Схема базы данных доступна по ссылке.

В случае, если есть расхождения с данной схемой, то информация здесь преобладает над схемой по ссылке.

1) Основные таблицы:

- **User**
- **UserEvent**
- **SubscribeWords**
- **Referral**
- **Invoice**
- **Payment**
- **SubscribePlan**
- **UserSubscribe**
- **Rubric**
- **RubricUserSubscribe**
- **Worker**
- **DigestNews**
- **Channel**
- **ChannelsNews**
- **OpenAiCost**
- **FeedBack** ### 2) Описание таблиц

User

Описание таблицы:
Таблица, хранящая информацию о пользователях системы.

Поля:

- `id` (BigInteger, Identity): Уникальный идентификатор пользователя. Это первичный ключ.
- `user_id` (BigInteger, unique): Уникальный идентификатор пользователя.
- `first_name` (Text): Имя пользователя.
- `username` (Text): Юзер пользователя.
- `chatgpt_flag` (Boolean, default=False): Флаг, указывающий, включен ли чат-бот GPT для пользователя.
- `created_at` (DateTime): Дата и время создания пользователя.
- `updated_at` (DateTime): Дата и время последнего обновления данных пользователя.
- `balance` (Float, default=0.0): Баланс пользователя.(не используется)
- `media_flag` (Boolean, default=True): Флаг, указывающий, включены ли медиа для пользователя.
- `recommendation_flag` (Boolean, default=False): Флаг, указывающий на активность рекомендации для пользователя.
- `channel_up` (Boolean, default=False): Флаг, указывающий, активирован ли канал пользователя.
- `language` (Text, default="ru"): Язык пользователя.(в данный момент функция не используется)
- `digest_time` (Text): Время получения дайджеста пользователем.
- `similar_news_filter` (Boolean, default=True): Флаг фильтрации похожих новостей.
- `utm_source` (Text): Источник UTM для пользователя(допустим media, referral).
- `ban_date` (DateTime): Дата блокировки бота пользователем.

Обязательные поля:

- `user_id`, остальные поля могут быть пустыми. ### UserEvent

Описание таблицы:
Таблица, хранящая информацию о событиях, связанных с пользователями.

Поля:

- `id` (BigInteger): Уникальный идентификатор события.
- `user_id` (ForeignKey): Внешний ключ, ссылающийся на пользователя, к которому относится событие.
- `event_type` (Text): Тип события.
- `event_timestamp` (DateTime, default=func.now() + timedelta(hours=3)): Время события.

Обязательные поля:

- `user_id`, `event_type`. ### SubscribeWords

Описание таблицы:
Таблица, связывающая подписки пользователей с определенными словами для фильтрации.

Поля:

- `id` (BigInteger, Identity): Уникальный идентификатор записи.
- `user_id` (ForeignKey): Внешний ключ, ссылающийся на пользователя.
- `subscribe_id` (ForeignKey): Внешний ключ, ссылающийся на подписку.
- `words` (Text): Слова, связанные с подпиской.

Обязательные поля:

- `user_id`, `subscribe_id`, `words`. ### Referral

Описание таблицы:
Таблица, хранящая информацию о рефералах (приглашенных пользователях).

Поля:

- `id` (BigInteger): Уникальный идентификатор.
- `referrer` (ForeignKey): Внешний ключ, ссылающийся на пользователя, который является рефералом.
- `referral` (ForeignKey): Внешний ключ, ссылающийся на пользователя, который был приглашен.

- `type` (Enum(ReferralType)): Тип реферала (новый, активный, оплаченный).
- `created_at` (DateTime): Дата и время создания записи.
- `updated_at` (DateTime): Дата и время последнего обновления записи.

Обязательные поля:

- `referrer`, `referral`, `type`. ### Payment

Описание таблицы:
Таблица, хранящая информацию о платежах пользователей.

Поля:

- `id` (Integer): Уникальный идентификатор платежа.
- `order_id` (String, unique): Идентификатор заказа.
- `amount` (Float): Сумма платежа.
- `status` (Enum(PaymentStatusType)): Статус платежа.
- `created_at` (DateTime): Дата и время создания записи.
- `updated_at` (DateTime): Дата и время последнего обновления записи.
- `user_id` (BigInteger): Идентификатор пользователя.
- `subscribe_type` (Enum(SubscribePlanType)): Тип подписки.
- `days_count` (Integer): Количество дней.
- `is_gift` (Boolean, default=False): Флаг подарочного платежа.
- `for_username` (String): Логин пользователя, для которого был осуществлен платеж.

Обязательные поля:

- `order_id`, `amount`, `status`, `user_id`, `subscribe_type`, `days_count`. ### SubscribePlan

Описание таблицы:
Таблица, хранящая информацию о подписках пользователей.

Поля:

- `id` (Integer): Уникальный идентификатор.
- `payment_id` (Integer, ForeignKey): Внешний ключ, ссылающийся на платеж.
- `created_at` (DateTime): Дата и время создания подписки.
- `updated_at` (DateTime): Дата и время последнего обновления подписки.
- `user_id` (BigInteger, ForeignKey): Идентификатор пользователя, связанного с подпиской.
- `paid_by_user_id` (BigInteger, ForeignKey): Идентификатор пользователя, оплатившего подписку.
- `subscribe_type` (Enum(SubscribePlanType)): Тип подписки.
- `days_count` (Integer): Количество дней подписки.
- `for_username` (String): Логин пользователя.
- `is_gift` (Boolean, default=False): Флаг подарочной подписки.
- `gift_hash` (String, unique): Хеш подарочного кода.
- `is_activated` (Boolean, default=True): Флаг активации подписки.
- `start_at` (DateTime): Дата начала подписки.
- `expired_at` (DateTime): Дата окончания подписки.

Обязательные поля:

- `subscribe_type`, `days_count`.

UserSubscribe

Описание таблицы:
Таблица, хранящая информацию о подписках пользователей на каналы.

Поля:

- `id` (BigInteger, Identity): Уникальный идентификатор записи.
- `channel_id` (BigInteger): Идентификатор канала.
- `user_id` (BigInteger): Идентификатор пользователя, подписавшегося на канал.
- `channel_name` (String): Название канала.
- `channel_link` (String): Ссылка на канал.
- `last_message_is_sended` (Boolean, default=False): Флаг, указывающий, отправлено ли последнее сообщение.
- `worker_id` (Integer, default=2): Идентификатор клиента, связанного с подпиской.
- `created_at` (DateTime): Дата и время создания подписки.
- `foreign_agent` (Boolean, default=False): Флаг, указывающий, является ли агентом внешний агент.
- `category` (String): Категория канала.

Обязательные поля:

- `channel_id`, `user_id`, `channel_name`, `channel_link`. ### Rubric

Описание таблицы:
Таблица, хранящая информацию о рубриках, которые могут быть связаны с подписками пользователей.

Поля:

- `id` (BigInteger): Уникальный идентификатор рубрики.
- `name` (String): Название рубрики.
- `user_id` (BigInteger, ForeignKey): Идентификатор пользователя, который создал рубрику.
- `created_at` (DateTime): Дата и время создания рубрики.

Обязательные поля:

- `name`, `user_id`. ### RubricUserSubscribe

Описание таблицы:
Таблица, связывающая пользователей с рубриками через подписки.

Поля:

- `id` (BigInteger): Уникальный идентификатор записи.
- `user_subscribe_id` (BigInteger, ForeignKey): Идентификатор подписки пользователя.
- `rubric_id` (BigInteger, ForeignKey): Идентификатор рубрики.

Обязательные поля:

- `user_subscribe_id`, `rubric_id`. ### Worker

Описание таблицы:
Таблица, хранящая информацию о воркерах, их сессиях и других данных.

Поля:

- `id` (Integer): Уникальный идентификатор воркера.
- `session` (String): Уникальная сессия воркера.
- `api_id` (BigInteger): Идентификатор API воркера.
- `api_hash` (String): Хэш API воркера.
- `device_model` (String): Модель устройства воркера.
- `system_version` (String): Версия операционной системы.
- `has_proxy` (Boolean, default=False): Флаг, указывающий, используется ли прокси.
- `proxy_type` (String, default="http"): Тип прокси.
- `proxy_address` (String): Адрес прокси.
- `proxy_port` (Integer): Порт прокси.
- `proxy_username` (String): Имя пользователя для прокси.
- `proxy_password` (String): Пароль для прокси.
- `username` (Text): Имя воркера.
- `channels_count` (Integer, default=0): Количество каналов, с которыми работает воркера.
- `has_limit` (Boolean, default=False): Флаг, указывающий, есть ли лимит для воркера.
- `is_active` (Boolean, default=False): Флаг, указывающий, активен ли воркера.
- `created_at` (DateTime): Дата и время создания записи о воркера.
- `updated_at` (DateTime): Дата и время последнего обновления воркера.

Обязательные поля:

- `session`, `api_id`, `api_hash`, `device_model`, `system_version`. ### DigestNews

Описание таблицы:
Таблица, хранящая информацию о новостях дайджеста, отправленных пользователям.

Поля:

- `id` (Integer): Уникальный идентификатор новости.
- `user_id` (BigInteger): Идентификатор пользователя, для которого была отправлена новость.
- `text` (Text): Текст новости.
- `created_at` (DateTime): Дата и время создания новости.
- `sended` (Boolean, default=False): Флаг, указывающий, была ли новость отправлена.
- `theme` (Text): Тема новости.
- `symbol_difference` (BigInteger): Разница в символах.

Обязательные поля:

- `text`, `created_at`. ### Channel

Описание таблицы:
Таблица, хранящая информацию о каналах, которые были напаршены заранее с помощью tg_stat.

Поля:

- `id` (Integer): Уникальный идентификатор канала.
- `title` (Text): Название канала.
- `link` (Text): Ссылка на канал.
- `language` (Text): Язык канала.
- `category` (Text): Категория канала.

Обязательные поля:

- `link`. ### ChannelsNews(не используется) ### OpenAiCost

Описание таблицы:
Таблица, хранящая информацию о стоимости запросов к OpenAI.

Поля:

- `id` (Integer): Уникальный идентификатор записи.
- `user_id` (BigInteger): Идентификатор пользователя.
- `type_of_requests` (String(255)): Тип запросов к OpenAI.
- `input_tokens` (BigInteger): Количество токенов во входном запросе.
- `output_tokens` (BigInteger): Количество токенов в ответе.
- `created_at` (DateTime): Дата и время создания записи.
- `cost` (Float): Стоимость запроса.

Обязательные поля:

- `user_id`, `type_of_requests`, `input_tokens`, `output_tokens`, `cost`. ### FeedBack

Описание таблицы:
Таблица, хранящая информацию о отзывах пользователей.

Поля:

- `id` (Integer): Уникальный идентификатор отзыва.
- `user_id` (BigInteger): Идентификатор пользователя, оставившего отзыв.
- `added_channels` (Integer): Количество добавленных каналов.
- `digest_time` (Text): Время получения дайджеста.
- `text` (Text): Текст отзыва.
- `created_at` (DateTime): Дата и время создания отзыва.

Обязательные поля:

- `user_id`, `text`.

Тут будут идеи

1. Общая архитектура

База данных содержит 16 взаимосвязанных таблиц, реализующих: - Управление пользователями и их профилями - Систему подписок на каналы и рубрики - Платежи и реферальную программу - Работу с контентом и новостными рассылками - Интеграцию с внешними сервисами (Telegram, OpenAI)

2. Основные сущности

2.1 Пользователи (users)

Назначение: Хранение основной информации о пользователях системы
Ключевые поля:

- `chatgpt_flag` - доступ к ChatGPT
- `media_flag` - разрешение медиаконтента
- `balance` - баланс(не используется, в будущем будет удален)
- `digest_time` - предпочтительное время рассылки

Связи:

- С платежами (payments)
- С рефералами (referrals)
- С рубриками (rubrics) ### 2.2 Подписки (users_subscribes)

Назначение: Управление подписками пользователей на Telegram-каналы
Особенности:

- Распределение по воркерам (worker_id)
- Флаг иностранных агентов(является ли канал иностранным агентом) (foreign_agent)
- Система категорий и рубрик

Связи:

- Многие-ко-многим с rubrics
- Связь с workers через worker_id

2.3 Платежи и подписки (payments, subscribe_plans)

Назначение: Управление платными подписками и платежами
Жизненный цикл: 1. Создание инвойса (invoices) 2. Обработка платежа (payments) 3. Активация подписки (subscribe_plans) ### 2.4 Работа с контентом

Каналы (channels): - Каталог доступных для подписки каналов - Классификация по языкам и категориям

Новости (channels_news):

- Архив полученных новостей
- Связь с оригинальными постами через link_of_news

Дайджесты (digest_news): - Система отложенной отправки (sended flag) ## 3. Вспомогательные системы

3.1 Реферальная программа (referrals)

- Трехуровневая система статусов (new/active/paid)
- Защита от самоссылок (referrer ≠ referral) ### 3.2 Воркеры (workers)

Назначение: Управление Telegram-клиентами для сбора новостей
Конфигурация:

- Настройки прокси
- Лимиты каналов на воркер
- Статистика использования (channels_count) ### 3.3 Аналитика и метрики

UserEvent:
Трекинг действий пользователей (открытие приложения, настройки)

OpenAICost:
Учет расходов на генерацию контента с детализацией:

- Типы запросов
- Использование токенов
- Стоимость операций

Feedback:
Система сбора обратной связи с привязкой к:

- Добавленным каналам
- Времени рассылки
- Произвольным комментариям пользователя ## 4. Бизнес-логика ### 4.1 Механика рекомендаций

1. Анализ имеющихся подписок
2. Учет рубрик и категорий
3. Фильтрация через similar_news_filter
4. Формирование персонализированного дайджеста ### 4.2 Модерация контента

- Автоматическая маркировка foreign_agent
- Ручная категоризация каналов
- Фильтрация через subscribe_words ## 5. Особенности реализации

Временные метки:

- Все таблицы содержат created_at/updated_at
- Автоматическое обновление при изменении записи
- Учет московского времени (+3 часа)

Безопасность: - Отдельное хранение платежных данных (order_id) - Система банов через ban_date - Валидация gift_hash (уникальные 8-символьные хеши)

Производительность: - Индексы на внешних ключах - Партиционирование по датам для channels_news

Все взаимодействия с базой данных, происходят путем использования класса {username}DAO(Base), вот пример такого класса ```class
UserDAO(BaseDAO):
model = User

```
@classmethod
def create_user(cls, user_id, first_name, username, utm_source):
    with session_maker() as session:
        query = insert(User).values(
            user_id=user_id,
            first_name=first_name,
            username=username,
            chatgpt_flag=False,
            balance=0.0,
            created_at=func.now() + timedelta(hours=3),
            updated_at=func.now() + timedelta(hours=3),
            utm_source=utm_source,
        )
        session.execute(query)
        session.commit()

        return cls.find_one_or_none(user_id=user_id)
```

```
Открывать сессию вне класса СТРОГО запрещено

[[Требования к бэкэнду]]
[[Как запустить проект в первый раз]]
[[Взаимодействие с базой данных]]

## Установка проекта
Клонируйте репозиторий и установите его в удобную вам папку. Далее запросите у коллег файл виртуального окружения(.env) для запуска проекта
Для запуска установите docker dekstop
## Запуск проекта
Проект можно запустить локально на компьютере таким образом:
#### 1. Создать файл postgres.yaml(заполнить его можно таким образом)
```

name: digest
services:
db:
container_name: db
image: postgres
restart: always
user: postgres
volumes:
- digest-db-data:/var/lib/postgresql/data
ports:
- “5432:5432”
command: -p 5432
environment:
- POSTGRES_DB=digest
- POSTGRES_PASSWORD=postgres
healthcheck:
test: [“CMD”, “pg_isready”]
interval: 1s
timeout: 1s
retries: 50
adminer:
image: adminer
restart: always
ports:
- “8080:8080”
environment:

- ADMINER_DEFAULT_SERVER=db
- ADMINER_DEFAULT_PORT=5432
volumes:
digest-db-data: ``` Этот файл запускает два контейнера: 1) База данных 2) Админка для регулировки этой базы данных(доступна по ссылке localhost:8080) Ниже пример входа

![[Pasted image 20250302005651.png]] После того как он создан впишите команду в консоли `docker-compose -f postgres.yaml up -d` #### 2. Установить зависимости Пример для windows(команды вписать в консоли, в директории проекта)

```
python -m venv venv
pip install -r requirements.txt
```

3. Запустить проект

Запустить проект можно вписать в консоли(в корневой папке) команду `python .\main.py` #### 4. Подготовка базы данных

```
from sqlalchemy import create_engine

from models import Base


DATABASE_URL = "postgresql+psycopg2://postgres:postgres@localhost/digest"
engine = create_engine(DATABASE_URL)

Base.metadata.drop_all(engine)
Base.metadata.create_all(engine)

print("Таблицы успешно созданы!")
```

Запустите этот код, далее зайдите в базу данных и заполните таблицу workers, без нее бот не сможет обрабатывать каналы!

Данные требование необходимо соблюдать в будущем!

Код

1. Названия функций и переменных должно отражать их предназначение.
2. Названия всех функций и методов классов, которые должны быть приватными в пакете/классе должны начинаться либо с _, либо с __
3. Каждая функция, класс должны иметь документацию в определенном формате. Данный формат должен быть обсужден
4. Наименования функций, классов должно иметь единый стиль в пакете.
5. Не стоит зависеть от общих классов. Исключением может являться объект, являющийся репрезентацией сущности с которой мы работаем. Для каждого хендлера должен быть собственный класс запроса и ответа, даже если мы имеем одинаковые ответы в разных хендлерах, классы ответов должны быть разными.
6. Все функции и методы должны быть покрыты UNIT-тестами. Желательно иметь покрытие 60-80%.
7. Для каждого хендлера желательно иметь подробное описание всех возможных ответов с примером ответа. ## GIT
8. Все ветки должны иметь в своем названии информацию по типу ветки, краткому описанию и идентификатору задачи в таск менеджеру

- `{prefix}/{name}-{identifier}` Префикс может быть один из следующих:
- `feature`
- `fix`
- `update`

1. Перед выпуском обновления в prod необходимо провести старшему специалисту code review # Особенности кода
2. При взаимодействии со временем необходимо переводить UTC время в МСК. По умолчанию в проекте у нас время по МСК
3. При запуске проекта необходимо добавить воркера в базу данных, в ином случае проект запустится, но не сможет обрабатывать каналы

[[Команды]]

Функционал бота

Настройки

[[Отображение медиа]] [[Похожие публикации]] [[Рекомендации для пользователя]] [[Сокращенный текст новостей]]

Дайджест бот позволяет пользователям собирать персонализированную ленту на основе его подписок, присылать сокращенные новости, рекомендации, добавлять ключевые слова для фильтрации текста и многое другое, что будет расписано в дальнейших разделах документации (так же написать про админку)

Описание

Дайджест ко времени - позволяет пользователю получать сводку новостей

Значение по умолчанию

По умолчанию включено у пользователя ## Как работает? Добавляет медиа, к новости отсылаемой ботом, если оно существует(видео, фото, аудио)

Значение по умолчанию

По умолчанию включены у пользователя ## Как работает? Мы используем модель, которая отбирает новости по базе данных для пользователя, если находит похожую новость(в пределах 12 часов), то он не отправляет ее пользователю

Замечания на будущее

Пока что работает не всегда, требуется переработка функции

Значение по умолчанию

По умолчанию отключены у пользователя ## Как работает? Рекомендации берутся из нашей базы данных, бот пытается определить категорию канала и:

Если определяет категорию, то отправляет похожие каналы В ином случае, отправляет самые популярные каналы из бота(ориентируется на подписки пользователей)

Как мы находим каналы?

Для нахождения каналов используется сервис TG STAT, мы забираем около 80 каналов для каждой категории, чтобы выдавать их пользователю

Значение по умолчанию

По умолчанию отключено у пользователя

Как работает?

Используется GPT, для выделения основной мысли текста, которая поступает из новости.

[[Команды]]

[[Обработка сообщений]]

Каналы

`back_to_start` [[Возврат к кнопке start]]
`channel_guide` [[Инструкция по добавлению каналов]]
`show_channel_info` [[Информация о канале]]
`list_channels` [[Список каналов пользователя]]
`delete_channel` [[Удаление канала]]

Ключевые слова

`add_word` [[Добавление ключевого слова]]
`add_new_word` [[Добавление ключевого слова]]
`list_word` [[Список ключевых слов]]
`delete_word` [[Удаление ключевого слова]]

Общий файл callback

`account` [[Аккаунт]]
`pay_subscribe` [[Оплата подписки]]
`handle_second_mes_pagination` [[Пагинация видео]]
`handle_callback_query` [[Оплата подписки]]

Флаги

`toggle_media` [[Переключение флага медиа]]
`toggle_media2` [[Переключение флага медиа]]
`toggle_channel_up` [[Переключение флага название канала]]
`toggle_similar` [[Переключение флага похожие каналы]]
`toggle_chatgpt` [[Переключение флага gpt]]

Дайджест

`digest` [[Режим дайджеста]]
`digest_off` [[Режим дайджеста]]
`get_now` [[Режим дайджеста]]

Рекомендации

`toggle_recommendation` [Рекомендации]
`handle_subscription` [Рекомендации]

Меню

`back_to_account` [[Вернуться к аккаунту]]
`regulate_channel` [[Регулировка каналов]]
`regulate_settings` [[Регулировка настроек]]

Подписка

`recharge_balance` [[Стандартная подписка]]
`recharge_balance2` [[Подписка с другим планом]]
`recharge_balance3` [[Подписка с ограничением по времени]]
`recharge_balance_with_free_days` [[Подписка с бесплатными днями]]

Рефералы

`referrals_info` [[Информация о рефералах]]
`open_gift` [[Общий подарок]]
`recharge_balance (gift-invoice)` [[Оплата подарка]]
`personal_gift` [[Персональный подарок]]
`referrals_btn` [[Реферальная ссылка]]
`referrals_stat` [[Статистика по рефералам]]
`gift_type` [[Тип подарка]]

Рубрики

`rubric_add_channel` [[Добавление канала в рубрику]]
`add_rubric` [[Добавление рубрики]]
`show_rubric_info` [[Информация о рубрике]]
`rubric_channels_query` [[Каналы рубрики]]
`confirm_delete_rubric` [[Подтверждение удаления рубрики]]
`list_rubrics` [[Список рубрик]]
`handle_page_rubric` [[Список рубрик с пагинацией]]
`rubric_rem_channel` [[Удаление канала с рубрики]]
`delete_rubric` [[Удаление рубрики]]

1. Команда `/start`

Назначение: Первичная инициализация пользователя в системе
Логика работы:

1. Обработка UTM-меток:
    - `giftXXXX` - активация подарочной подписки
    - цифровой ID - реферальная программа
2. Создание нового пользователя с:
    - Пробной 7-дневной подпиской
    - Отложенными уведомлениями (через 4 часа и 3 дня)
    - Генерацией персонального изображения профиля
3. Для существующих пользователей:
    - Снятие бана при наличии
    - Отправка адаптированного приветствия

Особенности: - Автозагрузка видео-инструкции с fallback механизмом - Интеграция с реферальной системой - Поддержка разных сценариев входа (медийные источники/обычный) ## 2. Команда `/account`

Назначение: Управление персональным аккаунтом
Функционал: - Отображение: - Персонализированного изображения профиля - Основного меню управления - Статуса подписки **Обработка ошибок:** - Автоматическая перегенерация изображения при отсутствии - Логирование проблем с доступом к медиафайлам ## 3. Команда `/language`(не используется)

Назначение: Смена языка интерфейса
Реализация:

- Поддерживаемые языки:
    - □□ Русский (`set_language_ru`)
    - □□ English (`set_language_en`)
- Особенность: Требуется доработка механизма обновления интерфейса после смены ## 4. Команда `/search`(не используется)

Назначение: Активация голосового режима (экспериментальный)
Функции: - Перевод бота в состояние `VoiceStates.waiting_for_voice` - Предупреждение о “коварстве ИИ” □ - Используется для: - Голосового поиска каналов - Управления через аудиосообщения ## 5. Команда `/add`

Назначение: Добавление новых каналов
Логика: - Фиксация события `add_command` - Отправка инструкций по формату ## 6. Команда `/pay`

Назначение: Управление платными подписками
Сценарии использования: 1. Для активной подписки: - Показ вариантов продления - Кнопки “Месячный”/“Годовой” план 2. Для неактивной подписки: - Отображение условий оплаты - Выбор тарифного плана ## 7. Команда `/support`

Назначение: Связь с технической поддержкой
Реализация: - Перенаправление в Telegram-чат `@digest_support` - Фиксация события `support_command` ## 8. Команда `/options`

Назначение: Настройка параметров системы
Особенности: - Динамическое меню настроек: - Управление рассылками - Фильтры контента - Настройки ChatGPT - Система генерации изображения профиля: - Автоматическое создание при первом входе - Кэширование в `media/image/{user_id}.jpg` ## 9. Команда `/channels`

Назначение: Просмотр текущих подписок
Логика отображения: Генерация интерактивного списка - Обработка пустого списка: - Отправка инструкции по добавлению - Кнопка возврата в аккаунт ## 10. Команда `/refresh`

Назначение: Сброс аккаунта (только для тестирования)
Функционал:

- Полное удаление пользовательских данных
- Принудительный рестарт через `/start`
- Логирование операций в `TestUserRefresh` ## Системные особенности:

1. **Механизм событий:**
   - Все действия фиксируются в `UserEvent`
   - Примеры событий: `start_text`, `pay_command`, `get_settings`
2. **Работа с медиа:**
   - Видео кэшируется через `upload_video()`
   - Использование `file_id` Telegram для оптимизации
3. **Безопасность:**
   - Валидация пользователя при каждом запросе
   - Защита от SQL-инъекций через ORM
   - Изоляция тестовых данных
4. **Логирование:**
   - Детальный трекинг ошибок
   - Запись ключевых пользовательских действий
   - Использование структурированных логов

1. Обработка голосовых сообщений

Эндпоинт: `@register_message_handler(content_types=["voice"])`
Состояние: Требуется активный стейт (после `/search`) ### Логика работы: 1. Распознавание речи через ASR-сервис 2. Обработка запроса через GPT 3. Маршрутизация ответа: - **Дайджест**: `digest-тема` - **Поиск**: `search-ключевые_слова` - **Произвольный ответ**: прямая отправка **Особенности:** - Автоматическое удаление промежуточных сообщений - Поддержка Markdown в ответах(на данный момент) ## 2. Обработка подарков (username)

Состояние: `GiftUserStates.waiting_for_username`

Активация: После выбора “Персональный подарок” ### Валидация имени:

1. Удаление `@` в начале
2. Проверка на:
   - Спецсимволы
   - Пробелы
   - Совпадение с отправителем **Ошибки:**

- `gift_username_invalid` - неверный формат
- `gift_username_self_error` - попытка самоподарка ## 3. Обработка ключевых слов **Состояние:** `UserWordStates.waiting_for_word`
  Контекст: Добавление фильтров к каналу **Особенности:**
- Регистронезависимое хранение
- Максимум 5 фраз на канал
- Автоматическая тримминг пробелов ## 4. Создание рубрик

Состояние: `AddRubricStates.waiting_for_rubric_name`
Лимиты:

- Пагинация по 10 рубрик на пользователя **Интеграция:**

1. Создание рубрики
2. Привязка к существующим подпискам
3. Генерация клавиатуры управления ## 5. Обработка медиаконтента

Поддерживаемые типы: - Текст - Фото/Видео - Документы - Голосовые/Видеосообщения

Тут будет реализован план рефакторинга

back_to_start(call, bot)

Описание

Этот обработчик вызывается, когда пользователь нажимает кнопку с данными `back_to_start`. Он возвращает пользователя в начальное состояние с возможностью вернуться к инструкциям или выбрать другие действия.

Логика

- Проверяется язык пользователя.
- Если каналы не добавлены, пользователю показывается инструкция по добавлению канала или текст, в зависимости от UTM-метки.
- Если каналы есть, пользователь возвращается к стартовому экрану.

Параметры:

- `call`: объект, содержащий информацию о callback-запросе от пользователя.
- `bot`: объект бота, через который отправляются сообщения.

add_word(call, bot)

Описание

Этот обработчик вызывается, когда пользователь нажимает на кнопку для добавления ключевого слова для канала (например, с данными `add_word|{channel_id}`). Он предоставляет интерфейс для добавления ключевого слова или отображения существующих.

Логика

- Проверяется, есть ли уже добавленные ключевые слова.
- Если нет ключевых слов, бот переходит в режим ожидания, где пользователь может ввести новое ключевое слово.
- Если ключевые слова уже есть, отправляется список доступных слов с возможностью их удалить или изменить.

Параметры:

- `call`: объект, содержащий информацию о callback-запросе от пользователя.
- `bot`: объект бота, через который отправляются сообщения.

add_new_word(call, bot)

Описание

Этот обработчик вызывается, когда пользователь нажимает кнопку для добавления нового ключевого слова (например, с данными `add_new_word|{channel_id}`). Он проверяет, не превышает ли количество ключевых слов лимит (максимум 5) и позволяет добавить новое слово.

Логика

- Проверяется, не добавлено ли уже больше 5 ключевых слов для канала.
- Если лимит превышен, выводится сообщение с предупреждением.
- В противном случае, пользователь переходит в режим ожидания для ввода нового слова.

Параметры:

- `call`: объект, содержащий информацию о callback-запросе от пользователя.
- `bot`: объект бота, через который отправляются сообщения.

channel_guide(call, bot)

Описание

Этот обработчик вызывается, когда пользователь нажимает кнопку с данными `channel_guide`. Он отображает инструкцию по добавлению канала, если каналы не были найдены, или список каналов, если они уже существуют.

Логика

- Если у пользователя нет подписанных каналов:
  - Отправляется инструкция по добавлению канала.
  - Добавляется кнопка для возврата в главное меню.
- Если каналы есть:
  - Отправляется список всех доступных каналов.
  - Добавляется возможность вернуться к списку каналов.

Параметры:

- `call`: объект, содержащий информацию о callback-запросе от пользователя.
- `bot`: объект бота, через который отправляются сообщения.

show_channel_info(call, bot)

Описание

Этот обработчик вызывается, когда пользователь нажимает на канал для просмотра его информации (например, с кнопки типа `channel_{channel_id}`). Он отображает подробности канала с возможностью удалить канал или изменить ключевые слова.

Логика

- Извлекается информация о канале из базы данных.
- Пользователь может удалить канал или просматривать/добавлять ключевые слова для канала.
- Отправляется соответствующее сообщение с кнопками для дальнейших действий.

Параметры:

- `call`: объект, содержащий информацию о callback-запросе от пользователя.
- `bot`: объект бота, через который отправляются сообщения.

Описание

Этот раздел описывает callback-обработчики, отвечающие за управление каналами в боте. Обработчики позволяют пользователям добавлять каналы, просматривать их информацию, а также удалять. # Общая информация

- Все обработчики используют информацию о пользователе для персонализированных сообщений.
- Логирование используется для отслеживания действий пользователя и выявления ошибок.
- Все обработчики взаимодействуют с базой данных для получения и изменения информации о каналах и ключевых словах.
- В будущем будет переработано

list_channels(call, bot)

Описание

Этот обработчик вызывается, когда пользователь нажимает кнопку с данными `list_channels`. Он отправляет список каналов, на которые подписан пользователь.

Логика

- Если у пользователя нет подписанных каналов:
  - Отправляется инструкция по добавлению канала.
  - Добавляется кнопка для возврата в аккаунт.

- Если каналы есть:
    - Отправляется список всех каналов пользователя. ### Параметры:
- `call`: объект, содержащий информацию о callback-запросе от пользователя.
- `bot`: объект бота, через который отправляются сообщения.

list_word(call, bot)

Описание

Этот обработчик вызывается, когда пользователь нажимает на ключевое слово для получения подробной информации о нем (например, с данными `word_{word_id}`). Он отображает информацию о выбранном слове и предоставляет возможность его удалить или изменить.

Логика

- Загружается информация о ключевом слове.
- Отправляется сообщение с информацией о слове и возможностью для пользователя удалить или изменить его.

Параметры:

- `call`: объект, содержащий информацию о callback-запросе от пользователя.
- `bot`: объект бота, через который отправляются сообщения.

delete_channel(call, bot)

Описание

Этот обработчик вызывается, когда пользователь нажимает кнопку для удаления канала (например, с данными `delete_channel_{channel_id}`). Он удаляет канал из базы данных и обновляет список доступных каналов.

Логика

- Из базы данных удаляется выбранный канал.
- Отправляется сообщение с подтверждением удаления канала.
- Обновляется список каналов пользователя.

Параметры:

- `call`: объект, содержащий информацию о callback-запросе от пользователя.
- `bot`: объект бота, через который отправляются сообщения.

delete_word(call, bot)

Описание

Этот обработчик вызывается, когда пользователь нажимает кнопку для удаления ключевого слова (например, с данными `delete_word|{word_id}`). Он удаляет ключевое слово из базы данных и отображает соответствующее сообщение.

Логика

- Удаляется выбранное ключевое слово.
- Пользователь получает подтверждение о том, что слово было удалено.

Параметры:

- `call`: объект, содержащий информацию о callback-запросе от пользователя.
- `bot`: объект бота, через который отправляются сообщения.

back_to_account

Назначение:

Обработчик для callback'а, который отвечает за возвращение пользователя в главное меню аккаунта. Этот callback вызывается, когда пользователь нажимает кнопку для возврата в основное меню.

Действия:

1. Логирует начало выполнения.
2. Извлекает информацию о пользователе из базы данных.
3. Проверяет наличие фото пользователя. Если фото отсутствует, пытается загрузить его заново.
4. Отправляет обновленное сообщение с изображением профиля и основной клавиатурой меню.

Используемые компоненты:

- **`logger`**: Для логирования действий пользователя.
- **`UserDAO`**: Для получения данных о пользователе.
- **`UserEventDAO`**: Для логирования события пользователя (возвращение в аккаунт).
- **`generate_main_menu_keyboard`**: Генерация клавиатуры для главного меню.
- **`update_image`**: Функция для обновления изображения пользователя, если оно отсутствует.
- **`types.InputMediaPhoto`**: Отправка изображения профиля пользователя.

Описание

Этот набор callback-обработчиков управляет действиями пользователей в меню, связанными с настройками и каналами. В основном, они включают в себя навигацию по различным разделам меню и работу с изображениями пользователей.

Закомментированный Callback (В будущем возможно будет реализован)

Назначение:

Этот callback, в будущем, будет использоваться для обработки часто задаваемых вопросов (FAQ) и поддержки пользователей. Кнопка будет выводить текст с часто задаваемыми вопросами и поддерживающей клавиатурой.

Действия:

1. Получение информации о пользователе.
2. Отправка текста с часто задаваемыми вопросами.
3. Добавление клавиатуры с опциями для дальнейших действий пользователя.

Используемые компоненты:

- **`get_support_menu_keyboard`**: Генерация клавиатуры с поддержкой.
- **`get_user_info`**: Получение текста с информацией для пользователя.

regulate_channel

Назначение:

Обработчик для callback'а, который отвечает за управление каналами пользователя. Этот callback вызывается, когда пользователь нажимает на кнопку для настройки или выбора каналов.

Действия:

1. Логирует начало выполнения.
2. Извлекает информацию о пользователе из базы данных.
3. Проверяет наличие фото пользователя. Если фото отсутствует, пытается загрузить его заново.
4. Отправляет обновленное сообщение с изображением профиля и клавиатурой для управления каналами.

Используемые компоненты:

- **`logger`**: Для логирования действий пользователя.
- **`UserDAO`**: Для получения данных о пользователе.
- **`generate_channels_menu_keyboard`**: Генерация клавиатуры для меню каналов.
- **`update_image`**: Функция для обновления изображения пользователя, если оно отсутствует.
- **`types.InputMediaPhoto`**: Отправка изображения профиля пользователя.

regulate_settings

Назначение:

Обработчик для callback'а, который отвечает за отображение меню настроек пользователя. Этот callback вызывается, когда пользователь нажимает на кнопку для управления настройками.

Действия:

1. Логирует начало выполнения.
2. Извлекает информацию о пользователе из базы данных.
3. Проверяет наличие фото пользователя. Если фото отсутствует, пытается загрузить его заново.
4. Отправляет обновленное сообщение с изображением профиля и клавиатурой для настроек.

Используемые компоненты:

- **`logger`**: Для логирования действий пользователя.
- **`UserDAO`**: Для получения данных о пользователе.
- **`UserEventDAO`**: Для логирования события пользователя (получение настроек).
- **`generate_settings_menu_keyboard`**: Генерация клавиатуры для меню настроек.
- **`update_image`**: Функция для обновления изображения пользователя, если оно отсутствует.
- **`types.InputMediaPhoto`**: Отправка изображения профиля пользователя.

account

- **Назначение**: Отображение аккаунта пользователя с обновленным профилем.
- **Действия**:
  - Загружает и отображает основное меню с персонализированным изображением профиля.
  - Логирует запрос и отправляет ответ пользователю.
- **Используемые компоненты**:
  - `generate_main_menu_keyboard` для формирования меню.
  - `update_image` для обновления изображения. ## start_btn
- **Назначение**: Обработка нажатия кнопки старт.
- **Действия**:
  - Отправляет пользователю сообщение приветствия, выполняя команду `/start`.
- **Используемые компоненты**:
  - `send_welcome` для отправки приветственного сообщения.

Назначение

Обработчики команд в этом модуле реализуют действия при нажатии пользователем кнопок в интерфейсе бота, управляют состоянием пользователя и обновляют контент в чатах. ### Структура

Каждый обработчик взаимодействует с определенной кнопкой и выполняет действия в зависимости от состояния пользователя или команды. Обработчики используют `telebot` для работы с Telegram API и базы данных SQLAlchemy для хранения состояния пользователей.

Общие особенности

1. **Логирование**: Все обработчики логируют действия пользователей для дальнейшего анализа и улучшения функционала.
2. **Подписка**: Для большинства функций, таких как фильтрация новостей, рекомендации или использование ChatGPT, проверяется наличие активной премиум-подписки.
3. **Переводы**: Все сообщения и уведомления генерируются с учетом языка пользователя, что позволяет поддерживать локализацию для различных языков.
4. **Интерактивность**: Все команды сопровождаются интерактивными кнопками для управления состоянием и перехода между различными интерфейсами. ## set_language(не используется)

- **Назначение**: Установка языка для пользователя.
- **Действия**:
  - Изменяет язык пользователя в базе данных и вызывает отправку приветственного сообщения.
- **Используемые компоненты**:
  - `send_welcome` для отправки приветственного сообщения. # В будущем будет переработано

Назначение

Обработчики команд в этом модуле реализуют действия при нажатии пользователем кнопок в интерфейсе бота, управляют состоянием пользователя и обновляют контент в чатах. ### Структура

Каждый обработчик взаимодействует с определенной кнопкой и выполняет действия в зависимости от состояния пользователя или команды. Обработчики используют `telebot` для работы с Telegram API и базы данных SQLAlchemy для хранения состояния пользователей.

pay_subscribe

- **Назначение**: Платеж за подписку.
- **Действия**:
  - Проверяет наличие подписки и отображает опции для подписки (ежемесячная или ежегодная).
  - Логирует информацию о подписке.
  - Отправляет текст с деталями подписки и кнопками выбора.
- **Используемые компоненты**:
  - `InlineKeyboardButton` для создания кнопок подписки.
  - `translations` для локализации текста.(не используется)

handle_second_mes_pagination

- **Назначение**: Пагинация видео-сообщений в процессе подписки.
- **Действия**:
  - Загружает видео и текст, отображая их в зависимости от выбранной страницы.
  - Логирует действия пользователя.
- **Используемые компоненты**:
  - `upload_video` для загрузки видео.

handle_callback_query (channels_page_)

- **Назначение**: Обработка пагинации страниц канала.
- **Действия**:
  - Переход к следующей или предыдущей странице списка каналов.
  - Отправка обновленного списка каналов пользователю.
- **Используемые компоненты**:
  - `generate_channel_list` для формирования списка каналов.

toggle_chatgpt

- **Назначение**: Переключение флага использования ChatGPT.
- **Действия**:
  - Проверяет подписку и состояние дайджеста, затем переключает флаг `chatgpt_flag`.
  - Обновляет медиафайл профиля пользователя.
  - Логирует событие и обновляет интерфейс с изображением.
- **Используемые компоненты**:
  - `update_image` для обновления изображения профиля.
  - `generate_settings_menu_keyboard` для обновления интерфейса.

1. toggle_media(1 версия)

- **Назначение**: Переключение флага показа медиа-материалов.
- **Действия**:
  - Проверяет подписку пользователя и обновляет флаг `media_flag`.
  - Логирует событие и обновляет меню пользователя.
- **Используемые компоненты**: -`generate_settings_menu_keyboard` для обновления интерфейса.
  - `UserDAO.check_subscribe` для проверки подписки. ## 2. toggle_media2(2 версия)
- **Назначение**: Переключение флага отображения медиа-контента для пользователя.
- **Действия**:
  - Переключает значение флага `media_flag` пользователя.
  - Обновляет кнопку с текстом в зависимости от состояния флага.

    - Логирует событие и отправляет обновленную кнопку в чат.
- **Используемые компоненты**:
    - `UserEventDAO.create_user_event` для записи события.
    - `translations` для перевода текста.
    - `bot.edit_message_reply_markup` для обновления кнопки.

Определяет, где будет название канала, сверху или снизу ### toggle_channel_up

- **Назначение**: Переключение флага канала для пользователя.
- **Действия**:
    - Переключает флаг `channel_up` пользователя.
    - Логирует изменения и обновляет меню канала.
- **Используемые компоненты**:
    - `generate_channels_menu_keyboard` для обновления интерфейса.

toggle_similar

- **Назначение**: Переключение фильтра похожих новостей.
- **Действия**:
    - Проверяет наличие премиум-подписки у пользователя.
    - Переключает флаг фильтра похожих новостей.
    - Логирует изменение и обновляет интерфейс пользователя.
- **Используемые компоненты**:
    - `UserDAO.check_subscribe` для проверки подписки.
    - `generate_settings_menu_keyboard` для обновления интерфейса.
    - `translations` для сообщений об ошибке.

digest

- **Назначение**: Включение режима дайджеста.
- **Действия**:
    - Активирует режим дайджеста и отправляет информацию с обновленным меню.
- **Используемые компоненты**:
    - `generate_digest_keyboard` для формирования клавиатуры.

digest_off

- **Назначение**: Отключение режима дайджеста.
- **Действия**:
    - Отключает режим дайджеста и сбрасывает `digest_time`.
    - Отправляет обновленное сообщение.
- **Используемые компоненты**:
    - `digest_off_scheduler` для сброса таймера дайджеста.
    - `generate_digest_keyboard` для обновления интерфейса.

get_now

- **Назначение**: Получение дайджеста в текущий момент.
- **Действия**:
    - Отправляет дайджест пользователю.
- **Используемые компоненты**:
    - `send_digest` для отправки дайджеста.

toggle_recommendation

- **Назначение**: Переключение флага рекомендаций для пользователя.
- **Действия**:
    - Проверяет подписку пользователя и переключает флаг `recommendation_flag`.
    - Логирует событие и обновляет меню настроек.
- **Используемые компоненты**:
    - `generate_settings_menu_keyboard` для обновления интерфейса. ## handle_subscription
- **Назначение**: Обработка обновлений рекомендаций для пользователя.
- **Действия**:
    - Обновляет рекомендации для указанной категории (например, “наши каналы” или другая категория если найдена).
- **Используемые компоненты**:
    - `generate_our_recommendation_keyboard` и `generate_recommendation_keyboard` для обновления интерфейса.

Описание

Этот набор callback-обработчиков управляет действиями пользователей в подписке, связанной с покупкой/продлением тарифа. В основном, они включают в себя генерацию платежей

recharge_balance_with_free_days

Назначение:

Обработчик для callback’а, который управляет пополнением баланса с дополнительными бесплатными днями, предоставляемыми пользователю в рамках акции.

Действия:

1. Логирует начало выполнения пополнения баланса с бесплатными днями.
2. Проверяет, прошел ли необходимый срок для использования акции.
3. Извлекает информацию о пользователе из базы данных.
4. Генерирует уникальный платежный ID.
5. Определяет тип подписки и рассчитывает количество дней, включая бесплатные.
6. Добавляет запись о платеже в базу данных.
7. Создает ссылку на оплату и формирует сообщение с деталями для пользователя.
8. Отправляет сообщение пользователю с ссылкой на оплату.

Используемые компоненты:

- **`logger`**: Для логирования процесса пополнения.
- **`UserDAO`**: Для получения данных о пользователе.
- **`UserEventDAO`**: Для проверки, может ли пользователь воспользоваться акцией.
- **`PaymentDAO`**: Для добавления записи о платеже.
- **`create_invoice`**: Для генерации ссылки на оплату.
- **`translations`**: Для получения перевода сообщения для пользователя.
- **`settings`**: Для получения цены и характеристик плана с учетом бесплатных дней.

recharge_balance2

Назначение:

Обработчик для callback'а, который выполняет пополнение баланса по аналогии с первым обработчиком, но для другого типа подписки с возможностью использования другого плана с дополнительными параметрами.

Действия:

1. Логирует начало выполнения операции пополнения для второго типа плана.
2. Извлекает информацию о пользователе.
3. Генерирует уникальный платежный ID.
4. Определяет тип подписки и количество дней в зависимости от выбранного плана.
5. Добавляет информацию о платеже в базу данных.
6. Создает ссылку на оплату и формирует сообщение для пользователя.
7. Отправляет пользователю сообщение с ссылкой для оплаты.

Используемые компоненты:

- **`logger`**: Для логирования действий и информации о процессе пополнения.
- **`UserDAO`**: Для получения данных о пользователе.
- **`PaymentDAO`**: Для сохранения данных о платеже.
- **`create_invoice`**: Для генерации платежной ссылки.
- **`translations`**: Для получения перевода сообщения на язык пользователя.
- **`settings`**: Для определения цены и характеристик плана.

3. recharge_balance3

Назначение:

Обработчик для callback'а, который управляет пополнением баланса для подписки, при этом учитывает дополнительные условия акции (например, ограничение по времени для использования скидки).

Действия:

1. Логирует начало выполнения операции пополнения для третьего типа плана.
2. Проверяет, прошел ли необходимый срок для использования акции.
3. Извлекает информацию о пользователе из базы данных.
4. Генерирует уникальный ID для платежа.
5. Рассчитывает стоимость и количество дней в зависимости от выбранного плана.
6. Добавляет запись о платеже в базу данных.
7. Создает ссылку на оплату и формирует сообщение с подробностями для пользователя.
8. Отправляет сообщение пользователю с ссылкой на оплату.

Используемые компоненты:

- **`logger`**: Для логирования действий пользователя.
- **`UserDAO`**: Для получения информации о пользователе.
- **`UserEventDAO`**: Для проверки доступности акции для пользователя.
- **`PaymentDAO`**: Для сохранения информации о платеже.
- **`create_invoice`**: Для генерации ссылки на оплату.
- **`translations`**: Для создания сообщения на языке пользователя.
- **`settings`**: Для получения цены подписки.

recharge_balance

Назначение:

Обработчик для callback'а, который отвечает за пополнение баланса пользователя с выбором между месячным или годовым планом. Он создает уникальный платежный ID, добавляет информацию о платеже в базу данных, генерирует ссылку для оплаты и отправляет сообщение пользователю с деталями оплаты.

Действия:

1. Логирует начало выполнения операции пополнения.
2. Извлекает информацию о пользователе из базы данных.
3. Генерирует уникальный платежный ID.
4. Определяет тип подписки (месячная или годовая) и рассчитывает количество дней для подписки.
5. Добавляет запись о платеже в базу данных.
6. Создает ссылку для оплаты и формирует сообщение для пользователя с подробностями платежа.
7. Отправляет пользователю сообщение с ссылкой на оплату и суммой.

Используемые компоненты:

- `logger`: Для логирования действий пользователя и информации о процессе оплаты.
- `UserDAO`: Для получения данных о пользователе (язык и ID).
- `PaymentDAO`: Для добавления записи о платеже в базу данных.
- `create_invoice`: Функция для генерации ссылки на оплату.
- `translations`: Для выбора текста сообщения в зависимости от языка пользователя.(не используется сейчас)
- `settings`: Для получения информации о ценах планов.

referrals_info

Назначение:

Обработчик для callback'а, который предоставляет информацию о рефералах пользователя, включая количество приглашенных друзей и ссылку для реферала.

Действия:

1. Логирует начало обработки запроса на информацию о рефералах.
2. Извлекает данные пользователя из базы данных.
3. Генерирует ссылку для реферала на основе ID пользователя.
4. Рассчитывает количество приглашенных рефералов.
5. Формирует сообщение с количеством рефералов и ссылкой для приглашений.
6. Отправляет обновленное сообщение с реферальной ссылкой и количеством рефералов, используя клавиатуру с управлением рефералами.

Используемые компоненты:

- `logger`: Для логирования событий и действий пользователя.
- `UserDAO`: Для получения данных о пользователе.
- `ReferralDAO`: Для получения информации о рефералах пользователя.
- `translations`: Для формирования сообщения с учетом языка пользователя(не используется).
- `settings`: Для генерации реферальной ссылки.
- `get_referral_keyboard`: Для создания клавиатуры с возможностью управления рефералами.

Описание

Этот набор обработчиков callback'ов управляет действиями пользователей, связанными с реферальной программой и подарками

open_gift

Назначение:

Обработчик для callback'а, который предоставляет информацию о возможных планах и ценах, если пользователь решит взять "публичный" подарок.

Действия:

1. Логирует запрос на открытие подарка.
2. Извлекает данные пользователя.
3. Отправляет информацию о стоимости планов и их вариантах для подарка.

Используемые компоненты:

- `logger`: Для логирования действий пользователя.
- `UserDAO`: Для получения данных о пользователе.
- `translations`: Для формирования сообщения с учетом языка пользователя.(не используется)
- `get_gift_plans_kb`: Для создания клавиатуры с планами для подарков.

recharge_balance (gift-invoice)

Назначение:

Обработчик для callback'а, который обрабатывает пополнение баланса для подарков и создает платежные ссылки для пользователей, которые хотят подарить подписку другому пользователю.

Действия:

1. Логирует запрос на пополнение баланса для подарка.
2. Извлекает данные о выбранном плане и пользователе, которому будет подарена подписка.
3. Генерирует уникальный платежный ID.
4. Добавляет информацию о платеже в базу данных.
5. Создает ссылку на оплату для подарка и отправляет информацию пользователю.

Используемые компоненты:

- `logger`: Для логирования действий пользователя.
- `UserDAO`: Для получения данных о пользователе.
- `ReferralDAO`: Для получения информации о рефералах (если это необходимо).
- `PaymentDAO`: Для добавления записи о платеже в базу данных.
- `create_invoice`: Для генерации платежной ссылки.
- `translations`: Для формирования сообщения с учетом языка пользователя.(не используется)
- `settings`: Для получения информации о ценах планов.

personal_gift

Назначение:

Обработчик для callback'а, который запрашивает у пользователя персональные данные (например, имя) для оформления персонального подарка.

Действия:

1. Логирует запрос на персональный подарок.
2. Извлекает данные пользователя.
3. Запрашивает у пользователя информацию (например, имя) для подарка.
4. Изменяет состояние пользователя и отправляет сообщение с инструкциями для ввода.

Используемые компоненты:

- `logger`: Для логирования действий пользователя.
- `UserDAO`: Для получения данных о пользователе.
- `translations`: Для формирования сообщения с учетом языка пользователя.(не используется)
- `bot.set_state`: Для установки состояния пользователя для дальнейшего ввода.
- `get_return_to_gift_type_kb`: Для создания клавиатуры для возврата к выбору типа

referrals_btn

Назначение:

Обработчик для callback'а, который отправляет пользователю информацию о реферальной программе и предоставляет возможность использовать реферальные кнопки.

Действия:

1. Логирует событие клика по кнопке рефералов.
2. Извлекает данные пользователя из базы данных.
3. Генерирует реферальную ссылку для пользователя.
4. Отправляет сообщение с текстом о реферальной программе и кнопками для использования рефералов.

Используемые компоненты:

- `logger`: Для логирования действий пользователя.
- `UserDAO`: Для получения данных о пользователе.
- `translations`: Для формирования текста сообщения с учетом языка пользователя.(не используется)
- `settings`: Для получения информации о реферальной ссылке.
- `get_btn_referral_keyboard`: Для создания клавиатуры с кнопками для использования рефералов.

referrals_stat

Назначение:

Обработчик для callback'а, который предоставляет статистику о рефералах пользователя, включая количество рефералов, количество подписанных и платящих пользователей.

Действия:

1. Логирует запрос на статистику по рефералам.
2. Извлекает данные пользователя и информацию о рефералах.
3. Рассчитывает статистику по общему количеству рефералов, подписавшихся пользователей и платящих рефералов.
4. Формирует сообщение с статистикой и отправляет его пользователю.

Используемые компоненты:

- `logger`: Для логирования действий пользователя.
- `UserDAO`: Для получения данных о пользователе.
- `ReferralDAO`: Для получения статистики по рефералам.
- `translations`: Для формирования сообщения с учетом языка пользователя.(не используется)

gift_type

Назначение:

Обработчик для callback'а, который запрашивает у пользователя выбор типа подарка, предоставляя клавиатуру для дальнейших действий.

Действия:

1. Логирует начало запроса на выбор типа подарка.
2. Извлекает данные пользователя.

3. Отправляет пользователю сообщение с предложением выбрать тип подарка и отображает соответствующую клавиатуру.

Используемые компоненты:

- `logger`: Для логирования действий пользователя.
- `UserDAO`: Для получения данных о пользователе.
- `translations`: Для формирования сообщения с учетом языка пользователя.(не используется)
- `get_gift_type_keyboard`: Для создания клавиатуры с выбором типа подарка.

rubric_add_channel

Назначение:

Обработчик для callback'а, который добавляет канал в рубрику и обновляет список каналов для отображения.

Действия:

1. Логирует добавление канала в рубрику.
2. Добавляет канал в рубрику.
3. Обновляет сообщение с новым списком каналов рубрики.

Используемые компоненты:

- `logger`: Для логирования действий пользователя.
- `get_session`: Для получения сессии с базой данных.
- `RubricUserSubscribe`: Для добавления канала в рубрику.
- `rubric_generate_channel_list`: Для генерации обновленного списка каналов.

add_rubric

Назначение:

Обработчик для callback'а, который инициирует процесс создания новой рубрики.

Действия:

1. Логирует запрос на создание новой рубрики.
2. Отправляет сообщение с инструкциями по созданию рубрики.
3. Устанавливает состояние пользователя для ввода имени рубрики.

Используемые компоненты:

- `logger`: Для логирования действий пользователя.
- `translations`: Для локализации сообщения.(не используется)
- `bot.set_state`: Для установки состояния пользователя.

show_rubric_info

Назначение:

Обработчик для callback'а, который отображает информацию о рубрике, включая возможность редактировать каналы, удалить рубрику и вернуться к списку рубрик.

Действия:

1. Логирует запрос на информацию о рубрике.
2. Извлекает данные о рубрике и пользователе.
3. Отправляет сообщение с информацией о рубрике и кнопками для редактирования или удаления рубрики.

Используемые компоненты:

- `logger`: Для логирования действий пользователя.
- `get_session`: Для получения сессии с базой данных.
- `Rubric`: Для получения информации о рубрике.
- `User`: Для получения данных о пользователе.
- `translations`: Для локализации сообщения.(не используется)
- `types.InlineKeyboardButton`: Для создания кнопок с действиями по рубрике.

rubric_channels_query

Назначение:

Обработчик для callback'а, который отображает страницу каналов рубрики, генерирует список каналов и позволяет перейти на другие страницы с каналами.

Действия:

1. Логирует запрос страницы каналов.
2. Извлекает информацию о рубрике и пользователя.
3. Генерирует список каналов рубрики для текущей страницы.
4. Отправляет сообщение с каналами рубрики, используя клавиатуру для навигации между страницами.

Используемые компоненты:

- `logger`: Для логирования действий пользователя.
- `get_session`: Для получения сессии с базой данных.
- `Rubric`: Для получения информации о рубрике.
- `User`: Для получения данных о пользователе.
- `rubric_generate_channel_list`: Для генерации списка каналов рубрики.
- `translations`: Для локализации сообщения.(не используется)

Описание

Этот набор обработчиков callback'ов управляет действиями пользователей, связанными с рубриками и их каналами. Пользователи могут создавать, редактировать, удалять рубрики, а также добавлять и удалять каналы внутри рубрик. Эти обработчики обеспечивают навигацию по страницам рубрик, отображение информации о рубриках, добавление каналов в рубрики и их удаление. Также поддерживаются функции для подтверждения удаления рубрик и управления списками рубрик.

confirm_delete_rubric

Назначение:

Обработчик для callback'а, который подтверждает удаление рубрики.

Действия:

1. Логирует запрос на подтверждение удаления рубрики.
2. Удаляет рубрику из базы данных.
3. Отправляет сообщение с обновленным списком рубрик.

Используемые компоненты:

- `logger`: Для логирования действий пользователя.
- `get_session`: Для получения сессии с базой данных.
- `Rubric`: Для удаления рубрики.
- `translations`: Для локализации сообщения.(не используется)

handle_page_rubric

Назначение:

Обработчик для callback'а, который отображает страницу рубрик с возможностью перехода между страницами.

Действия:

1. Логирует запрос страницы рубрик.
2. Генерирует и отправляет список рубрик для указанной страницы.

Используемые компоненты:

- `logger`: Для логирования действий пользователя.
- `generate_rubric_list`: Для генерации списка рубрик.

list_rubrics

Назначение:

Обработчик для callback'а, который выводит список рубрик, если они существуют, и предлагает добавить новую рубрику, если их нет.

Действия:

1. Логирует запрос на вывод списка рубрик.
2. Извлекает данные о пользователе.
3. Отправляет сообщение с доступными рубриками или инструкциями по добавлению рубрики.

Используемые компоненты:

- `logger`: Для логирования действий пользователя.
- `get_session`: Для получения сессии с базой данных.
- `generate_rubric_list`: Для генерации списка рубрик.
- `translations`: Для локализации сообщения.(не используется)

rubric_rem_channel

Назначение:

Обработчик для callback'а, который удаляет канал из рубрики и обновляет список каналов для отображения.

Действия:

1. Логирует удаление канала из рубрики.
2. Удаляет канал из рубрики.
3. Обновляет сообщение с новым списком каналов рубрики.

Используемые компоненты:

- `logger`: Для логирования действий пользователя.
- `get_session`: Для получения сессии с базой данных.
- `RubricUserSubscribe`: Для удаления канала из рубрики.
- `rubric_generate_channel_list`: Для генерации обновленного списка каналов.

delete_rubric

Назначение:

Обработчик для callback'а, который инициирует процесс удаления рубрики.

Действия:

1. Логирует запрос на удаление рубрики.
2. Отправляет сообщение с запросом подтверждения удаления рубрики.

Используемые компоненты:

- `logger`: Для логирования действий пользователя.
- `get_session`: Для получения сессии с базой данных.
- `Rubric`: Для получения информации о рубрике.
- `translations`: Для локализации сообщения.(не используется)