Автономная некоммерческая организация
Дополнительного профессионального образования
Компьютерная Академия «ТОП» филиал «Академия ТОП Уфа»

Дипломная работа

по курсу «Web-разработка на python»

на тему: «Разработка Telegramm-бота для взаимодействия с VPN - сервисом»

Выполнил(а) студент(ка) Компьютерной Академии ТОП

Прошкин А.Е.
(ФИО) (подпись)

Дипломная работа допущена к защите и проверена на объем заимствования:

Директор филиала
«Академия Топ Уфа»

_____\ Е.И.Гиниатуллина

Руководитель
«Академия Топ Уфа»
_____\А.Т.Ахмадуллин

Уфа, 2024г.

Содержание

Введение

Актуальность темы

Современный мир предъявляет повышенные требования к конфиденциальности и защите данных в сети Интернет. С каждым годом количество кибератак, случаев утечки информации и несанкционированного доступа растет. Для защиты пользователей и их данных широко используются виртуальные частные сети (VPN), которые обеспечивают безопасность передачи данных и сохраняют анонимность в сети.

Особенно актуальным становится создание инструментов, которые позволяют пользователям легко и удобно подключаться к VPN. В этом контексте Telegram-боты предоставляют идеальную платформу для взаимодействия, благодаря своей популярности, простоте использования и поддержке интеграции с внешними сервисами. Разработка Telegram-бота, автоматизирующего процесс управления VPN-сервисом, позволяет значительно упростить подключение пользователей к сети и управление их подписками.

Цель работы

Целью дипломной работы является разработка Telegram-бота для взаимодействия с VPN-сервисом, который будет:

- автоматизировать процесс регистрации и управления пользователями;
- предоставлять удобный интерфейс для выбора тарифного плана и оплаты;
- генерировать конфигурационные файлы и QR-коды для подключения к VPN;
- обеспечивать надежное хранение и управление данными о пользователях с помощью базы данных.

Задачи работы

Для достижения поставленной цели необходимо решить следующие задачи:

1. Провести анализ существующих решений для управления VPN-сервисами с помощью Telegram-ботов.
2. Выбрать и обосновать использование технологий: WireGuard, PiVPN, Tmax, Python, aiogram 3, asyncio, asyncSession, MySQL, ORM SQLAlchemy и subprocess.
3. Разработать архитектуру системы, включающую взаимодействие между ботом, сервером VPN и базой данных.
4. Реализовать функционал Telegram-бота:
    - приветствие пользователей и выбор тарифного плана;
    - обработку платежей;
    - автоматическое создание конфигурационных файлов и QR-кодов;

    - управление статусом подписок пользователей.
5. Провести тестирование разработанного бота, включая проверку производительности и надежности.
6. Оценить результаты работы и предложить направления для дальнейшего развития системы.

Ожидаемые результаты

Результатом работы станет Telegram-бот, способный:

- предоставлять пользователям удобный интерфейс для взаимодействия с VPN-сервисом;
- автоматизировать процессы управления подписками и подключения;
- гарантировать безопасность и надежность работы системы.

Кроме того, работа будет включать анализ существующих решений, описание архитектуры и реализацию проекта, а также рекомендации по его дальнейшему развитию.

Обзор существующих решений

Существующие подходы к управлению VPN через Telegram-ботов

На текущий момент существует ряд решений, позволяющих управлять VPN-сервисами через Telegram-ботов. Большинство из них реализует базовую функциональность, такую как создание конфигурационных файлов, выдача ключей доступа и предоставление базовой статистики. Однако такие решения часто ограничены в функционале или требуют значительных усилий для настройки и поддержки. Например:

1. **Боты для WireGuard**: Некоторые проекты предлагают простые скрипты для интеграции WireGuard с Telegram, позволяя создавать ключи и конфигурации. Тем не менее, такие боты редко включают поддержку подписок или автоматизацию оплаты.
2. **Кастомные VPN-системы**: Существуют системы, которые предоставляют более сложный интерфейс для управления пользователями, но они часто имеют закрытый исходный код и требуют покупки лицензии.

Анализ технологий

Для реализации проекта были выбраны следующие технологии:

1. **WireGuard**
    - Современный и высокопроизводительный VPN-протокол.
    - Простота настройки и низкая нагрузка на систему.
    - Высокий уровень безопасности благодаря использованию современных криптографических методов.
2. **PiVPN**
    - Инструмент для автоматизации настройки WireGuard.
    - Удобство управления пользователями и конфигурациями.
3. **Tmax**
    - Используется для мониторинга и управления системными процессами.
4. **Python и aiogram 3**
    - Python — универсальный язык программирования с обширной экосистемой библиотек.
    - Aiogram 3 — удобный и мощный фреймворк для создания Telegram-ботов.
5. **База данных MySQL и ORM SQLAlchemy**
    - MySQL — производительная реляционная база данных.
    - ORM SQLAlchemy упрощает взаимодействие с базой данных за счет использования объектно-ориентированного подхода.
6. **asyncio и asyncSession**
    - Обеспечивают асинхронную обработку запросов, что позволяет повысить производительность бота.
7. **subprocess**
    - Используется для выполнения системных команд, связанных с настройкой VPN через PiVPN.

Сравнение с существующими решениями

Предлагаемый в работе подход отличается от существующих решений за счет:

- полной автоматизации процесса управления VPN;
- интеграции с системой подписок и оплаты;

- использования современных технологий для обеспечения высокой производительности и надежности системы;
- наличия проработанной админ-панели для управления клиентами и сервером.

Разработка Telegram-бота

Архитектура системы

Система включает несколько ключевых компонентов: Telegram-бот, сервер VPN (использующий WireGuard и PiVPN), и база данных MySQL, которая хранит информацию о пользователях и их подписках. Взаимодействие между этими компонентами реализуется через API, которое обеспечивает удобный интерфейс для взаимодействия с ботом и управления пользователями.

Описание API и функциональности бота

Основная функциональность бота включает:

- Приветственное сообщение и регистрация пользователей.
- Выбор тарифного плана.
- Обработка платежей.
- Генерация конфигурационных файлов и QR-кодов для подключения к VPN.
- Управление подписками и статусом пользователей в базе данных.

Выбор инструментов и технологий

- **Python** выбран за свою простоту и гибкость, а также богатую экосистему библиотек.
- **aiogram 3** используется для удобной разработки асинхронных Telegram-ботов.
- **ORM SQLAlchemy** обеспечивает удобный и безопасный способ взаимодействия с базой данных.
- **WireGuard и PiVPN** — оптимальные решения для настройки VPN-сервисов.

Реализация

Основные шаги реализации включают:

- ***Регистрацию пользователя через бота.***

Добавление нового клиента в Базу Данных MySQL *рис.1*

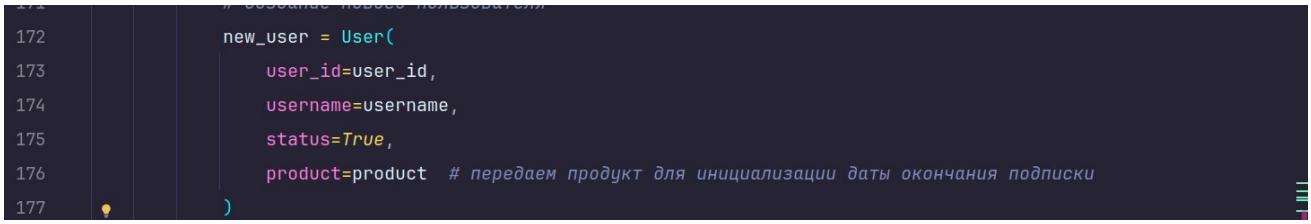


рис. 1

Получение id клиента и прописываем пути для сохранения .conf(qr) данных так же создаем username для **PiVPN** *рис. 2*

рис. 2

Добавление клиента(**PiVPN**) на сервере через **subprocess** *рис. 3*

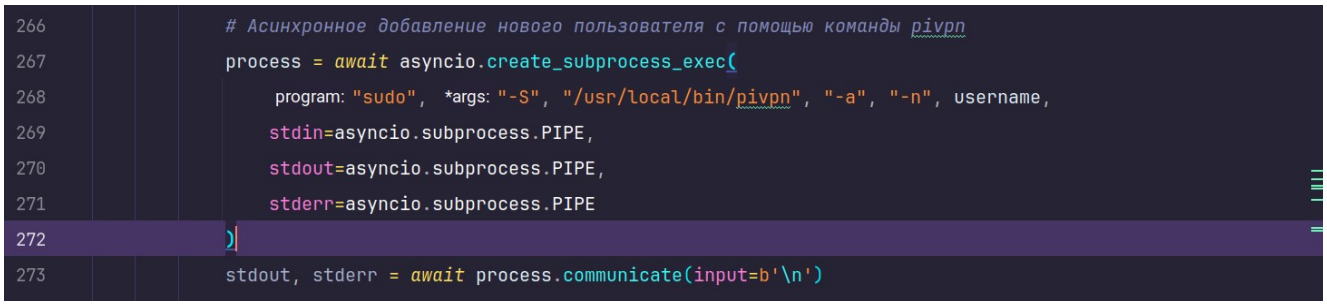


рис. 3

Модель пользователя (User) *рис. 4*

Модель User представляет собой таблицу пользователей, которые подключаются к VPN-сервису через Telegram-бота. Она включает информацию о пользователе, его подписке и связи с продуктом (тарифным планом). Модель реализует основные атрибуты, которые необходимы для управления пользователями в контексте работы с VPN.

рис. 4

Пояснение к полям модели:

1. **id**:
    - Тип: Integer
    - Описание: Уникальный идентификатор пользователя, автоматически увеличиваемый при добавлении нового пользователя в базу данных.
2. **user_id**:
    - Тип: Integer
    - Описание: Идентификатор пользователя, уникальный для каждого пользователя. Это поле важно для связи с Telegram-ботом, который использует идентификатор для общения с пользователем.
    - Ограничение: Поле обязательно для заполнения и уникально в таблице.
3. **username**:
    - Тип: String(255)
    - Описание: Имя пользователя, полученное из Telegram. Это поле может быть пустым, если пользователь не указал имя или оно отсутствует в Telegram.
4. **rate**:
    - Тип: Integer
    - Описание: Ссылка на тарифный план пользователя, представленный в таблице Product. Это поле может быть пустым (если подписка еще не выбрана).
    - Ограничение: Внешний ключ на поле id в таблице Product. При удалении продукта, значение поля rate будет установлено в NULL.
5. **status**:
    - Тип: Boolean

- Описание: Статус подписки пользователя, где True — активный пользователь, а False — неактивный. Это поле по умолчанию имеет значение False.

6. **subscription_start**:
    - Тип: DateTime
    - Описание: Дата начала подписки пользователя. Поле может быть пустым, если подписка не была активирована.
7. **subscription_end**:
    - Тип: DateTime
    - Описание: Дата окончания подписки. Рассчитывается на основе продолжительности выбранного тарифного плана и устанавливается при создании записи о пользователе.
8. **product**:
    - Тип: relationship("Product")
    - Описание: Связь с тарифным планом (моделью Product). Это поле описывает, к какому тарифу относится пользователь, и позволяет организовать двустороннюю связь между пользователями и тарифами.

Конструктор класса:

```
54      def __init__(self, user_id, product: Product, username=None, status=False):  AlexPro
55          self.user_id = user_id
56          self.username = username
57          self.status = status
58          self.subscription_start = datetime.utcnow()
59
60          # Устанавливаем дату окончания подписки в зависимости от count_day продукта
61          if product.count_day:
62              self.subscription_end = self.subscription_start + timedelta(days=product.count_day)
63          else:
64              self.subscription_end = None  # Если count_day не указан
65
66          self.product = product
```

рис. 5

Описание конструктора:

- Конструктор модели принимает параметры:
    - user_id: Идентификатор пользователя из Telegram.
    - product: Объект класса Product, представляющий выбранный тариф.

- username: Имя пользователя в Telegram (опционально).
- status: Статус пользователя (по умолчанию — False, то есть неактивен).

При создании экземпляра модели автоматически устанавливаются:

- subscription_start: Дата начала подписки (устанавливается на текущее время).
- subscription_end: Дата окончания подписки рассчитывается на основе количества дней, указанных в тарифе (count_day)

Модель **User** играет ключевую роль в управлении пользователями, их подписками и взаимодействии с тарифами. Она позволяет хранить информацию о пользователе, его статусе и выбранном тарифе, а также следить за сроком действия подписки. Эта информация используется для автоматизации процессов управления пользователями через Telegram-бота.

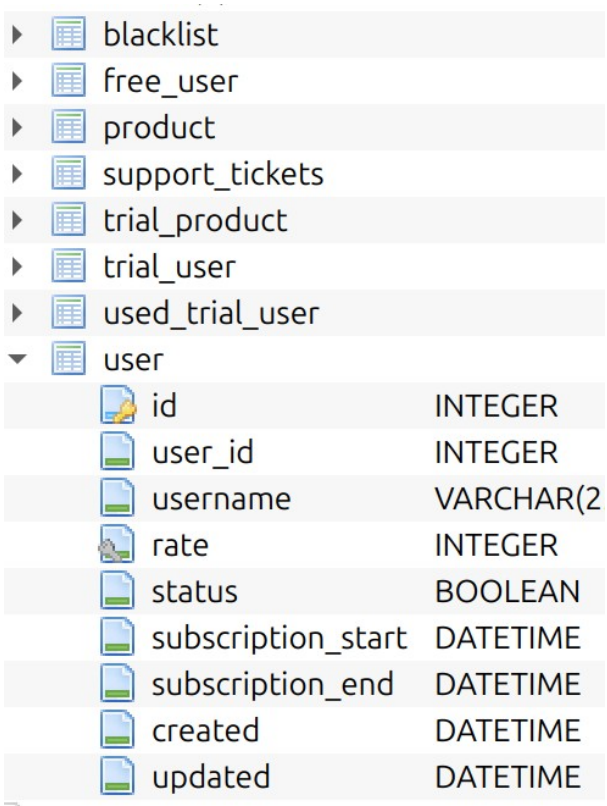


рис. 7

Так же создан **Base** класс от которого наследуются все остальные классы.

```
6   # Базовый класс с полями для отслеживания времени создания и обновления
7   class Base(DeclarativeBase):  11 usages  AlexPro
8       created: Mapped[DateTime] = mapped_column(DateTime, default=func.now())
9       updated: Mapped[DateTime] = mapped_column(DateTime, default=func.now(), onupdate=func.now())
10
```

рис. 8

Выбор тарифа и оплату через интеграцию с платёжной системой.

Необходимо создать модель для добавления самого продукта.

Модель продукта (Product)
Модель Product представляет собой таблицу, в которой хранятся данные о тарифных планах, доступных пользователям для подписки. Она включает информацию о названии продукта, цене и продолжительности подписки, которая определяет количество дней, на которые приобретается тариф.

Структура модели

```
13  class Product(Base):  18 usages  AlexPro
14      __tablename__ = 'product'
15
16      id: Mapped[int] = mapped_column(Integer,primary_key=True)
17      name: Mapped[str] = mapped_column(String(15), nullable=False)
18      price: Mapped[int] = mapped_column(Integer,nullable=False) # Работат
19      count_day: Mapped[int] = mapped_column(Integer, nullable=True)  # Пр
20
21      def __init__(self, name: str, price: float, count_day: int = None):
22          self.name = name
23          self.price = int(round(price * 100))  # Конвертация цены в центы
24          self.count_day = count_day
```

рис. 9

Пояснение к полям модели:

1. **id**:
    - Тип: Integer
    - Описание: Уникальный идентификатор продукта, автоматически увеличиваемый при добавлении нового продукта в базу данных.
2. **name**:
    - Тип: String(15)
    - Описание: Название тарифного плана. Это поле обязательно для заполнения и не может быть пустым. Размер поля ограничен 15 символами, что обеспечивает краткость названия.
3. **price**:

- Тип: Integer
- Описание: Цена тарифного плана в целых числах, представленных в центах (например, тариф за 5.99 будет записан как 599).
- Примечание: Цена конвертируется в центы при добавлении в базу данных, чтобы избежать проблем с точностью при работе с десятичными значениями (float).

4. **count_day**:
    - Тип: Integer
    - Описание: Количество дней, на которые действительна подписка. Это поле может быть пустым, если продолжительность подписки не указана, и может служить индикатором бессрочного или неопределённого периода.

Конструктор класса:

Описание конструктора:

- Конструктор класса принимает следующие параметры:
    - **name**: Название тарифного плана.
    - **price**: Цена тарифного плана в формате float, которая затем конвертируется в целое число (цены в центах).
    - **count_day**: Количество дней подписки. Если параметр не передан, то значение по умолчанию будет равно None.

При инициализации экземпляра модели, цена будет округлена до целых чисел и преобразована в центы для хранения в базе данных.

Назначение модели в проекте

Модель **Product** является ключевой для управления тарифами в системе. Она позволяет хранить данные о доступных тарифах для пользователей, включая цену и продолжительность подписки. Связь с моделью **User** позволяет каждому пользователю быть привязанным к конкретному тарифу, а также вычислять сроки действия подписки на основе длительности тарифа.

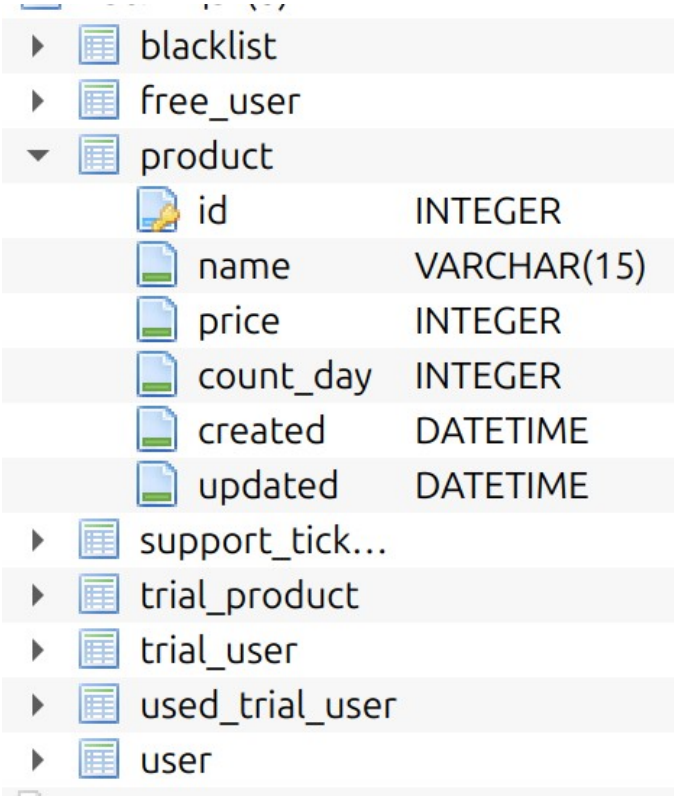


Рис. 10

Создаем в **Telegram-боте** кнопку Тарифы

```
# Главное меню
@user_private_router.callback_query(F.data == 'menu_')  AlexPro
async def back_callback(callback_query: types.CallbackQuery, state: FSMContext):
    # Очищаем состояние FSM, если нужно
    await state.clear()
    # Отправляем сообщение с главным меню
    await callback_query.message.answer(
        text: '<b>Выберите действие:</b>',
        parse_mode='HTML',
        reply_markup=get_keyboard(
            *btns: "💼 Тарифы",  # Информация о тарифах
            "🎁 Пробный период",  # Предложение попробовать бесплатно
            "👤 Личный кабинет",  # Данные пользователя
            "📖 Инструкции",  # Как пользоваться
            "🛠 Поддержка",  # Помощь и обратная связь
            placeholder="Что вас интересует?",
            sizes=(2, 2, 1),  # Логически сгруппировано: 2 верхних, 2 нижних, 1 пос
        ),
    )
```

При взаимодействии с кнопкой, клиент проверяется есть ли он в БД и ему выводиться список продуктов. Так же тут идет проверка для выбора кнопки.

```python
# Варианты подписки
@user_private_router.message(F.text == "💼 Тарифы")  AlexPro
async def menu_cmd(message: types.Message, session: AsyncSession):
    # Проверяем, зарегистрирован ли пользователь
    user_id = message.from_user.id
    query = select(User).where(User.user_id == user_id)
    result = await session.execute(query)
    existing_user = result.scalar()
    button_text = 'Продлить' if existing_user else 'Купить'
    products = await orm_get_products(session)
```

Рис. 12

Отправка информации о тарифах (продуктах) пользователю

Этот фрагмент кода отвечает за отправку информации о тарифах (или продуктах) пользователю через Telegram-бота. Когда бот должен показать список доступных товаров (например, тарифных планов), он выполняет следующие действия:

```python
    if other_products:
        print(f"Остальные товары: {[product.name for product in other_products]}")
        for product in other_products:
            formatted_price = f"{product.price / 100:.2f}"  # Пересчитываем цену
            response_message = (f"<strong>{product.name}</strong>\n"
                                f"Цена: {formatted_price} руб.\n"
                                f"Кол-во дней: {product.count_day}\n\n")
            await message.answer(
                response_message,
                reply_markup=get_inlineMix_btns(btns={
                    button_text: f'pay_{product.id}'
                })
            )
            await asyncio.sleep(0.1)  # Даём Telegram API время обработать сообщение
```

рис . 13

1. **Проверка наличия товаров (if other_products)**:
    - Сначала происходит проверка, есть ли в списке other_products товары (продукты), которые необходимо отправить пользователю. Это условие гарантирует, что сообщения с товарами отправляются только в случае, если такие товары существуют.
2. **Вывод в консоль списка товаров**:
    - Для отладки или логирования выводится список всех названий продуктов, которые будут отправлены пользователю. Это помогает отслеживать, какие товары обрабатываются в текущий момент.
3. **Цикл по товарам**:
    - Если список товаров не пуст, то для каждого продукта из other_products выполняется следующее:
        - Цена товара (**product.price**) преобразуется из целых чисел (центов) в десятичные числа с двумя знаками после запятой, чтобы корректно отобразить её в рублях. Для этого цена делится на 100 и форматируется в строку с двумя знаками после запятой.
        - Формируется сообщение response_message, которое включает:
            - Название товара, выделенное жирным шрифтом с помощью HTML-тегов.
            - Цена товара в рублях.
            - Количество дней действия подписки для данного продукта.
        - Далее сообщение отправляется пользователю через Telegram API с помощью метода message.answer. В этом сообщении также прикрепляется инлайн-кнопка, с текстом, генерируемым функцией **get_inlineMix_btns()**. Эта кнопка будет использоваться для оплаты товара (или подписки), и её действие связано с уникальным идентификатором продукта (**product.id**).

4. **Интервал между отправками сообщений**:
    - Внутри цикла после отправки каждого сообщения делается пауза в 0.1 секунды с помощью **asyncio.sleep(0.1)**. Это даёт **Telegram API** достаточно времени для обработки и отправки сообщений, чтобы избежать чрезмерных нагрузок на сервер и не превышать лимиты Telegram.

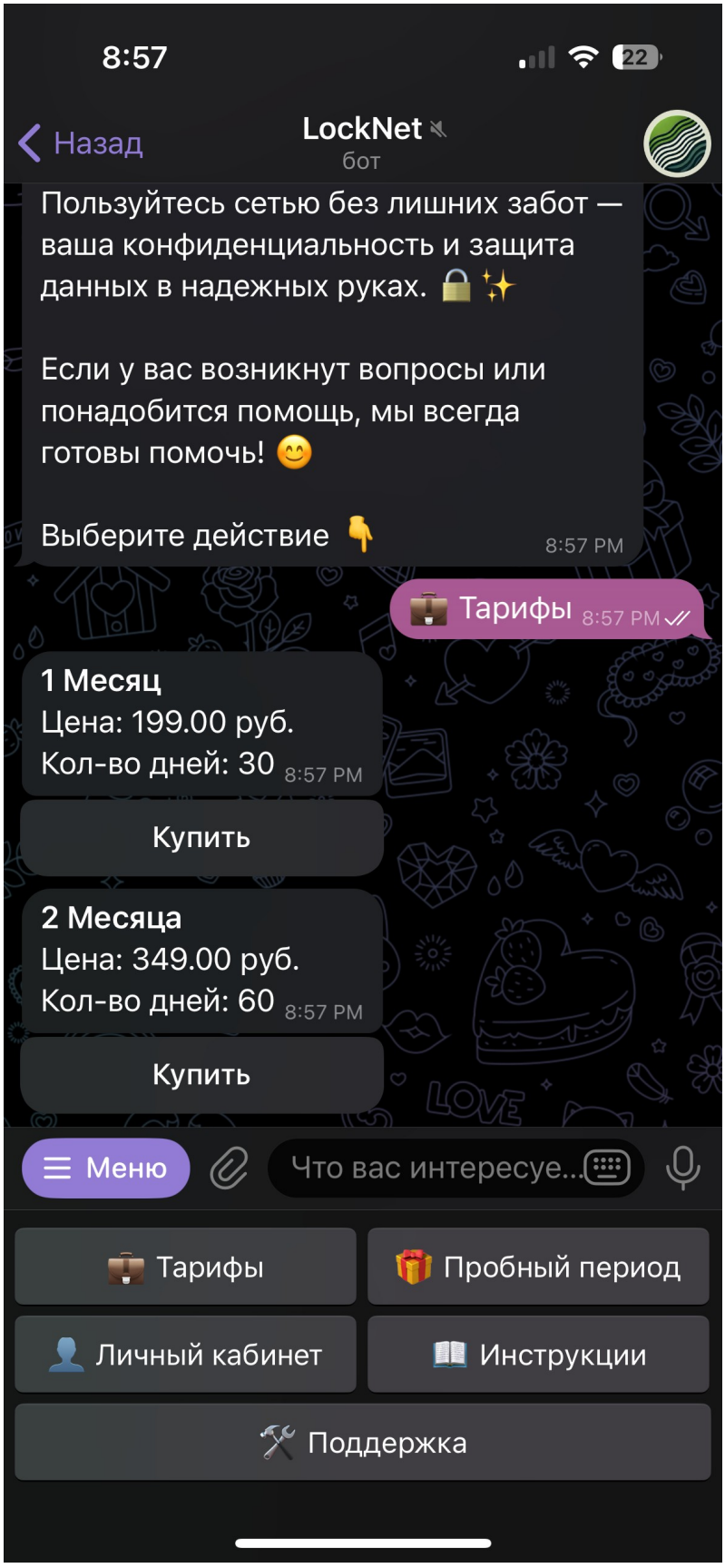
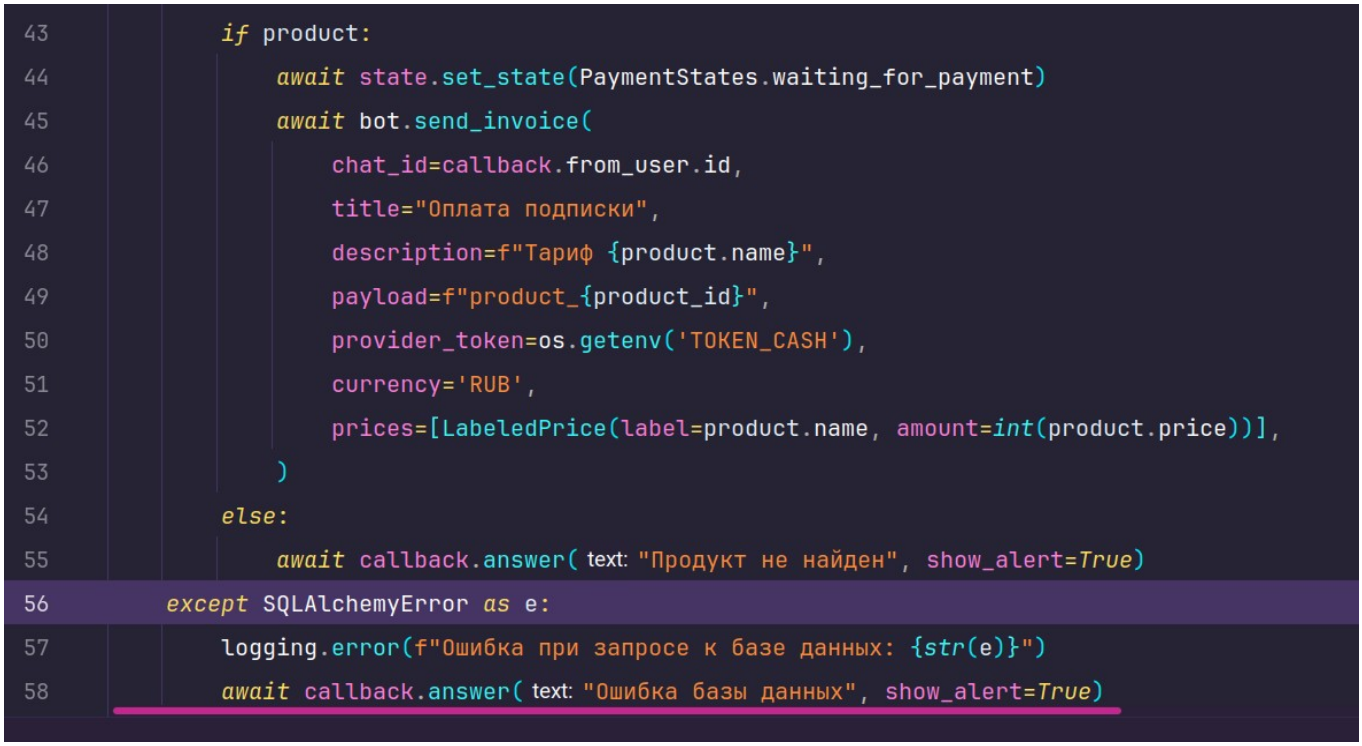

8:57
22
Назад
LockNet
бот
Пользуйтесь сетью без лишних забот — ваша конфиденциальность и защита данных в надежных руках. 🔒✨
Если у вас возникнут вопросы или понадобится помощь, мы всегда готовы помочь! 😊
Выберите действие 👇
8:57 PM
💼 Тарифы 8:57 PM
1 Месяц
Цена: 199.00 руб.
Кол-во дней: 30 8:57 PM
Купить
2 Месяца
Цена: 349.00 руб.
Кол-во дней: 60 8:57 PM
Купить
Меню
Что вас интересуе...
💼 Тарифы
🎁 Пробный период
👤 Личный кабинет
📖 Инструкции
🛠️ Поддержка

Рис. 14

Интеграция с Юkassa для приема платежей



```
43          if product:
44              await state.set_state(PaymentStates.waiting_for_payment)
45              await bot.send_invoice(
46                  chat_id=callback.from_user.id,
47                  title="Оплата подписки",
48                  description=f"Тариф {product.name}",
49                  payload=f"product_{product_id}",
50                  provider_token=os.getenv('TOKEN_CASH'),
51                  currency='RUB',
52                  prices=[LabeledPrice(label=product.name, amount=int(product.price))],
53              )
54          else:
55              await callback.answer( text: "Продукт не найден", show_alert=True)
56      except SQLAlchemyError as e:
57          logging.error(f"Ошибка при запросе к базе данных: {str(e)}")
58          await callback.answer( text: "Ошибка базы данных", show_alert=True)
```

рис. 15

Инициация процесса оплаты:

- Если продукт найден в базе данных, происходит следующее:
  - С помощью **await state.set_state(PaymentStates.waiting_for_payment)** меняется состояние пользователя на ожидание оплаты.
  - Затем вызывается метод **send_invoice**, который инициирует процесс оплаты через Telegram:
    - **chat_id=callback.from_user.id** — отправка счета пользователю.
    - **title и description** — заголовок и описание счета, которые содержат информацию о продукте.
    - **payload** — дополнительная информация, которая передаётся обратно боту при успешной оплате.
    - **provider_token** — токен для оплаты, получаемый из переменных окружения.
    - **currency и prices** — валюта (в рублях) и цена продукта. Цена преобразуется в копейки (умножается на 100).

Процесс оплаты прошел успешно

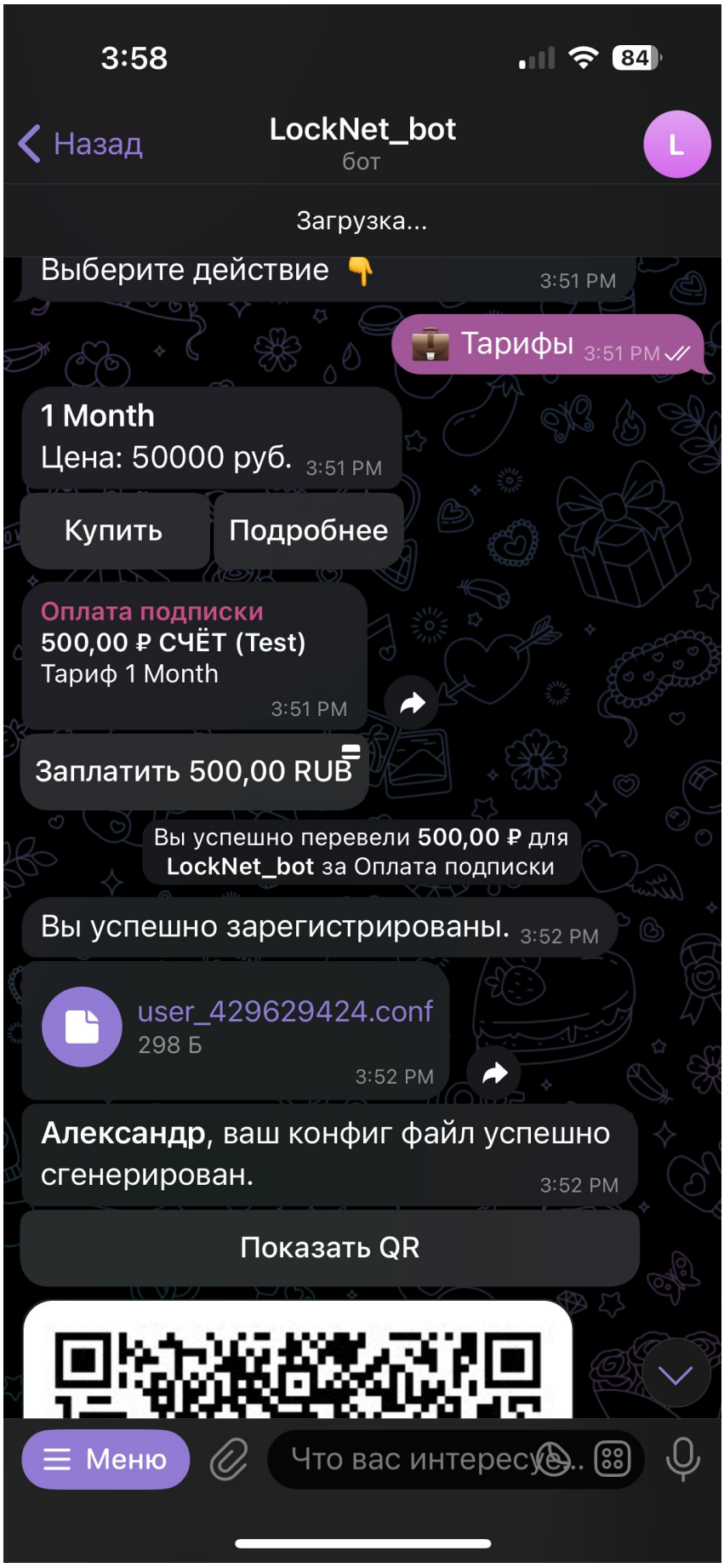


рис. 16

Генерацию конфигурационных файлов и QR-кодов для подключения.

```python
# Асинхронное добавление нового пользователя с помощью команды pivpn
process = await asyncio.create_subprocess_exec(
    program: "sudo",  *args: "-S", "/usr/local/bin/pivpn", "-a", "-n", username,
    stdin=asyncio.subprocess.PIPE,
    stdout=asyncio.subprocess.PIPE,
    stderr=asyncio.subprocess.PIPE
)
stdout, stderr = await process.communicate(input=b'\n')
```

Рис. 17

```python
stdout, stderr = await process.communicate(input=b'\n')
if process.returncode != 0:
    try:
        document = FSInputFile(config_path)
        # Отправка конфигурационного файла пользователю
        await bot.send_document(chat_id=message.chat.id, document=document)
        # Сообщение об успешной генерации конфигурации
        await message.answer(
            text: f"<strong>{message.from_user.first_name}</strong>, ваш конфиг файл успешно
            reply_markup=get_inlineMix_btns(btns={"Показать QR": f"qr_{user_id}"})
        )
        return  # Завершаем выполнение после успешной отправки документа
    except Exception as e:
        # Если документ не найден или произошла ошибка при отправке
        await message.answer("Ваши файлы не найдены. Обратитесь в техподдержку!")
        return
```

Рис. 18

Этот фрагмент кода отвечает за асинхронное добавление нового пользователя в VPN-сервис с помощью команды pivpn и последующую генерацию конфигурационного файла для подключения.

Основные этапы:

1. **Добавление пользователя через pivpn**:
    - Используется команда pivpn -a -n <username> для создания нового пользователя.
    - Асинхронное выполнение команды обеспечивается через asyncio.create_subprocess_exec.

- Ввод передаётся через stdin=asyncio.subprocess.PIPE, а вывод обрабатывается через stdout и stderr.

2. **Проверка результата команды**:
   - Если команда завершается с кодом ошибки (process.returncode != 0), считается, что пользователь не был успешно добавлен.
   - Если команда выполняется успешно, бот пытается отправить сгенерированный конфигурационный файл.
3. **Отправка конфигурационного файла**:
   - Файл считывается через FSInputFile, и отправка документа происходит с помощью метода bot.send_document.
   - Пользователю отправляется сообщение о успешной генерации конфигурации и кнопка для отображения QR-кода.
4. **Обработка ошибок**:
   - Если файл конфигурации не найден или возникает ошибка при отправке, пользователю отправляется сообщение с просьбой обратиться в техподдержку.

Генерация QR-кода через PiVPN:

```
# Асинхронное создание QR-кода из конфигурационного файла
process = await asyncio.create_subprocess_exec(
    program: "sudo",  *args: "-S", "qrencode", "-o", qr_path, "-r", config_path,
    stdout=asyncio.subprocess.PIPE,
    stderr=asyncio.subprocess.PIPE
)
stdout, stderr = await process.communicate()
if process.returncode != 0:
    await message.answer(f"Ошибка при создании QR-кода: {stderr.decode()}")
    return
```

- Используется команда **qrencode** для создания QR-кода из конфигурационного файла.
- Асинхронное выполнение команды осуществляется через **asyncio.create_subprocess_exec**, где:
  - **-o <qr_path>** — указывает путь для сохранения QR-кода.
  - **-r <config_path>** — указывает путь к конфигурационному файлу.
- Результаты выполнения команды (стандартный вывод и ошибки) считываются через **stdout** и **stderr**.

- **Обработка ошибок при генерации QR-кода**:
- Если команда завершается с ошибкой (**process.returncode != 0**), бот отправляет сообщение с текстом ошибки (декодированное содержимое **stderr**) и завершает выполнение функции.

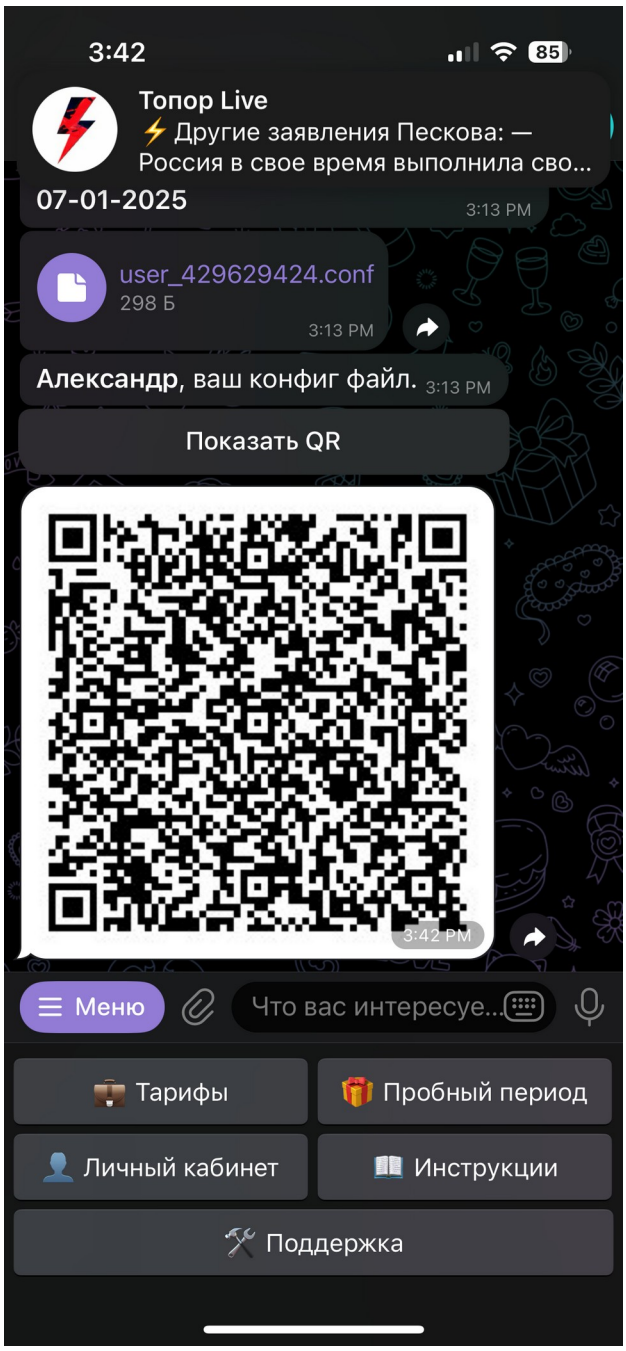
```
    # Отправка конфиг-файла пользователю
    document = FSInputFile(config_path)
    await bot.send_document(chat_id=message.chat.id, document=document)
    await message.answer(
        text: f"<strong>{message.from_user.first_name}</strong>, ваш конфиг файл успешно сгенерирован.",
        reply_markup=get_inlineMix_btns(btns={"Показать QR": f"qr_{user_id}"})
    )
    await state.clear()

except Exception as e:
    await message.answer(f"Произошла неизвестная ошибка: {e}")
    await session.rollback()  # Откат транзакции в случае ошибки
    await state.clear()
else:
    await session.commit()
```

Отправка конфигурационного файла пользователю:

- Конфигурационный файл передаётся в бота через **FSInputFile**.
- С помощью **bot.send_document** файл отправляется пользователю.
- Пользователю также отправляется сообщение об успешной генерации файла, и предоставляется кнопка для отображения QR-кода.

- **Обработка исключений**:
- Если во время выполнения возникает ошибка, бот:
    - Отправляет сообщение с описанием ошибки.
    - Откатывает изменения в базе данных через **session.rollback()**.
    - Сбрасывает состояние машины состояний с помощью **state.clear()**.
- **Коммит изменений**:
- Если выполнение кода завершилось без ошибок, изменения в базе данных фиксируются через **session.commit()**.



Этот код автоматизирует процесс генерации QR-кодов для подключения к **VPN**, упрощая для пользователя получение данных для настройки. Генерация QR-кодов предоставляет удобный способ подключения к **VPN**, особенно для мобильных устройств. Кроме того, код обрабатывает ошибки, гарантируя стабильность и корректное взаимодействие с пользователем.

Управление статусами подписки и удаление неактивных пользователей.

Этот код представляет собой асинхронную функцию, которая автоматически проверяет статус подписок пользователей в базе данных и отключает тех, у кого подписка истекла. Функция запускается в бесконечном цикле и выполняет проверку через заданные интервалы времени.

Запуск проверки подписок:

- Цикл начинается с уведомления о запуске проверки подписок.
- Используется асинхронный запрос для получения списка всех пользователей через функцию **orm_query_users.orm_get_users**.
- **Обработка отсутствия пользователей**:
- Если в базе данных нет пользователей, функция ждет 12 часов (43200 секунд) и повторяет проверку.
- **Проверка подписок пользователей**:

- Для каждого пользователя проверяется дата окончания подписки (**user.subscription_end**).
- Если дата истекла и пользователь имеет активный статус (**user.status == True**), он деактивируется:
    - Поле **status** изменяется на **False**.
    - Формируется имя пользователя (**username**), которое используется в команде **pivpn**.

```python
async def check_subscriptions(session: AsyncSession):  2 usages  AlexPro
                    user.status = False  # Деактивируем пользователя
                    username = f"user_{user.user_id}"

                    try:
                        process = await asyncio.create_subprocess_exec(
                             program: "sudo",  *args: "-S", "/usr/local/bin/pivpn", "-off","-n", username, "-y",
                            stdin=asyncio.subprocess.PIPE,
                            stdout=asyncio.subprocess.PIPE,
                            stderr=asyncio.subprocess.PIPE
                        )

                        stdout, stderr = await process.communicate(input=b'\n')

                        if process.returncode == 0:
                            print(f"Пользователь {user.username} с ID {user.user_id} деактивирован.\n"
                                  f"Вывод: {stdout.decode()}")
                        else:
                            print(f"Ошибка выполнения команды для {username}: {stderr.decode()}")

                    except asyncio.TimeoutError:
                        print(f"Команда для пользователя {username} превысила время ожидания.")
                    except Exception as e:
                        print(f"Ошибка при выполнении команды для {username}: {e}")
```

Деактивация пользователя через pivpn:

- Выполняется команда **pivpn -off -n <username>** для отключения пользователя.
- Асинхронное выполнение команды осуществляется через **asyncio.create_subprocess_exec.**
- Обрабатываются стандартный вывод (**stdout**) и ошибки (**stderr**).
- Если команда завершается успешно **(process.returncode == 0)**, в лог записывается информация об успешной деактивации пользователя. В противном случае фиксируются ошибки выполнения команды.
- **Обработка ошибок**:
- При превышении времени ожидания команды или других исключениях информация о проблемах записывается в лог.

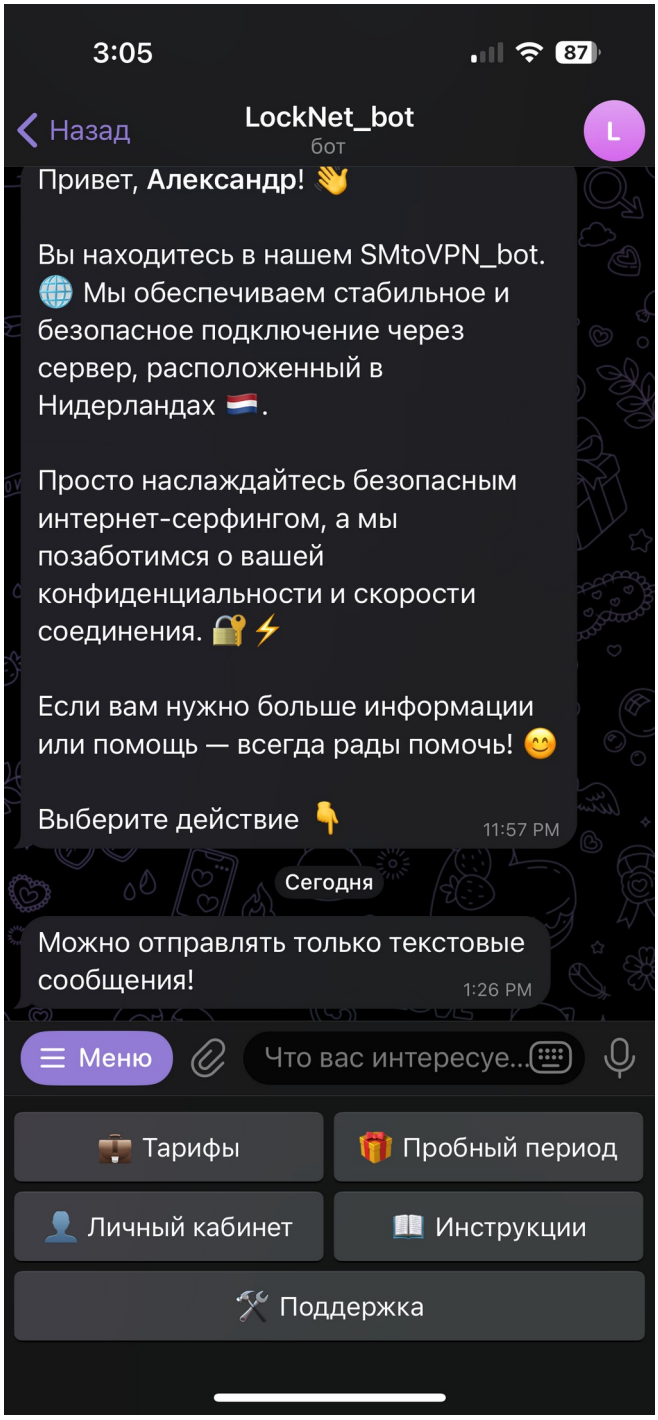
- **Фиксация изменений**:
- После обработки всех пользователей изменения фиксируются в базе данных через **session.commit().**
- В случае ошибки происходит откат изменений с помощью **session.rollback().**
- **Интервалы проверки**:
- После завершения цикла функция ждет 1 час (3600 секунд) перед повторной проверкой.

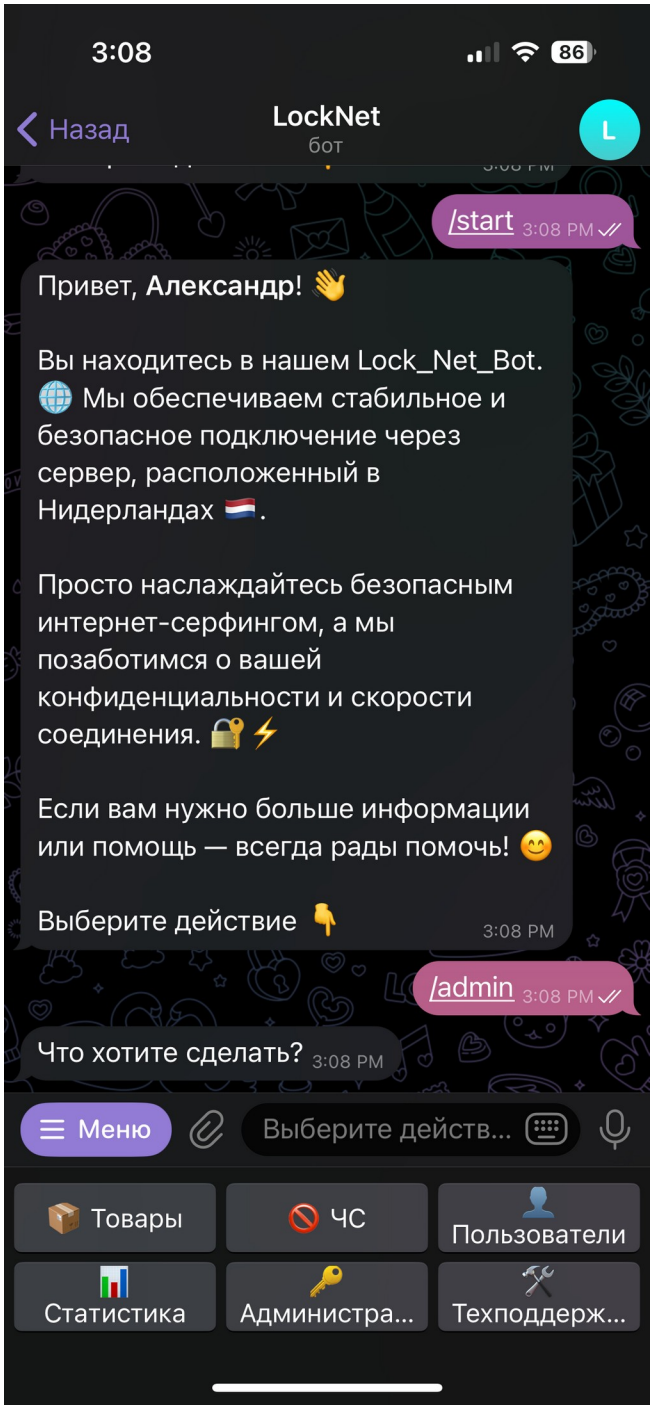
Эта функция автоматизирует процесс проверки и управления подписками, обеспечивая:

- Постоянный мониторинг статуса пользователей.
- Деактивацию пользователей с истекшими подписками.
- Надежность и защиту базы данных благодаря обработке ошибок и откату транзакций при сбоях.

Функция упрощает администрирование VPN-сервиса, минимизируя ручной труд и обеспечивая своевременное отключение неактивных пользователей. Это улучшает управление ресурсами и помогает поддерживать порядок в системе.

Демонстрация работы Telegram-бота:





Описание административной панели:

Административная панель предоставляет удобный интерфейс для управления различными функциями системы. В панели доступны следующие разделы:

1. □ **Товары**

   Раздел предназначен для управления товарами. Здесь администратор может

добавлять, редактировать или удалять товары, а также просматривать их наличие и статус.

2. □ **ЧС (Черный список)**
   В этом разделе можно управлять черным списком пользователей. Администратор может добавлять пользователей в ЧС, удалять их оттуда и просматривать текущий список заблокированных аккаунтов.
3. □ **Пользователи**
   Этот раздел предоставляет инструменты для управления пользователями. Администратор может просматривать список пользователей, редактировать их профили, назначать роли или блокировать доступ.
4. □ **Статистика**
   В разделе статистики можно анализировать ключевые показатели системы. Доступны отчеты и графики, позволяющие отслеживать активность пользователей, количество заказов, продажи и другие метрики.
5. □ **Администраторы**
   Раздел для управления администраторами и их правами доступа. Позволяет добавлять новых администраторов, редактировать их роли и удалять их из системы.
6. □ **Техподдержка**
   Инструмент для взаимодействия с технической поддержкой и решения проблем пользователей. Позволяет администратору управлять тикетами, отвечать на запросы и отслеживать статус обращения.

Дополнительные особенности панели:

- **Интуитивный интерфейс**: Кнопки удобно сгруппированы и распределены в сетке (размеры 3x3), что обеспечивает быстрый доступ к основным функциям.
- **Placeholder**: Панель сопровождается подсказкой "Выберите действие □", которая помогает сориентироваться пользователю при первом использовании панели.

Административная панель разработана для повышения эффективности управления системой и упрощения рутинных задач администратора.

Описание клиентского меню:

Меню клиента предоставляет удобный доступ к различным функциям и возможностям личного кабинета. В нем представлены следующие разделы:

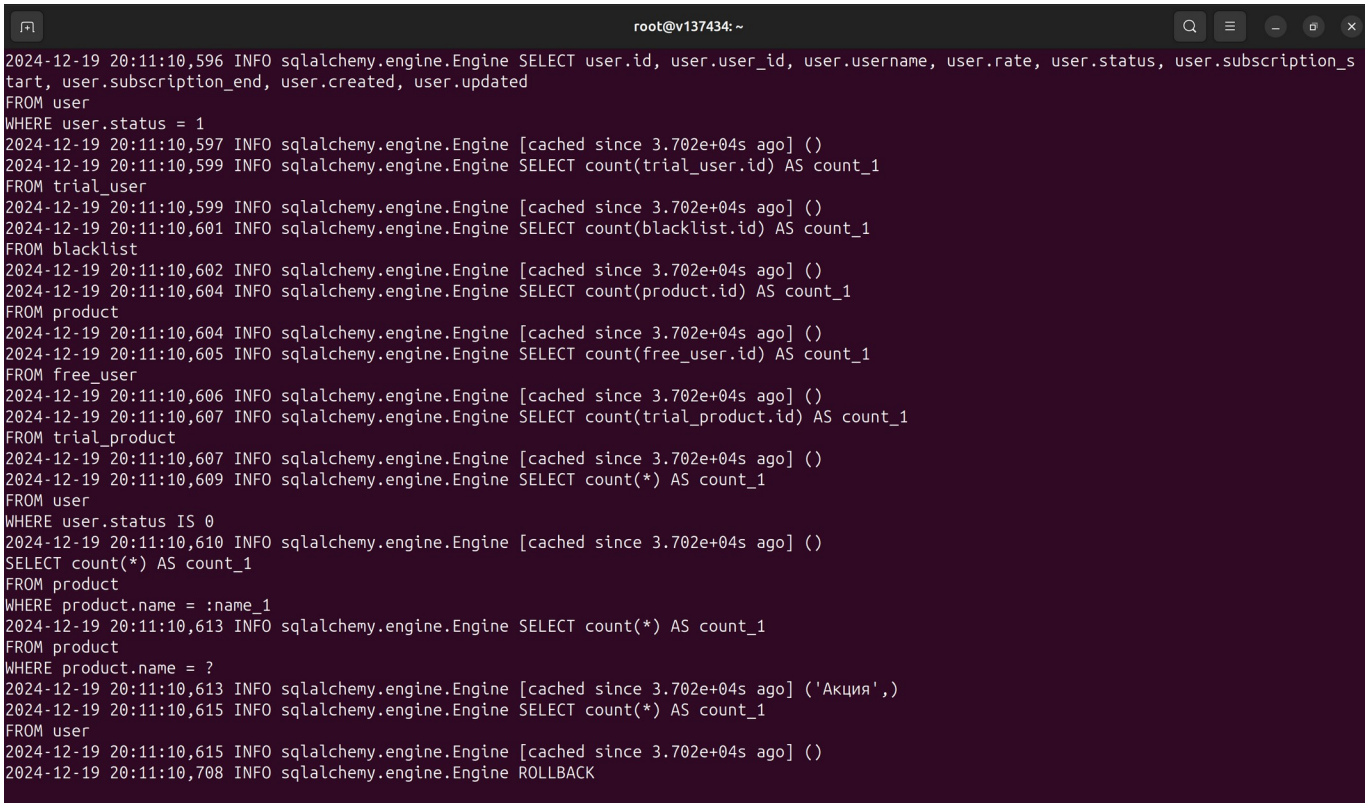
1. □ **Тарифы**
   В этом разделе клиент может ознакомиться с доступными тарифами на услуги, а также выбрать подходящий тариф в зависимости от своих потребностей.
2. □ **Пробный период**
   Раздел, посвященный пробному периоду, где клиент может активировать или ознакомиться с условиями предоставления бесплатного тестового доступа к услугам или продуктам.
3. □ **Личный кабинет**
   Здесь клиент может просматривать и редактировать свои данные, управлять учетной записью, а также отслеживать свои действия и настройки, связанные с использованием сервиса.
4. □ **Инструкции**
   В разделе с инструкциями клиент найдет полезную информацию и шаги по использованию сервиса, а также ответы на частые вопросы и рекомендации по оптимизации работы.
5. □ **Поддержка**
   Раздел для связи с технической поддержкой. Клиент может задать вопросы, отправить запрос на помощь или получить решение для возникающих проблем.

Особенности клиентского меню:

- **Интуитивный интерфейс**: Кнопки удобно расположены в сетке 2x2 для основных функций и 1 кнопка для раздела поддержки, что делает интерфейс логичным и легким для восприятия.
- **Placeholder**: Подсказка "Что вас интересует?" помогает пользователю выбрать нужный раздел и начать взаимодействие с меню.

Меню клиента оптимизировано для удобства использования, чтобы пользователь мог быстро находить нужную информацию и взаимодействовать с сервисом.

Скриншот сервера:



Заключение

В ходе выполнения дипломного проекта была разработана система автоматизированного взаимодействия пользователей с VPN-сервисом через Telegram-бота. Целью проекта было создание удобного инструмента для управления подписками, подключениям к VPN-сети и автоматизации процессов, таких как регистрация пользователей, выбор тарифных планов, оплата и генерация конфигурационных файлов. В рамках работы была реализована интеграция с современными технологиями, такими как WireGuard, PiVPN, а также использование популярного фреймворка для разработки Telegram-ботов — aiogram.

Основные результаты проекта:

1. **Разработка Telegram-бота:** Бот был реализован с использованием Python и библиотеки aiogram. Он позволяет пользователям легко и удобно управлять своими подписками, выбирать тарифные планы и оплачивать их через Telegram. Функционал бота включает автоматическую генерацию конфигурационных файлов для подключения к VPN и создание QR-кодов, что значительно упрощает настройку VPN-клиентов.

2. **Интеграция с VPN-сервисом:** Бот взаимодействует с VPN-сервисом через PiVPN и WireGuard. Процесс добавления нового пользователя и генерация конфигурационных файлов осуществляется автоматически с помощью системных команд, что позволяет минимизировать время на настройку и повышает удобство для пользователя.
3. **Автоматизация проверок:** Разработана система для регулярной проверки подписок пользователей. Бот автоматически отключает пользователей с истекшими подписками, поддерживая актуальность и надежность системы.
4. **Безопасность и надежность:** В процессе разработки были учтены важные аспекты безопасности, такие как защита данных пользователей и использование современных методов криптографии для создания защищенных конфигурационных файлов. Также уделено внимание обработке ошибок, что гарантирует стабильную работу системы.
5. **Тестирование:** Бот был протестирован на различных этапах разработки, включая тестирование функционала генерации конфигурационных файлов и QR-кодов, а также тестирование процесса оплаты и подписки. Были проведены нагрузочные тесты для проверки масштабируемости системы, а также тестирование надежности взаимодействия с сервером VPN.

Оценка эффективности и результатов:

Результаты работы подтверждают достижение заявленных целей проекта. Созданный Telegram-бот эффективно автоматизирует процессы управления подписками и подключениям к VPN. Система показала высокую стабильность и производительность при обработке запросов, что делает её удобной и безопасной для использования конечными пользователями.

Возможности для дальнейшего развития:

1. **Расширение функционала бота:** В дальнейшем возможно добавление новых функций, таких как интеграция с другими VPN-протоколами, улучшенная аналитика пользователей и добавление возможности создания нескольких типов тарифных планов с различными уровнями доступа.
2. **Оптимизация производительности:** С учетом возможного роста числа пользователей, стоит рассмотреть улучшение производительности системы, включая оптимизацию работы с базой данных и более эффективное управление сессиями.
3. **Поддержка разных платёжных систем:** Расширение возможностей по оплате подписок с интеграцией других платёжных систем (например, через API банковских карт или криптовалюты).

Вывод:

В ходе работы был успешно разработан и реализован Telegram-бот для автоматизированного взаимодействия с VPN-сервисом. Он удовлетворяет все требования, поставленные в начале проекта, и обеспечит удобство и безопасность для пользователей. Полученные результаты подтверждают актуальность выбранной темы и значимость автоматизации процесса подключения и управления подписками. В дальнейшем проект может быть улучшен и масштабирован для более широкого применения в реальных условиях.

Список литературы

1. **Греков, А. В.** Основы разработки программного обеспечения для веб-сервисов / А. В. Греков. — Москва: Наука, 2021. — 300 с.
2. **Шабалин, А. П.** Теория и практика использования VPN-сетей в информационной безопасности / А. П. Шабалин. — Санкт-Петербург: Питер, 2019. — 180 с.
3. **Иванов, В. И.** Python для разработчиков / В. И. Иванов. — Москва: Диалектика, 2020. — 350 с.
4. **Репнин, М. А.** Разработка Telegram-ботов на Python с использованием библиотеки aiogram / М. А. Репнин. — Москва: БХВ-Петербург, 2022. — 250 с.
5. **Документация WireGuard**. https://www.wireguard.com. [Дата обращения: 19.12.2024].
6. **Документация PiVPN**. https://pivpn.io. [Дата обращения: 19.12.2024].
7. **Bari, A.**, **Khan, A.** VPNs and Their Role in Securing Communication: A Survey / A. Bari, A. Khan. — Journal of Information Security, 2020. — Vol. 35, No. 2. — P. 123-145.
8. **Python Software Foundation**. Python 3 Documentation. https://docs.python.org/3/. [Дата обращения: 19.12.2024].
9. **Gorib, M.** Telegram Bots: A Developer's Guide / M. Gorib. — 2nd ed. — New York: Packt Publishing, 2021. — 450 p.
10. **Морозов, И. В.** Основы работы с базами данных MySQL / И. В. Морозов. — Москва: Бином, 2020. — 220 с.
11. **Системы и сети виртуальных частных сетей (VPN)** / Под ред. В. М. Савченко. — Москва: Техносфера, 2019. — 350 с.