**Федеральное государственное бюджетное образовательное учреждение высшего образования «Ярославский государственный университет им. П.Г. Демидова»**

Факультет: «Информатики и вычислительной техники»

Кафедра: «Дискретного анализа»

КУРСОВАЯ РАБОТА

Тема: «Разработка VPN и Telegram бота для защиты персональных данных и обеспечения конфиденциальности»

Студент: Кузнецов Александр Васильевич

Группа: ИВТ-31БО

Научный руководитель:  
Богомолов Юрий Викторович

**Ярославль – 2023г.**

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc138420891)

[ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 4](#_Toc138420892)

[1.1 Принцип работы VPN. 4](#_Toc138420893)

[1.2 Wireguard. 5](#_Toc138420894)

[ЗАКЛЮЧЕНИЕ 14](#_Toc138420895)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ: 16](#_Toc138420896)

[ПРИЛОЖЕНИЯ 17](#_Toc138420897)

# ВВЕДЕНИЕ

В современном обществе уже почти каждый человек повседневно использует в своей жизни интернет. Масса людей, использующих его для работы, общения и хранения личных данных, с каждым разом становится все больше. Помимо этого, также с каждым годом растет и количество утечек информации и кибератак на пользователей. Так, одним из важных вопросов, интересующих пользователей, является вопрос о защите их персональных данных и обеспечения конфиденциальности, поскольку утечка личной информации может привести к серьезным последствиям, таким как : кража личных сбережений, мошенничество, киберпреследование и другое.

Актуальность работы заключается в том, что создание надежных механизмов защиты является одним из приоритетных направлений развития информационных технологий. Для того, чтобы пользователи не сталкивались с проблемами при использовании интернета без защиты, стоит подчеркнуть необходимость их создания.

Целью выполнения курсовой работы является изучение и создание системы защиты персональных данных для обеспечения конфиденциальности в интернете.

Цель обусловила постановку и последовательное решение в работе следующих задач:

1. Рассмотреть принцип работы VPN.
2. Изучить Wireguard.
3. Арендовать и настроить сервер.
4. Организовать базу данных пользователей с информацией о их подписках.
5. Создать Telegram бота, который бы выдавал файлы пользователю, необходимые для работы VPN.

# ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

# 1.1 Принцип работы VPN.

VPN (Virtual Private Network) - это технология, которая позволяет создавать безопасное зашифрованное подключение пользователя к сети, с которым он может обходить локальные ограничения и сохранять конфиденциальность. VPN создает виртуальное пространство для работы в сети, объединяя локальные сети или отдельные машины, подключенные к сети общего пользования, в единую сеть, обеспечивающую секретность и целостность передаваемой по ней информации. Рассмотрим основные задачи VPN, его плюсы и минусы.

Основные задачи и плюсы VPN включают:

* Скрытие личности в интернете: VPN скрывает IP-адрес пользователя, что позволяет обеспечить анонимность в сети.
* Обход географических ограничений: VPN позволяет обходить геоблокировку и получать доступ к заблокированным сайтам и интернет-контенту.
* Защита данных: VPN защищает данные пользователя в интернете от хакеров или правительственной слежки путем шифрования интернет-трафика и сокрытия реального IP-адреса.
* Удаленный доступ: VPN позволяет создать частную удаленную сеть избранных устройств, которая идеально подходит для удаленных работников. Благодаря защищенному соединению команда будет оставаться на связи, даже если ее участников разделяют километры или континенты.
* Конфиденциальность: VPN-сервисы используются для безопасной отправки данных в Интернете. Если не используется виртуальная частная сеть, то третьи лица могут записывать и продавать персональные данные пользователя, такие как пароли, данные кредитной карты и журнал браузера. Для сохранения этой конфиденциальной информации в тайне, особенно при подключении с помощью общественных сетей Wi-Fi, сети VPN используют шифрование.

Недостатки VPN:

* Замедление скорости интернет-соединения: VPN может замедлять скорость интернет-соединения из-за шифрования и дополнительной обработки данных.
* Ограниченный выбор серверов: некоторые VPN-сервисы могут иметь ограниченный выбор серверов, что может привести к проблемам с подключение.
* Стоимость: некоторые VPN-сервисы могут быть дорогими.

Исходя из выше сказанного, можно сделать вывод, что VPN - это полезный инструмент для обеспечения безопасности и конфиденциальности в интернете, а также для обхода географических ограничений и получения доступа к заблокированным сайтам и интернет-контенту. Однако, при выборе VPN-сервиса необходимо учитывать его преимущества и недостатки, а также стоимость и доступность серверов.

# 1.2 Wireguard.

WireGuard VPN - это новый протокол виртуальной частной сети, который был разработан для замены IPsec и OpenVPN. WireGuard является свободным и открытым программным обеспечением с открытым исходным кодом, который реализует зашифрованные виртуальные частные сети (VPN). WireGuard использует современную криптографию, такую как Noise protocol framework, Curve25519, ChaCha20, Poly1305, BLAKE2, SipHash24, HKDF и т.д. . Он быстрее и проще в использовании, чем IPsec и OpenVPN, и имеет меньшую поверхность атаки. WireGuard также имеет компактный и читаемый код, что делает его проще исследовать на уязвимости.

Noise framework - это фреймворк для криптографических протоколов, основанных на соглашении о ключе Диффи-Хеллмана. Noise framework может описывать протоколы, состоящие из одного или нескольких этапов, которые могут включать в себя шифрование, аутентификацию, скрытие идентичности и другие функции. WireGuard использует Noise framework для создания ключей, которые используются для шифрования трафика между устройствами. Noise framework использует современные алгоритмы шифрования, такие как Curve25519, ChaCha20 и Poly1305, которые обеспечивают безопасность передачи данных.Noise framework также поддерживает взаимную аутентификацию и скрытие идентичности, что делает WireGuard более безопасным и защищенным. Noise framework также обеспечивает минимальную поверхность атаки, что делает WireGuard менее уязвимым к атакам.

WireGuard использует принципы шифрования и аутентификации для защиты данных, передаваемых через VPN-туннель. WireGuard использует публичные и приватные ключи для шифрования и расшифровки данных, а также для аутентификации и проверки целостности данных. WireGuard берет на себя смелость принимать некоторые решения за пользователя, что делает программу не такой гибкой, как IPsec и OpenVPN, но зато позволяет ей быть более безопасной.

WireGuard также обладает высокой производительностью и низкой задержкой, что делает его идеальным для использования в сетях с высокими требованиями к скорости передачи данных. WireGuard также имеет кроссплатформенную простоту использования и может быть использован на Android, iOS, Linux, FreeBSD, NetBSD, OpenBSD, macOS, Windows 7+ и других операционных системах.

В целом, WireGuard VPN является быстрым, безопасным и простым в использовании протоколом виртуальной частной сети, который может быть использован на различных операционных системах. Он использует современную криптографию, имеет компактный и читаемый код, а также обладает высокой производительностью и низкой задержкой. WireGuard может быть использован для защиты данных, передаваемых через VPN-туннель, и может быть использован в сетях с высокими требованиями к скорости передачи данных.

# ПРАКТИЧЕСКАЯ ЧАСТЬ

3. Настройка и аренда сервера.

Для работы моего бота необходим сервер, но перед тем, как его арендовать и запускать на нем бота, нужно протестировать все на локальной машине. Для запуска сервера на локальной машине, я использовал Hyper-V.  
Так как у меня установлена операционная система Windows 10, а нужен был сервер на Linux, то Hyper-V – это отличное решение. Сначала скачивается образ диска операционной системы Ubuntu версии 22.04.2 LTS для сервера. Он предназначен для установки операционной системы на сервер и содержит все необходимые файлы для этого. Далее в Hyper-V создаётся новая виртуальная машина и в ней производится дальнейшая установка операционной системы. Далее идет обычная установка, я пользовался руководством найденном в интернете.

Когда вся установка операционной системы завершена, нужно установить на сервер Wireguard. Вводим команду «apt install -y wireguard» (Приложение 1). Далее нам необходимо сгенерировать приватный и публичный ключ сервера:

* «wg genkey | tee /etc/wireguard/privatekey | wg pubkey | tee /etc/wireguard/publickey» (Приложение 1). Эта команда генерирует новый приватный ключ, сохраняет его в файл /etc/wireguard/privatekey, вычисляет соответствующий ему публичный ключ, сохраняет его в файл /etc/wireguard/publickey и выводит оба ключа на экран. Следующая команда - «chmod 600 /etc/wireguard/privatekey» делает возможность просмотра этого файла только для нас.

Теперь нам нужно сделать конфигурационный файл для сервера. После выполнения команды «nano /etc/wireguard/wg0.conf» откроется файл wg0.conf в редакторе nano, где можно редактировать настройки сервера WireGuard. В файле wg0.conf содержатся настройки интерфейса WireGuard, такие как приватный ключ, адреса IP-адреса и настройки пиров (Приложение 2). «echo "net.ipv4.ip\_forward=1" >> /etc/sysctl.conf» эта команда добавляет строку net.ipv4.ip\_forward=1 в конец файла /etc/sysctl.conf с помощью оператора >>.

После этого, изменения в файле /etc/sysctl.conf вступят в силу после перезагрузки системы или после выполнения команды «sysctl -p», которая перечитывает файл /etc/sysctl.conf и применяет изменения. IP-форвардинг позволяет Linux пересылать сетевые пакеты между различными сетевыми интерфейсами и сетями.

Далее выполняем три команды:

* «systemctl enable wg-quick@wg0.service» - включает автозапуск службы WireGuard при загрузке системы.
* «systemctl start wg-quick@wg0.service» - запускает службу WireGuard.
* «systemctl status wg-quick@wg0.service» - показывает статус службы WireGuard (Приложение 3).

Теперь нужно создать приватные и публичные ключи для клиента нашего VPN. Для этого вводим команду - «wg genkey | tee /etc/wireguard/user\_1privatekey | wg pubkey | tee /etc/wireguard/user1\_publickey». Далее заходим в конфигурацию нашего сервера и добавляем соответствующего пользователя: «nano /etc/wireguard/wg0.conf» (Приложение 4). Дальше выполняем следующие команды:

* «systemctl restart wg-quick@wg0» - перезапускает службу WireGuard.
* «systemctl status wg-quick@wg0» - показывает статус службы WireGuard (Приложение 5).

Все основные шаги по настройке сервера были сделаны, осталось только сделать сам конфигурационный файл, который будет выдаваться пользователю. Такой файл можно сделать в любом текстовом редакторе (Приложение 6).

После того, как файл был настроен, можем заходить в WireGuard. В нем мы нажимаем на кнопку «Добавить туннель» и выбираем файл конфигурации. Теперь наш VPN работает и наше соединение безопасно. В качестве доказательства работы Wireguard заходим на сайт, где можно посмотреть свой ip-адрес. Вот такой ip у нас без включенного VPN (Приложение 7), а вот такой уже после его включения (Приложение 8).

4. Организация базы данных.

Поскольку подписка на VPN подразумевает действие её в период какого-то времени, то возникает необходимость где-то хранить эти данные.

К тому же, как бот будет знать какой именно пользователь приобрел ту или иную подписку? Для решения этих проблем необходимо организовать базу данных.

В моем проекте используется такая база данных как SQLite. SQLite - это система управления реляционными базами данных, которая является быстрой, легковесной и простой в использовании. Она не требует установки сервера и может быть встроена в приложение, что упрощает ее использование. Вся база данных хранится в одном файле на диске, что облегчает ее переносимость и управление.

SQLite поддерживает SQL-запросы, что облегчает поиск и извлечение информации из базы данных. Она также поддерживает транзакции, что позволяет обеспечить целостность данных и избежать ошибок при одновременном доступе к базе данных. SQLite является автономным и не имеет внешних зависимостей, что делает ее удобной для использования в различных приложениях.

 В целом, SQLite - это простая и эффективная система управления реляционными базами данных, которая может быть использована для хранения информации в различных приложениях, включая Telegram ботов.

В базе данных есть таблица (configs), которая содержит информацию о самой подписке.

Таблица 1 - configs

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| name\_id | config | is\_use | cost | time | user\_id |
|  |  |  |  |  |  |

Данные поля:

* name\_id (тип данных INTEGER) – уникальный идентификатор самого файла;
* config (тип данных BLOB) – хранит в себе необходимый файл для Wireguard;
* is\_use (тип данных INT) – используется ли текущий файл кем-либо;
* cost (тип данных INT) – цена за которую пользователь приобрел подписку;
* time (тип данных INT) – время окончания подписки;
* user\_id (тип данных TEXT) - уникальный идентификатор пользователя в Telegram.

Всё управление базой данных организовано в файле sql.py. В приложении будет пояснение к каждой функции (Приложение 9). В текущей реализации, файлы в таблице формируются вручную, но в планах, конечно, автоматизировать этот процесс.

5. Создание Telegram бота.

Telegram-боты могут быть написаны на различных языках программирования, но одним из популярных выборов является Python с использованием библиотеки aiogram. Ниже перечислены причины, почему aiogram является хорошим выбором для создания Telegram-ботов на Python:

* Aiogram является асинхронной библиотекой, что позволяет обрабатывать несколько запросов одновременно и работать быстрее, чем синхронные библиотеки. Это означает, что при использовании aiogram бот может обрабатывать несколько запросов почти одновременно, что улучшает производительность и быстродействие бота
* Aiogram имеет простой и интуитивно понятный API, что упрощает создание и управление Telegram-ботами.
* Библиотека предоставляет широкий спектр функций, включая поддержку встроенных запросов, редактирование сообщений и загрузку файлов.
* Aiogram имеет хорошую документацию и активное сообщество, что облегчает поиск помощи и поддержки при необходимости.
* Aiogram поддерживает машину состояний, что позволяет создавать более сложные боты с различными состояниями и переходами между ними.
* Aiogram также поддерживает веб-хуки, что позволяет боту получать обновления в режиме реального времени.

Из вышеперечисленного можно сделать вывод что, aiogram является хорошим выбором для создания Telegram-ботов на Python благодаря своей асинхронной природе, простому API, широкому спектру функций и поддержке машины состояний и веб-хуков.

Перейдем к созданию самого бота. Первым делом необходимо зайти в Telegram и написать боту BotFather. Вводим команду /newbot и придумываем имя нашему боту. Далее нужно ввести уникальный идентификатор для бота, по которому другие пользователи смогут его находить. После того, как уникальное имя было придумано, и одобрено, BotFather высылает нам сообщение об удачном создании бота и выдает уникальный секретный токен бота, который нам и нужен для отправки запросов к API Telegram.

Первый этап пройден. Теперь необходимо приступить к написанию кода на Python. Импортируем в наш проект все необходимые модули, такие как – Bot, Dispatcher, execotur, types из библиотеки aiogram. (Приложение 11) Импорт этих модулей необходим для создания Telegram-бота на Python. Bot используется для создания экземпляра бота и взаимодействия с Telegram Bot API. Dispatcher используется для обработки входящих обновлений от сервера Telegram и маршрутизации их к соответствующим обработчикам. Executor используется для запуска бота и обработки обновлений асинхронно. Types содержит классы для всех типов объектов Telegram API, таких как сообщения, чаты и пользователи.

Создаем файл botToken.py и записываем в него тот самый токен, который выдал на BotFather. Теперь можно перейти непосредственно к написанию функционала бота.

Сначала опишем основные функции, которые будут использоваться в проекте:

1. @dp.message\_handler(commands=['''']) - это декоратор в библиотеке aiogram, который используется для реагирования на входящие сообщения. Он указывает, что функция, которая следует за декоратором, будет вызываться, когда бот получит сообщение. В качестве аргумента декоратор может принимать различные параметры, такие как текстовые команды, регулярные выражения и другие фильтры, чтобы определить, какие сообщения должны вызывать функцию.
2. @dp.callback\_query\_handler(text='') - это декоратор в библиотеке aiogram, который используется для обработки callback-запросов. Он указывает, что функция, которая следует за декоратором, будет вызываться, когда пользователь нажимает на кнопку с текстом. Сами кнопки были сделаны в отдельном файле. (Приложение 10) Callback-запросы - это способ взаимодействия пользователя с ботом, который позволяет пользователю отправлять запросы на сервер бота, не отправляя сообщения.
3. @dp.pre\_checkout\_query\_handler() - это декоратор в библиотеке aiogram, который используется для обработки pre-checkout запросов. Pre-checkout запросы - это запросы, которые отправляются при оформлении платежа пользователем. Декоратор указывает, что функция, которая следует за декоратором, будет вызываться, когда бот получает pre-checkout запрос. Функция, которая следует за декоратором, должна принимать объект PreCheckoutQuery и возвращать True или False. Если функция возвращает True, то бот отправляет запрос на подтверждение платежа. Если функция возвращает False, то бот отправляет сообщение об ошибке и отменяет платеж.

В моей работе бот имеет следующий функционал:

При вводе команды ''/start'' бот начинает свою работу и отправляет приветственное сообщение (Приложение 12), с последующим выбором действий:

1. Ознакомление с инструкцией по использованию VPN (Приложение 13).
2. Выбор интересуемой подписки (на 1 месяц или на 6) (Приложение 12, 13).
3. Выставление счета для оплаты интересуемой его подписки (Приложение 12, 13).
4. В случае удачной оплаты, информирование пользователя об успешной операции, выдача необходимого файла, добавление данного пользователя в базу данных (Приложение 15).
5. Предоставление информации о приобретённых подписках на VPN и сроках её действия (Приложение 14).

Все управление ботом сделано максимально просто для пользователя. После того, как он запускает бота, под каждым сообщением от бота появляется мини-клавиатура с кнопками (Приложение 10), которые отвечают за действия приведенные выше. Так же бот удаляет предыдущие сообщения, которые не несут важной информации, чтобы переписка не засорялась.

# ЗАКЛЮЧЕНИЕ

В ходе курсовой работы мной были рассмотрены вопросы разработки VPN и Telegram бота для защиты персональных данных и обеспечения конфиденциальности. Был настроен сервер под VPN WireGuard, изучены основы языка программирования Python и Telegram API, а также организована база данных.

Для защиты персональных данных и обеспечения конфиденциальности был выбран VPN-протокол WireGuard, который является быстрым, современным и безопасным. Был настроен сервер под этот протокол, что позволит пользователям обеспечить безопасный доступ к интернету.

Для разработки Telegram бота была использована библиотека aiogram для работы с Telegram API. Бот был создан на языке программирования Python, основы которого были изучены в ходе этой курсовой работы. Он позволяет пользователям получить подписку и всегда иметь актуальную информацию о её состоянии.

Для организации базы данных были изучены основы работы с БД. База данных позволяет хранить информацию упорядоченно и структурированно, что упрощает работу с данными и позволяет извлекать необходимые данные в любом сочетании. База данных используется для хранения информации о пользователях, которые используют VPN и Telegram бота.

Однако, необходимо отметить, что VPN и Telegram бот также имеют свои недостатки на текущий момент. Например :

* Недостаточная защита базы данных может привести к утечке конфиденциальной информации.
* Ручное формирование конфигурационных файлов для WireGuard может быть трудоемким и требовать определенных знаний.
* Тестовое состояние проекта, пока что он не готов к использованию в больших масштабах.

Но в будущем планируется доработать проект и устранить эти недостатки. Например, нужно будет изучить более надежную систему защиты базы данных, автоматизировать формирование конфигурационных файлов для WireGuard и провести более тщательное тестирование проекта перед его использованием в реальных условиях.

Таким образом, создание VPN и Telegram бота для защиты персональных данных и обеспечения конфиденциальности является актуальной и важной задачей в настоящее время. Использование VPN-протокола WireGuard, языка программирования Python и библиотеки aiogram позволяет создать надежное и безопасное решение для защиты персональных данных в интернете.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ:

<https://openvpn.net/faq/what-is-and-how-do-i-enable-ip-forwarding-on-linux/> - OpenVPN.

<https://help.stack-it.ru/202104/installation/Install_vue/01_install_ubuntu_posgresql.html> - Установка Ubuntu Server 18.04 на виртуальную машину Hyper-V.

<https://timeweb.com/ru/community/articles/vds-na-hyper-v-preimushchestva-i-nedostatki> - VDS на Hyper-V: преимущества и недостатки.

<https://docs.aiogram.dev/> - Документация aiorgam.

<https://habr.com/ru/articles/552788> - Подключение Sqlite3 к Telegram боту.

<https://docs.python.org/3/library/sqlite3.html> - Документация Sqlite3.

<https://proglib.io/p/20-qa-sqlite> - Подробности про SQLite.

<https://www.wireguard.com> – Официальный сайт WireGuard.

<https://ru.wikipedia.org/wiki/WireGuard> - дополнительная информация о WireGuard.

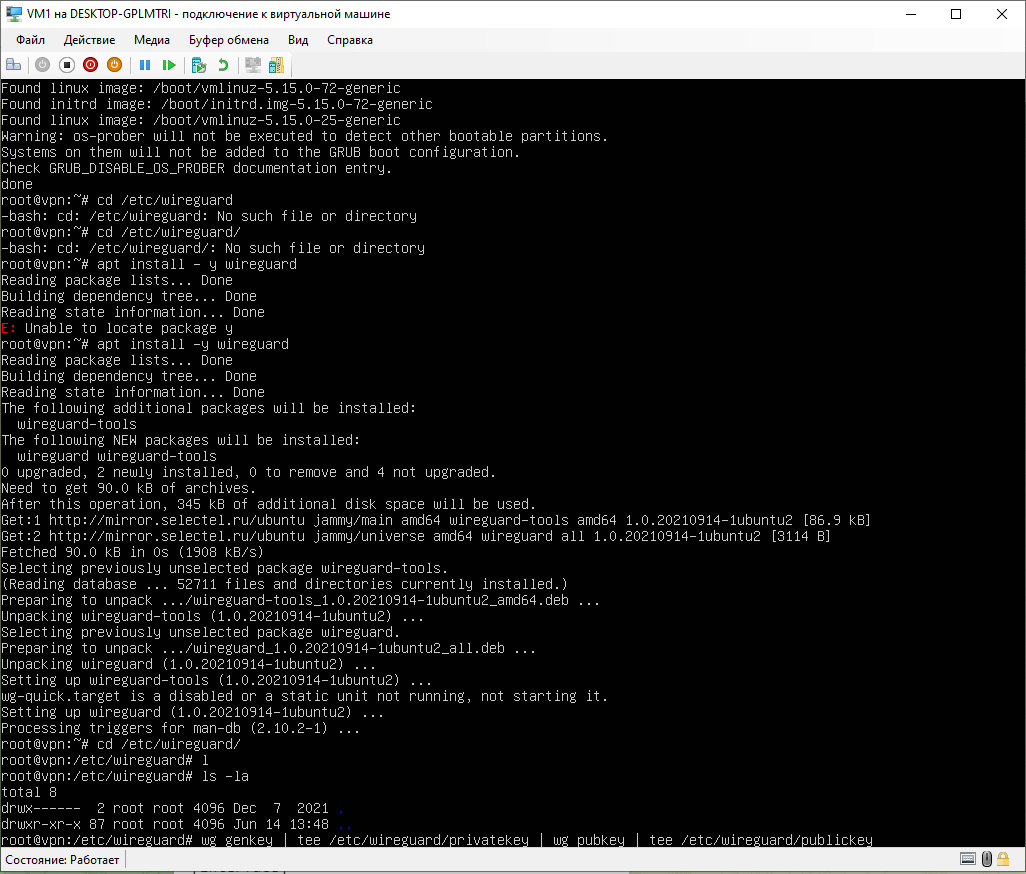
<https://yookassa.ru/developers> - Документация для приёма платежей.

<https://ru.wikipedia.org/wiki/VPN> - Информация про VPN.

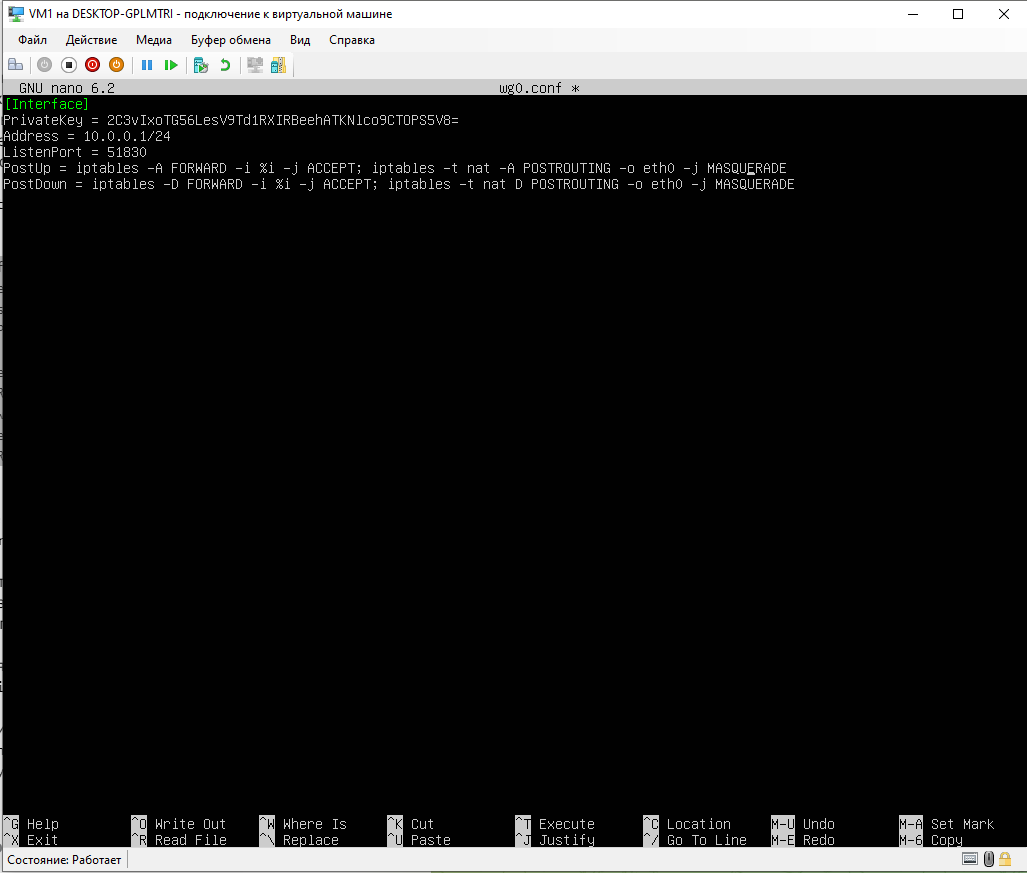
<https://habr.com/ru/articles/534250> - Информация про VPN.

# ПРИЛОЖЕНИЯ

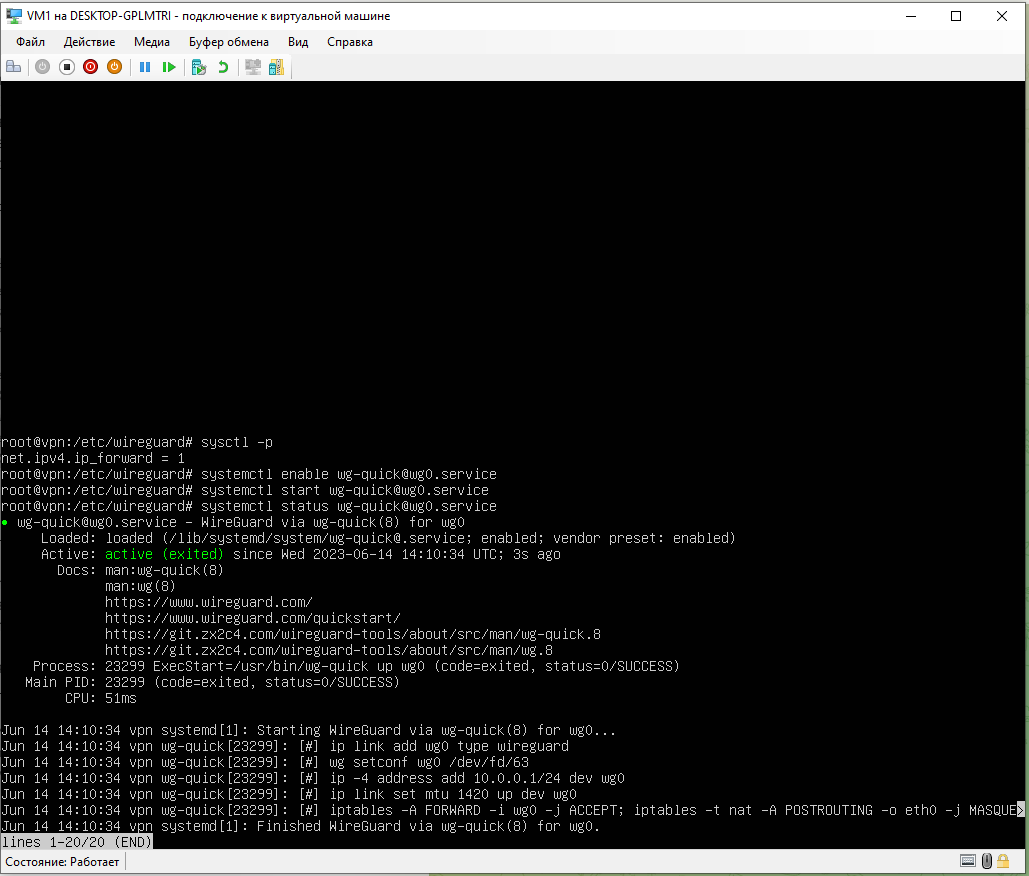
Приложение 1



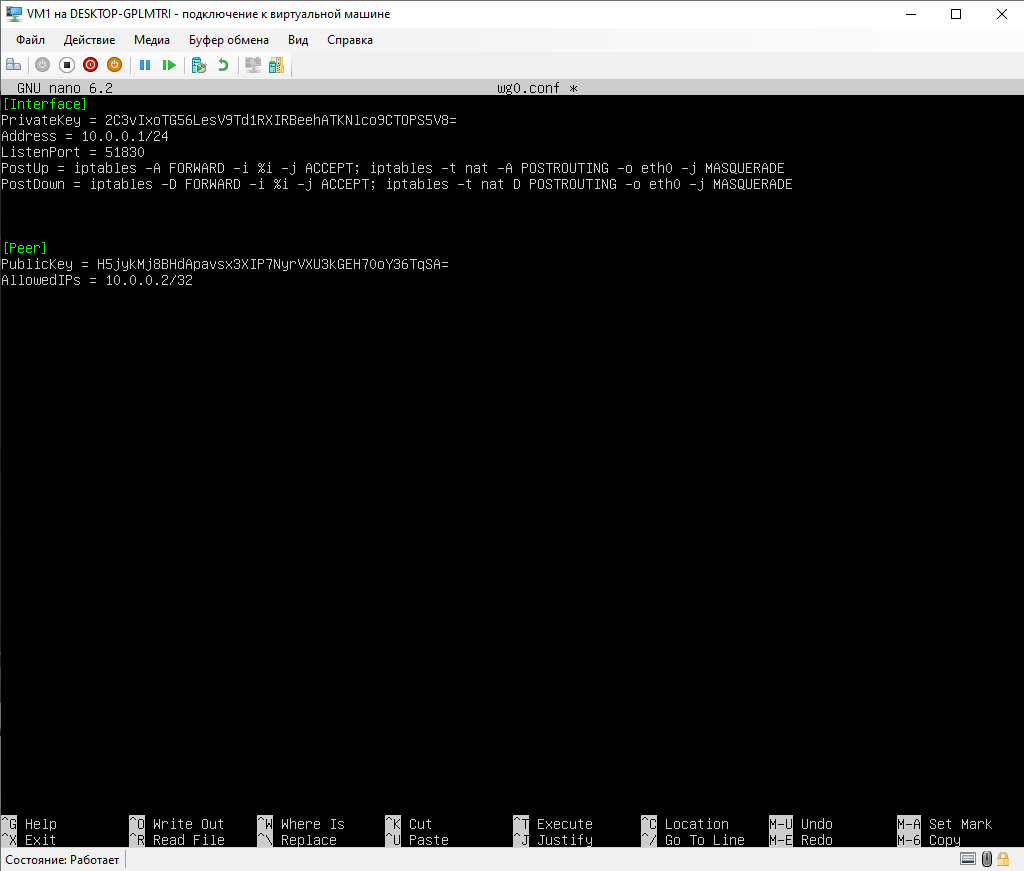
Приложение 2



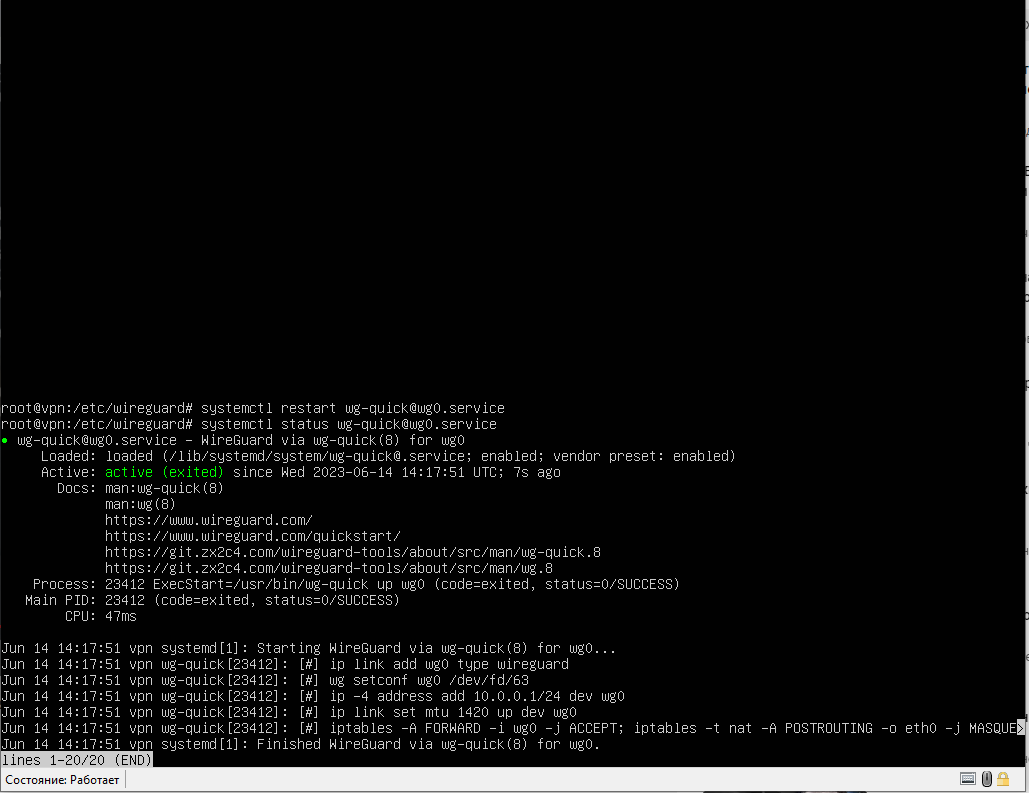
Приложение 3



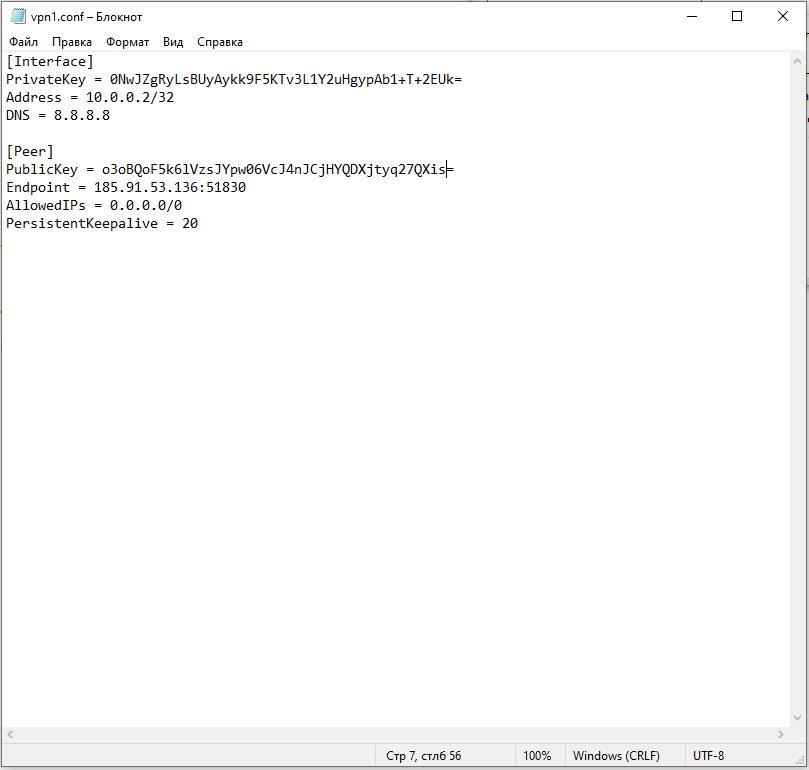
Приложение 4



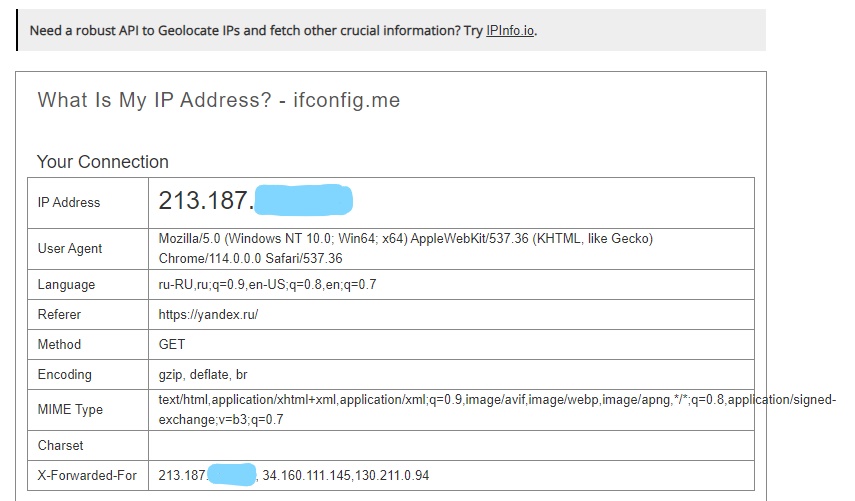
Приложение 5



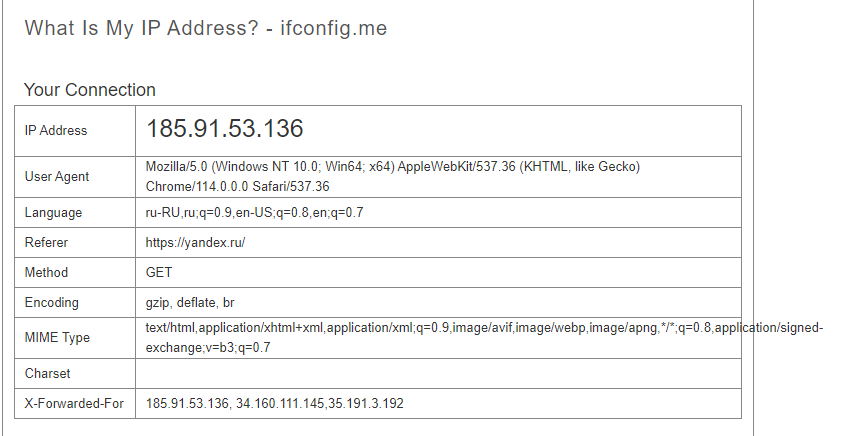
Приложение 6



Приложение 7



Приложение 8



Приложение 9

import sqlite3 as sq  
async def db\_start():  
 global db2, cur2  
  
 db2 = sq.connect('PeersDb.db') #Установления соединения с базой данных  
 cur2 = db2.cursor() #Создание объекта курсора для выполнения операций с базой данных  
 cur2.execute("CREATE TABLE IF NOT EXISTS configs(name\_id INTEGER PRIMARY KEY, config BLOB, is\_use INT, cost INT, time INT, user\_id TEXT)")  
  
 db2.commit() #Сохранение изменений в базе данных  
  
def get\_conf(isuse): #Функция для проверки наличия доступных файлов в базе данных  
 cur2.execute("SELECT \* FROM configs")  
 for row in cur2.fetchall(): #Цикл для обработки строки в которой содержится информация о доступности файла  
 if row[2] == 0: #Берется строка, в которой записана информация о состоянии доступности файла  
 cur2.execute("UPDATE configs SET is\_use = {} WHERE name\_id = {}".format(isuse, row[0]))  
 db2.commit()  
 return row[1] #Возвращается файл  
 return None  
  
def get\_sub\_time(user\_id, time\_now): #Функция для информирования пользователя о его подписке  
 subs\_list = []  
 cur2.execute("SELECT \* FROM configs")  
 for row in cur2.fetchall():  
 if row[5] == user\_id:  
 s =''  
 days = 24\*60\*60  
 s = s + f'✔Файл {row[1]} будет действовать ещё {(row[4] - time\_now) // days} дней'  
 subs\_list.append(s)  
  
 if len(subs\_list) > 0:  
 return subs\_list  
 else:  
 return None

Продолжение Приложения 9

def get\_file(isuse, user\_id, cost, time): #Функция, которая выдает файл пользователю  
 cur2.execute("SELECT \* FROM configs")  
 for row in cur2.fetchall():  
 if row[2] == 0:  
 cur2.execute("UPDATE configs SET is\_use = {}, cost ={}, time ={}, user\_id ={} WHERE name\_id = {}".format(isuse, cost, time, user\_id, row[0]))

db2.commit()  
 return row[1]  
 return None

Приложение 10

from aiogram.types import InlineKeyboardMarkup, InlineKeyboardButton  
  
kb\_catalog = InlineKeyboardMarkup(row\_width=2)  
One\_Month\_Sub\_Butt = InlineKeyboardButton("💸Месяц 100 руб💸", callback\_data="On\_Month\_Pay")  
Six\_Month\_Sub\_Butt = InlineKeyboardButton("🔥Полгода 400 руб🔥", callback\_data="Six\_Month\_Pay")  
Back\_MenuButt = InlineKeyboardButton(text="Меню", callback\_data="Menu")  
Sub\_Info\_Butt = InlineKeyboardButton(text="Состояние моих подписок", callback\_data="sub\_info")  
kb\_catalog.add(One\_Month\_Sub\_Butt, Six\_Month\_Sub\_Butt, Back\_MenuButt, Sub\_Info\_Butt)  
  
kb\_start = InlineKeyboardMarkup(row\_width=2)  
MenuButt = InlineKeyboardButton(text="Меню", callback\_data="Menu")  
Documentation\_Butt = InlineKeyboardButton(text="Инструкция", callback\_data="Doc")  
Catalog\_Butt = InlineKeyboardButton(text="Посмотреть подписки", callback\_data="Catalog")  
kb\_start.add(MenuButt, Documentation\_Butt, Catalog\_Butt)  
  
kb\_return = InlineKeyboardMarkup(row\_width=2)  
Return\_Butt = InlineKeyboardButton(text="Отмена", callback\_data="Cancel")  
kb\_return.add(Return\_Butt)

Приложение 11

import logging  
  
import KeyBord  
  
from aiogram import Bot, Dispatcher, executor, types  
from aiogram.types import CallbackQuery, ContentType  
  
import botToken  
from botToken import token  
  
from sql import db\_start, get\_conf, get\_file, get\_sub\_time  
  
import time  
import datetime  
  
kb = KeyBord  
  
YOOTOKEN = '1744374395:TEST:7584325955b32d8f177f'  
  
async def on\_startup(\_):  
 await db\_start()  
  
def days\_to\_seconds(days):  
 return days \* 24 \* 60 \* 60  
  
def subscription\_balance(get\_time):#Функция для получения информации о длительности подписки  
 time\_now = int(time.time())  
 middle\_time = get\_time - time\_now  
 if middle\_time < 0:  
 return False  
 else:  
 return str(datetime.timedelta(seconds=middle\_time))

Приложение 12

def tg\_bot\_2(token):  
 logging.basicConfig(level=logging.INFO)#Настраивание вывод логов на уровне INFO, что означает, что будут выводиться только сообщения с уровнем INFO и выше  
 bot = Bot(token=botToken.token)#Создает экземпляр класса Bot с использованием токена  
 dp = Dispatcher(bot) #Этот экземпляр класса используется для регистрации обработчиков сообщений и запуска бота  
  
 @dp.message\_handler(commands=["start"]) #Обработка команды /start  
 async def start\_message(message: types.message):  
 start\_text = "Привет!🖐\nЯ бот для тестовой покупки подписки на ВПН\nВыберите действие:"  
 await message.answer(start\_text)#Пользователю отправляется сообщение  
 await message.answer("Выберите Подписку:", reply\_markup=kb.kb\_start)#Вывод клавиатуры под сообщением  
  
 @dp.callback\_query\_handler(text='On\_Month\_Pay')#Обработка нажатия на кнопку "Месяц 100 руб"  
 async def pay(call: CallbackQuery):  
 if call.message.message\_id:  
 await bot.delete\_message(call.from\_user.id, call.message.message\_id) #Удаление предыдущего сообщения  
 if get\_conf(0): #Проверка, есть ли свободные файлы, в случае, если они есть, пользователю отправляется счет для оплаты,  
 # иначе выводится соответствующее сообщение с клавиатурой для выбора дальнейшего действия  
 await bot.send\_invoice(chat\_id=call.from\_user.id,  
 title="Подписка на 1 месяц ВПН",  
 description="Тестовое описание товара",  
 payload="first",  
 provider\_token=YOOTOKEN,  
 currency="RUB",  
 start\_parameter="test\_bot",  
 prices=[types.LabeledPrice(label='Test', amount=10000)]  
 )  
 else:  
 await bot.send\_message(call.from\_user.id, "К сожалению файлы закончились!\nОбратитесь в поддержку", reply\_markup=kb.kb\_start)

Приложение 13

@dp.callback\_query\_handler(text='Six\_Month\_Pay')#Обработка нажатия на кнопку "Пол года 400 руб"  
 async def pay1(call: CallbackQuery):  
 if call.message.message\_id:  
 await bot.delete\_message(call.from\_user.id, call.message.message\_id) #Удаление предыдущего сообщения  
 if get\_conf(0): #Проверка, есть ли свободные файлы, в случае, если они есть, пользователю отправляется счет для оплаты,  
 # иначе выводится соответствующее сообщение с клавиатурой для выбора дальнейшего действия  
 await bot.send\_invoice(chat\_id=call.from\_user.id,  
 title="Подписка на полгода ВПН",  
 description="Тестовое описание товара",  
 payload="second",  
 provider\_token=YOOTOKEN,  
 currency="RUB",  
 start\_parameter="test\_bot",  
 prices=[types.LabeledPrice(label='Test', amount=40000)]  
 )  
 else:  
 await bot.send\_message(call.from\_user.id, "К сожалению файлы закончились!\nОбратитесь в поддержку",  
 reply\_markup=kb.kb\_start)  
  
 @dp.callback\_query\_handler(text='Doc')#Обработка нажатия на кнопку "Инструкция"  
 async def get\_documentation(call: CallbackQuery):  
 if call.message.message\_id:  
 await bot.delete\_message(call.from\_user.id, call.message.message\_id)#Удаление предыдущего сообщения  
 await bot.send\_message(call.from\_user.id, "Здесь будет инструкция")#Отправка инструкции  
 await bot.send\_message(call.from\_user.id, "Доступные подписки:", reply\_markup=kb.kb\_catalog)  
  
 @dp.callback\_query\_handler(text='Cancel')#Обработка нажатия на кнопку "Меню"  
 async def cancel(call: CallbackQuery):  
 if call.message.message\_id:  
 await bot.delete\_message(call.from\_user.id, call.message.message\_id)#Удаление предыдущего сообщения  
 await bot.send\_message(call.from\_user.id, "Доступные подписки:", reply\_markup=kb.kb\_catalog)

Приложение 14

@dp.callback\_query\_handler(text='sub\_info')#Обработка нажатия на кнопку "Состояние моих подписок"  
 async def get\_sub\_info(call: CallbackQuery):  
 if call.message.message\_id:  
 await bot.delete\_message(call.from\_user.id, call.message.message\_id)  
  
 if get\_sub\_time(str(call.from\_user.id), int(time.time())): #Вызов функции для получения информации о длительности подписки  
 subs\_status = get\_sub\_time(str(call.from\_user.id), int(time.time()))  
 s =''  
 for el in subs\_status:  
 s = s + el + '\n'  
 s = s.replace('Configs/', '')  
 await bot.send\_message(call.from\_user.id, s)  
 else:  
 await bot.send\_message(call.from\_user.id, "У вас еще нет активных подписок😢")  
  
 await bot.send\_message(call.from\_user.id, "Доступные подписки:", reply\_markup=kb.kb\_catalog)  
  
 @dp.callback\_query\_handler(text='Catalog')#Обработка нажатия на кнопку "Посмотреть подписки"  
 async def get\_catalog(call: CallbackQuery):  
 if call.message.message\_id:  
 await bot.delete\_message(call.from\_user.id, call.message.message\_id)  
 await bot.send\_message(call.from\_user.id, "Доступные подписки:", reply\_markup=kb.kb\_catalog)#Отправка сообщения с клавиатурой, на который представлены подписки

Приложение 15

@dp.message\_handler(content\_types=ContentType.SUCCESSFUL\_PAYMENT)#Обработка успешного платежа  
 async def successful\_payment2(message: types.Message):  
 if message.successful\_payment.invoice\_payload == "first":#Для подписки на 30 дней  
 await bot.send\_message(message.chat.id,  
 f"✅Платёж на сумму {message.successful\_payment.total\_amount // 100} {message.successful\_payment.currency} прошел успешно!✅")  
 try:  
 time\_sub = int(time.time() + days\_to\_seconds(30))  
 await message.answer("Отлично!\nВот ваш конфигурационный файл:")  
 await message.answer\_document(open(get\_file(1, message.from\_user.id, message.successful\_payment.total\_amount//100, time\_sub)))#Выдача файла пользователю  
 await message.answer(f"Срок действия вашей подписки: 30 дней")  
 await message.answer("Выберите действие:", reply\_markup=kb.kb\_start)  
 except TypeError:  
 await message.answer("К сожалению файлы закончились!\nОбратитесь в поддержку", reply\_markup=kb.kb\_start)  
  
 elif message.successful\_payment.invoice\_payload == "second":# Для подписки на 6 месяцев  
 await bot.send\_message(message.chat.id,  
 f"✅Платёж на сумму {message.successful\_payment.total\_amount // 100} {message.successful\_payment.currency} прошел успешно!✅")  
 try:  
 time\_sub = int(time.time() + days\_to\_seconds(180))  
 await message.answer("Отлично!\nВот ваш конфигурационный файл:")  
 await message.answer\_document(  
 open(get\_file(1, message.from\_user.id, message.successful\_payment.total\_amount // 100, time\_sub)))  
 await message.answer(f"Срок действия вашей подписки: 180 дней")  
 await message.answer("Выберите действие:", reply\_markup=kb.kb\_start)  
 except TypeError:  
 await message.answer("К сожалению файлы закончились!\nОбратитесь в поддержку", reply\_markup=kb.kb\_start)  
 @dp.pre\_checkout\_query\_handler()#Обработка перед оплатой  
 async def pre\_checkout\_query(pre\_checkout\_q: types.PreCheckoutQuery):  
 if get\_conf(0):#Проверка, на наличие доступных файлов  
 logging.info('Received pre-checkout query: %s', pre\_checkout\_query)  
 await bot.answer\_pre\_checkout\_query(pre\_checkout\_q.id, ok=True)  
 else:  
 logging.info('Received pre-checkout query: %s', pre\_checkout\_query)  
 await bot.answer\_pre\_checkout\_query(pre\_checkout\_q.id, error\_message="Файлы закончились!", ok=False)

Приложение 16

@dp.callback\_query\_handler(text='Menu')#Обработка нажатия на кнопку "Меню"  
 async def start\_message(call: CallbackQuery):  
 if call.message.message\_id:  
 await bot.delete\_message(call.from\_user.id, call.message.message\_id)  
 await call.message.answer("Выберите действие:", reply\_markup=kb.kb\_start)  
  
  
 executor.start\_polling(dp, skip\_updates=False, on\_startup=on\_startup)#Запускает бота в режиме long-polling  
  
if \_\_name\_\_ == '\_\_main\_\_':#Конструкция в Python, которая позволяет запустить код только при запуске файла, а не при импорте его в другой файл  
 tg\_bot\_2(token)