🚀 Миграция на PostgreSQL - Пошаговая инструкция

📋 Обзор новой архитектуры

Что изменилось?

1. **SQLite → PostgreSQL** - надёжная продакшн БД
2. **Разделение логики** на сервисы:
   - **Collector** - сбор сообщений
   - **Processor** - рерайт и сборка постов
   - **Publisher** - публикация по расписанию
3. **Умная склейка медиа+текст** - обходим ограничения Telegram
4. **Автоподпись** для медиа без текста
5. **InputMedia** вместо forward - профессиональная пересылка

📁 Новая структура проекта

```
project/
├── app/
│   ├── models/
│   │   ├── __init__.py
│   │   ├── base.py          # Base SQLAlchemy
│   │   ├── source.py        # Источники (каналы)
│   │   ├── message.py       # Очередь сообщений
│   │   └── post.py          # Готовые посты
│   ├── database/
│   │   ├── __init__.py
│   │   └── engine.py        # Подключение к PostgreSQL
│   ├── services/
│   │   ├── __init__.py
│   │   ├── collector.py     # Сборщик сообщений
│   │   ├── processor.py     # Обработчик (рерайт + сборка)
│   │   └── publisher.py     # Публикатор
│   ├── ai.py                # AI рерайт (без изменений)
│   ├── config.py            # Конфигурация (обновлена)
│   ├── prompts.py           # Промпты (без изменений)
│   ├── utils.py             # Утилиты (split_text)
│   └── bot_logic.py         # Главный координатор
├── alembic/
│   ├── versions/            # Миграции
│   ├── env.py               # Конфигурация Alembic
```

```
│   └── script.py.mako
├── data/
│   ├── sources_ids.txt       # Миграция в PostgreSQL
│   └── sources_links.txt
├── logs/
│   └── bot_work.log
├── alembic.ini               # Конфиг Alembic
├── main.py                   # Точка входа
├── requirements.txt          # Зависимости
└── .env                      # Конфигурация
```

🔧 Шаг 1: Установка PostgreSQL

Вариант A: Локально (Linux/Mac)

```bash
# Ubuntu/Debian
sudo apt update
sudo apt install postgresql postgresql-contrib

# Запуск
sudo systemctl start postgresql
sudo systemctl enable postgresql

# Создание БД и пользователя
sudo -u postgres psql

# В psql консоли:
CREATE DATABASE telegram_bot;
CREATE USER botuser WITH PASSWORD 'secure_password';
GRANT ALL PRIVILEGES ON DATABASE telegram_bot TO botuser;
\q
```

Вариант B: Docker (рекомендуется для разработки)

```bash
```

```
# docker-compose.yml
version: '3.8'
services:
  db:
    image: postgres:15
    environment:
      POSTGRES_DB: telegram_bot
      POSTGRES_USER: botuser
      POSTGRES_PASSWORD: secure_password
    ports:
      - "5432:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data

volumes:
  postgres_data:

# Запуск
docker-compose up -d
```

Вариант C: Облачный PostgreSQL

- **Railway.app** - бесплатный тариф
- **Render.com** - бесплатный PostgreSQL
- **Supabase** - бесплатный с ограничениями
- **ElephantSQL** - бесплатный 20MB

📦 Шаг 2: Установка зависимостей

```
bash

# Обновляем pip
pip install --upgrade pip

# Устанавливаем зависимости
pip install -r requirements.txt

# Или вручную:
pip install sqlalchemy[asyncio] asyncpg alembic psycopg2-binary
```

⚙️ Шаг 3: Настройка .env

Создай `.env` файл в корне проекта:

```bash
# Копируем пример
cp .env.example .env

# Редактируем
nano .env
```

Обязательно измени:

```env
# PostgreSQL (ВАЖНО!)
DATABASE_URL=postgresql+asyncpg://botuser:secure_password@localhost:5432/telegram_bot

# Для Docker:
# DATABASE_URL=postgresql+asyncpg://botuser:secure_password@db:5432/telegram_bot

# Для облака (например, Railway):
# DATABASE_URL=postgresql+asyncpg://user:pass@host.railway.app:5432/railway
```

🗄️ Шаг 4: Инициализация Alembic

```bash
# Инициализация (ОДИН РАЗ!)
alembic init alembic

# Создание первой миграции
alembic revision --autogenerate -m "Initial schema"

# Применение миграций
alembic upgrade head
```

Проверка:

```bash
```

```
# Подключись к PostgreSQL
psql -U botuser -d telegram_bot

# Список таблиц
\dt

# Должны быть:
# - sources
# - message_queue
# - posts
# - post_media
# - alembic_version

\q
```

🔄 Шаг 5: Миграция данных из старой БД

Если у тебя есть данные в `sources_ids.txt`:

```
python
```

```python
# Скрипт миграции: migrate_sources.py

import asyncio
from pathlib import Path
from app.database.engine import SessionLocal
from app.models.source import Source

async def migrate_sources():
    ids_file = Path("data/sources_ids.txt")

    if not ids_file.exists():
        print("Файл sources_ids.txt не найден")
        return

    async with SessionLocal() as session:
        with open(ids_file, "r") as f:
            for line in f:
                chat_id = int(line.strip())

                # Проверяем, нет ли уже
                from sqlalchemy import select
                stmt = select(Source).where(Source.chat_id == chat_id)
                result = await session.execute(stmt)
                existing = result.scalar_one_or_none()

                if not existing:
                    new_source = Source(
                        chat_id=chat_id,
                        is_active=True
                    )
                    session.add(new_source)
                    print(f"✅ Добавлен: {chat_id}")
                else:
                    print(f"⏭️ Уже есть: {chat_id}")

        await session.commit()
        print("🎉 Миграция завершена!")

if __name__ == "__main__":
    asyncio.run(migrate_sources())
```

Запуск:

```bash
```

```
python migrate_sources.py
```

🚀 Шаг 6: Запуск бота

```
bash

# Проверка конфигурации
python -c "from app.config import DATABASE_URL; print(DATABASE_URL)"

# Запуск
python main.py
```

Что должно произойти:

```
2024-01-25 12:00:00 [INFO] root: ==========================================
2024-01-25 12:00:00 [INFO] root: 🚀 Запуск Telegram бота с PostgreSQL
2024-01-25 12:00:00 [INFO] root: ==========================================
2024-01-25 12:00:01 [INFO] app.database.engine: ✅ База данных инициализирована
2024-01-25 12:00:02 [INFO] app.bot_logic: ✅ База и папки готовы
2024-01-25 12:00:03 [INFO] app.bot_logic: 🔄 Проверка подписок (5 источников)...
2024-01-25 12:00:05 [INFO] app.bot_logic: 🚀 Бот запущен и слушает каналы...
```

📊 Как работает новая архитектура?

Поток данных:

```
1. СБОР (Collector)
   └─ Входящее сообщение → message_queue
      ├─ Медиа + текст → сразу в очередь
      ├─ Только медиа → awaiting_text = True (ждём 10 сек)
      └─ Только текст → ищем ожидающее медиа → склеиваем

2. РЕРАЙТ (Processor - фон каждые 30 сек)
   └─ message_queue (rewrite_status = 'pending')
      └─ AI рерайт → rewrite_status = 'done'

3. ЗАКРЫТИЕ ОЖИДАЮЩИХ (Processor - фон каждые 15 сек)
   └─ message_queue (awaiting_text = True, timeout)
      └─ awaiting_text = False
         └─ Если текста нет → "медиа без подписи"
```

```
4. СБОРКА ПОСТОВ (Processor - фон каждые 45 сек)
   └─ message_queue (rewrite_status = 'done', ready_to_post = False)
      ├─ Группировка по grouped_id (альбомы)
      └─ Создание Post + PostMedia

5. ПУБЛИКАЦИЯ (Publisher - фон каждую минуту)
   └─ posts (status = 'scheduled')
      ├─ Медиа БЕЗ caption → send_file()
      ├─ Пауза 1.5 сек
      └─ Текст отдельно → send_message()
```

🔍 Мониторинг и отладка

Проверка очереди сообщений

sql

```sql
-- Статус рерайтов
SELECT
    rewrite_status,
    COUNT(*)
FROM message_queue
GROUP BY rewrite_status;

-- Ожидающие текст
SELECT
    source_id,
    message_id,
    awaiting_until
FROM message_queue
WHERE awaiting_text = TRUE;

-- Готовые к публикации
SELECT
    id,
    status,
    scheduled_at
FROM posts
WHERE status = 'scheduled'
ORDER BY scheduled_at;
```

Логи

bash

```bash
# Следим за логами в реальном времени
tail -f logs/bot_work.log

# Фильтрация ошибок
grep "ERROR" logs/bot_work.log

# Фильтрация конкретного сервиса
grep "Processor" logs/bot_work.log
```

⚠️ Частые проблемы

1. "could not connect to server"

```bash
# Проверь, запущен ли PostgreSQL
sudo systemctl status postgresql

# Для Docker
docker-compose ps
```

2. "relation does not exist"

```bash
# Применяем миграции
alembic upgrade head
```

3. "password authentication failed"

Проверь `.env`:

- Правильный пользователь?
- Правильный пароль?
- Правильная БД?

4. Медиа не отправляются

Проблема: `file_reference` устарел (Telegram меняет их)

Решение: Хранить оригинальные сообщения дольше, чтобы успеть опубликовать.

🎯 Проверка работы

1. Добавь тестовый источник

```python
# В Python консоли
import asyncio
from app.bot_logic import TGBot

async def test():
    bot = TGBot()
    await bot.client.start(phone=PHONE)
    await bot.add_source_by_link("@test_channel")
    await bot.client.disconnect()

asyncio.run(test())
```

2. Отправь тестовое сообщение в источник

- Фото без текста → должна добавиться подпись
- Фото + текст → текст должен рерайтнуться
- Альбом → все медиа + рерайтнутый текст

3. Проверь БД

```sql
-- Сообщения
SELECT * FROM message_queue ORDER BY collected_at DESC LIMIT 10;

-- Посты
SELECT * FROM posts ORDER BY created_at DESC LIMIT 5;
```

🗂️ Полезные команды Alembic

```bash
```

```
# Создать новую миграцию
alembic revision --autogenerate -m "Описание изменений"

# Применить все миграции
alembic upgrade head

# Откатить последнюю миграцию
alembic downgrade -1

# Показать текущую версию
alembic current

# История миграций
alembic history

# Откатить до конкретной версии
alembic downgrade <revision_id>
```

🎉 Готово!

Бот теперь работает на PostgreSQL с профессиональной архитектурой!

Что дальше?

1. Настрой автозапуск (systemd/supervisor)
2. Настрой бэкап PostgreSQL
3. Мониторинг через Grafana/Prometheus
4. Добавь админ-панель для управления источниками

Нужна помощь? Пиши вопросы!