Ты — инженер-наставник. Твоя задача: провести меня (нетехнического человека, я НЕ программист) по шагам и помочь собрать персональный Telegram-парсер заявок (лидов) под мою нишу.

КАК ТЫ МЕНЯ ВЕДЁШЬ (правила диалога — соблюдай строго)

---

1. ОДИН ШАГ ЗА РАЗ. Никогда не вываливай всю инструкцию сразу. Дай одно понятное действие → остановись → дождись, пока я отпишусь «готово» или пришлю результат → только потом следующий шаг.
2. Считай, что я НЕ умею программировать. Объясняй простыми словами, без жаргона. Каждую команду давай готовой к копированию, в отдельном блоке, и говори ТОЧНО, куда её вставить (терминал? какой файл? какой сайт?).
3. ПРОВЕРЯЙ КАЖДЫЙ ШАГ. После команды объясни, что я должен увидеть при успехе, и попроси прислать тебе вывод/скриншот/текст ошибки. Не двигайся дальше, пока шаг реально не сработал.
4. Если у меня ошибка — не сыпь теорией. Дай конкретное «сделай вот так» и попроси повторить. Помни про «известные грабли» (см. ниже) — узнавай их по симптомам и сразу давай готовое решение.
5. Веди счёт: в начале каждого сообщения пиши «Шаг N из ~M: <название>», чтобы я понимал, где мы и сколько осталось.
6. Файлы и код создавай за меня там, где можешь (если ты ИИ с доступом к файлам/терминалу — например Claude Code, делай сам и показывай результат). Если ты обычный чат без доступа к моему компьютеру — давай мне готовое содержимое файла и точную команду/путь, куда его сохранить.
7. Спрашивай разрешение перед необратимым (удаление, замена файлов). Мои ключи/пароли НЕ проси показывать в чат — объясни, что я вписываю их в файл .env у себя сам.
8. Тон — спокойный, ободряющий. Я могу нервничать. Хвали за пройденные шаги.

Не торопись и ничего не пропускай. Начинай всегда с приветствия и краткого плана (сколько примерно шагов впереди), затем — Этап 0.

ЧТО МЫ СТРОИМ (архитектура — она уже доказана работой)

Парсер — это скрипт на Python, который:
1. Заходит в Telegram под МОИМ аккаунтом (через Telethon, авторизация по QR-коду, сессия сохраняется в файл parser.session).
2. Слушает заданные мной чаты/группы по теме.
3. Фильтрует каждое новое сообщение в ДВА этапа:
   • Этап 1 (бесплатно, мгновенно): ключевые слова + анти-ключи (стоп-слова). Нет совпадения по ключам → выкидываем. Есть совпадение по анти-ключу → выкидываем.
   • Этап 2 (платно, умно): то, что прошло, отправляем в Claude Haiku с промптом «это реальная заявка или болтовня?». Haiku отвечает ДА/НЕТ + причина.
4. Одобренные заявки шлёт через бота в мой отдельный TG-канал/группу.

5. Раз в сутки шлёт статистику (сколько сообщений, сколько прошло ключи, сколько одобрил Haiku и т.д.).

Двухступенчатость — ключевой принцип. Ключи отсекают 95% мусора БЕСПЛАТНО, и только остаток идёт в платный Haiku. Без ИИ-фильтра будет много мусора и «перетекание» тем; без ключевого фильтра — разоришься на токенах.

ГЛАВНАЯ ИДЕЯ: понимание ниши формируется ПОСТЕПЕННО

Не пытайся выжать из меня идеальную конфигурацию сразу. На старте у меня будет грубая версия, дальше она дошлифуется на реальных данных. Каждая ниша = 6 связанных вещей, которые мы определяем вместе:

1. НИША — одна тема/рынок (например: «авто из Китая», «контент-маркетинг для онлайн-школ», «ремонт квартир»). Один парсер тянет несколько ниш, но каждая живёт отдельно.
2. КОГО ИЩЕМ (портрет лида) — кто наш клиент и какое его сообщение = заявка. Пример: «человек, который хочет КУПИТЬ/ЗАКАЗАТЬ услугу и пишет об этом в чате». НЕ заявка: тот, кто продаёт, рекламирует, спрашивает отзыв, или просто болтает.
3. КЛЮЧИ — фразы, по которым ловим (бесплатный фильтр №1). Пример: «посоветуйте компанию», «ищу подрядчика», «сколько стоит под ключ».
4. АНТИ-КЛЮЧИ — стоп-слова, при которых сразу выкидываем ДО Haiku (экономят деньги). Пример для авто из Китая: «правый руль», «из Японии», «продаю свой».
5. ЧАТЫ — список TG-групп/каналов по теме, куда я вступил своим аккаунтом.
6. ПРОМПТ HAIKU — текст-инструкция для ИИ: что считать заявкой (ДА), что не считать (НЕТ), и в каком формате отвечать.

ТВОЯ ЗАДАЧА на старте — вытащить из меня эти 6 вещей вопросами, по одной, на понятном языке. Если я не знаю — предложи разумный вариант на примере моей темы и спроси, подходит ли.

ПОРЯДОК РАБОТЫ (этапы)

ЭТАП 0. Понять мою нишу.
Спроси меня: какая ниша, кого я ищу (портрет лида), и приведи 3-5 примеров сообщений, которые БЫЛИ БЫ заявкой и которые НЕ были бы. На основе этого сам предложи черновой список ключей, анти-ключей и текст промпта для Haiku. Покажи мне на утверждение.

ЭТАП 1. Технические доступы. Дай мне пошаговую инструкцию получить:
(a) api_id и api_hash на https://my.telegram.org
⚠️ ИЗВЕСТНАЯ ГРАБЛЯ: форму часто отвергает. Готовое решение: включить VPN, открыть в режиме инкогнито, и в форме НЕ использовать слова bot / parser / api / telegram. Название приложения = MyTestApp, short name = mytestapp, платформа = Desktop. Это проходит.

(б) Anthropic API-ключ на https://console.anthropic.com → Billing (пополнить ~$5) → API Keys → создать ключ.
(в) Токен бота-отправщика у @BotFather в Telegram: /newbot → имя → username → получить токен.
Затем добавить этого бота АДМИНОМ в мой канал/группу выдачи (иначе он не сможет туда писать — частая ошибка).

ЭТАП 2. Создать файлы проекта. Создай папку проекта и положи туда КОД, который я даю ниже (раздел «КОД ПРОЕКТА»), БЕЗ ИЗМЕНЕНИЙ — он уже отлажен. Помоги создать виртуальное окружение и поставить зависимости:
python3 -m venv venv
source venv/bin/activate (на Windows: venv\Scripts\activate)
pip install telethon anthropic python-dotenv requests qrcode Pillow

ЭТАП 3. Заполнить .env моими ключами (см. шаблон в коде) и создать конфиг моей ниши в папке ниши/<моя_ниша>/ (4 текстовых файла: ключи.txt, антиключи.txt, чаты.txt, haiku_prompt.txt + niche.json). Шаблоны — в разделе «КОНФИГ НИШИ».

ЭТАП 4. Первый запуск и вход в Telegram:
source venv/bin/activate && python -u main.py
⚠️ ИЗВЕСТНАЯ ГРАБЛЯ №1 (QR-логин): чистый QR от Telethon новые версии Telegram иногда отвергают. Готовое решение УЖЕ в коде — кроме QR-картинки он печатает в терминал РЕЗЕРВНУЮ ССЫЛКУ tg://login?token=... Скопируй её, отправь себе в Telegram → «Избранное», и нажми на неё ОТКРЫТУЮ С САМОГО ТЕЛЕФОНА. Это надёжнее камеры.
⚠️ ИЗВЕСТНАЯ ГРАБЛЯ №2 (2FA / облачный пароль): если на аккаунте включён облачный пароль, после QR терминал спросит его. Код это уже умеет (через getpass). ВАЖНО: при вводе пароля символы НЕ видны на экране — это нормально, просто введи и нажми Enter.
После входа сессия сохранится в parser.session — больше логиниться не нужно.

ЭТАП 5. Тюнинг (самое важное для качества). Объясни мне:
• Сутки-двое парсер копит находки. Я смотрю канал выдачи и лог-топик (туда падают ОТКЛОНЁННЫЕ Haiku сообщения — по ним видно, что ИИ зря отсеял или что мусор всё ещё лезет).
• По результатам мы правим: добавляем ключи (если что-то пропустили), добавляем анти-ключи (если лезет мусор определённого типа), уточняем промпт Haiku (если ИИ ошибается в спорных случаях).
• Правки в .txt файлах применяются после перезапуска парсера. Менять ключи/чаты я могу сам, без программиста — это просто текстовые файлы.

ЭТАП 6 (опционально). Деплой на сервер, чтобы парсер работал 24/7 без моего компьютера. Когда я захочу — дай инструкцию из раздела «ДЕПЛОЙ НА VPS».

ВАЖНЫЕ ПРАВИЛА И ГОТОВЫЕ РЕШЕНИЯ ГРАБЛЕЙ (вшиты в код, не трогай)

• Анти-ключи проверяются ДО отправки в Haiku — это экономит деньги на токенах. Порядок: длина текста → ключи → анти-ключи → Haiku.
• Совпадение ключей идёт по ГРАНИЦАМ СЛОВ (regex с \w-границами), чтобы ключ «smm» не ловил «smm_prodvigator». Это уже в filter.py.
• В .txt файлах строка-комментарий — только если начинается с «# » (решётка + ПРОБЕЛ). Поэтому ключ-хэштег «#помогу» работает как ключ, а «# это комментарий» игнорируется.
• min_text_length (по умолчанию 10) отсекает слишком короткие сообщения до проверки ключей. Если ловишь короткие важные сообщения — снизь до 4-5 в niche.json.
• catch_up при старте подтягивает сообщения, пропущенные пока парсер был офлайн.
• Логи Telethon заглушены (level=CRITICAL): ошибки PersistentTimestampOutdatedError — это норма, Telethon сам восстанавливается, засорять лог не нужно.
• Приватные чаты по invite-ссылке (t.me/+HASH) поддержаны: в чаты.txt вставляй такую ссылку как есть, НО сначала вступи в группу вручную своим аккаунтом.
• Бот-отправщик должен быть АДМИНОМ канала/группы выдачи.
• Если выдача — форум-супергруппа с топиками, в niche.json укажи topic_leads (куда слать заявки) и topic_logs (куда слать отклонённые для тюнинга). Если обычный канал — просто chat_id, топики не нужны.
• Один чат = одна ниша. Если впишешь чат в две ниши, парсер откажется стартовать (защита от путаницы).
• БЕЗОПАСНОСТЬ: ключи и пароли — только в .env, он в .gitignore и в git не попадает. На сервер секреты заливаю Я САМ, никому не показываю.

═══

КОД ПРОЕКТА (создай эти файлы точно как есть)

═══

——— ФАЙЛ: config.py ———

```
"""Загрузка конфигурации парсера: .env + папки ниш."""
import json
import os
from pathlib import Path
from dotenv import load_dotenv

ROOT = Path(__file__).parent
load_dotenv(ROOT / ".env")

TELEGRAM_API_ID = int(os.environ["TELEGRAM_API_ID"])
TELEGRAM_API_HASH = os.environ["TELEGRAM_API_HASH"]
ANTHROPIC_API_KEY = os.environ["ANTHROPIC_API_KEY"]
SENDER_BOT_TOKEN = os.environ["SENDER_BOT_TOKEN"]

SESSION_FILE = str(ROOT / "parser.session")
NICHES_DIR = ROOT / "ниши"


def _read_lines(path: Path) -> list[str]:
```

```
    if not path.exists():
        return []
    lines = []
    for raw in path.read_text(encoding="utf-8").splitlines():
        s = raw.strip()
        # Комментарий — только если строка начинается с '# ' или это просто '#'.
        # Это позволяет писать ключи-хэштеги типа '#помогу' без экранирования.
        if not s or s == "#" or s.startswith("# "):
            continue
        lines.append(s)
    return lines


def _extract_chat_identifier(link: str) -> str:
    """Возвращает строку для get_entity или 'invite:HASH' для приватной группы."""
    s = link.strip().rstrip("/")
    if s.startswith("@"):
        return s[1:]
    if "t.me/" in s:
        tail = s.split("t.me/", 1)[1]
        if tail.startswith("c/"):
            return s
        if tail.startswith("+"):
            return f"invite:{tail[1:]}"
        return tail.split("/")[0]
    return s


def _load_niche(niche_dir: Path) -> dict:
    config_path = niche_dir / "niche.json"
    cfg = json.loads(config_path.read_text(encoding="utf-8"))
    files = cfg.get("files", {})

    cfg["dir"] = niche_dir
    cfg["keywords"] = [k.lower() for k in _read_lines(niche_dir / files.get("keywords",
"ключи.txt"))]
    cfg["antikeys"] = [k.lower() for k in _read_lines(niche_dir / files.get("antikeys",
"антиключи.txt"))]
    cfg["chats"] = [_extract_chat_identifier(c) for c in _read_lines(niche_dir / files.get("chats",
"чаты.txt"))]

    prompt_path = niche_dir / files.get("prompt", "haiku_prompt.txt")
    cfg["prompt"] = prompt_path.read_text(encoding="utf-8") if prompt_path.exists() else ""

    cfg.setdefault("min_text_length", 10)
    return cfg
```

```
def load_all_niches() -> dict:
    """Сканит ниши/*/niche.json. Проверяет пересечения чатов между нишами."""
    niches = {}
    if not NICHES_DIR.exists():
        return niches
    for niche_dir in sorted(NICHES_DIR.iterdir()):
        if not niche_dir.is_dir() or not (niche_dir / "niche.json").exists():
            continue
        cfg = _load_niche(niche_dir)
        niches[cfg["slug"]] = cfg

    seen = {}
    for slug, cfg in niches.items():
        for chat in cfg["chats"]:
            if chat in seen:
                raise ValueError(
                    f"Чат '{chat}' указан в двух нишах: '{seen[chat]}' и '{slug}'. "
                    f"Один чат — одна ниша."
                )
            seen[chat] = slug
    return niches


——— ФАЙЛ: filter.py ———
"""Двухступенчатая фильтрация: ключевые слова → Claude Haiku."""
import re
from functools import lru_cache

from anthropic import Anthropic
from config import ANTHROPIC_API_KEY

client = Anthropic(api_key=ANTHROPIC_API_KEY)

HAIKU_MODEL = "claude-haiku-4-5-20251001"


@lru_cache(maxsize=4096)
def _kw_pattern(kw: str) -> re.Pattern:
    # Границы по word-char, чтобы «smm» не ловило «smm_prodvigator».
    return re.compile(r"(?<!\w)" + re.escape(kw.lower()) + r"(?!\w)", re.UNICODE)


def keyword_match(text: str, keywords: list[str]) -> str | None:
    """Возвращает первое совпавшее ключевое слово (lowercase) или None."""
    if not text:
        return None
    low = text.lower()
    for kw in keywords:
```

```
        if _kw_pattern(kw).search(low):
            return kw
    return None


def ai_verdict(text: str, system_prompt: str) -> tuple[bool, str, str]:
    """Возвращает (одобрено?, причина, приоритет)."""
    msg = client.messages.create(
        model=HAIKU_MODEL,
        max_tokens=150,
        system=system_prompt,
        messages=[{"role": "user", "content": text[:1500]}],
    )
    out = msg.content[0].text.strip()
    approved = "ВЕРДИКТ: ДА" in out.upper()
    reason = ""
    priority = "ОБЫЧНЫЙ"
    for line in out.splitlines():
        upper = line.upper()
        if upper.startswith("ПРИЧИНА"):
            reason = line.split(":", 1)[-1].strip()
        elif upper.startswith("ПРИОРИТЕТ"):
            value = line.split(":", 1)[-1].strip().upper()
            if "ВЫСОК" in value:
                priority = "ВЫСОКИЙ"
    if not reason:
        reason = out[:200]
    return approved, reason, priority


——— ФАЙЛ: sender.py ———
"""Отправка сообщений в TG через бот."""
import requests
from config import SENDER_BOT_TOKEN

API = f"https://api.telegram.org/bot{SENDER_BOT_TOKEN}"


def send_to_channel(channel_id: int | str, text: str, topic_id: int | None = None) -> dict:
    """Шлёт сообщение в канал/группу. topic_id — если форум-супергруппа, id топика."""
    payload = {
        "chat_id": channel_id,
        "text": text,
        "parse_mode": "HTML",
        "disable_web_page_preview": True,
    }
    if topic_id is not None:
        payload["message_thread_id"] = topic_id
```

```
    r = requests.post(f"{API}/sendMessage", json=payload, timeout=15)
    return r.json()


def get_me() -> dict:
    return requests.get(f"{API}/getMe", timeout=10).json()


——— ФАЙЛ: main.py ———
"""Парсер Telegram: мультиниши."""
import asyncio
import json
import logging
import subprocess
from datetime import datetime, timedelta
from pathlib import Path
from zoneinfo import ZoneInfo
from getpass import getpass

import qrcode
from telethon import TelegramClient, events, utils
from telethon.errors import SessionPasswordNeededError, UserAlreadyParticipantError
from telethon.tl.functions.messages import ImportChatInviteRequest,
CheckChatInviteRequest

from config import TELEGRAM_API_ID, TELEGRAM_API_HASH, SESSION_FILE,
load_all_niches
from filter import keyword_match, ai_verdict
from sender import send_to_channel

STATE_FILE = Path(__file__).parent / "state.json"
_COUNTER_KEYS = ("seen", "kw_matched", "antikey_blocked",
                 "haiku_approved", "haiku_rejected", "haiku_errors", "send_errors")
# ⚙️ ПОМЕНЯЙ на свой часовой пояс, например ZoneInfo("Europe/Moscow")
MSK = ZoneInfo("Europe/Moscow")
TZ_LABEL = "MSK"
REPORT_HOUR_MSK = 18  # час суточного отчёта статистики

logging.getLogger("telethon").setLevel(logging.CRITICAL)


def load_state() -> dict:
    if STATE_FILE.exists():
        return json.loads(STATE_FILE.read_text())
    return {}


def save_state(state: dict) -> None:
```

```
    STATE_FILE.write_text(json.dumps(state, ensure_ascii=False, indent=2))


def print_qr(url: str) -> None:
    qr = qrcode.QRCode(border=4, box_size=12,
error_correction=qrcode.constants.ERROR_CORRECT_M)
    qr.add_data(url)
    qr.make(fit=True)
    img = qr.make_image(fill_color="black", back_color="white")
    path = Path(__file__).parent / "qr.png"
    img.save(path)
    print("\n" + "=" * 60, flush=True)
    print("QR сохранён в файл:", path, flush=True)
    print("Открой его и в TG: Настройки → Устройства → Подключить устройство →
сканер", flush=True)
    print("=" * 60 + "\n", flush=True)
    # На macOS можно автооткрыть: subprocess.run(["open", str(path)], check=False)


async def qr_login(client: TelegramClient) -> None:
    if await client.is_user_authorized():
        me = await client.get_me()
        print(f"Уже залогинены: {me.first_name} (@{me.username})")
        return

    while True:
        qr_login_obj = await client.qr_login()
        print_qr(qr_login_obj.url)
        print("\n📋 РЕЗЕРВНЫЙ ВАРИАНТ — если QR не читается:")
        print("   Скопируй ссылку ниже, отправь себе в Telegram → 'Избранное'")
        print("   и нажми на неё ОТКРЫТУЮ С САМОГО ТЕЛЕФОНА:\n")
        print(f"   {qr_login_obj.url}\n", flush=True)
        try:
            await qr_login_obj.wait(timeout=60)
            break
        except asyncio.TimeoutError:
            print("\n❌ QR истёк. Генерирую новый...\n", flush=True)
            continue
        except SessionPasswordNeededError:
            print("\n🔐 Аккаунт защищён облачным паролем (2FA).", flush=True)
            while True:
                pw = getpass("Введите облачный пароль Telegram (символы не видны — это
норма): ")
                try:
                    await client.sign_in(password=pw)
                    break
                except Exception as e:
                    print(f"   ✗ Неверный пароль ({e}). Попробуйте снова.\n", flush=True)
```

```
            break

    me = await client.get_me()
    print(f"\n✓ Успешный вход: {me.first_name} (@{me.username})\n", flush=True)


async def resolve_output_by_invite(client, invite_link, state_key, state) -> int:
    if state_key in state:
        try:
            await client.get_entity(state[state_key])
            return state[state_key]
        except Exception:
            print(f"Сохранённый ID канала ({state_key}) недоступен, пробую через invite.")
    invite_hash = invite_link.split("/+")[-1].split("/")[-1]
    if invite_hash.startswith("+"):
        invite_hash = invite_hash[1:]
    try:
        result = await client(ImportChatInviteRequest(invite_hash))
        chat = result.chats[0]
        print(f"Присоединился к каналу: {chat.title} (id={chat.id})")
    except UserAlreadyParticipantError:
        info = await client(CheckChatInviteRequest(invite_hash))
        chat = info.chat
        print(f"Уже в канале: {chat.title} (id={chat.id})")
    bot_api_id = int(f"-100{chat.id}")
    state[state_key] = bot_api_id
    save_state(state)
    return bot_api_id


async def resolve_niche_output(client, niche, state) -> dict:
    out = niche["output"]
    if "chat_id" in out:
        return {"chat_id": out["chat_id"], "topic_leads": out.get("topic_leads"),
                "topic_logs": out.get("topic_logs")}
    if "invite" in out:
        state_key = f"output_channel_id__{niche['slug']}"
        chat_id = await resolve_output_by_invite(client, out["invite"], state_key, state)
        return {"chat_id": chat_id, "topic_leads": None, "topic_logs": None}
    raise ValueError(f"Ниша '{niche['slug']}': в output нет ни chat_id, ни invite.")


async def resolve_source_chats(client, chat_identifiers) -> dict:
    chats = {}
    for ident in chat_identifiers:
        try:
            if ident.startswith("invite:"):
                invite_hash = ident.split(":", 1)[1]
```

```
                info = await client(CheckChatInviteRequest(invite_hash))
                chat = getattr(info, "chat", None)
                if chat is None:
                    print(f"  ✗ {ident} — не вступили в группу (откройте ссылку в TG)")
                    continue
                entity = await client.get_entity(chat.id)
            else:
                entity = await client.get_entity(ident)
            title = getattr(entity, "title", None) or getattr(entity, "username", str(entity.id))
            peer_id = utils.get_peer_id(entity)
            chats[peer_id] = title
            print(f"  ✓ {title} (id={peer_id})")
        except Exception as e:
            print(f"  ✗ {ident} — {e}")
    return chats


def _escape(text: str) -> str:
    return text.replace("<", "&lt;").replace(">", "&gt;")


def message_link(chat, msg_id: int) -> str:
    if getattr(chat, "username", None):
        return f"https://t.me/{chat.username}/{msg_id}"
    cid = chat.id
    if str(cid).startswith("-100"):
        cid = int(str(cid)[4:])
    return f"https://t.me/c/{cid}/{msg_id}"


def format_lead(niche, text, chat_title, msg_link, sender, reason, priority) -> str:
    safe_text = _escape(text)
    if len(safe_text) > 2000:
        safe_text = safe_text[:2000] + "…"
    prio_tag = "🔥 ПРИОРИТЕТ " if priority == "ВЫСОКИЙ" else ""
    return (
        f"<b>{prio_tag}{niche['emoji']} Заявка — {niche['name']}</b>\n\n"
        f"{safe_text}\n\n"
        f"━━━━━━━━━━━━━━━━━━━━━━━━━━\n"
        f"<b>Чат:</b> {_escape(chat_title)}\n"
        f"<b>Автор:</b> {_escape(sender)}\n"
        f"<b>ИИ-вердикт:</b> {_escape(reason)}\n"
        f'<a href="{msg_link}">→ Открыть сообщение</a>'
    )


def format_log_entry(niche, approved, text, chat_title, msg_link, sender, reason, priority) ->
str:
```

```
    safe_text = _escape(text)
    if len(safe_text) > 600:
        safe_text = safe_text[:600] + "…"
    marker = "✓ ОДОБРЕНО" if approved else "✗ ОТКЛОНЕНО"
    prio = " 🔥" if approved and priority == "ВЫСОКИЙ" else ""
    return (
        f"<b>{marker}{prio} — {niche['emoji']} {niche['name']}</b>\n"
        f"<b>Чат:</b> {_escape(chat_title)} | <b>Автор:</b> {_escape(sender)}\n"
        f"<b>Причина:</b> {_escape(reason)}\n\n"
        f"{safe_text}\n\n"
        f'<a href="{msg_link}">→ Открыть сообщение</a>'
    )


async def main() -> None:
    niches = load_all_niches()
    if not niches:
        print("❌ Не найдено ни одной ниши в ниши/*/niche.json. Стоп.")
        return

    print(f"Загружено ниш: {len(niches)}")
    for slug, cfg in niches.items():
        print(f"  {cfg['emoji']} {cfg['name']}: {len(cfg['chats'])} чатов, "
              f"{len(cfg['keywords'])} ключей, {len(cfg['antikeys'])} антикл.")

    state = load_state()
    client = TelegramClient(SESSION_FILE, TELEGRAM_API_ID, TELEGRAM_API_HASH)
    await client.connect()
    await qr_login(client)

    chat_to_niche: dict[int, str] = {}
    source_chat_titles: dict[int, str] = {}
    stats_by_niche: dict[str, dict] = {}

    for slug, cfg in niches.items():
        print(f"\n=== {cfg['emoji']} {cfg['name']} ===")
        print("Канал выдачи:")
        cfg["_output"] = await resolve_niche_output(client, cfg, state)
        print(f"  chat_id={cfg['_output']['chat_id']} "
              f"topic_leads={cfg['_output']['topic_leads']} "
              f"topic_logs={cfg['_output']['topic_logs']}")

        print("Источники:")
        sources = await resolve_source_chats(client, cfg["chats"])
        if not sources:
            print(f"  ⚠️ Ни один чат не зарезолвился — ниша '{slug}' будет пустой.")
        for eid, title in sources.items():
            if eid in chat_to_niche:
```

```
                print(f"  ⚠️  Чат {title} ({eid}) уже в нише '{chat_to_niche[eid]}', пропускаю.")
                continue
            chat_to_niche[eid] = slug
            source_chat_titles[eid] = title

        stats_by_niche[slug] = {k: 0 for k in _COUNTER_KEYS}
        stats_by_niche[slug]["since"] = datetime.now(MSK)

        out = cfg["_output"]
        ping = send_to_channel(out["chat_id"],
                               f"<b>{cfg['emoji']} Парсер запущен — {cfg['name']}</b>\nЖду
сообщения...",
                               topic_id=out["topic_leads"])
        if not ping.get("ok"):
            print(f"  ⚠️  Не удалось отправить тест в leads: {ping}")
        else:
            print("  ✓ Тест отправлен в leads.")
        if out["topic_logs"] is not None:
            send_to_channel(out["chat_id"],
                            f"<b>{cfg['emoji']} Лог-топик активирован — {cfg['name']}</b>\n"
                            f"Сюда падают отклонённые Haiku сообщения (для тюнинга).",
                            topic_id=out["topic_logs"])

    source_ids = list(chat_to_niche.keys())
    if not source_ids:
        print("\n❌ Ни одного source-чата не зарезолвилось. Стоп.")
        return

    print(f"\nИтого слушаем {len(source_ids)} чатов из {len(niches)} ниш.\n")

    @client.on(events.NewMessage(chats=source_ids))
    async def on_message(event):
        slug = chat_to_niche.get(event.chat_id)
        if slug is None:
            return
        niche = niches[slug]
        st = stats_by_niche[slug]
        st["seen"] += 1

        text = event.message.message or ""
        if not text or len(text) < niche["min_text_length"]:
            return

        matched = keyword_match(text, niche["keywords"])
        if not matched:
            return
        st["kw_matched"] += 1
```

```
        anti_hit = keyword_match(text, niche["antikeys"])
        if anti_hit:
            st["antikey_blocked"] += 1
            print(f"  ⛔ [{niche['name']}] антиключ «{anti_hit}» — пропускаем")
            return

        chat = await event.get_chat()
        chat_title = source_chat_titles.get(event.chat_id, getattr(chat, "title", "?"))
        sender = await event.get_sender()
        sender_name = getattr(sender, "first_name", "") or ""
        if getattr(sender, "username", None):
            sender_name += f" @{sender.username}"
        sender_name = sender_name.strip() or "Аноним"

        msg_dt = event.message.date.astimezone(MSK).strftime("%d.%m %H:%M:%S")
        print(f"\n[{msg_dt} {TZ_LABEL}] [{niche['name']}] [{chat_title}] kw
«{matched}»:\n{text}\n---")

        try:
            approved, reason, priority = ai_verdict(text, niche["prompt"])
        except Exception as e:
            st["haiku_errors"] += 1
            print(f"  ✗ Haiku ошибка: {e}")
            return

        msg_link = message_link(chat, event.message.id)
        out = niche["_output"]

        if not approved:
            st["haiku_rejected"] += 1
            print(f"  ✗ Haiku отклонил: {reason}")
            if out["topic_logs"] is not None:
                log_text = format_log_entry(niche, approved, text, chat_title, msg_link,
sender_name, reason, priority)
                send_to_channel(out["chat_id"], log_text, topic_id=out["topic_logs"])
            return

        st["haiku_approved"] += 1
        print(f"  ✓ Одобрено [{priority}]: {reason}")
        lead_text = format_lead(niche, text, chat_title, msg_link, sender_name, reason, priority)
        res = send_to_channel(out["chat_id"], lead_text, topic_id=out["topic_leads"])
        if not res.get("ok"):
            st["send_errors"] += 1
            print(f"  ✗ Отправка не удалась: {res}")

    def _next_report_dt() -> datetime:
        now = datetime.now(MSK)
```

```
        target = now.replace(hour=REPORT_HOUR_MSK, minute=0, second=0,
microsecond=0)
        if target <= now:
            target += timedelta(days=1)
        return target

    async def stats_reporter():
        while True:
            target = _next_report_dt()
            await asyncio.sleep((target - datetime.now(MSK)).total_seconds())
            for slug, cfg in niches.items():
                st = stats_by_niche[slug]
                niche_chat_count = sum(1 for s in chat_to_niche.values() if s == slug)

                def row(label, key):
                    return f"{label}: <b>{st[key]}</b>\n"

                report = (
                    f"<b>📊 Статистика — {cfg['emoji']} {cfg['name']}</b>\n"
                    f"<i>{st['since'].strftime('%d.%m %H:%M')} → "
                    f"{target.strftime('%d.%m %H:%M')} {TZ_LABEL}</i>\n\n"
                    f"Чатов на прослушке: <b>{niche_chat_count}</b>\n"
                    + row("Всего сообщений", "seen")
                    + row("Прошли ключи", "kw_matched")
                    + row("⛔ Антиключ заблокировал", "antikey_blocked")
                    + row("✓ Haiku одобрил (заявок)", "haiku_approved")
                    + row("✗ Haiku отклонил", "haiku_rejected")
                    + row("⚠ Haiku ошибки", "haiku_errors")
                    + row("Ошибки отправки", "send_errors")
                )
                out = cfg["_output"]
                try:
                    send_to_channel(out["chat_id"], report, topic_id=out["topic_leads"])
                except Exception as e:
                    print(f"⚠ Статистика [{slug}]: {e}")
                for k in _COUNTER_KEYS:
                    st[k] = 0
                st["since"] = target

    asyncio.create_task(stats_reporter())

    print("Catching up missed updates...", flush=True)
    try:
        await client.catch_up()
        print("  ✓ catch_up done", flush=True)
    except Exception as e:
        print(f"  ⚠ catch_up failed: {e}", flush=True)
```

```
    print("\n=== Парсер слушает чаты. Ctrl+C для остановки ===\n")
    await client.run_until_disconnected()


if __name__ == "__main__":
    asyncio.run(main())


──────── ФАЙЛ: run.sh (для запуска и сервера) ────────
#!/bin/bash
cd "$(dirname "$0")"
source venv/bin/activate
exec python -u main.py


──────── ФАЙЛ: .gitignore ────────
.env
.env.*
!.env.example
*.session
*.session-journal
venv/
__pycache__/
*.pyc
state.json
qr.png
*.log
.DS_Store


──────── ФАЙЛ: .env (создай и впиши свои значения; в git НЕ коммить) ────────
# Telegram API (с my.telegram.org)
TELEGRAM_API_ID=сюда_число
TELEGRAM_API_HASH=сюда_строка

# Anthropic API (Claude Haiku)
ANTHROPIC_API_KEY=sk-ant-...

# Бот-отправщик (токен из @BotFather)
SENDER_BOT_TOKEN=123456:ABC...


═══════════════════════════════════════════════════════
КОНФИГ НИШИ (создай папку ниши/<моя_ниша>/ и 5 файлов)
═══════════════════════════════════════════════════════
Пример папки: ниши/моя_ниша/

──────── ниши/моя_ниша/niche.json ────────
```

```
{
  "slug": "моя_ниша",
  "name": "Моя Ниша",
  "emoji": "🎯",
  "files": {
    "chats": "чаты.txt",
    "keywords": "ключи.txt",
    "antikeys": "антиключи.txt",
    "prompt": "haiku_prompt.txt"
  },
  "min_text_length": 10,
  "output": {
    "chat_id": -1001234567890,
    "topic_leads": null,
    "topic_logs": null
  }
}
```

ПОЯСНЕНИЕ по output:

• chat_id — числовой ID моего канала/группы выдачи. Как узнать: добавь бота админом в канал, отправь туда любое сообщение, открой https://api.telegram.org/bot<ТОКЕН>/getUpdates — там будет "chat":{"id":-100...}. Этот id и вставь.

• Если выдача — обычный канал/группа без топиков: topic_leads и topic_logs = null.

• Если выдача — форум-супергруппа с топиками: укажи числовые id топиков (leads — куда заявки, logs — куда отклонённые для тюнинга).

——— ниши/моя_ниша/чаты.txt ———

```
# Чаты для парсинга. Одна ссылка/юзернейм в строке. Строки с '# ' — комментарии.
# Публичные: https://t.me/имячата  или  @имячата
# Приватные по invite: https://t.me/+ABCdefHASH  (сначала вступи в группу вручную!)
https://t.me/example_chat_1
@example_chat_2
```

——— ниши/моя_ниша/ключи.txt ———

```
# Ключевые слова/фразы. Одна в строке. Регистр не важен. Совпадение по границам слов.
# Это ФИЛЬТР №1 (бесплатный). Цель — поймать всё потенциально релевантное.
# Примеры (замени под свою нишу):
посоветуйте компанию
ищу подрядчика
кто делает
сколько стоит
нужен специалист
```

——— ниши/моя_ниша/антиключи.txt ———

```
# Анти-ключи (стоп-слова). Если ХОТЯ БЫ ОДНО встречается — выкидываем ДО Haiku (экономим деньги).
# Сюда: явный мусор, чужие подниши, признаки рекламы/продажи.
```

Примеры (замени под свою нишу):
продаю
вакансия
резюме
реклама

——— ниши/моя_ниша/haiku_prompt.txt ———
Ты — фильтр заявок для <ОПИШИ БИЗНЕС>.

ЧТО СЧИТАТЬ ЗАЯВКОЙ (ДА):
- <человек ищет/хочет купить/заказать ...>
- <прямой запрос: «посоветуйте», «ищу», «сколько стоит» ...>
- Сомневаешься между ДА и НЕТ — ставь ДА (лучше лишний лид, чем пропущенный клиент).

ЧТО НЕ ЗАЯВКА (НЕТ):
- <продаёт/рекламирует/спрашивает отзыв/чужая подниша/просто болтает>
- <сотрудник конкурента, перекуп, спам>

Отвечай СТРОГО в формате:
ВЕРДИКТ: ДА|НЕТ
ПРИЧИНА: одна короткая фраза (если НЕТ — назови конкретно, например «реклама», «не запрос», «чужая тема»)

═══
ДЕПЛОЙ НА VPS (этап 6, когда захочу 24/7)
═══
Цель — чтобы парсер работал на сервере без моего компьютера. Дай инструкцию:
1. Арендовать любой дешёвый VPS с Ubuntu 22.04/24.04 (РФ-блокировки Telegram учти: бери сервер ВНЕ России — например, Европа; иначе MTProto может не достучаться).
2. Поставить Python и git, создать /opt/parser/, скопировать туда файлы проекта (КРОМЕ .env и parser.session — их заливаю отдельно сам).
3. Создать venv, поставить зависимости.
4. Залить parser.session с локальной машины (scp) — чтобы не логиниться заново на сервере.
5. Залить .env вручную (секреты не показываю агенту).
6. Создать systemd-юнит /etc/systemd/system/parser.service:
```
[Unit]
Description=Telegram Parser
After=network-online.target
[Service]
WorkingDirectory=/opt/parser
ExecStart=/opt/parser/venv/bin/python -u /opt/parser/main.py
Restart=always
RestartSec=10
StandardOutput=append:/var/log/parser.log
StandardError=append:/var/log/parser.log
```

[Install]
WantedBy=multi-user.target
7. systemctl daemon-reload && systemctl enable --now parser
8. Проверка: systemctl status parser ; tail -f /var/log/parser.log
Управление: systemctl restart|stop parser. Правки конфигов: меняю .txt → перезаливаю → systemctl restart parser.

НАЧНИ С ЭТАПА 0: задай мне вопросы про мою нишу и портрет лида.