Введение

Система amoCRM — один из самых популярных инструментов для автоматизации продаж. Она предоставляет множество полезных функций для организации работы с клиентами, среди которых выделяется модуль чатов. Чаты позволяют пользователям amoCRM обмениваться сообщениями с клиентами, причем клиенты могут использовать свои привычные мессенджеры для коммуникации.

Однако, несмотря на обширный функционал и удобство использования, amoCRM предоставляет примеры интеграции только на PHP, что вводит дополнительные сложности для разработчиков, работающих с другими языками программирования.

В нашем проекте как раз возникла задача интеграции amoCRM API чатов в Telegram бота, написанного на Python. Это позволило пользователям переводить диалог на оператора: бот создавал диалог, направляя сообщения пользователя из Telegram в amoCRM и передавая ответы менеджера наоборот из amoCRM в Telegram. Для реализации этого решения нам понадобилось адаптировать документацию с PHP на Python.

В этой статье мы рассмотрим, как интегрировать amoCRM API чатов на Python, чтобы разработчики могли быстро и эффективно использовать этот функционал в своих проектах.

Практическая часть

Подготовка

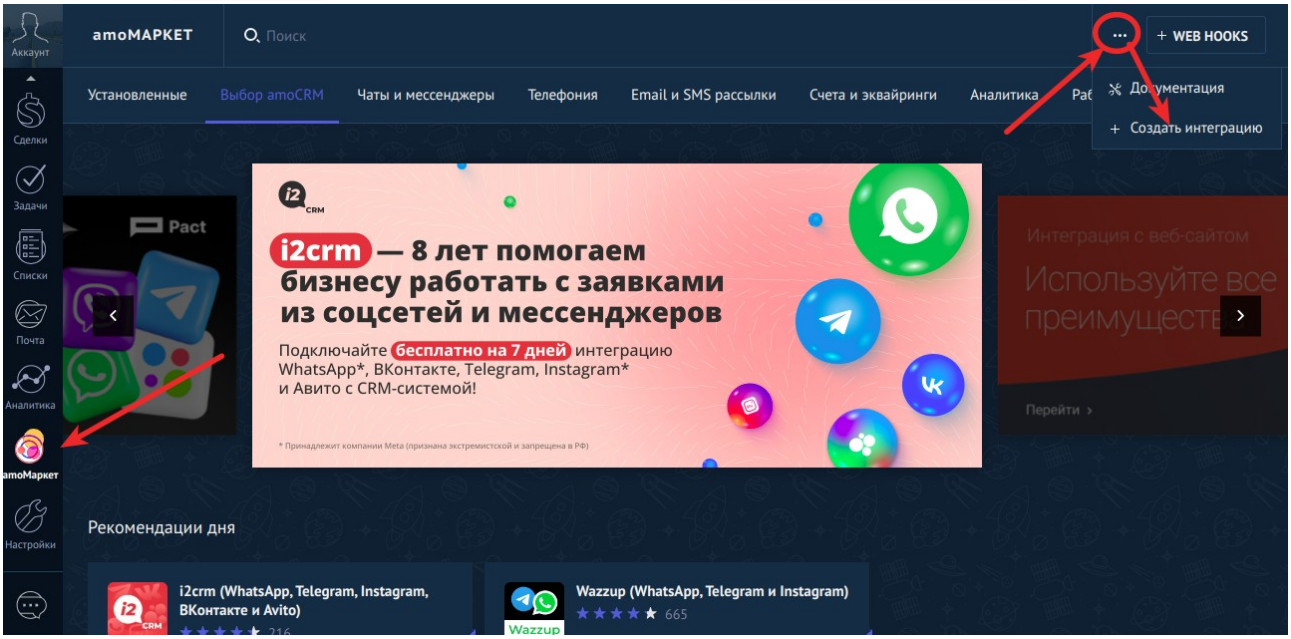
Прежде чем приступить к интеграции, необходимо ознакомиться с документацией по API чатов amoCRM. Вместо пересказа содержания этой документации, я отмечу пару неочевидных моментов:

1. Сервер, на котором будет открытый публичный endpoint (Webhook URL), должен иметь SSL-сертификат. При этом наличие домена не обязательно, что было подтверждено поддержкой;
2. Нужно создать внешнюю интеграцию и указать `client_uuid` интеграции в заявке на регистрацию канала (пункт 8). Ниже более подробно обсудим, как создать такую интеграцию;

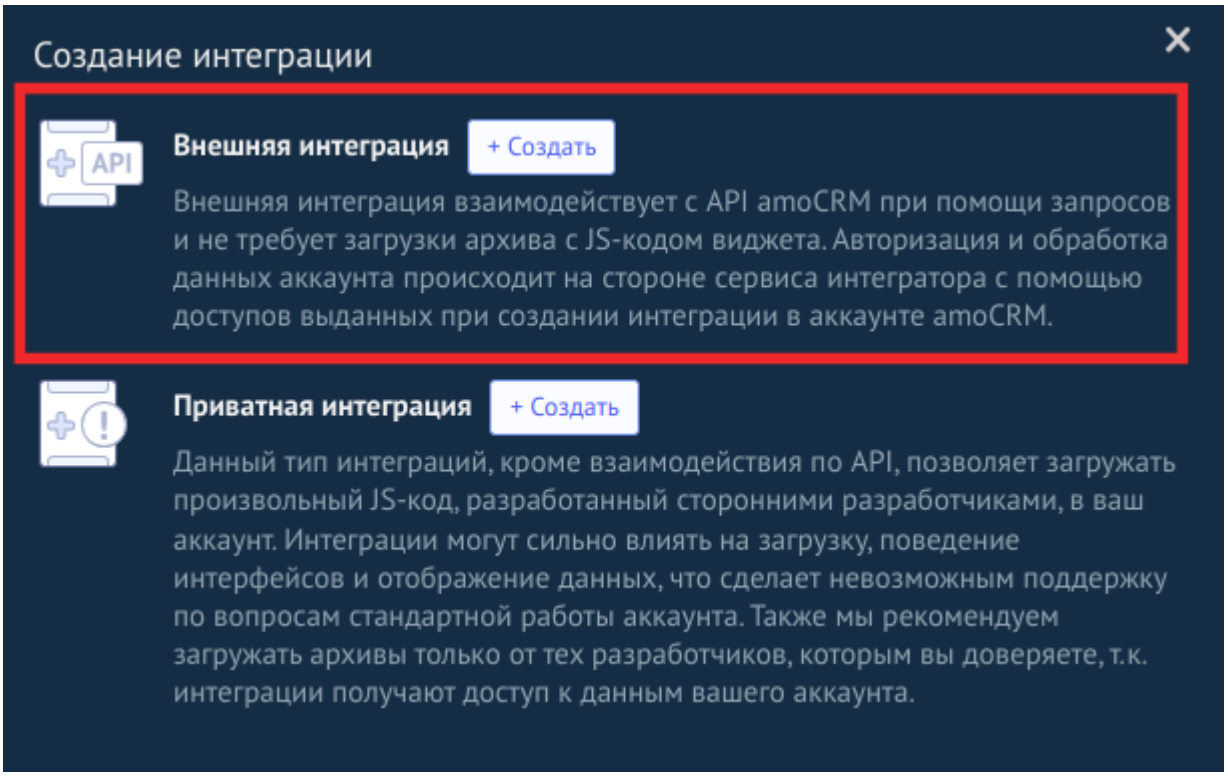
Создание внешней интеграции

Для создания внешней интеграции нужно:

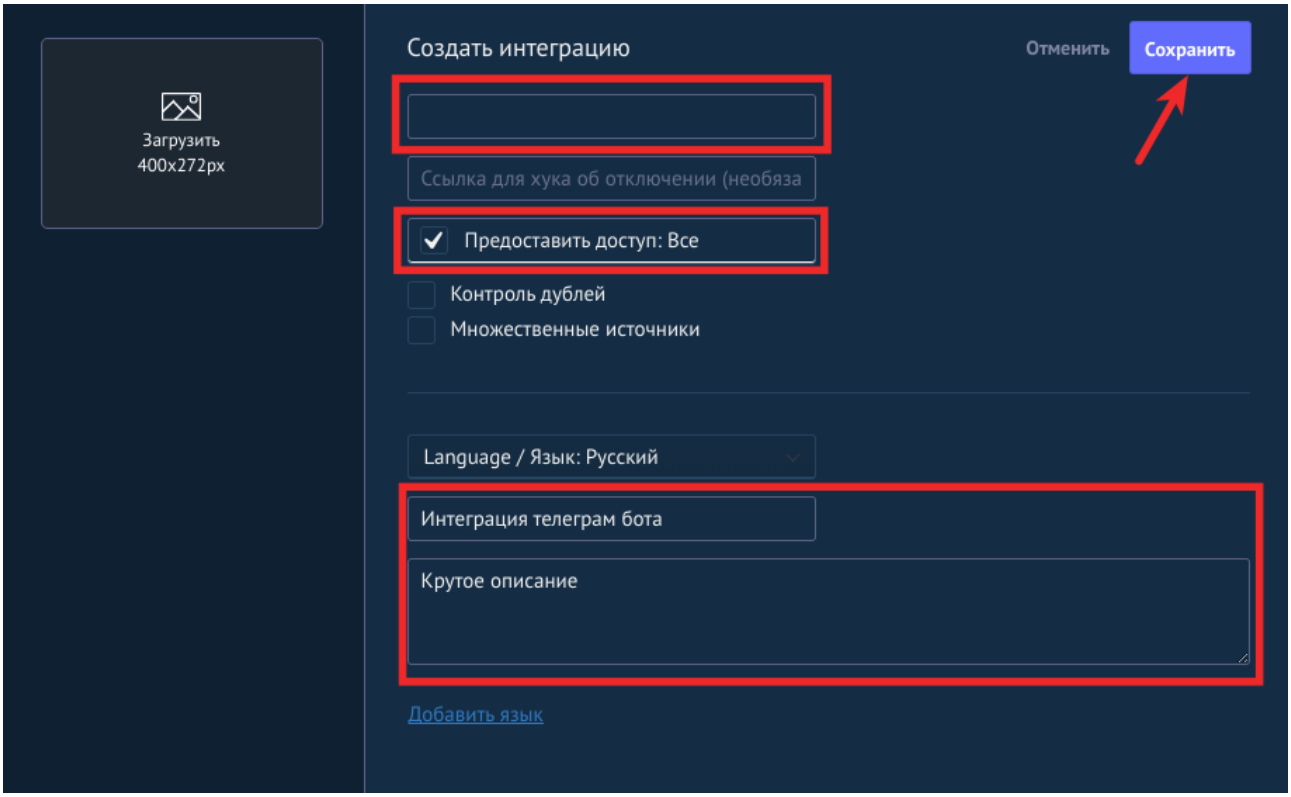
1. Перейти в `amoМаркет` и далее в меню выбрать `Создать интеграцию`:



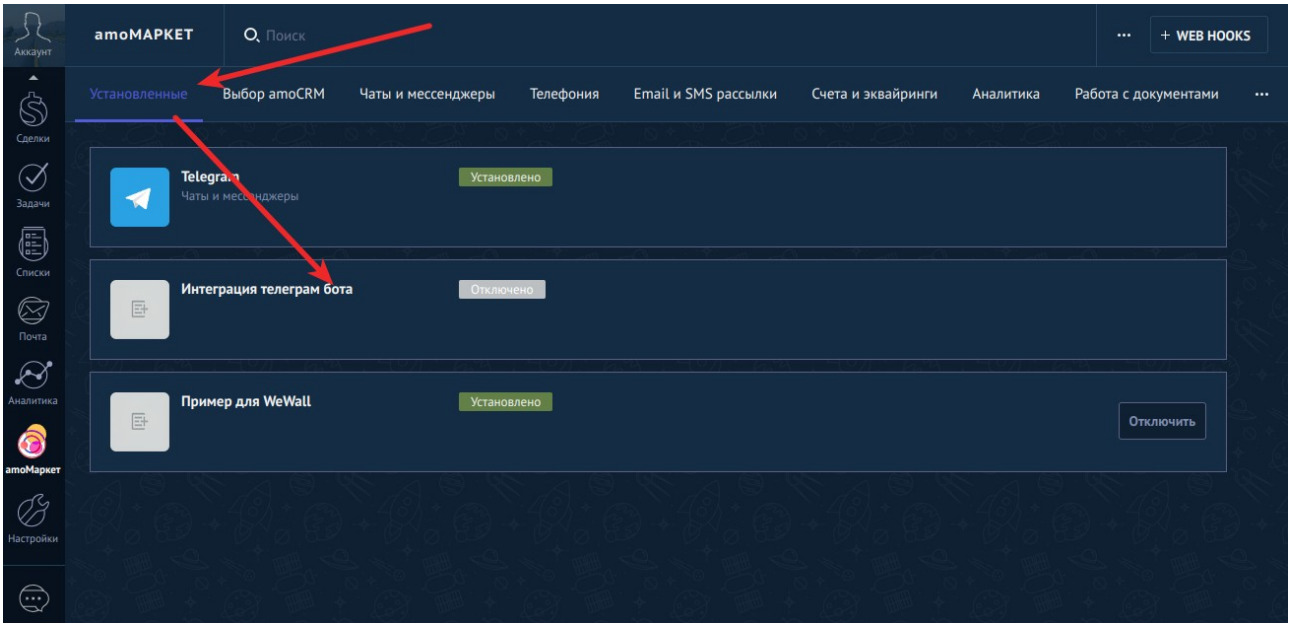
2. Выбрать внешнюю интеграцию:

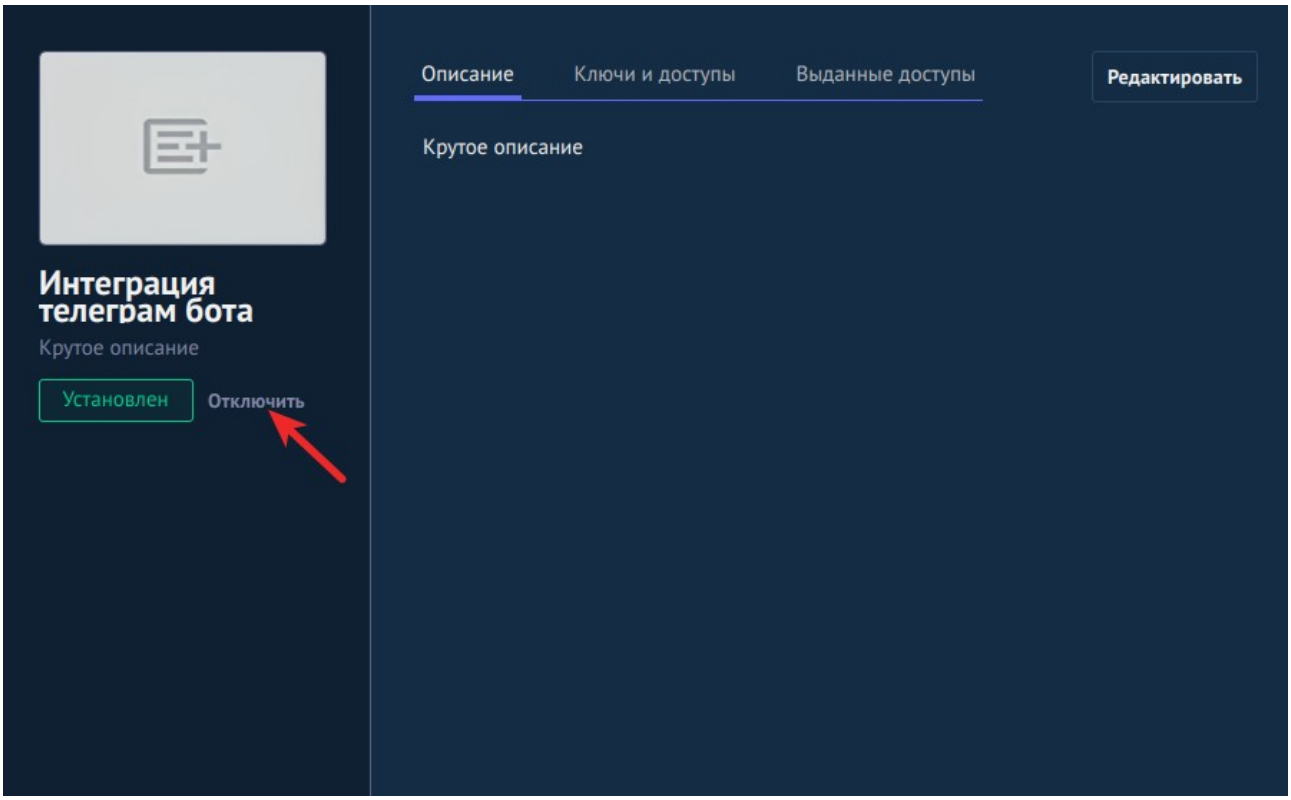


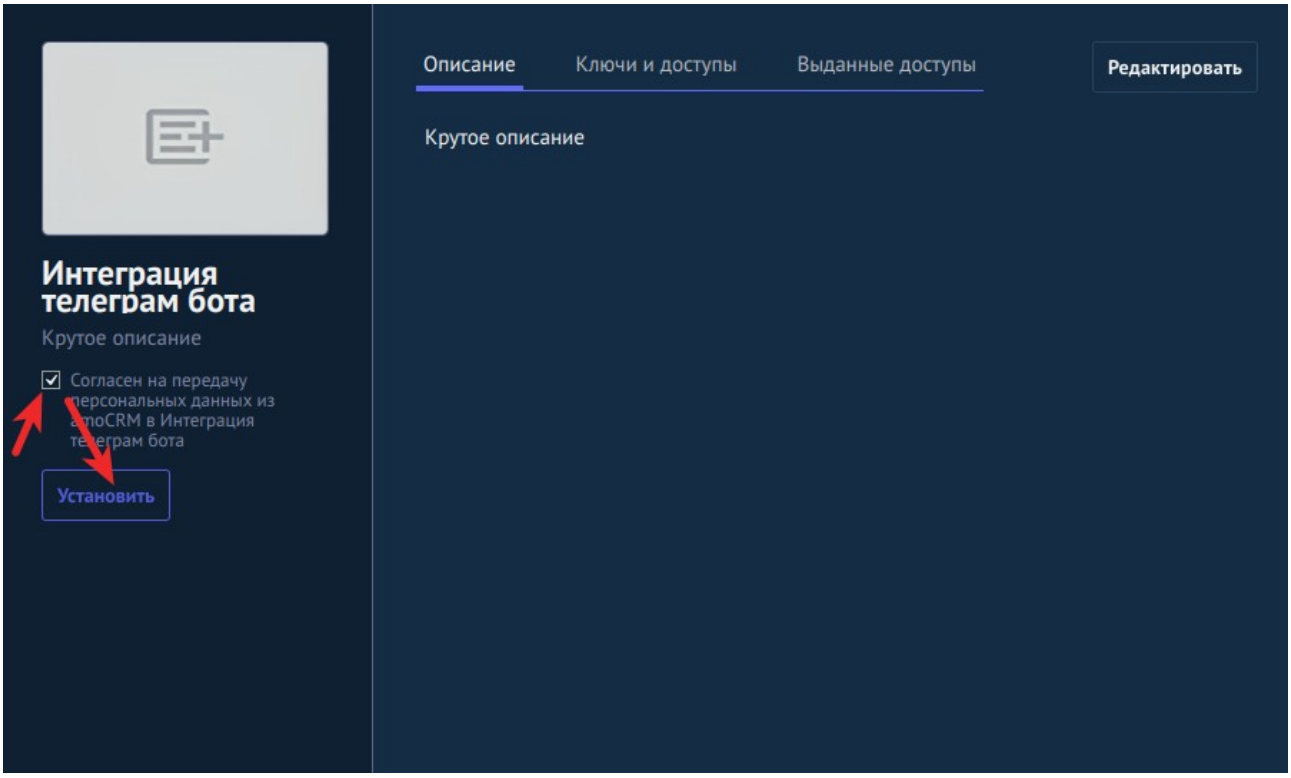
3. Заполнить выделенные поля и нажать “Сохранить”:



4. Включить интеграцию. Для этого переходим в “Установленные”, выбираем созданную интеграцию, нажимаем “Отключить”, а потом включаем:







Запрос на регистрацию канала

Наконец, для непосредственной регистрации канала потребуется связаться с технической поддержкой amoCRM, предоставив информацию по 14 пунктам из документации. Самая простая конфигурация следующая:

```
1. {название сервиса}
2. {ваш URL}
3. {ID вашего аккаунта}
4. Нет
5. Нет
6. {ваш email для связи}
7. {приложить файл с иконкой}
8. {client_uuid внешней интеграции}
9. Не нужен, так как внешняя интеграция
10. Нет
11. Нет
12. Нет
13. Нет
14. Нет
```

Через некоторое время поддержка предоставит данные созданного канала, включая ID интеграции, код канала, и секретный ключ:

```
{
  "id": "{ID канала}",
  "code": "amo.ext.{ID аккаунта}",
  "secret_key": "{секретный ключ}"
}
```

Если потребуется отправлять сообщения от имени бота, запросите параметры бота зарегистрированного канала, которые вам вышлют в виде:

```
"bot": {
  "id": "{ID бота}",
  "name": "Telegram",
  "is_bot": true
}
```

Реализация интеграции

Теперь, имея все необходимые данные, перейдем к реализации интеграции. Код доступен в GitHub.

Интеграция разделяется на две основные части: отправка и обработка сообщений.

Отправка сообщений в amoCRM

Начнем с отправки сообщений. Прежде всего определим необходимые зависимости:

```
amocrm-api==2.6.1  # amocrm api
httpx==0.27.0  # async requests
pydantic-settings==2.1.0  # for env
```

Далее подключим канал к аккаунту amoCRM. Это нужно сделать единожды, сохранив `scope_id` для использования в последующих запросах. Чтобы подключить канал к аккаунту, создадим шлюз для взаимодействия с API, используя паттерн одиночки (singleton):

```python
"""API Chats gateway."""

import hashlib
import hmac
import json
from email.utils import formatdate

import httpx
from amocrm.v2.interaction import BaseInteraction

from env_settings import env


def singleton(cls):
    """Create a singleton instance of a class."""
    instances = {}

    def wrapper(*args, **kwargs):
        if cls not in instances:
            instances[cls] = cls(*args, **kwargs)
        return instances[cls]

    return wrapper

@singleton
class ChatsAPI:
    def __init__(self) -> None:
        self.channel_secret = env.AMOCRM_CHANNEL_SECRET
        self.channel_id = env.AMOCRM_CHANNEL_ID
        self.base_url = "https://amojo.amocrm.ru"

    def __create_body_checksum(self, body: str) -> str:
        return hashlib.md5((body).encode("utf-8")).hexdigest()
```

```
    def __create_signature(
        self, check_sum: str, api_method: str, http_method: str = "POST",
content_type: str = "application/json"
    ) -> str:
        now_in_RFC2822 = formatdate()
        string_to_hash = "\n".join([http_method.upper(), check_sum,
content_type, now_in_RFC2822, api_method])
        return hmac.new(
            bytes(self.channel_secret, "UTF-8"), string_to_hash.encode(),
digestmod=hashlib.sha1
        ).hexdigest()

    def __prepare_headers(
        self, check_sum: str, signature: str, content_type: str =
"application/json"
    ) -> dict[str, str]:
        headers = {
            "Date": formatdate(),
            "Content-Type": content_type,
            "Content-MD5": check_sum.lower(),
            "X-Signature": signature.lower(),
            "User-Agent": "amocrm-py/v2",
        }

        return headers

    async def __request(
        self, payload: dict[str, str], api_method: str
    ) -> httpx.Response | None:
        check_sum = self.__create_body_checksum(json.dumps(payload))
        signature = self.__create_signature(
            check_sum=check_sum,
            api_method=api_method,
        )

        headers = self.__prepare_headers(check_sum=check_sum,
signature=signature)

        async with httpx.AsyncClient() as client:
            response = await client.post(
                self.base_url + api_method,
                headers=headers,
                json=payload,
            )
            response.raise_for_status()
```

```
        return response

    async def _connect_channel_to_account(self) -> str:
        # get account ID in chats
        response = BaseInteraction().request("get", "account?with=amojo_id")
        account_id = response[0]["amojo_id"]

        # connect
        payload = {
            "account_id": account_id,
            "title": "ChatsIntegration",
            "hook_api_version": "v2",
        }
        response: httpx.Response = await self.__request(
            payload=payload,
api_method=f"/v2/origin/custom/{self.channel_id}/connect"
        )
        return response.json()["scope_id"]
```

Здесь стоит отметить, что мы используем библиотеку `amocrm.v2` для взаимодействия с основным API amoCRM, и чтобы она правильно работала необходимо авторизироваться в соответствии с документацией.

Таким образом, нужно вызвать метод `_connect_channel_to_account` и записать полученный `scope_id` в переменные среды.

Создание нового чата и отправка сообщений

После подключения канала к аккаунту можно создать новый чат, связать его с контактом и отправлять сообщения от имени пользователя или бота. Для этого расширим наш класс соответствующими методами:

```
    async def create_new_chat(self, chat_id: str, contact_id: int) -> str:
        contact = Contact.objects.get(object_id=contact_id)
        payload = {
            "conversation_id": chat_id,
            "user": {
                "id": str(contact_id),
                "name": str(contact.name),
            },
        }
        response: httpx.Response = await self.__request(
            payload=payload,
```

```
api_method=f"/v2/origin/custom/{self.scope_id}/chats"
        )
        return response.json()["id"]

    def connect_chat_to_contact(self, amocrm_chat_id: str, contact_id: int)
-> None:
        payload = [
            {
                "contact_id": contact_id,
                "chat_id": amocrm_chat_id,
            }
        ]

        response = BaseInteraction().request(
            "post", "contacts/chats", data=payload, headers={"Content-Type":
"application/json"}
        )
        if response[1] != 200:
            raise Exception("The chat could not be linked to the contact!")

    async def send_message_to_chat_as_user(self, text: str, chat_id: str,
contact_id: int) -> None:
        contact = Contact.objects.get(object_id=contact_id)
        payload = {
            "event_type": "new_message",
            "payload": {
                "timestamp": int(time.time()),
                "msec_timestamp": int(time.time() * 1000),
                "msgid": str(uuid4()),
                "conversation_id": chat_id,
                "sender": {
                    "id": str(contact_id),
                    "name": str(contact.name),
                },
                "message": {
                    "type": "text",
                    "text": text,
                },
                "silent": True,
            },
        }
        await self.__request(payload=payload,
api_method=f"/v2/origin/custom/{self.scope_id}")

    async def send_message_to_chat_as_bot(self, text: str, chat_id: str,
contact_id: int) -> None:
```

```
        contact = Contact.objects.get(object_id=contact_id)
        payload = {
            "event_type": "new_message",
            "payload": {
                "timestamp": int(time.time()),
                "msec_timestamp": int(time.time() * 1000),
                "msgid": str(uuid4()),
                "conversation_id": chat_id,
                "sender": {
                    "id": self.bot_client_id,
                    "name": self.bot_name,
                    "ref_id": self.bot_id,
                },
                "receiver": {
                    "id": str(contact_id),
                    "name": str(contact.name),
                },
                "message": {
                    "type": "text",
                    "text": text,
                },
                "silent": True,
            },
        }
        await self.__request(payload=payload,
api_method=f"/v2/origin/custom/{self.scope_id}")
```

После того, как мы определили все необходимые методы, можно создавать чат и отправлять сообщения:

```
async def main(chat_id: str, contact_id: int) -> None:
    amocrm_chat_id = await ChatsAPI().create_new_chat(chat_id=chat_id,
contact_id=contact_id)
    ChatsAPI().connect_chat_to_contact(amocrm_chat_id=amocrm_chat_id,
contact_id=contact_id)

    await ChatsAPI().send_message_to_chat_as_bot(chat_id=chat_id, text="any
message", contact_id=contact_id)
    await ChatsAPI().send_message_to_chat_as_user(chat_id=chat_id, text="any
message", contact_id=contact_id)


if __name__ == "__main__":
    asyncio.run(main(chat_id="1", contact_id=1))
```

Обработка сообщений из amoCRM

Когда менеджер отправляет сообщение из amoCRM, события поступают на Webhook URL, указанный при регистрации канала. Для начала определим необходимые зависимости:

```
fastapi==0.110.3 # API
pydantic-settings==2.1.0  # for env
uvicorn==0.29.0  # ASGI web server
```

Затем реализуем обработчик, который будет обрабатывать входящие события:

```python
"""amoCRM chats router."""

from fastapi import APIRouter, Request, Response, status

amocrm_router = APIRouter(
    tags=["amoCRM"],
)


@amocrm_router.post(
    "/location/{scope_id}",
    description="Processing message from amoCRM chats.",
)
async def amocrm_handler(scope_id: str, request: Request):
    json_body = await request.json()

    chat_id = json_body["message"]["conversation"]["client_id"]
    message = json_body["message"]["message"]["text"]

    # any message processing

    return Response(status_code=status.HTTP_200_OK)
```

Саму инициализацию FastAPI приложения можете посмотреть в GitHub. Также стоит отметить, что нужно верифицировать запрос с помощью проверки подписи, а также учитывать, что endpoint публичный. Но в этой статье мы не будем затрагивать тему безопасности.

На этом интеграция amoCRM API чатов завершена. Теперь у вас есть готовый шаблон, который можно использовать для общения с клиентами через amoCRM и ваши привычные каналы связи.