МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

КУРСОВАЯ РАБОТА

по дисциплине «Алгоритмы и структуры данных»

Тема: Трекер цен на сайтах

Мирошников Н.Ю.

Студенты гр. 0362 Радионов Р.С.

Рядинский А.П.

Преподаватель _____ Спиридонов Р.Е.

Санкт-Петербург

2021

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студенты Мирошников Н.Ю., Радионов Р.С., Рядинский А.П.

Группа 0362

Тема работы: Трекер цен на сайтах

Содержание пояснительной записки:

- Содержание
- Введение
- Описание функций парсера, Telegram-бота и работы с БД
- Примеры работы программы
- Исследование реализованных алгоритмов
- Заключение
- Список использованных источников

Предполагаемый объем пояснительной записки:

Не менее 10 страниц.

Дата выдачи задания: 30.11.2021

Дата сдачи курсовой работы: 15.12.2021

Дата защиты курсовой работы: 15.12.2021

Студенты		Мирошников Н.Ю. Радионов Р.С. Рядинский А.П.
Преподаватель		Спиридонов Р.Е.

АННОТАЦИЯ

В ходе работы была реализована программа на языке программирования Python с использованием библиотек, обеспечивающих парсинг цен с сайтов, работу с csv файлами и работу с Telegram. Программа реализована в виде бота в мессенджере Telegram.

SUMMARY

During the work the program was implemented in the Python programming language using libraries that provide parsing prices from the website, working with csv files and working with Telegram. The program is implemented as a Telegram bot.

СОДЕРЖАНИЕ

ЗАДАНИЕ

Отслеживать динамику цен товаров на сайте.

Сохранять между запусками программы какие товары нужно отслеживать. Выводить результаты работы за прошлые дни. По кнопке выполнять запросы с получением актуальных цен. В настройках программы дать возможность выполнять автоматический сбор данных после запуска программы.

Функции меню:

- Добавить товар по ссылке
- Удалить товар по номеру в списке
- Вывод отслеживаемых товаров
- Получение своего CSV-файл

ВВЕДЕНИЕ

Целью данной работы является создание программы способной отслеживать динамику цен товаров на сайтах (М.Видео, DNS, ситилинк, Эльдорадо, ozon), написанной на языке Python с использованием библиотек, обеспечивающих возможность парсинга цен с сайта, работы с csv файлами и работы с Telegram.

Для достижения поставленной цели требуется решить следующие задачи:

- Реализация получения цен по ссылке;
- Реализация методов для работы с CSV-файлами;
- Создание Telegram-бота как интерфейса для взаимодействия с программой;
- Сборка и тестирование программы.

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Многие веб-приложения используют API для подключения к различным сторонним сервисам. Используя API, вы можете получать доступ к таким данным как информация о погоде, результаты спортивных состязаний, рейтинги фильмов, твиты, результаты поиска в поисковых системах и изображения. Также вы можете использовать API для добавления функционала в ваше приложение, например платежей, планирования, отправки сообщений электронной почты, переводов, карт и передачи файлов. Самостоятельное создание таких инструментов может занять очень много времени, а API позволяют за считанные минуты подключиться к источнику и получить доступ к его функциям и данным.

Запросы HTTP лежат в основе всемирной сети. Каждый раз, когда вы открываете веб-страницу, ваш браузер направляет множество запросов на сервер этой веб-страницы. Сервер отвечает на них, пересылая все необходимые данные для вывода страницы, и ваш браузер отображает страницу, чтобы вы могли увидеть ее.

В составе запроса клиент отправляет данные по методу запроса. Наиболее распространенными методами запроса являются GET, POST и PUT. Запросы GET обычно предназначены только для чтения данных без их изменения, а запросы POST и PUT обычно предназначаются для изменения данных на сервере. Например, Stripe API позволяет использовать запросы POST для тарификации, чтобы пользователь мог купить что-нибудь в вашем приложении.

Selenium WebDriver — это программная библиотека для управления браузерами. WebDriver представляет собой драйверы для различных браузеров и клиентские библиотеки на разных языках программирования, предназначенные для управления этими драйверами.

По сути своей использование такого веб-драйвера сводится к созданию бота, выполняющего всю ручную работу с браузером автоматизировано.

Beautiful Soup — это парсер для синтаксического разбора файлов HTML/XML, написанный на языке программирования Python, который может

преобразовать даже неправильную разметку в дерево синтаксического разбора. Он поддерживает простые и естественные способы навигации, поиска и модификации дерева синтаксического разбора. В большинстве случаев он поможет программисту сэкономить часы и дни работы. Написанный на языке программирования Ruby порт называется Rubyful Soup.

РЕАЛИЗАЦИЯ ПРОГРАММЫ. ОПИСАНИЕ ФУНКЦИЙ

1. Функция getFilename(user)

Вызывается для получения имя файла конкретного пользователя.

2. Функция CheckSameURL(user, URL)

Вызывается для проверки на наличие указанной ссылки в БД.

3. Функция CheckUserCSV(user)

Вызывается для проверки на существование БД пользователя.

4. Функция CreateCSV(user)

Вызывается для создания БД пользователя.

5. Функция SaveInCSV(user, product, store, URL)

Вызывается для парсинга цены товара и добавления строки, содержащей название товара, магазин, ссылку и цены на товар в БД.

6. Функция DeleteFromCSV(user, product)

Вызывается для удаления строки товара из БД по названию.

7. Функция AddCurPrice(user, index, price)

Вызывается для добавления цены товара в столбец с сегодняшней датой в строку под номером index

8. Функция getUserList(user)

Вызывается для получения имён, ссылок и актуальных цен всех товаров из БД.

9. Функция UpdateAll()

Вызывается для обновления цен всех товаров во всех БД.

10. Функция UpdateDB(user)

Вызывается для построчного обновления цен товаров в конкретной БД.

11. Функция processing(silk)

Функция проверяет полученную строку на то, является ли она ссылкой на товар на одном из указанных сайтов.

12. Функция get_data_with_selenium(url)

Функция открывает браузер Chrome, переходит по указанному url и парсит html-код страницы.

13. Функция parse(url, website_name)

Функция производит парсинг html-кода страницы и последующую обработку с целью получения цены товара.

14. Функция update()

Функция запускает обновление цен всех отслеживающихся товаров.

15. Функция scheduler()

Функция реализует метод планировки операций ввода/вывода I/O Scheduling.

16. Функция on_startup(_)

Функция запускает сопрограмму(задание) sheduler(), тем самым реализуя многопоточность и позволяя обновлять все цены во всех базах данных раз в сутки в установленное время в автоматическом режиме

В моменты, когда программа не совершает никаких других действий, функция проверяет условия выполнения, указанные в scheduler и в случае совпадения - запускает её.

17. Функция load_animation(message: types.Message)

Функция отправляет пользователю сообщения, уведомляющие о начале поиска файла и добавления последнего в базу данных.

18. Функция process_start_command(message: types.Message)

Функция достаёт массив с id пользователей, уже использовавших бота, из файла Users.json, проверяет наличие id пользователя в этом массиве, в случае отсутствия — добавляет id в массив и загружает его назад в Users.json. Так же функция добавляет/изменяет (в случае если пользователь уже использовал бота) все диалоговые состояния пользователя на False.

19. Функция updater(message: types.Message)

Функция предназначена для ручного запуска обновления всех БД администраторами.

20. Функция no_type_message(msg: types.Message)

Работу функции можно разделить на обработку текста сообщений, отправка которых вызвана нажатием встроенных кнопок и сообщений

неизвестного содержания. В случае, если пользовательское сообщение - текст встроенной кнопки, сообщение будет обработано соответственно тому, какая кнопка была выбрана. В противном случае в зависимости от диалогового состояния пользователя, сообщение будет расценено либо как ссылка на товар, либо как название добавляемого товара, либо как номер удаляемого товара в списке и будет обработано соответствующим образом.

21. Функция error_bot_blocked(update: types.Update, exception: BotBlocked)

Функция обрабатывает исключение, возникающее при блокировке пользователем бота во время его работы.

РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ ПРОГРАММЫ

На рис. 1 представлено сообщение, которым бот встречает новых пользователей

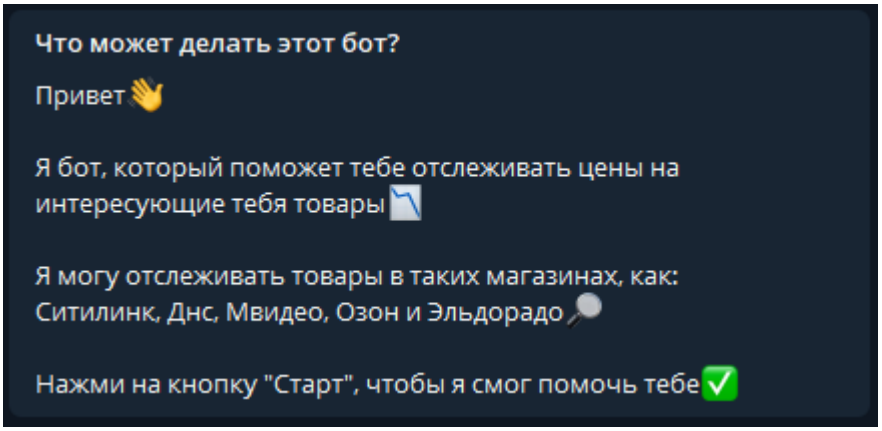

Рисунок 1 — Описание бота

На рис. 2 представлен пользовательский интерфейс программы как телеграмм-бот. Бот содержит функции добавления товара по ссылки, удаления по номеру в списке отслеживаемых, вывод списка товаров пользователя и получения своего CSV-файла.

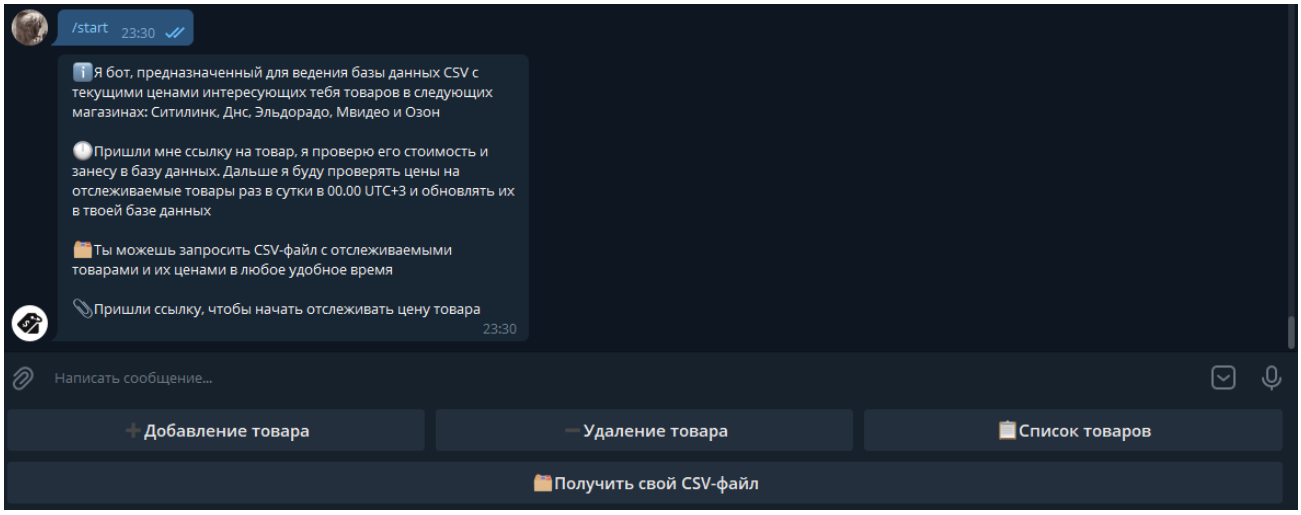

Рисунок 2 — Пользовательский интерфейс бота

На рис. 3 показана опция, добавления товара. Пользователь может добавить товар введя ссылку. А также назвав свой товар как пользователю захочется. Добавление товара происходит под приятную анимацию.

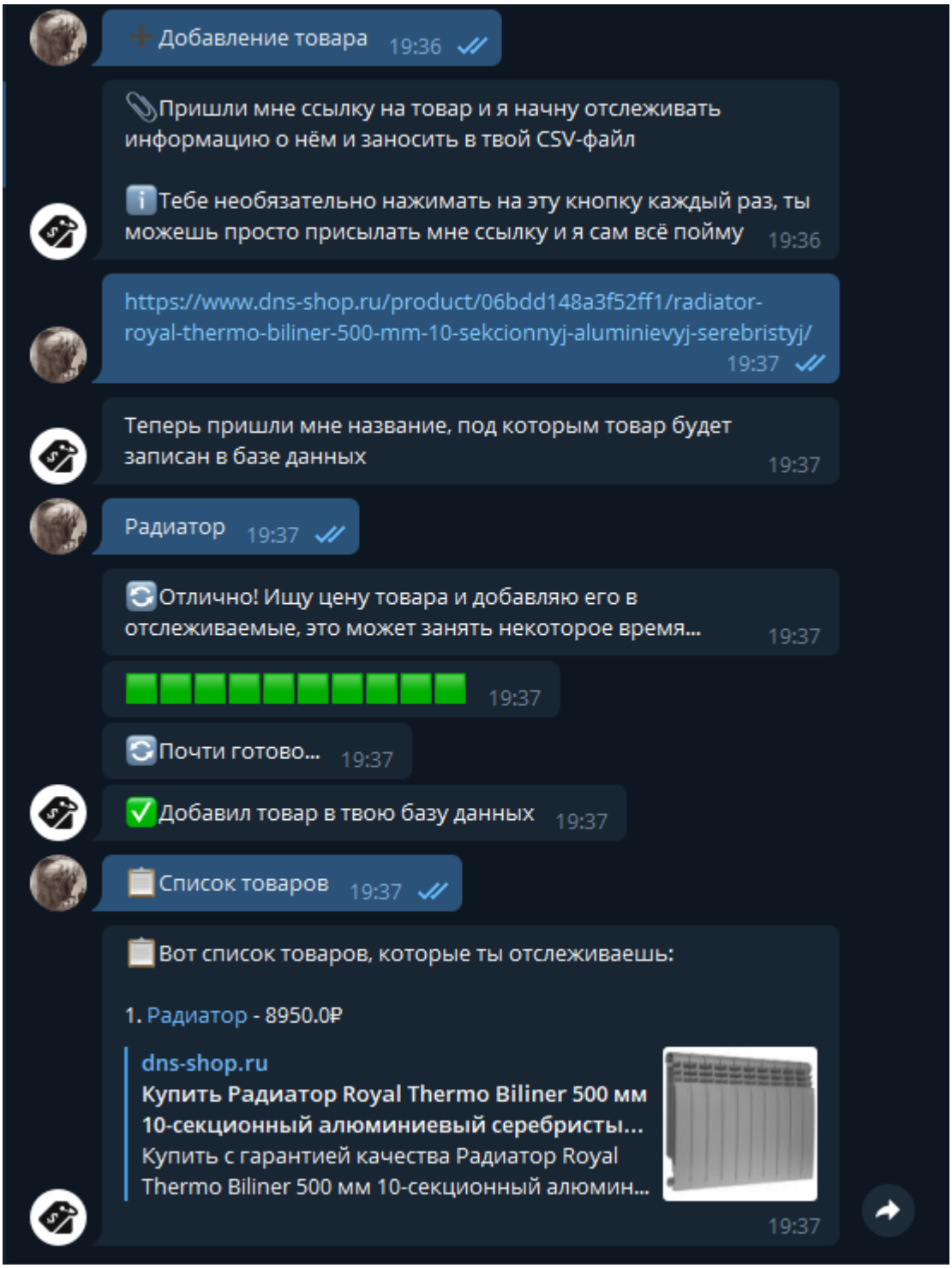

Рисунок 3 — Добавление товара

На рис. 4 показан вывод отслеживаемых товаров пользователя, название, которое вписал пользователь и его цену.

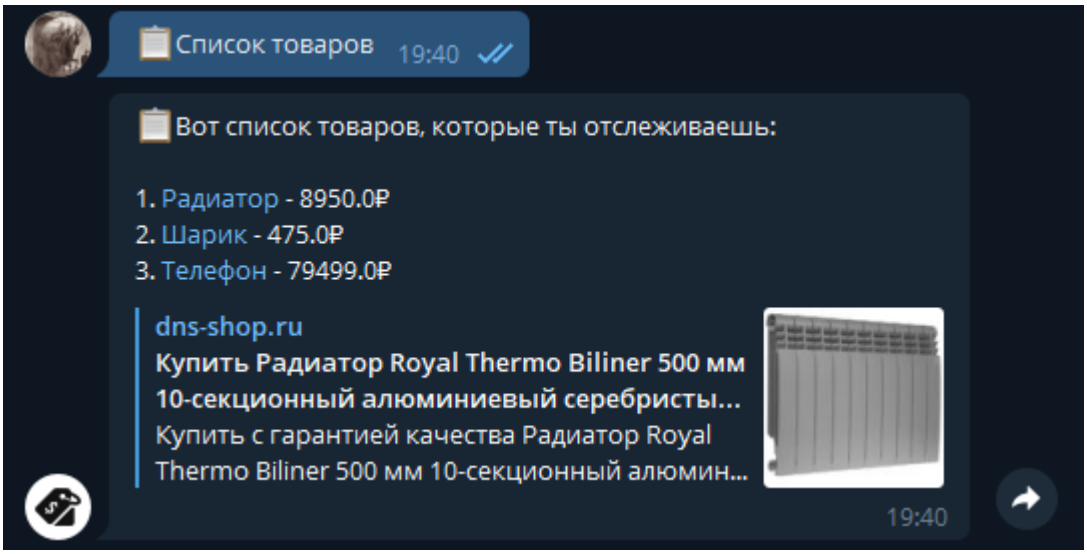

Рисунок 4 — Вывод списка товаров

На рис. 5 показано удаление из списка отслеживаемых путем ввода его номера в списке.

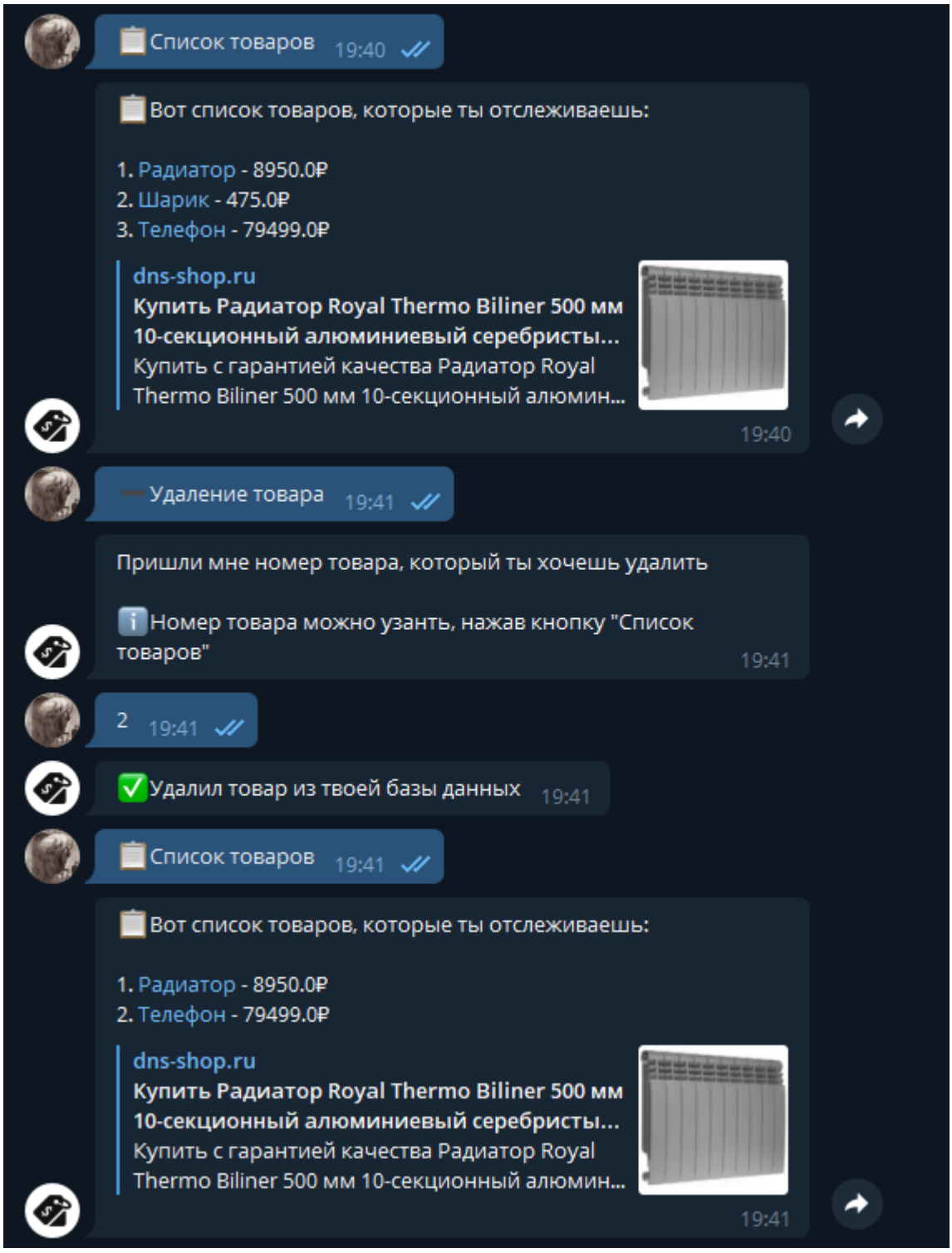

Рисунок 5 —Удаление из списка отслеживаемых

На рис. 6 показано получение своего CSV-файла с отслеживаемыми товарами.

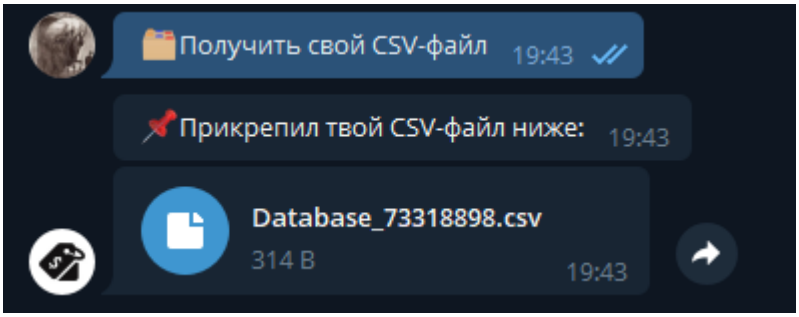

Рисунок 6 — Получение своего CSV-файла

На рис. 7 показано что будет если у вас нет отслеживаемых товаров, но вы нажали кнопку «Получить свой CSV-файл».

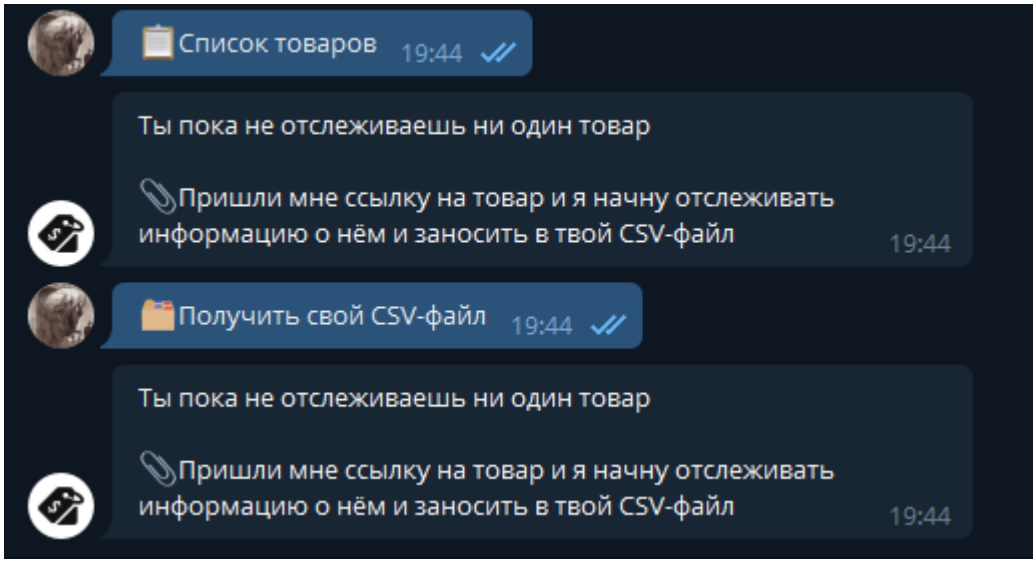

Рисунок 7 — Попытка получения пустого CSV-файла

ЗАКЛЮЧЕНИЕ

В ходе работы была реализована программа на языке программирования Python с использованием библиотек, обеспечивающих парсинг цен с сайта, работу с csv файлами и работу с Telegram. Для взаимодействия с программой был реализован бот в мессенджере Telegram.

Пользователю доступны функции для создания, изменения и сохранения файлов, просмотра результата в режиме реального времени. Также пользователю доступны инструменты экспорта результата в виде CSV-файла.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. https://habr.com/ru/post/248559/ —Selenium для Python
2. http://wiki.python.su/Документации/BeautifulSoup — BeautifulSoup для Python

ПРИЛОЖЕНИЕ А. РУКОВОДСТВО ПО НАСТРОЙКЕ

Для работы бот использует некоторые сторонние сервисы и программы. В данном руководстве указано, как запустить исходный код программы для корректной работы бота.

1. Бот разработан под язык Python версии 3.10

2. Для корректной работы бота на компьютере, с которого запущен бот, должен быть установлен браузер Google Chrome версии 96.0.4664.45 либо Google Chrome версии 96.0.4664.110

3. Также необходимо поместить в папку проекта специальный веб драйвер ChromeDriver версии 96.0.4664.45

4. Основным файлом проекта является файл «Bot.py». Именно его следует запускать при желании протестировать бота.

5. Поскольку проект реализован в формате бота для telegram, в коде должен быть указан токен бота. В прикреплённых исходных файлах данный токен удалён по соображениям безопасности. При желании протестировать исходный код, необходимо создать собственного telegram бота и использовать его токен.

ПРИЛОЖЕНИЕ Б. ИСХОДНЫЙ КОД ПРОГРАММЫ

Bot.py

```
import json
from aiogram.utils.exceptions import BotBlocked
from aiogram import Bot, Dispatcher, executor, types
from aiogram.types import KeyboardButton, ReplyKeyboardMarkup,
ChatActions
import bd
import urlcheck
import asyncio
import aioschedule


# Присваивание переменной TOKEN строки с токеном бота, полученным у
@BotFather (в целях безопасности токен,
# использовавшийся для теста программы удалён, для теста необходимо
самостоятельно получить токен у @BotFather)
TOKEN = ""

# Создание объекта бота bot с помощью функции Bot() сторонней
библиотеки aiogram и передача ему переменнной TOKEN
# в качестве параметра token
bot = Bot(token=TOKEN)

# Создание объекта диспетчера dp с помощью функции Discpatcher()
сторонней библиотеки aiogram
# и передача ему переменной bot в качестве параметра bot
dp = Dispatcher(bot=bot)


# Открытие и чтение массива, хранящегося в файле Users.json и
содержащего id пользователей, использовавших бота ранее
with open('Users', 'r') as file:
    from_json = json.load(file)  # массив с id пользователей
записывается в переменную from_json
file.close()


"""
 Словарь диалоговых состояний - словарь, в котором ключом является
id пользователя, а значением - True/False
 Словари диалоговых состояний необходимы для определения того, как
бот должен реагировать на сообщение пользователя
 если оно не является командой
"""

# Создание пустых словарей диалоговых состояний naming_products,
adding_products и deleting_products:
# 1 - Словарь naming_products отображает, ожидает ли бот от
пользователя название нового продукта
# 2 - Словарь adding_products отображает, производит ли бот парсинг
товара и сохранение его в CSV файл пользователя
```

```
# 2 - Словарь deleting_products отображает, ожидает ли бот от
пользователя номер удаляемого товара
naming_products = {}
adding_products = {}
deleting_products = {}

# Создание пустых словарей shops и urls:
# 1 - Словарь urls хранит последнюю корректную ссылку, присланною
пользователем для дальнейшей обработки, ключом для
#     каждой ссылки служит id пользователя
# 2 - Словарь shops хранит название магазина, на который ведёт
последняя корректная ссылка, присланная пользователем,
#     ключом для каждого названия является id пользователя
urls = {}
shops = {}

# Присваивание ключу, являющемуся id каждого пользователя, ранее
использовавшего бота, значения False в словарях
# naming_products, adding_products и deleting_products
for user in from_json:
    naming_products[user] = False
    adding_products[user] = False
    deleting_products[user] = False


async def update():
    """
    Функция запускает обновление цен всех отслеживающихся товаров
во всех БД с помощью функции
    UpdateAll(), описанной в файле bd.py

    :return: Функция ничего не возвращает
    """
    bd.UpdateAll()


async def scheduler():
    """
    Функция реализует метод планировки операций ввода/вывода I/O
Scheduling
    За счёт своей асинхронности функция позволяет программе
работать в многопоточном режиме

    :return:
    """
    aioschedule.every().day.at("00:00").do(update)
    while True:
        await aioschedule.run_pending()
        await asyncio.sleep(1)


async def on_startup(_):
    """
```

```
    Функция запускает сопрограмму(задание) sheduler(), тем самым
реализуя многопоточность и позволяя
    обновлять все цены во всех базах данных раз в сутки в
установленное время в автоматическом режиме
    В моменты, когда программа не совершает никаких других
действий, функция проверяет условия выполнения,
    указанные в scheduler и в случае совпадения - запускает её

    :param _:
    :return:
    """
    asyncio.create_task(scheduler())


async def load_animation(message: types.Message):
    """
    Функция отправляет пользователю сообщения, уведомляющие о
начале поиска файла и добавления последнего в базу данных
    Далее функция отправляет строку, состоящую из 10 белых
квадратов, после редактирет это сообщения, поочерёдно заменяя
    слева направо белые квадраты зелёными с интервалом в 0.3
секунды, тем самым имитируя загрузку
    В конце своей работы функция отправляет сообщение, уведомляющее
о скором завершении обработки запроса пользователя

    :param message: Объект типа Message сторонней библиотеки
aiogram - сообщение пользователя
    :return: Функция ничего не возвращает
    """
    await bot.send_message(message.from_user.id,
                           "🔄Отлично! Ищу цену товара и добавляю
его в отслеживаемые, это может занять некоторое время...")
    upload_message = await
bot.send_message(chat_id=message.from_user.id,
text="□□□□□□□□□□")
    await asyncio.sleep(1)

    # Цикл, в котором происходит редактирование сообщения с
интервалом 0.3 секунды
    for i in range(1, 11):
        await upload_message.edit_text(text="■" * i + (10 - i) *
"□")
        await asyncio.sleep(0.3)
    await bot.send_message(message.from_user.id, "🔄Почти
готово...")


# Хэндлер, ответственный за обработку сообщения с командой /start
@dp.message_handler(commands=['start'])
async def process_start_command(message: types.Message):
    """
```

```
    Функция достаёт массив с id пользователей, уже использовавших
бота, из файла User.json, проверяет наличие id
    пользователя в этом массиве, в случае отсутствия - добавляет id
в массив и загружает его назад в Users.json

    Так же функция добавляет/изменяет(в случае если пользоваетль
уже использовал бота) все диалоговые состояния
    пользователя на False

    :param message: Объект типа Message сторонней библиотеки
aiogram - сообщение пользователя
    :return: Функция ничего не возвращает
    """
    # Достаём масив, содержащий id пользователей из файла
Users.json
    with open('Users', 'r') as file:
        users = json.load(file)
    file.close()

    # Если id пользователя отсутствует в массиве в файле
Users.json, он добавлятся в этот массив
    if message.from_user.id not in users:

        # Запись пользователя в массив в файле Users.json
        with open('Users', 'w') as file:
            users.append(message.from_user.id)
            json.dump(users, file)
        file.close()

    # Присваивание ключу, являющемуся id пользователя значения
False в словарях naming_products, adding_products
    # и deleting_products
    naming_products[message.from_user.id] = False
    deleting_products[message.from_user.id] = False
    adding_products[message.from_user.id] = False

    await bot.send_message(message.from_user.id, "ℹ️Я бот,
предназначенный для ведения базы данных CSV с текущими ценами
интересующих тебя товаров в следующих магазинах: Ситилинк, Днс,
Эльдорадо, Мвидео и Озон\n\n🕐Пришли мне ссылку на товар, я проверю
его стоимость и занесу в базу данных. Дальше я буду проверять цены
на отслеживаемые товары раз в сутки в 00.00 UTC+3 и обновлять их в
твоей базе данных\n\n🗂Ты можешь запросить CSV-файл с
отслеживаемыми товарами и их ценами в любое удобное
время\n\n📎Пришли ссылку, чтобы начать отслеживать цену товара",
reply_markup=menu)


# Хэндлер, ответственный за обработку сообщения с командой
/doarickroll
@dp.message_handler(commands=['doarickroll'])
async def rickroller(msg: types.Message):
```

```
    """
    Функция отправляет пользователю сообщение со старой, но
навевающей воспоминания шуткой

    :param msg: Объект типа Message сторонней библиотеки aiogram -
сообщение пользователя
    :return: функция возвращает пользователя во времена лампового
интернета, Медведа и Упячки
    """
    await bot.send_message(msg.from_user.id, '<a
href="https://www.youtube.com/watch?v=dQw4w9WgXcQ">Never gonna give
you up\nNever gonna let you down\nNever gonna run around and desert
you\nNever gonna make you cry\nNever gonna say goodbye\nNever gonna
tell a lie and hurt you</a>', parse_mode="HTML")


@dp.message_handler(commands=['update'])
async def updater(msg: types.Message):
    """
    Функция предназначена для ручного обновления всех БД
администраторами
    :param msg:
    :return:
    """
    if msg.from_user.id == 276194719 or msg.from_user.id ==
73318898:
        await bot.send_message(msg.from_user.id, "Приветствую,
администратор!\n\nЗапускаю обновление всех БД в ручном режиме")
        await bot.send_message(276194719,
                               "ℹ️Администратор запустил обновление
БД в ручном режиме")
        await bot.send_message(73318898,
                               "ℹ️Администратор запустил обновление
БД в ручном режиме")
        bd.UpdateAll()
        await bot.send_message(276194719,
                               "✅Завершено обновление всех БД")
        await bot.send_message(73318898,
                               "✅Завершено обновление всех БД")


# Хэндлер, обрабатывающий сообщения, не являющиеся командами
@dp.message_handler()
async def no_type_message(msg: types.Message):
    """
    Работу функции можно разделить на обработку текста сообщений,
отправка которых вызвана нажатием встроенных кнопок
    и сообщений неизвестного содержания

    В случае, если пользовательское сообщение - текст встроенной
кнопки, сообщение будет обработано соответственно тому,
    какая кнопка была выбрана
```

```
    В противном случае в зависимости от диалогового состояния
пользователя, сообщение будет расценено либо как
    ссылка на товар, либо как название добавляемого товара, либо
как номер удаляемого товара в списке и будет обработано
    соответствующимм образом

    :param msg: Объект типа Message сторонней библиотеки aiogram -
сообщение пользователя
    :return: 0 - если пользователь присылает любое сообщение во
время парсинга и сохранения
             информации о товаре в базу данных
    """

    # Призваивание переменной user значение id пользователя для
дальнейшега удобства в использовании
    user = msg.from_user.id

    # Проверка на то, производится ли парсинг и сохранение
информации о товаре в базу данных пользователя
    if adding_products[user] is True:
        return 0

    # Обработка текста сообщения, присылаемого встроенной кнопкой
"➕Добавление товара"
    if msg.text == "➕Добавление товара":

        await bot.send_message(user, "📎Пришли мне ссылку на товар и
я начну отслеживать информацию о нём и заносить в твой CSV-
файл\n\nℹ️Тебе необязательно нажимать на эту кнопку каждый раз, ты
можешь просто присылать мне ссылку и я сам всё пойму")

    # Обработка текста сообщения, присылаемого встроенной кнопкой
"🗂Получить свой CSV-файл"
    elif msg.text == "🗂Получить свой CSV-файл":

        # Присваивание переменной file_name имени файла
пользователя с помощью функции getFilename(),
        # описанной в файле bd.py
        file_name = bd.getFilename(user)

        # Присваивание переменной products двумерного массива,
содержащего имена и url сайтов и цены товаров из
        # CSV файла пользователя с помощью функции getUserList(),
описанной в файле bd.py
        products = bd.getUserList(user)

        # Если у пользователя сейчас есть хотя бы один
отслеживаемый товар
        if len(products) != 0:
```

```
            await bot.send_message(user, "📌Прикрепил твой CSV-файл
ниже:")

            #Открытие CSV файла пользователя, отправка файла,
закрытие файла
            file_for_user = open(file_name, 'rb')
            await bot.send_chat_action(user,
ChatActions.UPLOAD_DOCUMENT)
            await bot.send_document(user, file_for_user)
            file_for_user.close()

        else:
            await bot.send_message(user, "Ты пока не отслеживаешь
ни один товар\n\n📎Пришли мне ссылку на товар и я начну отслеживать
информацию о нём и заносить в твой CSV-файл")

    # Обработка текста сообщения, присылаемого встроенной кнопкой
"➖Удаление товара"
    elif msg.text == "➖Удаление товара":

        # Если у пользователя сейчас есть хотя бы один
отслеживаемый товар
        if len(bd.getUserList(user)) != 0:

            # Присваивание пользовательскому id значения True в
словаре диалоговых состояний deleting_products,
            # что означает переключение бота в режим ожидания
номера удаляемого товара от пользователя
            deleting_products[user] = True
            await bot.send_message(user, 'Пришли мне номер товара,
который ты хочешь удалить\n\nℹ️Номер товара можно узнать, нажав
кнопку "Список товаров"')

        else:
            await bot.send_message(user,
                                   "Ты пока не отслеживаешь ни один
товар\n\n📎Пришли мне ссылку на товар и я начну отслеживать
информацию о нём и заносить в твой CSV-файл")

    # Обработка текста сообщения, присылаемого встроенной кнопкой
"📋Список товаров"
    elif msg.text == "📋Список товаров":

        # Присваивание переменной products двумерного массива,
содержащего имена и url сайтов и цены товаров из
        # CSV файла пользователя с помощью функции getUserList(),
описанной в файле bd.py
        products = bd.getUserList(user)

        # Если у пользователя сейчас есть хотя бы один
отслеживаемый товар
```

```
        if len(products) != 0:

            output = '📋Вот список товаров, которые ты
отслеживаешь:\n\n'

            # Цикл, в котором производится формирование сообщения в
переменной output, адресованного пользователю
            # и содержащего номер товара, его название(являющееся
гиперссылкой на сам товар) и цену в рублях
            product_index = 1
            for product, url, price in products:
                output += str(product_index) + '. ' + '<a href=' +
'"' + url + '">' + product + '</a>' + ' - ' + str(price)
                if price == 'Parsing Error' or price == 'Товар
временно отсутствует в продаже':
                    output += '\n'
                else:
                    output += '₽' + '\n'
                product_index += 1
            output += '\n'

            await bot.send_message(user, output, parse_mode="HTML")

        else:
            await bot.send_message(user, "Ты пока не отслеживаешь
ни один товар\n\n📎Пришли мне ссылку на товар и я начну отслеживать
информацию о нём и заносить в твой CSV-файл")

    # Если текст сообщения не является ни командой ни текстом
встроенных кнопопок
    else:
        # Если пользователь прислал не название добавляемого товара
и не номер удаляемого товара означает,
        # что пользователь прислал ссылку на товар
        if naming_products[user] is False and
deleting_products[user] is False:

            # Присваивание переменной url текста сообщения
пользователя
            url = msg.text

            # Присваивание переменной shop_name значения,
полученного с помощью функции processing(),
            # описанной в файле urlcheck.py
            # Значение - название магазина, в случае корректности
ссылки, -1 - в противном случае
            shop_name = urlcheck.processing(url)

            # Если ссылка, присланная пользователем корректна
            if shop_name != -1:

                # Если ссылка, присланная пользователем не
указывает на уже отслеживаемый товар
```

```
                # Проверка осуществляется с помощью функции
CheckSameURL(), описанной в файле bd.py
                if bd.CheckSameURL(user, url) is False:

                    # Сохранение ссылки пользователя в словарь urls
по ключу, представляющему id пользователя
                    urls[user] = url

                    # Сохранение названия магазина, указанного в
ссылке пользователя в словарь urls
                    # по ключу, представляющему id пользователя
                    shops[user] = shop_name

                    await bot.send_message(user,
                                           "Теперь пришли мне
название, под которым товар будет записан в базе данных")

                    # Присваивание пользовательскому id значения
True в словаре диалоговых состояний naming_products,
                    # что означает переключение бота в режим
ожидания названия нового товара
                    naming_products[user] = True

                else:
                    await bot.send_message(user, "⃠Товар,
находящийся по этой ссылке уже отслеживается")
            else:
                await bot.send_message(user, "⃠Ссылка, которую ты
прислал некорректна, попробуй ещё раз")

        # Если бот находится в режиме ожидания названия нового
товара от пользователя
        elif naming_products[user] is True:

            # Отключение режима ожидания названия нового товара от
пользователя
            naming_products[user] = False

            # Присваивание пользовательскому id значения True в
словаре диалоговых состояний adding_products,
            # что означает переключение бота в режим парсинга и
работы с бд
            # Этот режим необходим чтобы пользователь не мог
послать запрос боту пока тот парсит и работает с БД
            adding_products[user] = True

            # Проигрывание анимации с помощью функции
load_animation()
            await load_animation(msg)

            #
            bd.SaveInCSV(user, msg.text, shops[user], urls[user])
```

```
            # Удаление последних ссылки и названия магазина,
добавленных пользователем из словарей
            # shops и urls
            del shops[msg.from_user.id], urls[msg.from_user.id]

            await bot.send_message(user, "✅Добавил товар в твою
базу данных")

            # Отключение режима парсинга и работы с бд
            adding_products[user] = False

        # Если бот находится в режиме ожидания номера удаляемого
продукта от пользователя
        elif deleting_products[user] is True:

            # Присваивание переменной products двумерного массива,
содержащего имена и url сайтов и цены товаров из
            # CSV файла пользователя с помощью функции
getUserList(), описанной в файле bd.py
            products = bd.getUserList(user)

            # Если у пользователя сейчас есть хотя бы один
отслеживаемый товар
            if len(products) != 0:

                # Проверка номера продукта в списке, введённого
пользователем, на корректность
                if '1' <= msg.text <= str(len(products)):

                    # Удаление товара из БД с помощью функции
DeleteFromCSV, описанной в файле bd.DeleteFromCSV
                    bd.DeleteFromCSV(user, int(msg.text) - 1)

                    await bot.send_message(user, '✅Удалил товар из
твоей базы данных')
                else:
                    await bot.send_message(user, 'Упс! Товара с
таким номером нет в твоей базе данных\n\n📋Нажми кнопку "Список
товаров", чтобы увидеть номер нужного тебе товара и возвращайся
сюда')
            else:
                await bot.send_message(user,
                                       "Ты пока не отслеживаешь ни
один товар\n\n📎Пришли мне ссылку на товар и я начну отслеживать
информацию о нём и заносить в твой CSV-файл")

            # Отключение режима ожидания номера удаляемого товара
от пользователя
            deleting_products[user] = False


@dp.errors_handler(exception=BotBlocked)
```

```
async def error_bot_blocked(update: types.Update, exception:
BotBlocked):
    """
    Функция обрабатывает исключение, возникающее при блокировке
пользователем бота во время его работы

    :param update: Объект типа Update сторонней библиотеки aiogram
    :param exception: Объект типа BotBlocked сторонней библиотеки
aiogram
    :return: True во всех случаях
    """

    # Вывод в консоль сообщения о произошедшей блокировкой
пользователем бота
    print(f"Меня заблокировал пользователь!\nСообщение:
{update}\nОшибка: {exception}")

    return True


# ----Меню----
# Создание четырёх объектов кнопок меню с помощью функции
KeyboardButton() сторонней библиотеки aiogram
add_url_button = KeyboardButton("➕Добавление товара")
delete_product_button = KeyboardButton("➖Удаление товара")
recieve_csv_button = KeyboardButton("🗃Получить свой CSV-файл")
show_products_list_button = KeyboardButton("📋Список товаров")

# Создание меню встроенных кнопок с помощью функции
ReplyKeyboardMarkup() сторонней библиотеки aiogram
menu =
ReplyKeyboardMarkup(resize_keyboard=True).add(add_url_button,
delete_product_button, show_products_list_button,
recieve_csv_button)

# Запуск бота с помощью функции start_polling() сторонней
библиотеки aiogram
executor.start_polling(dp, skip_updates=True,
on_startup=on_startup)
```

bd.py

```
import csv
import parsing
import pandas as pd
import os
import json
from datetime import date


def getFilename(user):
    """
    Получаем название файла базы данных конкретного пользователя
```

```
    :param user: идентификатор пользователя
    """

    return 'Database_' + str(user) + '.csv'


def CheckUserCSV(user):
    """
    Проверяем существует ли файл базы данных конкретного
пользователя

    :param user: идентификатор пользователя
    """

    if os.path.exists(getFilename(user)):
        return True
    else:
        return False


def CheckSameURL(user, URL):
    """
    Проверяем, записан ли такой товар в базе данных пользователя,
    сверяя переданную ссылку со всеми значениями столбца 'URL'

    :param user: идентификатор пользователя
    :param URL: ссылка на товар
    """

    # Проверка на существование файла БД
    if not CheckUserCSV(user):
        return False

    df = pd.read_csv(getFilename(user), index_col='URL',
encoding='windows_1251')
    if URL in df.index:
        return True
    else:
        return False


def CreateCSV(user):
    """
    Создание файла базы данных конкретного пользователя, заполнение
основных заголовков

    :param user: идентификатор пользователя
    """

    with open(getFilename(user), 'w') as csvfile:
        fieldnames = ['Product', 'Store', 'URL']
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
```

```
            writer.writeheader()
        csvfile.close()


def SaveInCSV(user, product, store, URL):
    """
    Сохранение товара в базу данных конкретного пользователя.

    При сохранении товара, заполняются основные сведения о нём
    (Имя товара, магазин, в котором товар отслеживается и ссылка на
товар в этом магазине).

    После этого при помощи парсинга получаем актуальную цену товара
и записываем в столбец,
    заголовком которого будет актуальная дата.

    :param user: идентификатор пользователя
    :param product: имя товара
    :param store: магазин, в котором необходимо отслеживать товар
    :param URL: ссылка на товар
    """

    # Создание БД, если её не существует
    if not CheckUserCSV(user):
        CreateCSV(user)

    # Проверка, на наличие такого товара в БД
    if CheckSameURL(user, URL):
        return 1

    # Запись основной информации о товаре в БД
    with open(getFilename(user), 'a') as csvfile:
        fieldnames = ['Product', 'Store', 'URL']
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
        writer.writerow({'Product': product, 'Store': store, 'URL':
URL})
    csvfile.close()

    # Преобразование файла БД в словарь
    database = []  # Словарь, в который будет записана информация
из БД
    with open(getFilename(user), 'r') as csvfile:
        reader = csv.DictReader(csvfile)
        for row in reader:
            database.append(row)
    csvfile.close()

    price = parsing.parse(URL, store)  # Парсинг цены
    AddCurPrice(user, len(database) - 1, price)  # Запись цены в
столбец с актуальной датой


def DeleteFromCSV(user, index):
```

```
    """
    Удаление товара из файла базы данных по номеру

    :param user: идентификатор пользователя
    :param index: номер товара, который необходимо удалить
    """

    # Проверка на существование файла БД
    if not CheckUserCSV(user):
        return 2

    # Запись информации из файла БД в pandas DataFrame
    df = pd.read_csv(getFilename(user), engine='python',
encoding='windows_1251')

    # Проверка, на то, что такой товар действительно существует в
БД
    # if product not in df.index:
    #     return 1

    df.drop(index=[index], inplace=True)
    df.to_csv(getFilename(user), encoding='windows_1251',
index=False)


def AddCurPrice(user, index, price):
    """
    Добавление цены в столбец с сегодняшней датой в строку под
номером index.
    В случае, если такого столбца не существует он создастя
автоматически.

    :param user: идентификатор пользователя
    :param index: номер строки, для которой необходимо записать
цену
    :param price: значение цены, которое необходимо записать в БД
    """

    # Проверка на существование файла БД
    if not CheckUserCSV(user):
        return 1

    df = pd.read_csv(getFilename(user), engine='python',
encoding='windows_1251')
    df.at[index, str(date.today())] = price
    df.to_csv(getFilename(user), index=False,
encoding='windows_1251')


def getUserList(user):
    """
    Получение имени товара, ссылки на товар, а также последней
записанной цены из файла БД.
```

```
    Данные сохраняются в формате двумерного массива для каждой
строки в файле, кроме строки заголовков.

    :param user: идентификатор пользователя
    """

    # Проверка на существование товара
    if not CheckUserCSV(user):
        return []

    df = pd.read_csv(getFilename(user), usecols=['Product', 'URL',
str(date.today())], encoding='windows_1251')
    dataframe = df.to_numpy()
    return dataframe


def UpdateAll():
    """
    Данная функция запускает функцию обновления цен для всех файлов
баз данных пользователей.
    Идентификаторы пользователей получаем из файла Users.
    """

    with open('Users', 'r') as files:
        db_list = json.load(files)
    files.close()

    for user in db_list:
        if CheckUserCSV(user):  # Проверка на существование файла
данного пользователя
            UpdateDB(user)


def UpdateDB(user):
    """
    Данная функция получает актуальную цену каждого товара для
одного конкретного файла базы данных
    и записывает её в столбец с актуальной датой.

    :param user: идентификатор пользователя
    """

    # Преобразование файла БД в словарь
    database = []  # Словарь, в который будет записана информация
из БД
    with open(getFilename(user), 'r') as csvfile:
        reader = csv.DictReader(csvfile)
        for row in reader:
            database.append(row)
    csvfile.close()

    # Построчная запись актуальной цены в столбец с актуальной
датой
```

```
    for i in range(len(database)):
        price = parsing.parse(database[i]['URL'],
database[i]['Store'])
        AddCurPrice(user, i, price)
```

parsing.py

```
import requests
from bs4 import BeautifulSoup as bs
from fake_useragent import UserAgent
from selenium import webdriver
from unipath import Path
import time
import lxml


# Функция открывает браузер Chrome, переходит по
# указанному url и парсит html-код страницы
# Принимает на вход строку с URL адресом сайта
# Возвращает объект, в котором хранится дерево html-кода
# либо -1 в случае, если был пойман Exception

def get_data_with_selenium(url):
    """
    Функция открывает браузер Chrome, переходит по
    указанному url и парсит html-код страницы

    :param url: Строка
    :return result: Объект, хранящий дерево html-кода страницы
    :return -1: В случае возникновения любого исключения
    """
    try:
        # Получение абсолютной ссылки на драйвер и её изменение под
формат ссылок Windows
        path = Path("chromedriver.exe").absolute()   # Проверка на
наличие драйвера
        absolute_path = ''
        for i in path:
            if ord(i) == 92:
                absolute_path += chr(92) * 2
            else:
                absolute_path += i
    except Exception as ex:
        print(ex)
        return -1

    chrome_options = webdriver.ChromeOptions()  # Добавление
настроек запуска браузера

chrome_options.add_experimental_option("useAutomationExtension",
False)
    chrome_options.add_experimental_option("excludeSwitches",
["enable-automation"])
    chrome_options.add_argument("--no-sandbox")
```

```
    chrome_options.add_argument("--disable-dev-shm-usage")

    result = -1
    driver = -1

    try:
        driver = webdriver.Chrome(executable_path=path,
options=chrome_options)  # Запуск драйвера
        driver.get(url=url)  # Получение доступа к сайту
        result = driver.page_source  # Сохранение html-кода
страницы
        time.sleep(1)
    except Exception as ex:
        print(ex)
        return -1
    finally:
        if driver != -1:
            try:
                driver.close()  # Закрытие драйвера
            except Exception as ex:
                print(ex)
                return -1
            try:
                driver.quit()
            except Exception as ex:
                print(ex)
                return -1
            return result
        else:
            return -1

def parse(url, website_name):
    """
    Функция производит парсинг html-кода страницы и последующую
обработку с целью получения цены товара

    :param url: Строка, ссылка на сайт
    :param website_name: Строка, имя сайта
    :return price: Число, цена товара (в случае успеха)
    :return 'Parsing Error': Строка (в случае возникновения
исключений или необработанных вариантов)
    """
    responce = requests.get(url, headers={'User-Agent':
UserAgent().chrome})  # Отправка запроса на сервер
    if responce.status_code != 200:  # Статус 200 состояния HTTP
значит, что получен ответ от сервера
        print('Connection Error: ', responce.status_code)
        return -1
    else:
        # Разделение на разные методы обработки полученного html-
кода в зависимости от сайта
        if website_name == 'citilink':
```

```
            silenium_result = get_data_with_selenium(url)  #
Парсинг
            if silenium_result == -1:
                return -1
            soup = bs(silenium_result, 'lxml')
            result = soup.find_all('h2',
class_='ProductHeader__not-available-header')  # Проверяем товар на
наличие
            if result != []:
                return 'Товар временно отсутствует в продаже'
            else:
                try:
                    result = soup.find('span',

class_='ProductHeader__price-default_current-price js--
ProductHeader__price-default_current-price').text  # Ищем тег span
с указанным классом
                except AttributeError:
                    return 'Parsing Error'
                price = ''
                for letter in result:
                    for i in str(letter):
                        if '0' <= i <= '9':
                            price += i
                if price == '':
                    price = 'Parsing Error'
                return price

        elif website_name == 'dns':
            silenium_result = get_data_with_selenium(url)  #
Парсинг
            if silenium_result == -1:
                return -1
            soup = bs(silenium_result, 'lxml')
            try:
                result = eval(soup.find('script',
type='application/ld+json').text)
                price = result['offers']['price']
                return price
            except AttributeError:
                try:
                    result = soup.find('div',
                                       class_='product-
buy__price').text  # Ищем тег div с указанным классом
                except AttributeError:
                    return 'Товар временно отсутствует в продаже'
            price = ''
            for letter in result:
                for i in str(letter):
                    if '0' <= i <= '9':
                        price += i
            if price == '':
                price = 'Parsing Error'
```

```
            return price

        elif website_name == 'mvideo':
            silenium_result = get_data_with_selenium(url)  #
Парсинг
            if silenium_result == -1:
                return -1
            soup = bs(silenium_result, 'lxml')
            result = soup.find_all('p', class_='product-sold-out-
text')  # Проверяем товар на наличие
            if result != []:
                return 'Товар временно отсутствует в продаже'
            else:
                try:
                    result = soup.find('span', class_='price__main-
value').text  # Ищем тег span с указанным классом
                except AttributeError:
                    return 'Parsing Error'
                price = ''
                for letter in result:  # Очищаем указанную цену от
лишних символов
                    for i in str(letter):
                        if '0' <= i <= '9':
                            price += i
                if price == '':
                    price = 'Parsing Error'
                return price

        elif website_name == 'ozon':
            silenium_result = get_data_with_selenium(url)  #
Парсинг
            if silenium_result == -1:
                return -1
            soup = bs(silenium_result, 'lxml')
            result = soup.find_all('h2', class_='e7z1')  #
Проверяем товар на наличие
            if result != []:
                return 'Товар временно отсутствует в продаже'
            else:
                try:
                    result = soup.find('span', class_='c2h5').text
# Ищем тег span с указанным классом
                except AttributeError:
                    return 'Parsing Error'
                price = ''
                for letter in result:
                    for i in str(letter):
                        if '0' <= i <= '9':
                            price += i
                if price == '':
                    price = 'Parsing Error'
                return price
```

```
        elif website_name == 'eldorado':
            silenium_result = get_data_with_selenium(url)  #
Парсинг
            if silenium_result == -1:
                return -1
            soup = bs(silenium_result, 'lxml')
            result = soup.find_all('script',
type='text/javascript')  # Поиск скриптов типа 'text/javascript'

            result_text = ''
            for i in result:
                result_text += str(i)

            # Поиск переменной var dataLayer во всех полученных
скриптах, содержащей словарь с ценой и наличием
            for i in range(len(result_text)):
                if result_text[i] == 'v' and result_text[i + 1] ==
'a' and result_text[i + 2] == 'r' and \
                        result_text[i + 3] == ' ' and result_text[i
+ 4] == 'd' and result_text[i + 5] == 'a' and \
                            result_text[i + 6] == 't' and
result_text[i + 7] == 'a' and result_text[i + 8] == 'L' and \
                                result_text[i + 9] == 'a' and
result_text[i + 10] == 'y' and result_text[i + 11] == 'e' and \
                                    result_text[i + 12] == 'r' and
result_text[i + 13] == ' ' and result_text[i + 14] == '=':

                    # Сохранение позиции начала словаря
                    position = i + 17
                    vocab = ''

                    # Проверка на окончание словаря
                    while result_text[position] != ']':
                        vocab += result_text[position]
                        position += 1

                    # Превращение полученного текста в словарь
                    vocab = eval(vocab)

                    try:
                        if vocab['productAvailability'] ==
'not_available':
                            return 'Товар временно отсутствует в
продаже'
                        else:
                            try:
                                price =
vocab['ecommerce']['detail']['products']['price']
                            except Exception:
                                return 'Parsing Error'
                            return price
                    except Exception:
                        return 'Parsing Error'
```

urlcheck.py

```
import requests
from fake_useragent import UserAgent


def processing(silk):
    """
    Функция проверяет полученную строку на то, является ли она
ссылкой на товар на одном из
    указанных сайтов

    :param silk: Строка
    :return: Строка с названием магазина
    :return: -1 в случае ошибки
    """
    if silk.startswith("https://www"):
        response = requests.get(silk, headers={'User-Agent':
UserAgent().chrome})  # Отправка запроса сайту
        if response.status_code != 200:  # Статус 200 состояния
HTTP значит, что получен ответ от сервера
            return -1
        if "https://www.mvideo.ru/products/" in silk:
            return "mvideo"
        elif "https://www.dns-shop.ru/product/" in silk:
            return "dns"
        elif "https://www.ozon.ru/product/" in silk:
            return "ozon"
        elif "https://www.citilink.ru/product/" in silk or
"https://www.citilink.ru/amp/product/" in silk:
            return "citilink"
        elif "https://www.eldorado.ru/cat/detail/" in silk:
            return "eldorado"
        else:
            return -1
    else:
        return -1
```