

**Документация, содержащая информацию,
необходимую для установки программного обеспечения
Smart Platform**

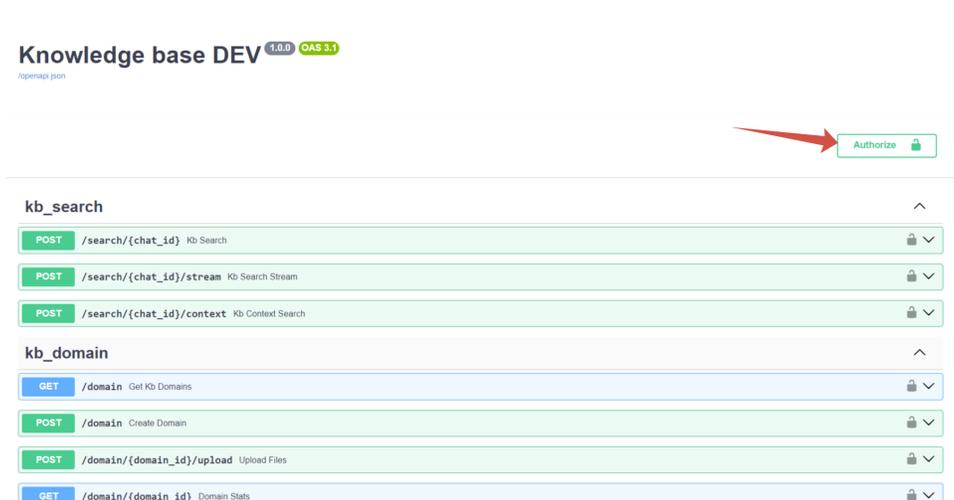
**ООО «Ресерч дата лаб»,
Москва, 2024**

Авторизация и получение токена

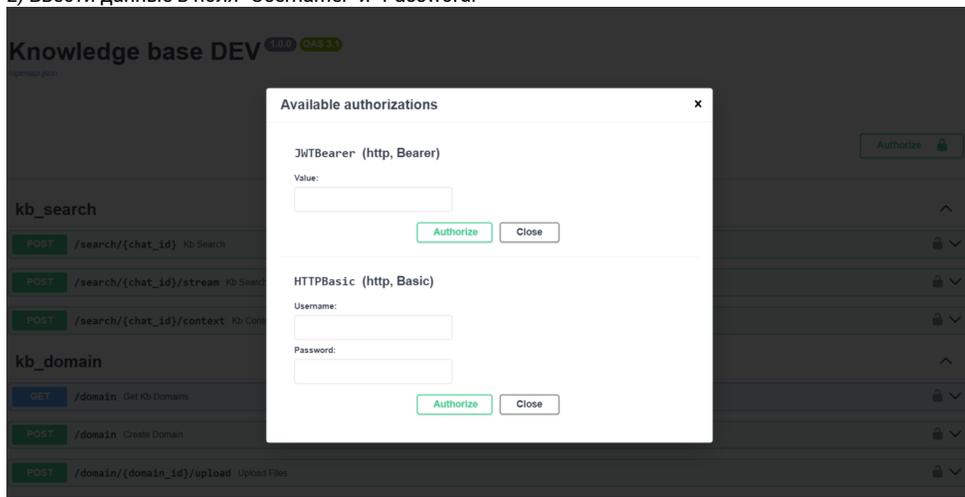
Для авторизации в API вам нужно использовать логин и пароль, которые были получены вами перед началом работы с API.

Эти логин и пароль вам нужно указать также, как это указано на скриншотах:

1) Нажимаем кнопку "Authorize"



2) Ввести данные в поля "Username:" и "Password:"



6) Нажимаем на кнопку "Execute"

The screenshot shows a REST client interface for a POST request to `/sign_in`. The "Parameters" section is empty. A large blue "Execute" button is visible. Below it, the "Responses" section shows a "200 Successful Response" with a media type of "application/json". An "Example Value" field displays a JSON object containing an "access_token" and a "user_id".

```
{
  "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIwIiwiaWF0IjoxNTE2MzIyMjQyLCJmcmVzIjoiIn0",
  "user_id": "string"
}
```

7) Пролыстываем метод чуть ниже и в окне "Response body", находим поле "access_token" и в нём находится наш токен для авторизации, копируем его

The screenshot shows the "Response body" section of the REST client. The response is a JSON object with the following structure:

```
{
  "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIwIiwiaWF0IjoxNTE2MzIyMjQyLCJmcmVzIjoiIn0",
  "user_id": "6545d3a11a92bed176e98",
  "token_type": "bearer"
}
```

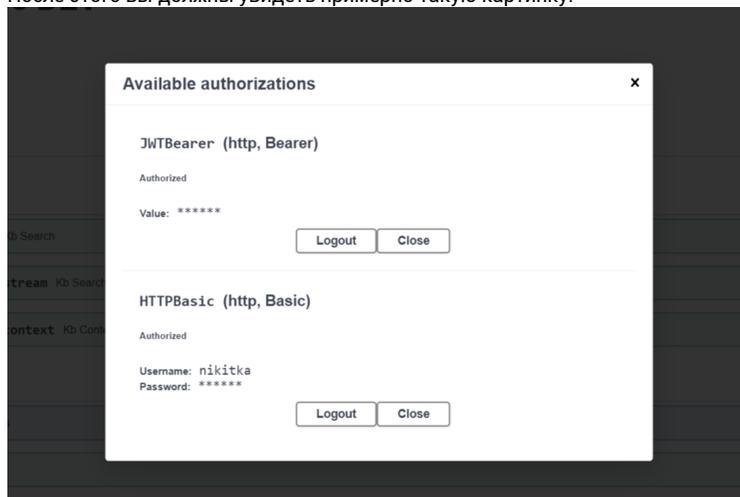
The "access_token" field is highlighted, and the "Copy" button is visible next to it.

8) Возвращаемся в начало и вновь нажимаем кнопку "Authorize", вводим скопированный токен авторизации в поле "Value" и нажимаем кнопку "Authorize" в этом окне

The screenshot shows an "Available authorizations" dialog box with two sections:

- JWTBearer (http, Bearer)**: A "Value:" field with a text input box, and "Authorize" and "Close" buttons.
- HTTPBasic (http, Basic)**: A section labeled "Authorized" with "Username: nikitka" and "Password: *****", and "Logout" and "Close" buttons.

После этого вы должны увидеть примерно такую картинку:



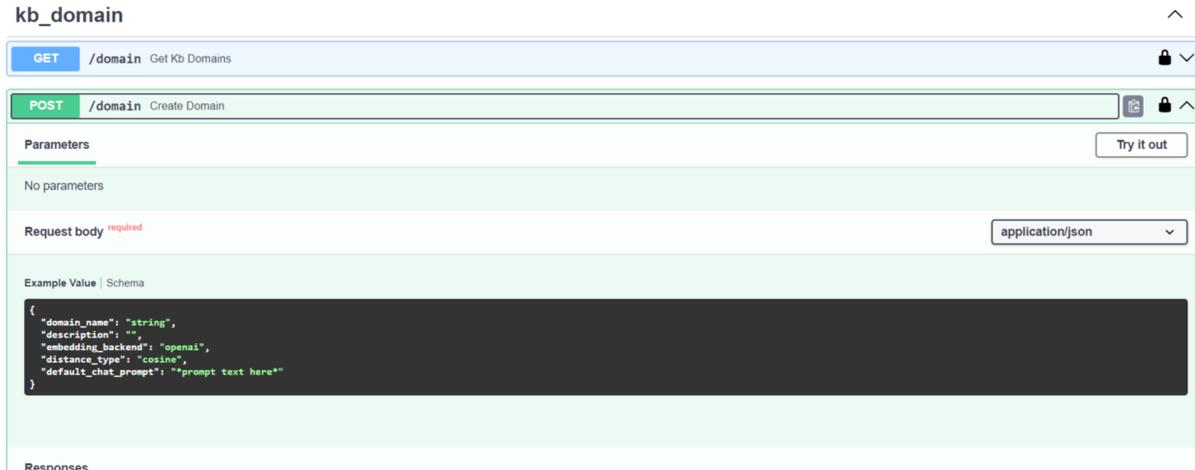
Это значит, что вы внесли все нужные для авторизации данные и теперь можете использовать API

Как создать домен и добавить на него данные

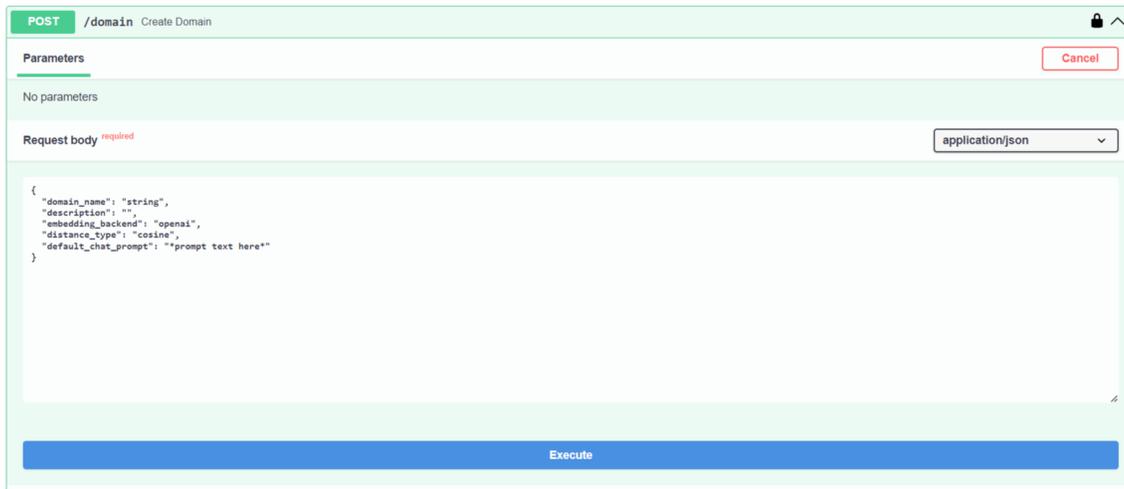
Для создания домена, вам нужно [авторизоваться](#), используя аккаунт, на котором вы хотите создать домен.

После авторизации следует действовать по этим шагам

1) Первое что нужно сделать, это найти на странице метод POST /domain, открыть его и нажать "Try it out"



2) Следующий шаг, заполнить данные в текстовом поле



Подробнее про эти поля:

```
{
  "domain_name": "string", - Название вашего нового домена
  "description": "", - Описание для нового домена (необязательно)
  "embedding_backend": "ndt", - Выбор системы эмбедингов, рекомендуется использовать значение ndt
  "distance_type": "cosine", - Выбор системы векторного поиска, рекомендуется использовать значение cosine
  "default_chat_prompt": "*prompt text here*" - Базовый промт для домена (необязательно)
}
```

3) После заполнения данных, нужно нажать на кнопку "Execute"

No parameters

Request body **required** application/json

```
{
  "domain_name": "New domain",
  "description": "From API",
  "embedding_backend": "ndt",
  "distance_type": "cosine",
  "default_chat_prompt": "*prompt text here*"
}
```

Execute

4) В случае, если всё создано верно, появится "domain_id", который потребуется далее для загрузки данных

Request URL
https://chat-api-dev.neuraldeep.tech/domain

Server response

Code Details

201

Response body

```
{
  "domain_id": "..."
}
```

Response headers

Responses

5) После этого, сохранив себе "domain_id" можно скрыть данный метод и открыть метод POST /domain{domain_id}/upload необходимый для загрузки данных в домен, нажимаем "Try it out"

POST /domain/{domain_id}/upload Upload Files Try it out

Parameters

Name	Description
domain_id required (path)	domain_id

Request body **required** multipart/form-data

files **required**
array

meta_information List of additional json info for every doc.Format: [{"meta1": "meta2":2}, ... {}, {}]

6) Вводим "domain_id" из прошлого метода в поле "domain_id"
kb_domain

GET /domain Get Kb Domains

POST /domain Create Domain

POST /domain/{domain_id}/upload Upload Files

Parameters

Cancel Reset

Name Description

domain_id * required (path)

Request body required multipart/form-data

files * required array Add string item

meta_information List of additional json info for every doc.Format: [{"meta1": "meta2:2"}, ... {}, {}]

meta_information

Send empty value

7) Используя кнопку Add string item в строке files добавляем поле для вноса документов, на каждый документ, который нужно добавить в домен, нужно добавить своё поле

POST /domain/{domain_id}/upload Upload Files

Parameters

Cancel Reset

Name Description

domain_id * required (path)

Request body required multipart/form-data

files * required array

meta_information List of additional json info for every doc.Format: [{"meta1": "meta2:2"}, ... {}, {}]

meta_information

Send empty value

Execute

Responses

8) После добавления всех документов, нажимаем "Execute"

POST /domain/{domain_id}/upload Upload Files

Parameters

Cancel Reset

Name Description

domain_id * required (path) 66a3857c61ef6d5ea4687576

Request body required multipart/form-data

files * required array

meta_information List of additional json info for every doc.Format: [{"meta1": "meta2:2"}, ... {}, {}]

meta_information

Send empty value

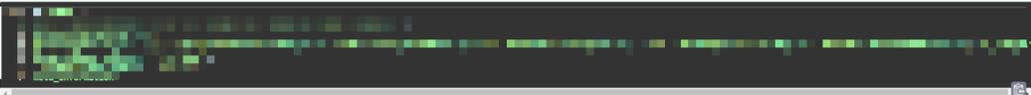
Execute

Responses

9) Ждем результата обработки и вот такой ответ

Responses

Curl



```
curl -X POST https://chat-api-dev.neuraldeep.tech/domain/66a3857c61ef6d5ea4687576/upload
```

Request URL

```
https://chat-api-dev.neuraldeep.tech/domain/66a3857c61ef6d5ea4687576/upload
```

Server response

Code	Details
201	<p>Response body</p> <pre>{ "message": "Files uploaded successfully" }</pre> <p>Response headers</p> <pre>HTTP/1.1 201 Created Content-Type: application/json Date: Wed, 14 Jun 2023 12:15:10 GMT Server: Apache/2.4.18 (Ubuntu)</pre>

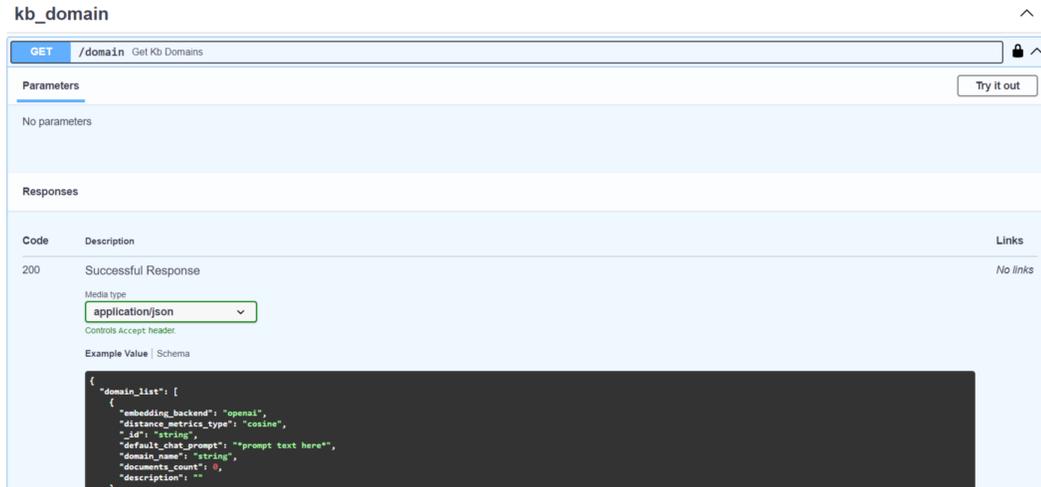
Всё, домен успешно создан и данные в него загружены.

Как узнать domain_id/Как создать чат

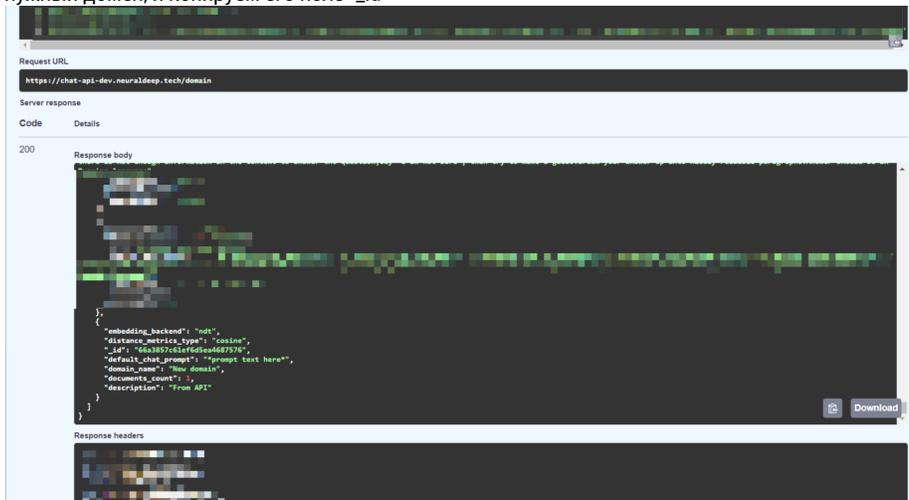
Для создания чата, вам нужно [авторизоваться](#), используя аккаунт, на котором вы хотите создать чат, а также получить domain_id необходимого домена, к которому будет добавлен чат. Если domain_id уже вам известен, переходите к 3 шагу.

После авторизации следует действовать по этим шагам:

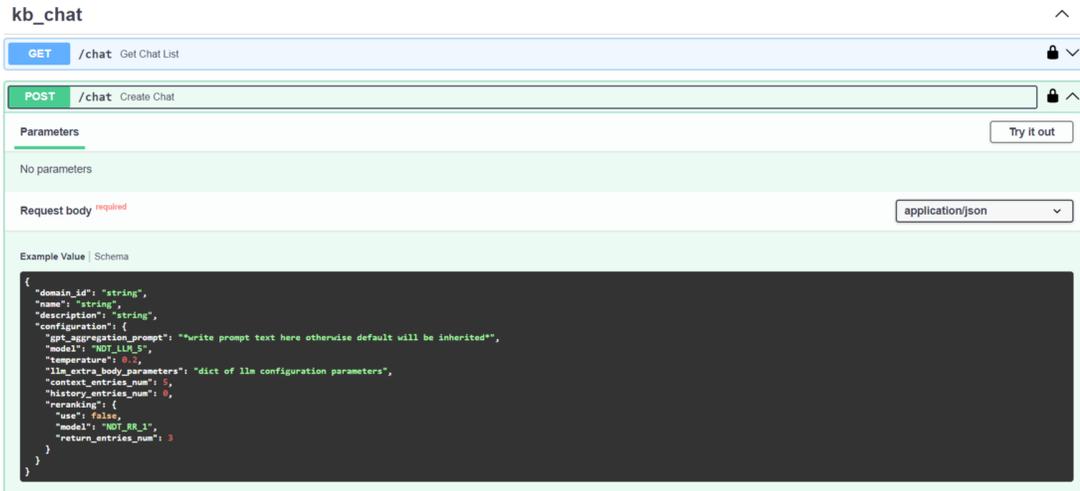
- 1) Для получения domain_id лучше всего использовать метод GET /domain kb_domain



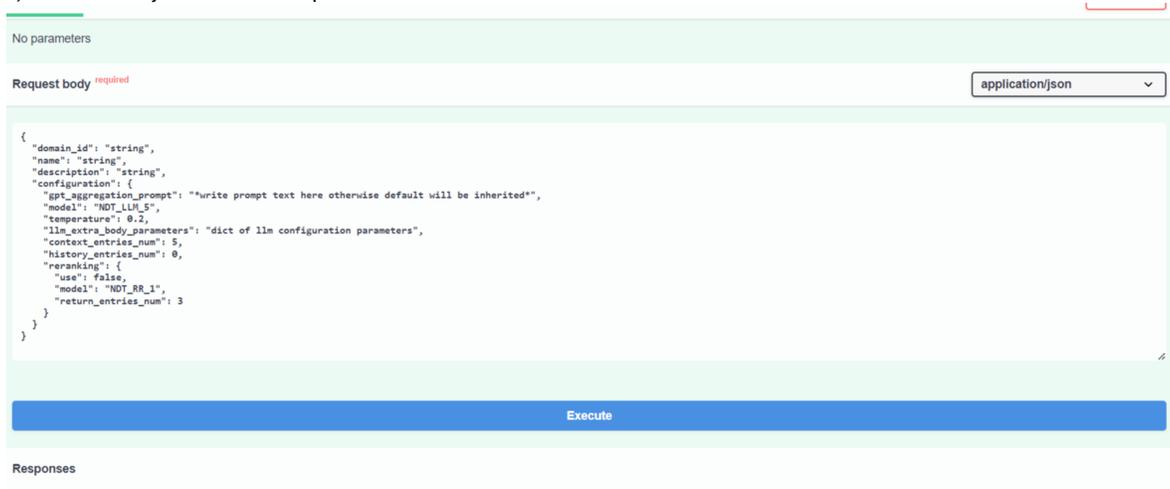
- 2) После нажатия "Try it out" нажимаем "Execute" и пролистываем вниз до окна ответа от API, с списком доступных доменов, где находим нужный домен, и копируем его поле "_id"



3) Используя полученный domain_id находим метод POST /chat и нажимаем кнопку "Try it out"



4) Заполняем пункты в поле запроса



Подробнее о пунктах при создании домена:

```
{
"domain_id": "string", - Тут нужно указать domain_id полученный ранее, для того, чтобы чат был привязан именно к этому домену
"name": "string", - Указываем название чата
"description": "string", - Указываем описание чата (необязательно)
"configuration": {
"gpt_aggregation_prompt": "*write prompt text here otherwise default will be inherited*", - Указываем "aggregation_prompt", инструкцию, которой
будет следовать модель в данном чате
"model": "NDT_LLM_5", - Указываем модель, которая будет использоваться в этом чате по базе (изменять необязательно)
"temperature": 0.2, - Указываем базовую температуру модели (потом можно без труда менять, поэтому можно не изменять)
"llm_extra_body_parameters": "dict of llm configuration parameters", - Поле для дополнительных настроек модели (необязательно, если не
используется, нужно удалить строчку)
"context_entries_num": 5, - Поле с значением контекста, устанавливает, сколько чанков из домена будет использоваться для каждого ответа (0
- общение с моделью без информации из домена, 3 - среднее значение точности, 5 - базовое значение, которое даёт хорошую точность
ответов)
"history_entries_num": 0, - Поле с значением истории, устанавливает, сколько прошлых сообщений будет помнить модель (лучше устанавливать
чётное значение, так как модель будет помнить и запрос к ней и свой ответ, например при установке значения на 6 модель будет помнить 3
прошлых запроса пользователя и 3 прошлых ответа)
"reranking": {
"use": false, - Использовать ли реранкер
"model": "NDT_RR_1", - Модель реранкера
"return_entries_num": 3 - Количество данных, которые возвращает реранкер
}
}
}
```

5) После указания всех данных в необходимых полях нажимаем кнопку "Execute" (на данном примере кстати видно, что "llm_extra_body_parameters": "dict of llm configuration parameters" я убрал из за ненадобности

No parameters

Request body **required** application/json

```
{
  "domain_id": "66a3857c61ef6d5ea4687576",
  "name": "Новый чат",
  "description": "Описание",
  "configuration": {
    "gpt_aggregation_prompt": "Ты умный помощник, помоги",
    "model": "NDT_LLH_5",
    "temperature": 0.2,
    "context_entries_num": 5,
    "history_entries_num": 0,
    "reranking": {
      "use": false,
      "model": "NDT_RR_1",
      "return_entries_num": 3
    }
  }
}
```

Execute

Responses

б) После всех вышесказанных действий, получаем chat_id, который является идентификатором данного чата и по которому потом можно будет обращаться к этому чату

```
"description": "Описание",
"configuration": {
  "gpt_aggregation_prompt": "Ты умный помощник, помоги",
  "model": "NDT_LLH_5",
  "temperature": 0.2,
  "context_entries_num": 5,
  "history_entries_num": 0,
  "reranking": {
    "use": false,
    "model": "NDT_RR_1",
    "return_entries_num": 3
  }
}
```

Request URL

https://chat-api-dev.neuraldeep.tech/chat

Server response

Code	Details
201	<p>Response body</p> <pre>{ "chat_id": "66a3958361ef6d5ea4687577" }</pre> <p>Response headers</p>

Таким образом мы создаём чат через API.

Как узнать chat_id/Как задать вопрос к чату через API

Для того чтобы задать вопрос через API, вам нужно [авторизоваться](#), используя аккаунт, на котором находится домен и чат, через которые вы хотите задать вопрос, а также получить chat_id необходимого чата, к которому будет задан вопрос. Если chat_id уже вам известен, переходите к 3 шагу.

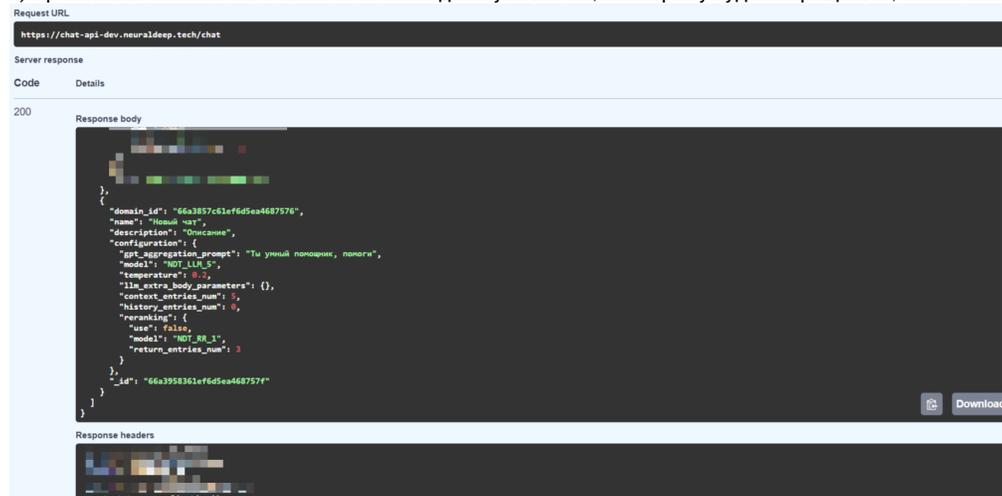
Для обращения к модели вам нужно действовать по этим шагам:

1) Для получения chat_id лучше всего использовать метод GET /chat, сначала открываем его, потом нажимаем "Try it out" и "Execute"



The screenshot shows the Swagger UI for the `GET /chat` endpoint. The interface includes a search bar, a 'Try it out' button, and a 'Responses' section. The response details show a 200 status code for a 'Successful Response' with a media type of 'application/json'.

2) Пролыстываем ниже и в ответе от API находим нужный чат, к которому будем обращаться, после чего копируем из него поле "_id"



The screenshot shows the Swagger UI displaying the response body for the `GET /chat` endpoint. The response is a JSON object containing chat details, including a `_id` field with the value `66a3958361ef6d5ea468757f`.

3) Находим метод POST /search/{chat_id} нажимаем "Try it out" и указываем "chat_id"

kb_search

POST /search/{chat_id} Kb Search

Parameters Try it out

Name	Description
chat_id * required (path)	chat_id

Request body required application/json

Example Value | Schema

```
{
  "text": "string",
  "return_debug_data": false,
  "meta_override": {
    "gpt_aggregation_prompt": "*write prompt text here otherwise default will be inherited*",
    "model": "NDT_LLM_1",
    "temperature": 0.2,
    "llm_extra_body_parameters": "dict of llm configuration parameters",
    "context_entries_num": 5,
    "history_entries_num": 0,
    "reranking": {
      "use": false,
      "model": "NDT_RR_1",
      "return_entries_num": 3
    }
  }
}
```

4) Указав "chat_id" переходим к полю запроса

Parameters Cancel

Name	Description
chat_id * required (path)	66a3958361ef6d5ea468757f

Request body required application/json

```
{
  "text": "string",
  "return_debug_data": false,
  "meta_override": {
    "gpt_aggregation_prompt": "*write prompt text here otherwise default will be inherited*",
    "model": "NDT_LLM_1",
    "temperature": 0.2,
    "llm_extra_body_parameters": "dict of llm configuration parameters",
    "context_entries_num": 5,
    "history_entries_num": 0,
    "reranking": {
      "use": false,
      "model": "NDT_RR_1",
      "return_entries_num": 3
    }
  }
}
```

Подробнее про эти атрибуты запроса:

```
{
  "text": "string", - Текст вашего запроса
  "return_debug_data": false, - Возвращать ли debug_data, в которых хранится дебаг информация (контекст из базы данных, из истории) false - не возвращать true - возвращать
  "meta_override": {
    "gpt_aggregation_prompt": "*write prompt text here otherwise default will be inherited*", - "aggregation_prompt" (Промт, инструкция для модели)
    "model": "NDT_LLM_1", - Модель LLM, к которой будут отправляться запросы
    "temperature": 0.2, - Температура, с которой будут отправляться запросы
    "llm_extra_body_parameters": "dict of llm configuration parameters", - Поле для дополнительных настроек модели (необязательно, если не используется, нужно удалить строчку)
    "context_entries_num": 5, - Поле с значением контекста, устанавливает, сколько чанков из домена будет использоваться для каждого ответа (0 - общение с моделью без информации из домена, 3 - среднее значение точности, 5 - базовое значение, которое даёт хорошую точность ответов)
    "history_entries_num": 0, - Поле с значением истории, устанавливает, сколько прошлых сообщений будет помнить модель (лучше устанавливать чётное значение, так как модель будет помнить и запрос к ней и свой ответ, например при установке значения на 6 модель будет помнить 3 прошлых запроса пользователя и 3 прошлых ответа)
    "reranking": {
      "use": false, - Использовать ли ранкер
      "model": "NDT_RR_1", - Модель ранкера
      "return_entries_num": 3 - Количество данных, которые возвращает ранкер
    }
  }
}
```

5) Указав всё необходимое нажимаем "Execute"

The screenshot shows a REST client interface. At the top, there is a 'Name' field with 'chat_id' and a 'Description' field with a long alphanumeric string. Below this is the 'Request body' section, which is set to 'application/json'. The request body contains a JSON object with various fields like 'text', 'return_debug_data', 'meta_override', 'apt_aggregation_prompt', 'model', 'temperature', 'context_entries_num', 'history_entries_num', 'reranking', 'use', and 'return_entries_num'. A red arrow points from the JSON body down to a blue 'Execute' button.

6) Дожидаемся ответа и получаем ответ на наш запрос и его идентификатор

The screenshot shows the same REST client interface after the request has been executed. The 'Request URL' field shows the full URL. The 'Server response' section is expanded, showing a 'Code' of 200 and a 'Response body' containing a JSON object with a 'response' field and a 'response_id' field. The 'Response headers' section is also visible but mostly obscured by a blurred image.

Запрос совершён, информация получена.

Удаление данных домена

Для выполнения данного действия, вам потребуется заранее узнать domain_id домена, который вы хотите отчистить от старых данных.

Внимание! При отчистке домена, удаляются все документы, которые были в нём и восстановлению внутри системы они не подлежат, не забывайте сохранять данные, чтоб не потерять важную информацию.

1) Находим функцию DELETE /domain/{domain_id}/clear_info и нажимаем Try it out.



DELETE /domain/{domain_id}/clear_info Clear Domain Info

Delete all domain data

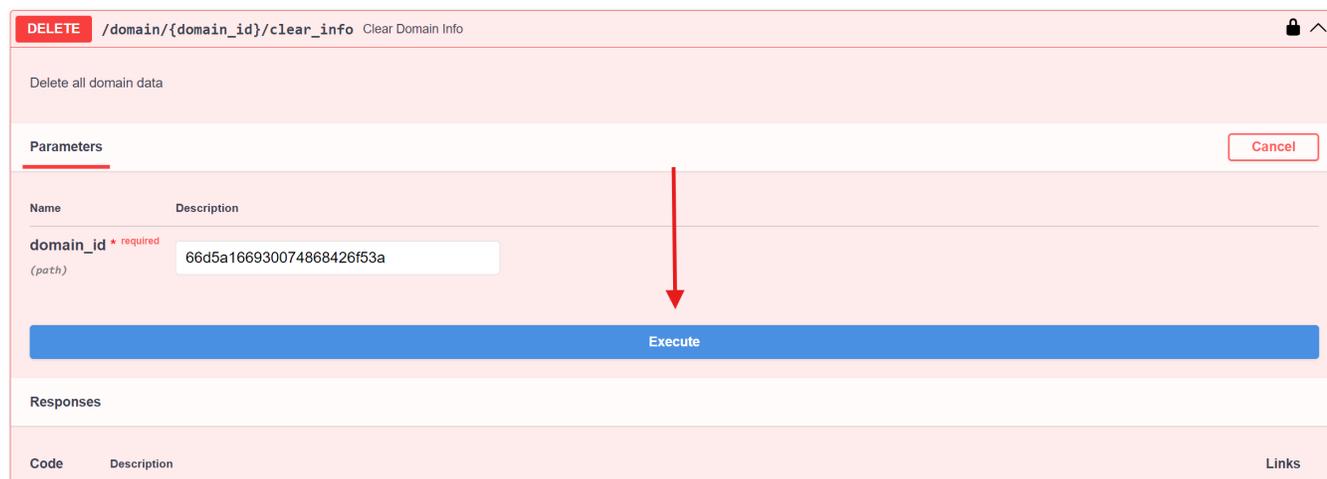
Parameters Try it out

Name	Description
domain_id * required (path)	domain_id

Responses

Code	Description	Links
------	-------------	-------

2) Вставляем domain_id и нажимаем Execute.



DELETE /domain/{domain_id}/clear_info Clear Domain Info

Delete all domain data

Parameters Cancel

Name	Description
domain_id * required (path)	66d5a166930074868426f53a

Execute

Responses

Code	Description	Links
------	-------------	-------

3) Получаем Successful Response.



Server response

Code	Details
204	<p>Response headers</p> <pre>access-control-allow-credentials: true access-control-allow-origin: * alt-svc: h3=;443; ma=86400 cf-cache-status: DYNAMIC cf-ray: 8bcb331d94266ca-AMS content-type: application/json date: Mon, 02 Sep 2024 11:37:18 GMT nel: {"success_fraction": 0, "report_to": "cf-nel", "max_age": 604800} report-to: [{"endpoints": [{"url": "https://a.nel.cloudflare.com/report/v4? s=70a1y8Q5V8h1DgK601x2f2J5QlgCFV53c%2F7kwJ3neL1Yl1fh35w6EkIT0KdqnVz1q4m15dn7L3DqU17XH87hQ3V4R65kgHfuVzVRtEqmsEgd2Bzo39D8Q7mLzQH8KuhOs7INSyY2FX1Q%2B%2FpzgqCKhu3Yb"}], "group": "cf- nel", "max_age": 1604800} server: cloudflare</pre>

Responses

Code	Description	Links
204	Successful Response	No links
	Example Value	
	"string"	
404	object not found	No links

Media type: application/json

Данные успешно удалены из базы знаний.

Отчистка истории чата

Перед началом нужно знать chat_id чата, историю которого вы хотите отчистить.

Внимание! При отчистке истории, удаляются все сообщения, которые были в нём и восстановлению они не подлежат, не забывайте сохранять данные, чтоб не потерять важную информацию.

1) Находим функцию DELETE /chat/{chat_id}/history и нажимаем Try it out.



DELETED /chat/{chat_id}/history Delete Chat History

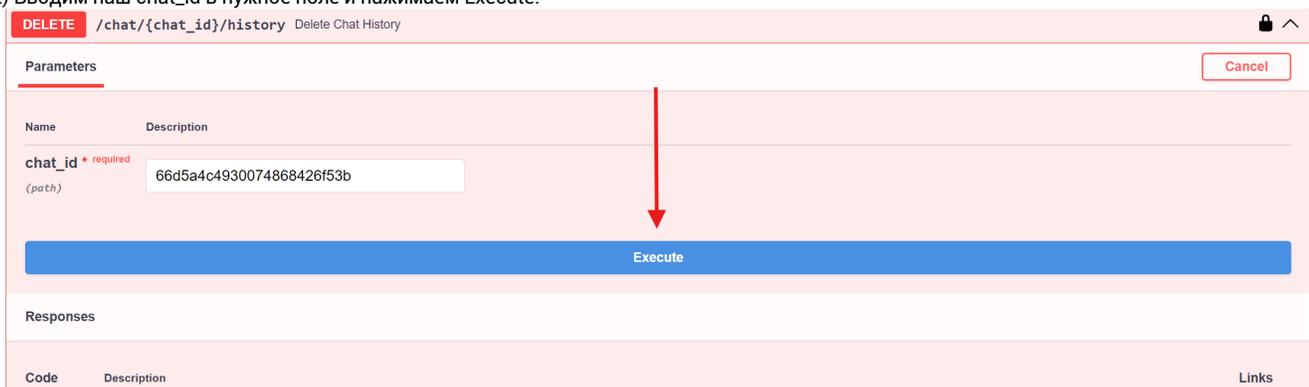
Parameters Try it out

Name	Description
chat_id * required (path)	chat_id

Responses

Code	Description	Links
------	-------------	-------

2) Вводим наш chat_id в нужное поле и нажимаем Execute.



DELETED /chat/{chat_id}/history Delete Chat History

Parameters Cancel

Name	Description
chat_id * required (path)	66d5a4c4930074868426f53b

Execute

Responses

Code	Description	Links
------	-------------	-------

3) Получаем Successful Response.



Code	Description	Links
204	Successful Response Example Value "string"	No links
404	object not found Media type application/json Example Value Schema { "detail": "check exception details here" }	No links
422	Validation Error	No links

История чата успешно отчищена.