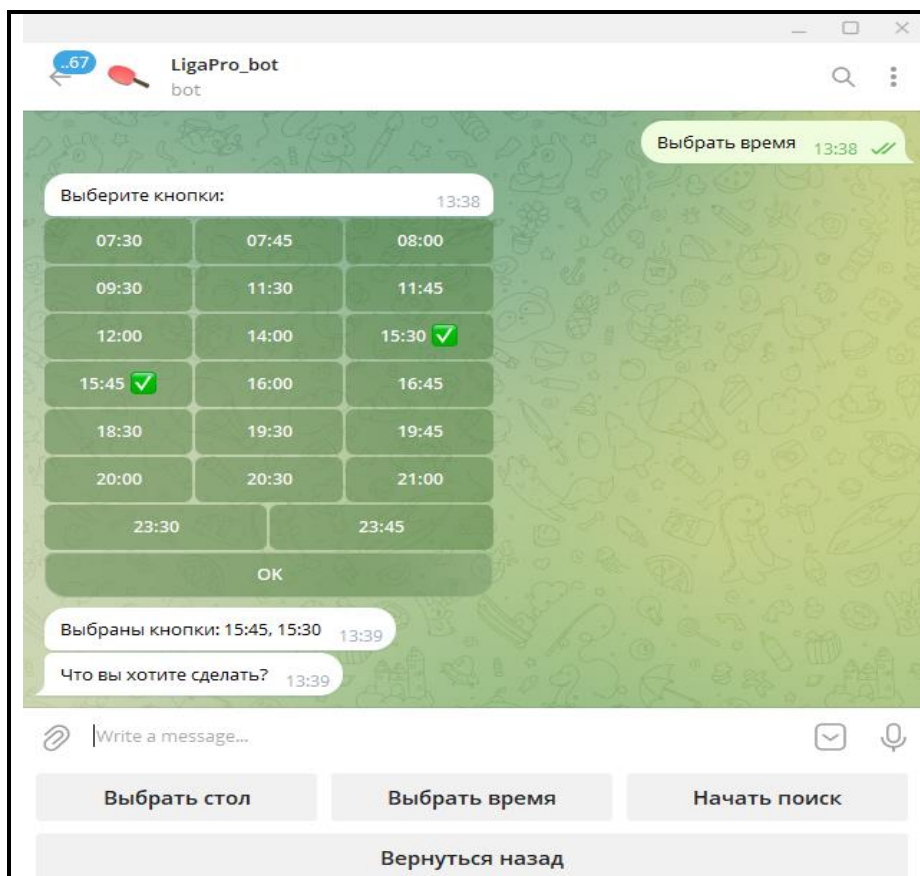
Отработка кнопки
«Поиск по
параметрам».
Предложение
выбора стола



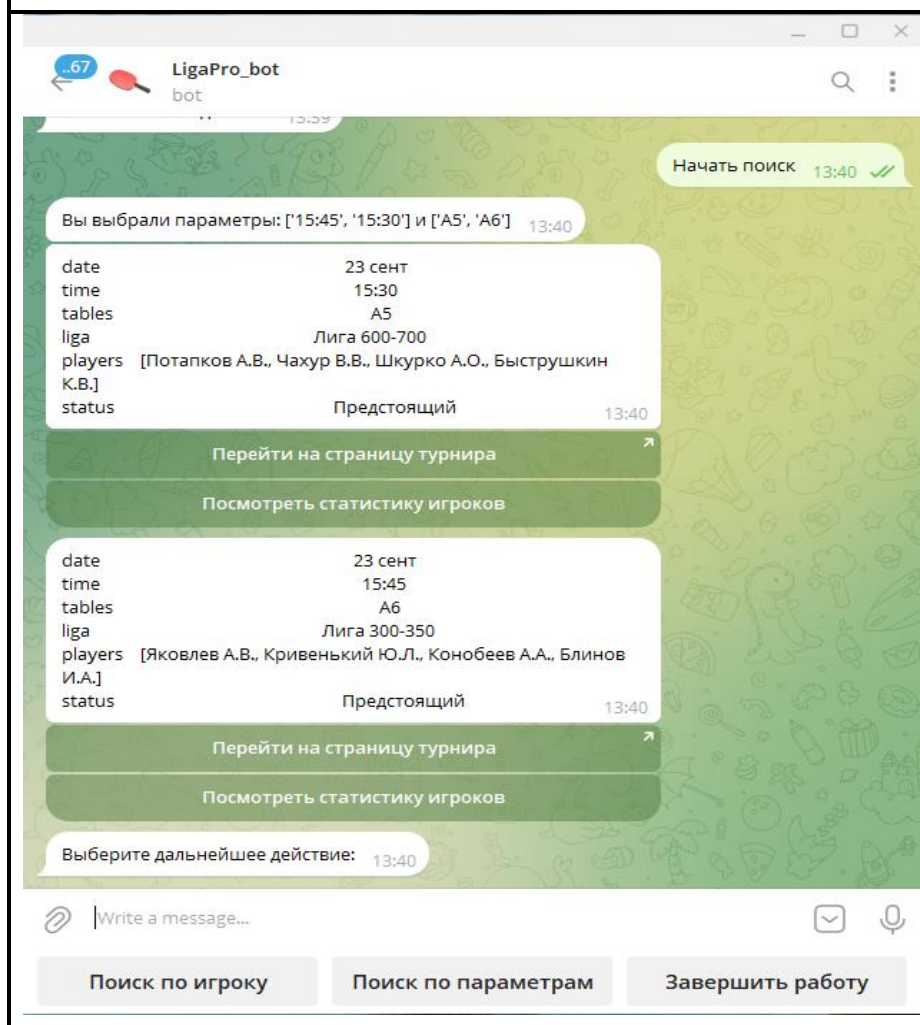
Нажатие на кнопки выбора стола, указание новой маркировки, нажатие кнопки ОК и получение ответа в виде списка выбранных столов



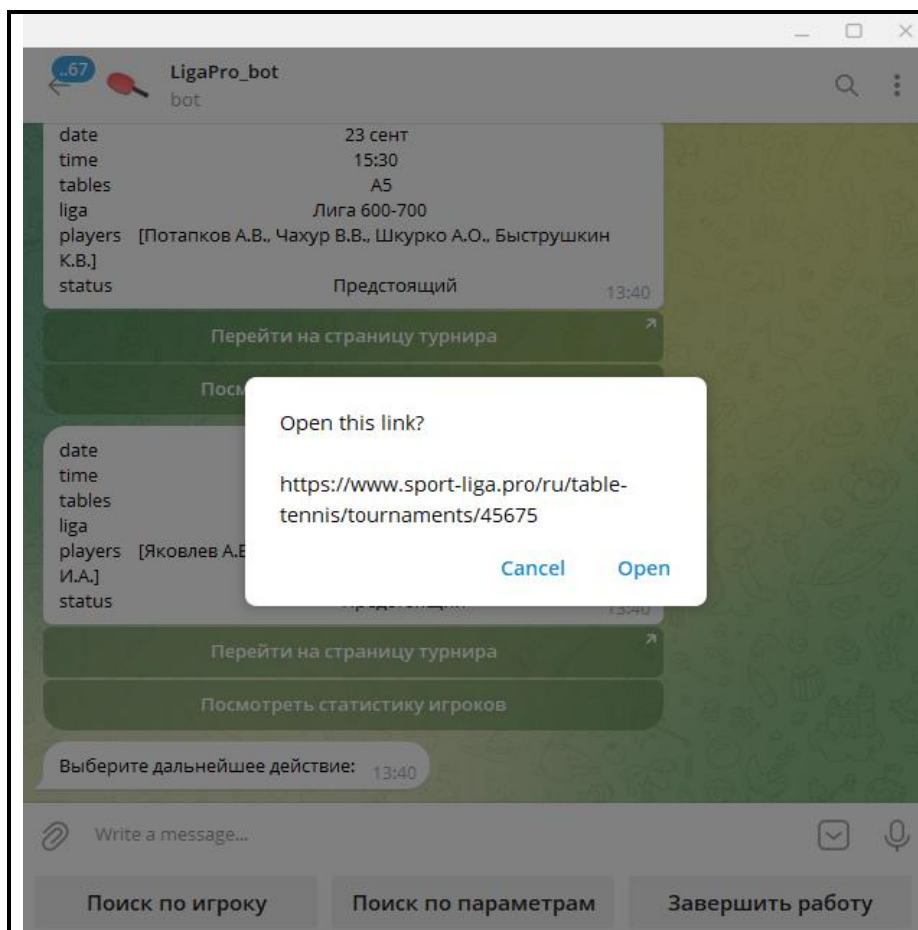
Нажатие на кнопку «Выборить время» и получение клавиатуры с выбором нужных кнопок



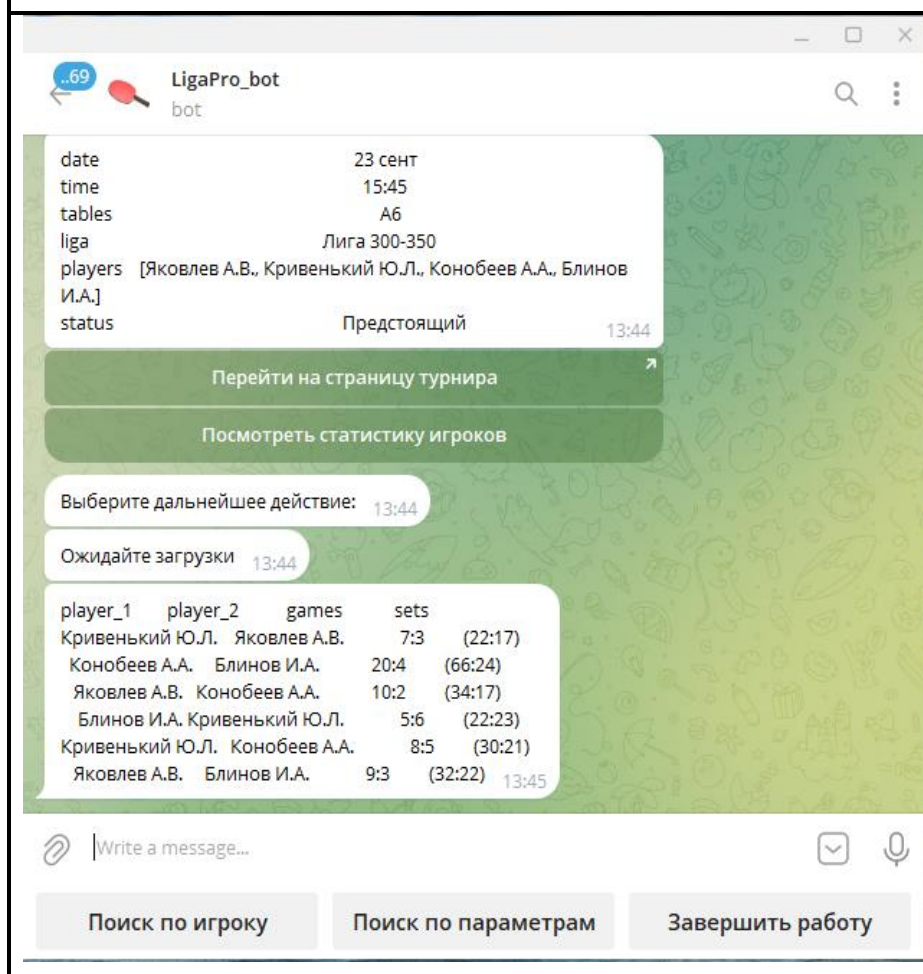
Отработка нажатия на кнопки нужного времени и кнопки «ОК», получение в ответ списка с выбранным временем



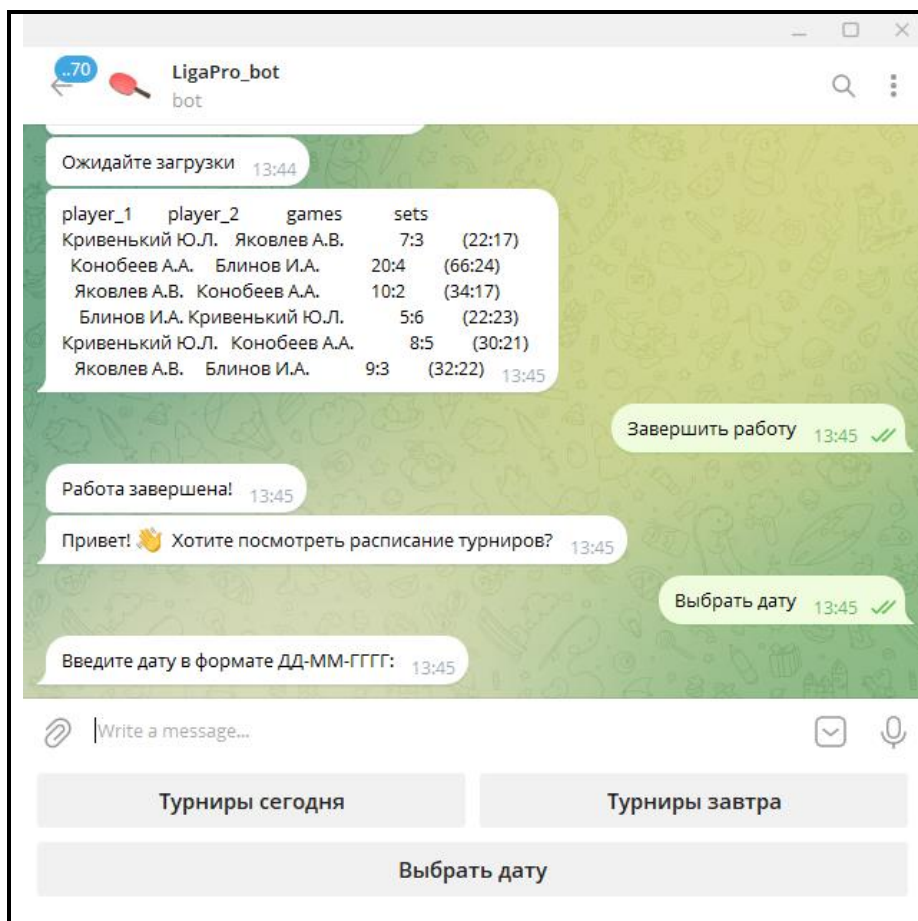
Нажатие на кнопки «Начать поиск» и получение информации о турнирах, соответствующих выбранным параметрам



Нажатие на кнопку «Перейти на страницу турнира», открытие окна с предложением перейти на сайт



Нажатие на кнопку «Посмотреть статистику игроков» и получение истории личных встреч между игроками, участвующих в турнире. Время загрузки составляет 30 секунд



Отработка кнопки
«Завершение
работы» и переход
в приветственное
меню