Introduction

amoCRM is one of the most popular tools for sales automation. It offers a wide range of features to streamline customer management, including the chat module. Chats enable amoCRM users to exchange messages with customers, while customers can use their preferred messengers for communication.

Despite its extensive functionality and ease of use, `amoCRM` only provides integration examples in `PHP`, posing additional challenges for developers working with other programming languages.

In our project, we faced the challenge of integrating the `amoCRM` chat API with a `Telegram` bot written in `Python`. This integration allowed users to escalate conversations to a human operator: the bot would initiate a chat, forward messages from `Telegram` to `amoCRM`, and relay responses from amoCRM back to `Telegram`. To achieve this, we adapted the documentation from `PHP` to `Python`.

In this article, we'll discuss how to integrate the `amoCRM` chat API using `Python` so that developers can use this functionality in their projects.

Practical Part

Preparation

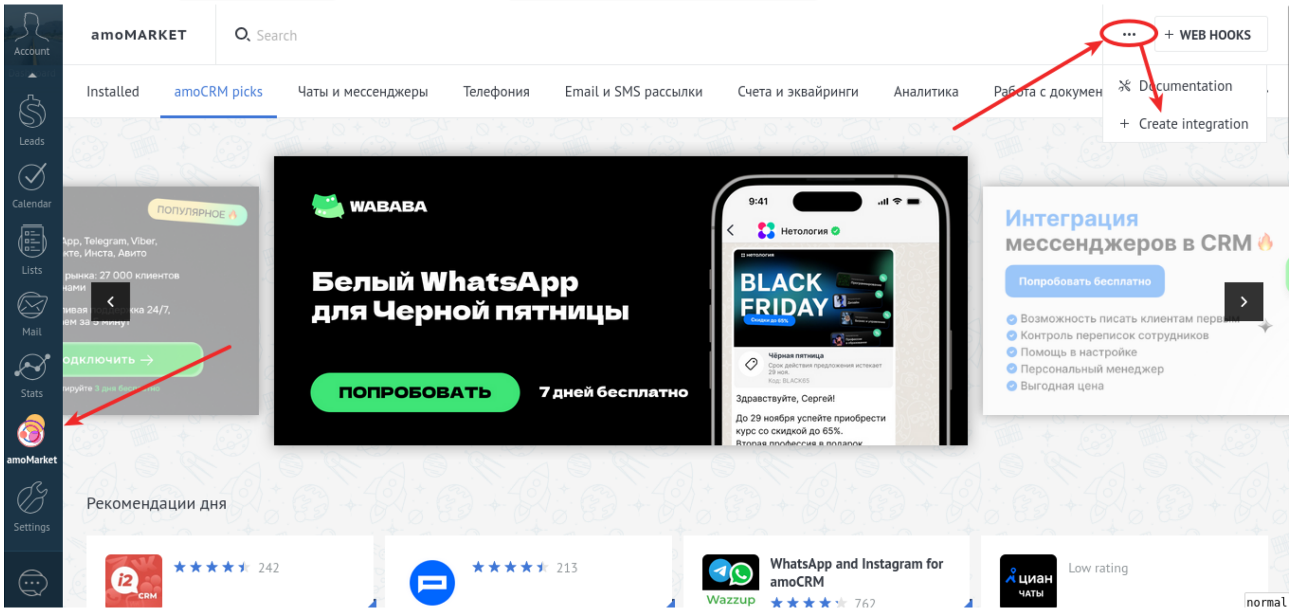
Before starting the integration, it's essential to familiarize yourself with the amoCRM chat API documentation. Instead of summarizing the documentation, here are some important points to note:

1. The server hosting the public endpoint (`Webhook URL`) must have an `SSL` certificate. However, a domain is not mandatory.
2. You need to create an external integration and provide the integration's `client_uuid` when applying for channel registration (Step 8). We'll discuss how to create such an integration below.

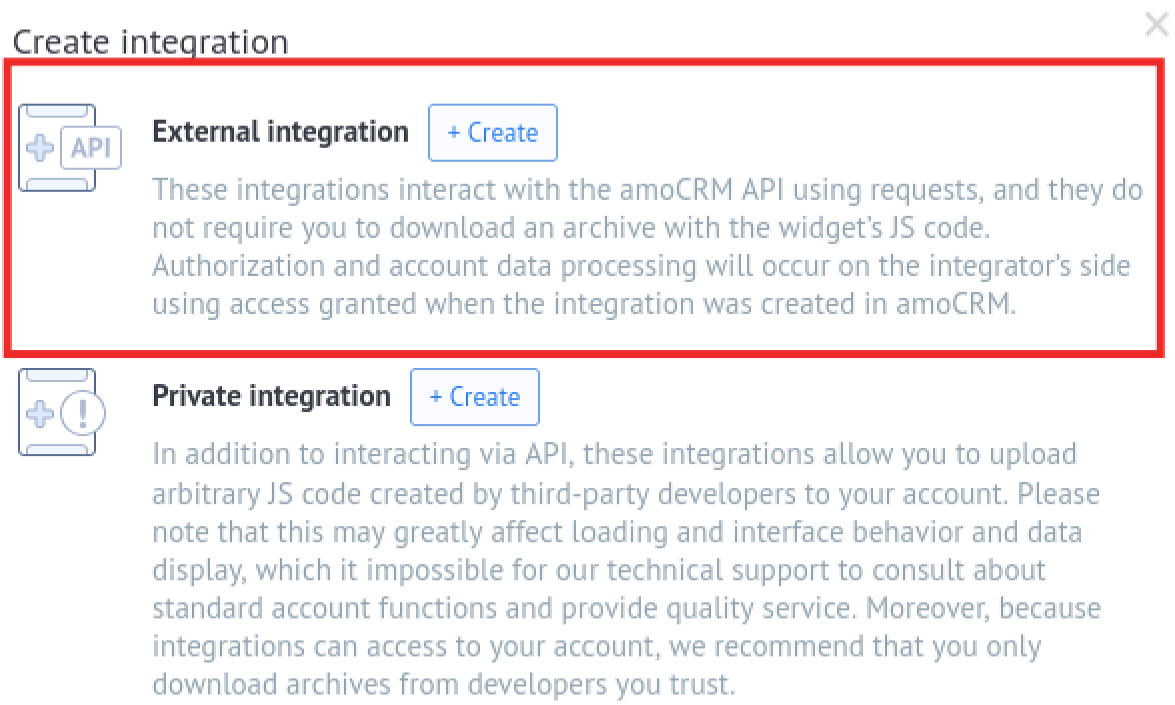
Creating an External Integration

To create an external integration:

1. Navigate to `amoMarket` and select `Create integration` in the menu:



2. Choose the external integration option:



Create integration

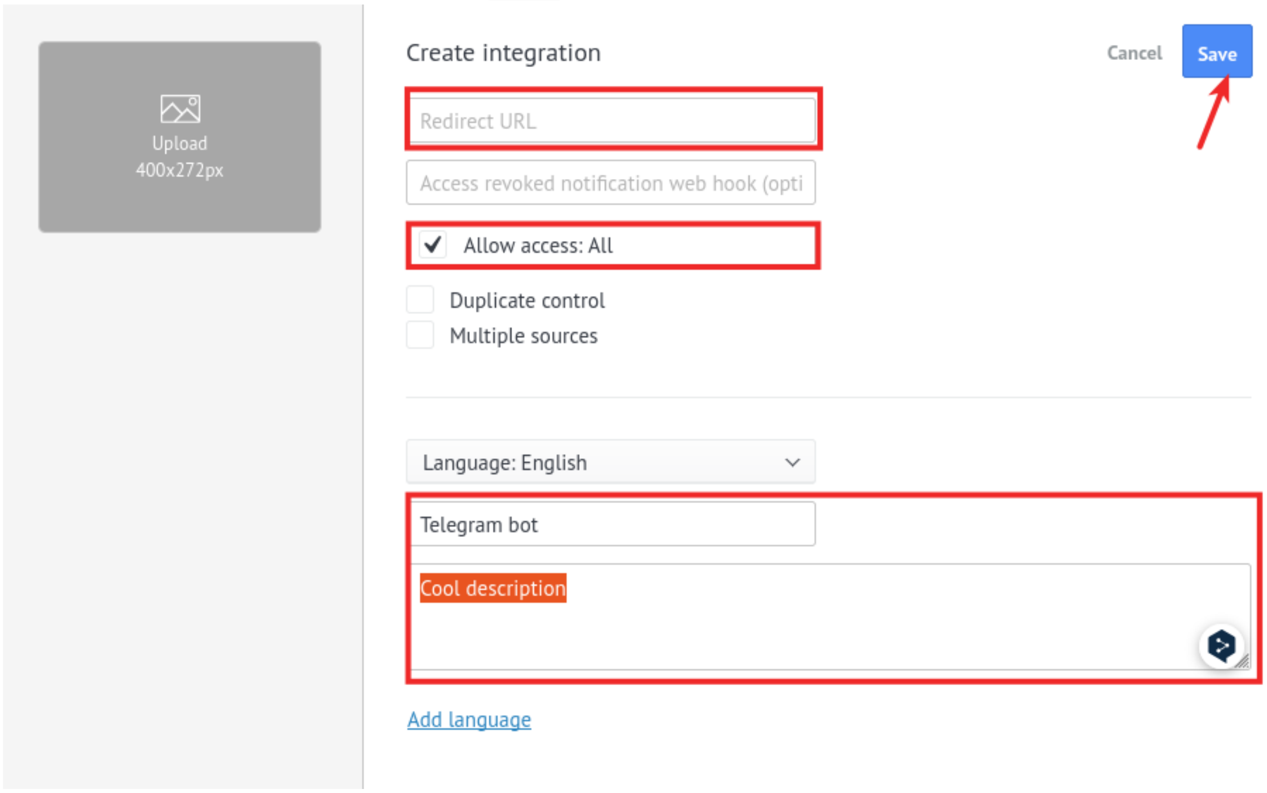
External integration + Create

These integrations interact with the amoCRM API using requests, and they do not require you to download an archive with the widget's JS code. Authorization and account data processing will occur on the integrator's side using access granted when the integration was created in amoCRM.

Private integration + Create

In addition to interacting via API, these integrations allow you to upload arbitrary JS code created by third-party developers to your account. Please note that this may greatly affect loading and interface behavior and data display, which it impossible for our technical support to consult about standard account functions and provide quality service. Moreover, because integrations can access to your account, we recommend that you only download archives from developers you trust.

3. Fill in the required fields and click `Save`:



Redirect URL

Access revoked notification web hook (opti

Allow access: All

Duplicate control

Multiple sources

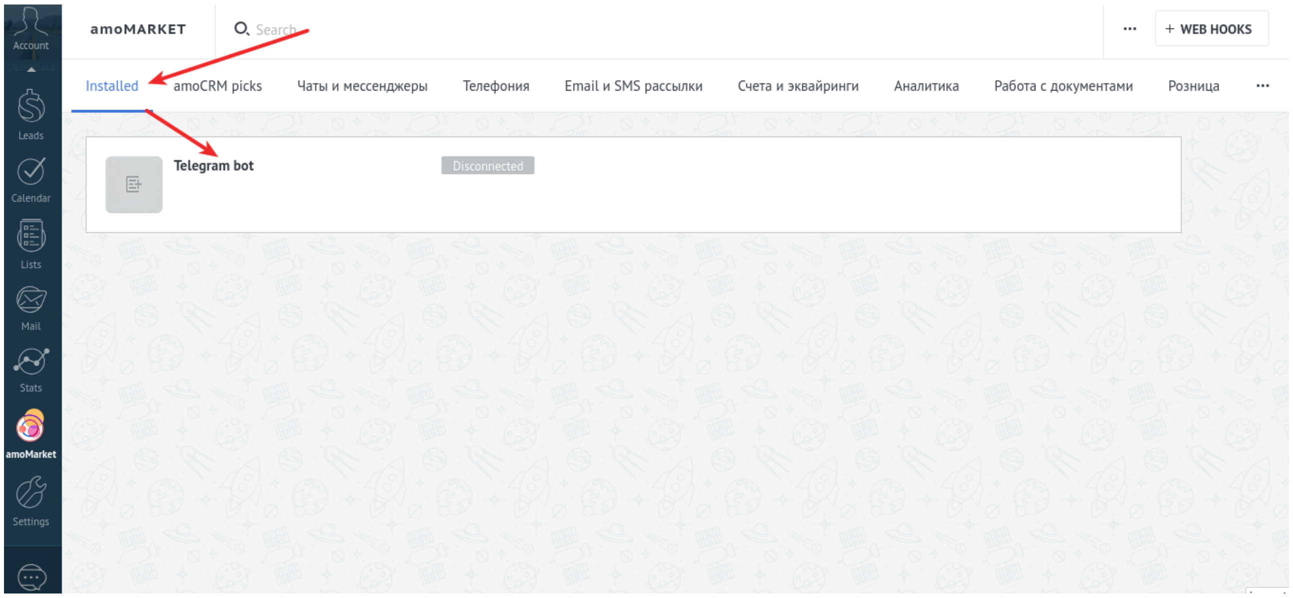
Language: English

Telegram bot

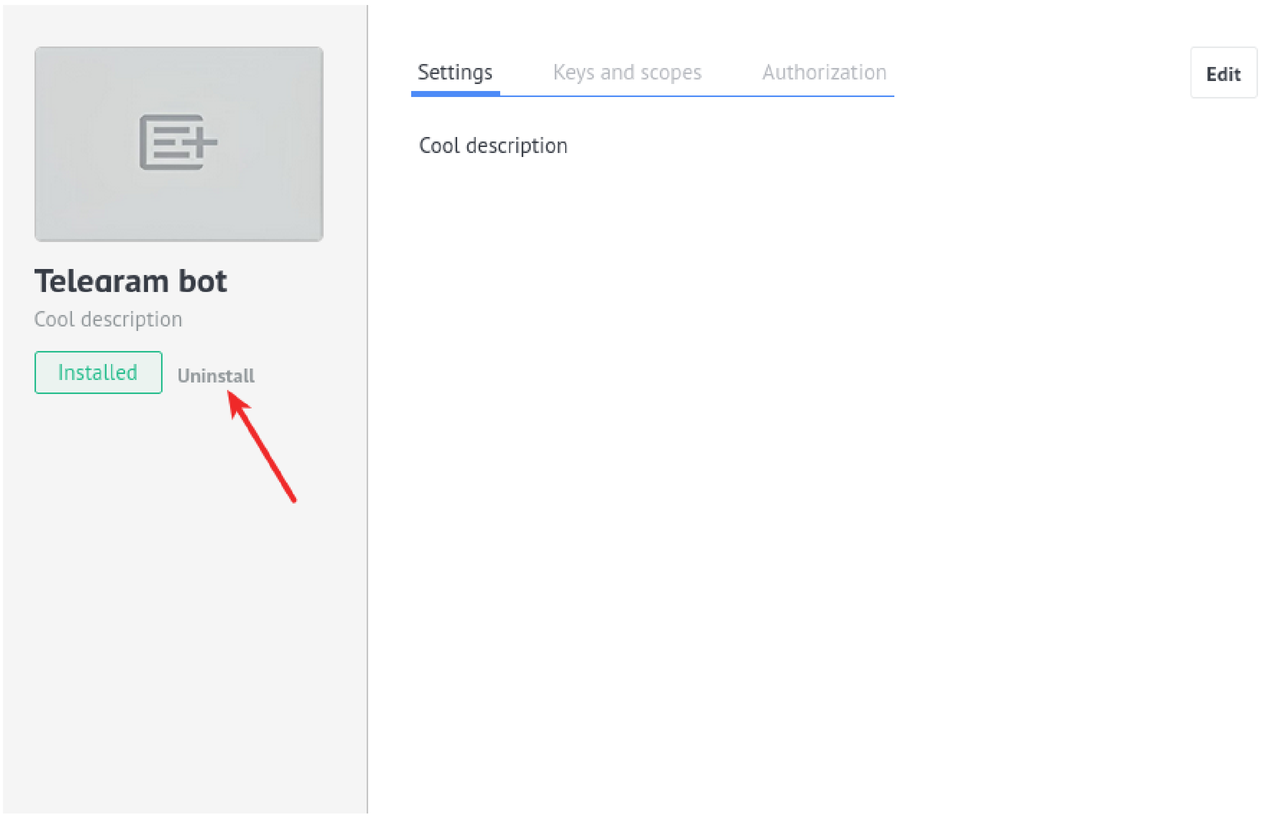
Cool description

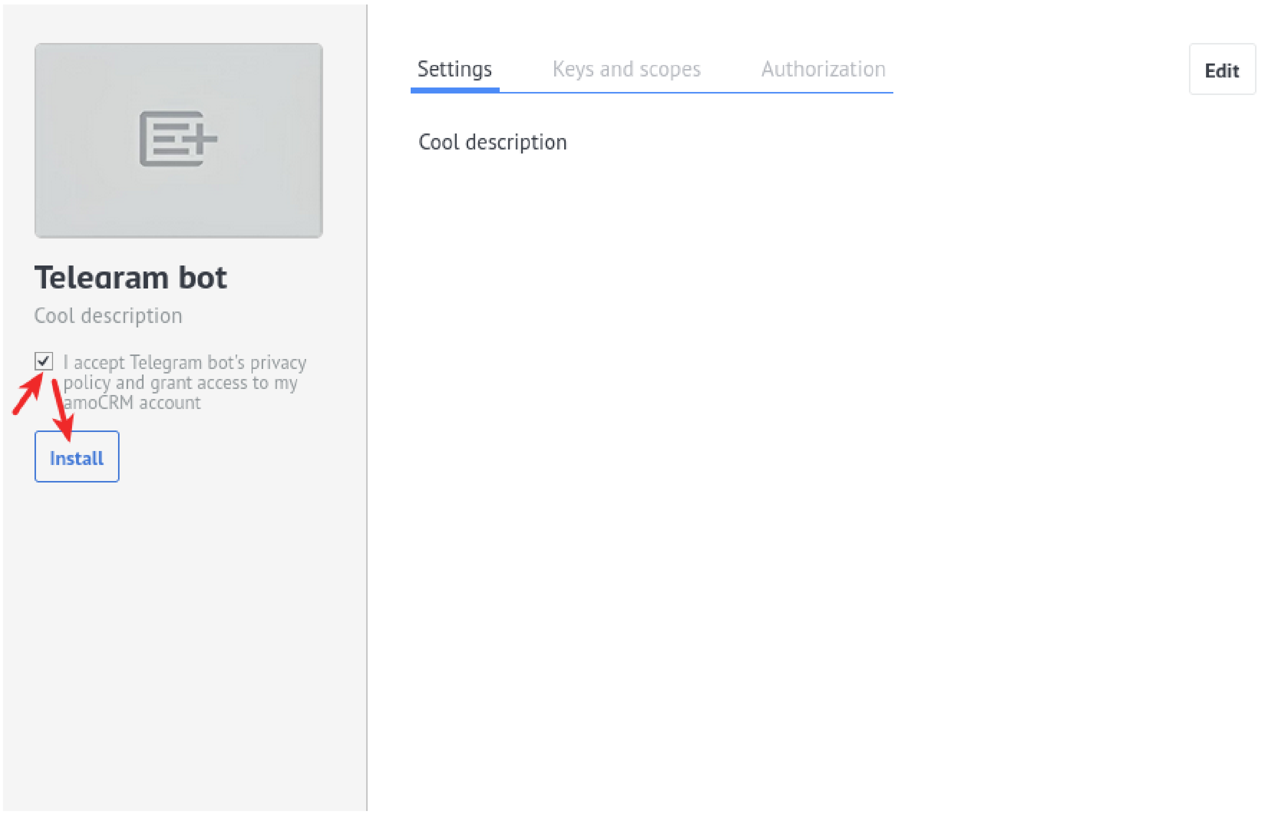
Add language

4. Enable the integration by navigating to `Installed`, selecting the created integration, disabling it, and then re-enabling it:



Telegram bot Disconnected





Channel Registration Request

To register the channel, you'll need to contact `amoCRM` support and provide information about the 14 points listed in the documentation. A simple configuration looks like this:

```
1. {service name}
2. {your URL}
3. {your account ID}
4. No
5. No
6. {your contact email}
7. {attach an icon file}
8. {client_uuid of the external integration}
9. Not needed since it's an external integration
10. No
11. No
12. No
13. No
14. No
```

Support will respond with the channel details, including the integration ID, channel code, and secret key:

```
{
  "id": "{channel ID}",
  "code": "amo.ext.{account ID}",
  "secret_key": "{secret key}"
}
```

If you need to send messages on behalf of a bot, request the bot's parameters for the registered channel. Support will provide them as follows:

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

Integration Implementation

Once you have the necessary details, you can proceed with the integration. The code is available on GitHub.

The integration is divided into two main parts: sending and processing messages.

Sending Messages to amoCRM

We'll start with sending messages. First, define the required dependencies:

```
amocrm-api==2.6.1  # amocrm API
httpx==0.27.0  # async requests
pydantic-settings==2.1.0  # for environment variables
```

Next, connect the channel to the `amoCRM` account. This is a one-time operation, and you'll need to save the `scope_id` for future requests. We'll create a gateway for interacting with the API using the singleton pattern:

```
"""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"]
```

Here it should be noted that `amocrm.v2` library is used to interact with the main API `amoCRM`, and for proper operation it is necessary to authorize according to documentation.

So, call `_connect_channel_to_account` method and write `scope_id` to environment variables.

Creating a New Chat and Sending Messages

After connecting the channel to the account, you can create a new chat, link it to a contact, and send messages as a user or bot. Here's how to extend the class with the necessary methods:

```
    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}")
```

After defining the required methods, call them as follows:

```
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))
```

Processing Messages from amoCRM

When a manager sends a message from `amoCRM`, events are sent to the `Webhook URL` specified during channel registration. Define the required dependencies:

```
fastapi==0.110.3  # API
pydantic-settings==2.1.0  # for environment variables
```

```
uvicorn==0.29.0  # ASGI web server
```

Then, implement a handler to process incoming events:

```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)
```

The `FastAPI` application initialization itself is provided in GitHub. It's also worth noting that you need to verify the request with signature verification and take into account that the endpoint is public, but we won't touch on security in this article.

Now the `amoCRM` chat API integration is complete. You have a ready-to-use template to enable communication with customers through `amoCRM`.