Intelligencia AI

API Quick Start Guide

Table of contents

Introduction

Welcome to Intelligencia's API. In this document you will find a quick start guide for our GraphQL API. A detailed list of available operations, types, and fields can be found in our documentation: https://docs.intelligencia.ai.

Contact us with questions, feedback or improvement suggestions: support@intelligencia.ai

Prerequisites

Before you can start experimenting with the API please make sure that you have received the following from us:

- API URL/Endpoint
- AUTH URL/Endpoint
- Client ID
- Client Secret
- Postman Collection

API URL/Endpoint, AUTH URL/Endpoint, Client ID, Client Secret are necessary to start working.

Intelligencia's API is based on GraphQL; GraphQL specifications can be found here: https://spec.graphql.org/

Authentication

To be able to use any of the API Operations, first you need to authenticate and generate an access token (JWT). To do so, perform a POST request against the authentication url which you have received. The request content type must be x-www-form-urlencoded, and the request body must include the following fields:

- grant_type=client_credentials
- client_id as provided by intelligencia
- client_secret as provided by Intelligencia

Below you can see a cURL example (A ready to use example exists in the Postman collection as well).

```
curl --location --request POST auth_url \
--header 'Content-Type: application/x-www-form-urlencoded' \
--data-urlencode 'grant_type=client_credentials' \
--data-urlencode 'client_id=client_id' \
--data-urlencode 'client_secret=client_secret'
```

When everything is correct, performing this request will result in a JSON response with a 200 status code. The response will include, among other information, a key value pair with the access token (access_token). Example response:

```
{"access_token":"some_access_token","expires_in":1800,"refresh_expires_in":3600,
"refresh_token":"some_refresh_token","token_type":"bearer","not-before-policy":0,
"session_state":"de223b60-7d0d-42ae-bffc-d3f44cdc84a6","scope":"profile email"}
```

This token must be used in the operations you want to perform against the API as an Authorization bearer request header. Below you can see a cURL example (requesting action types for the drug Nivolumab)

```
curl --location --request POST api_url \
--header 'Authorization: Bearer some_access_token' \
--header 'Content-Type: application/json' \
--data-raw '{"query":"query MyQuery($name: String) {\n  api_drug(where: {preferred_name: {_eq:
$name}}) {\n    action_types\n   }\n}\n","variables":{"name":"Nivolumab"}}'
```

Token Notes

- Please keep in mind that the token has a specific time to live (TTL) and this can be found in the authentication response in the field expires_in (number is in seconds). After the token expires, you should refresh it by simply following the same authentication process.
- Note that If you use an expired token and perform an operation, the operation will result in a status code 200 and the error message will appear in the body.

```
{
   "errors": [
       {
           "extensions": {
               "path": "$",
               "code": "invalid-jwt"
           },
           "message": "Could not verify JWT: JWTExpired"
       }
   ]
}
```

Executing Queries

Assuming you have created the token, you can now execute any GraphQL query against the API. Consult the API documentation on how to shape your queries, and check the provided Postman collection for examples. Below you can see a simple cURL example for requesting data for the drugs with a specific name or synonym.

```
curl --location --request POST 'api_url' \
--header 'Authorization: Bearer token' \
--header 'Content-Type: application/json' \
--data-raw '{"query":"query MyQuery($name: String, $synonym: jsonb) {\n  api_drug(where: \n  {\n
_or: [\n      {synonyms: {_contains: $synonym}},\n      {preferred_name: {_eq: $name}}\n      ]\n
}) {\n    drug_id\n    action_types\n    genes\n    mechanism_of_action\n    modalities\n
biological_pathways\n    preferred_name\n    synonyms\n    targets\n    protein_class\n
}\n}\n","variables":{"name":"Cisplatin","synonym":[{"name":"ONO-4538"}]}}'
```

The above will generate a response similar to:

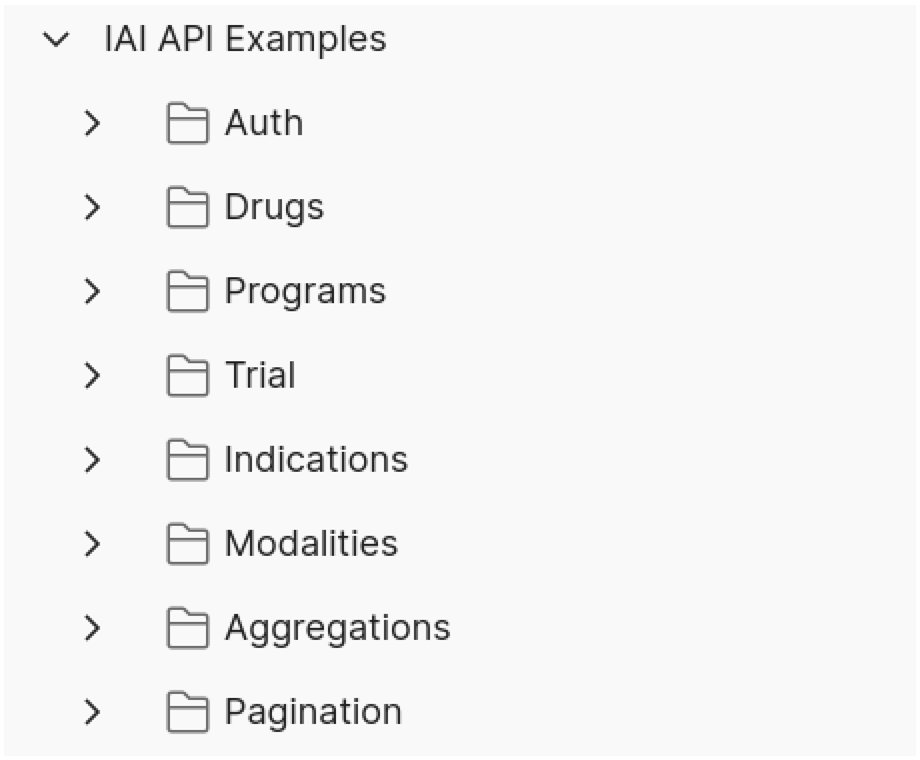
```
{"data":{"api_drug":[{"drug_id":4670,"action_types":[{"name": "Antagonist"}],"genes":[{"name":
"PDCD1"}],"mechanism_of_action":[{"name": "Programmed cell death protein 1
Antagonist"}],"modalities":[{"name": "Monoclonal antibody", "parent":
"Antibody"}],"biological_pathways":[{"name": "Adaptive Immune System"}, {"name": "Costimulation by
the CD28 family"}, {"name": "Immune System"}, {"name": "PD-1
signaling"}],"preferred_name":"Nivolumab","synonyms":[{"name": "NIVOLUMAB [Injectable/Intravenous;
Injectable/Subcutaneous; Injectable/Intraperitoneal; Injectable/Intramuscular]"}, {"name":
"ONO-0123"}, ],"targets":[{"name": "Programmed cell death protein 1"}],"protein_class":[{"name":
"Receptor"}]}, {"drug_id":1588,"action_types":[{"name": "Cross-Linking
Agent"}],"genes":null,"mechanism_of_action":[{"name": "DNA Cross-Linking
Agent"}],"modalities":[{"name": "Other small molecule", "parent": "Small
molecule"}],"biological_pathways":null,"preferred_name":"Cisplatin","synonyms":[{"name":
"Abiplatin"}],"targets":[{"name": "DNA"}],"protein_class":null}]}}
```

Postman

This section is focused on utilizing Postman in order to interact with the API. Postman is an API tool - an HTTP client essentially -, which allows setting up an environment, sharing request examples and more. You can find more info on Postman's website (https://www.postman.com), you can download it from here: https://www.postman.com/downloads/, or use the web application.

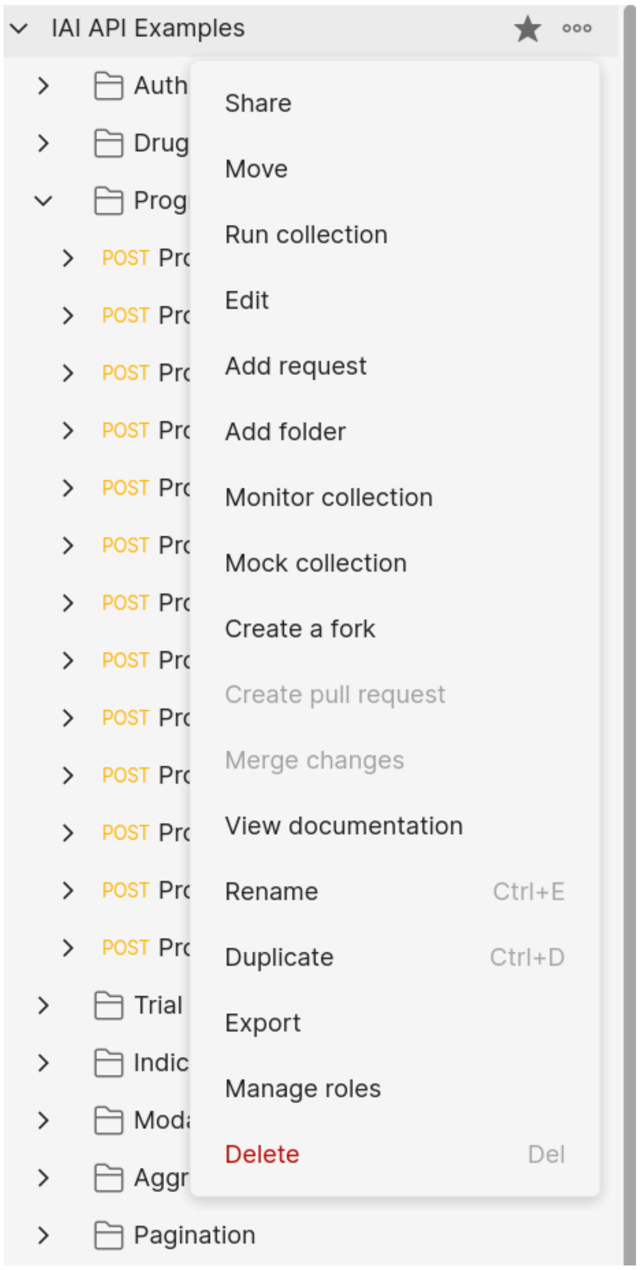
Collection

By now you should have received from us a link to Intelligencia's Postman collection. Please note that we are constantly improving our Postman collection, thus you may find differences between the actual collection and the screenshots presented later in this document regarding the collection itself (e.g number of requests shown).
The provided Postman collection includes examples for Authentication and Operations (queries). You need to create a Postman account in order to work with it.



Documentation

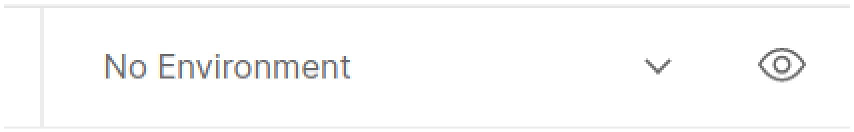
You can access collections' documentation via clicking on the three dots next to the collection name and selecting View Documentation.



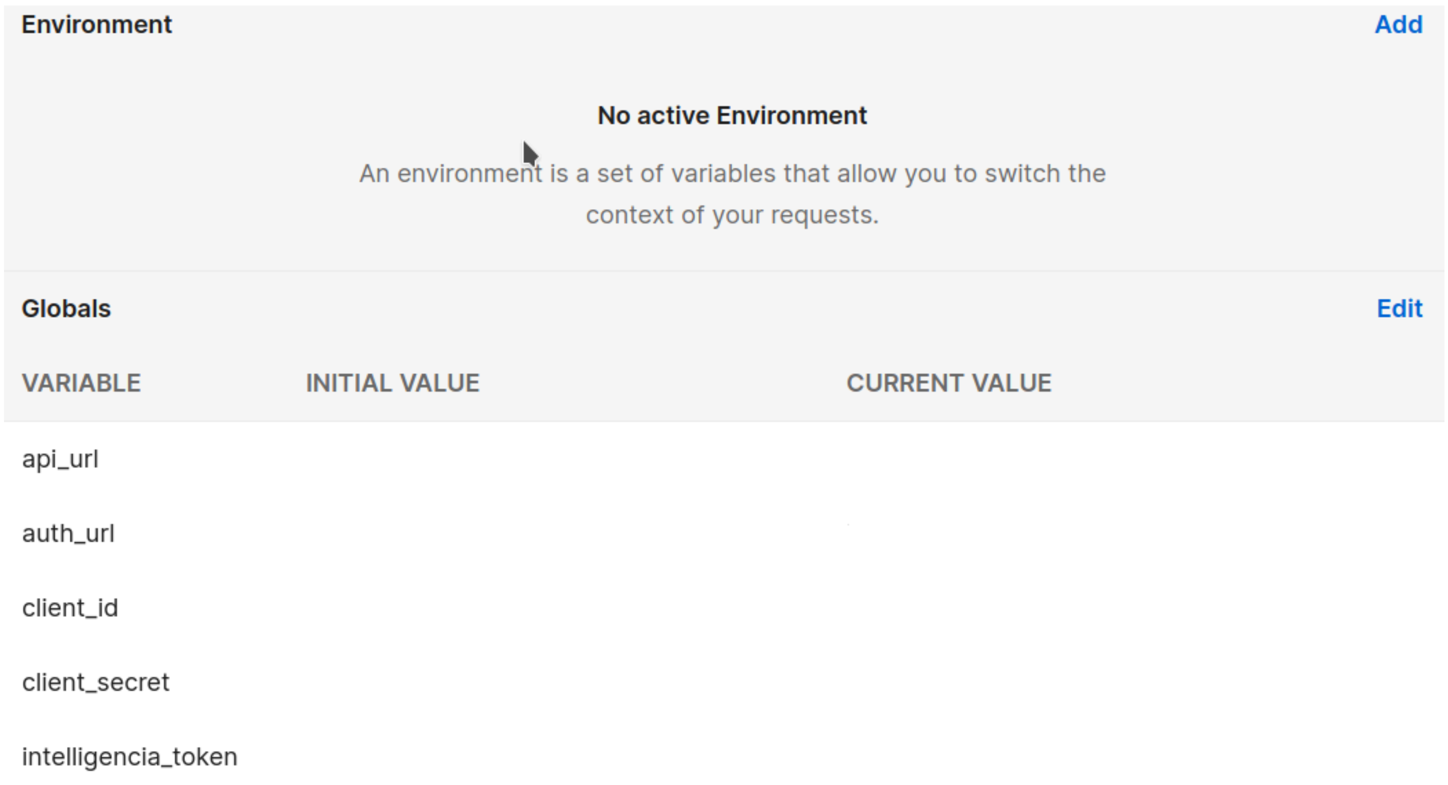
There you will find some notes on how some of the queries work and what is the logic behind some of them.

Setting up environment

Before you start executing any action in Postman, you need to set up your environment. In the upper right corner you should see:



Click on the eye icon and you should see some predefined variable names under the globals sub section.


Environment Add

No active Environment

An environment is a set of variables that allow you to switch the context of your requests.

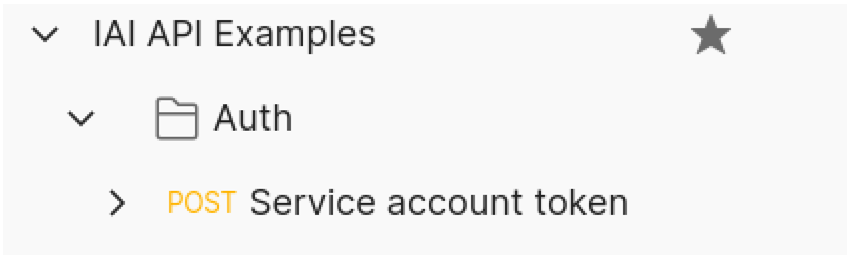
Globals Edit

VARIABLE	INITIAL VALUE	CURRENT VALUE
api_url		
auth_url		
client_id		
client_secret		
intelligencia_token		

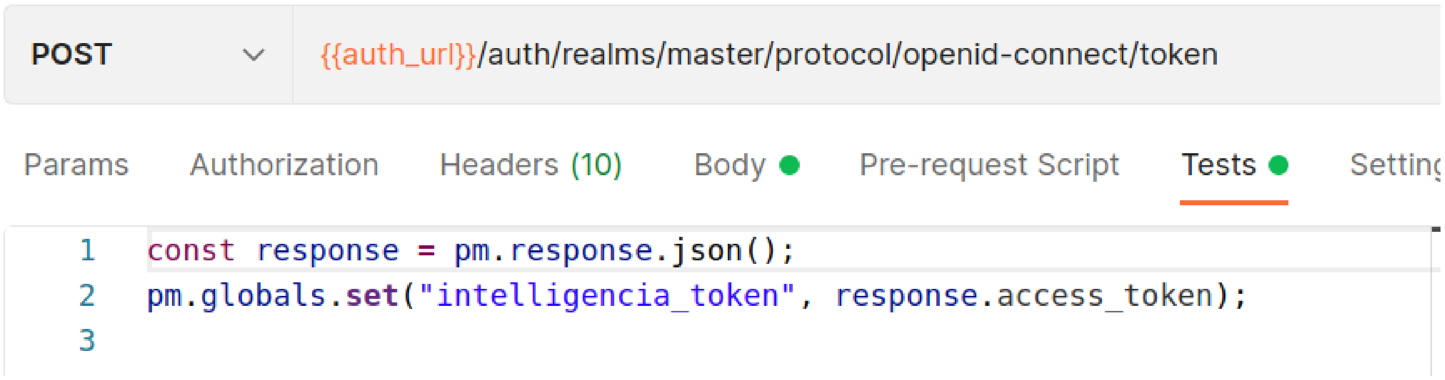
By clicking on the Edit button you are able to change the values of those variables. Fill in the CURRENT_VALUE of api_url, auth_url, cliend_id, and client_secret variables with the information you have received. The value of "intelligencia_token" should be filled with the response of the auth endpoint. (It is also filled in automatically if you use the sample auth request).

Running Queries

If you completed the previous step successfully you are now able to execute actions via Postman. The first step for this should always be to authenticate. Go to the Auth directory and execute the Service account token request.

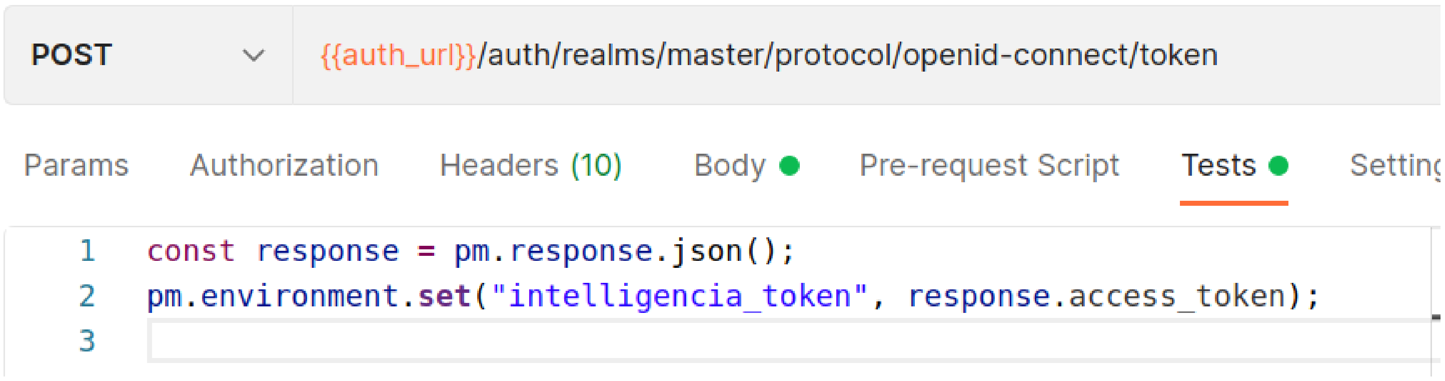


This will perform a POST request against the Auth url and if the request is successful it will also set up the returned token as intelligencia_token in the global environment variables of postman. You can see how this works if you go to this request's Tests section.


POST {{auth_url}}/auth/realms/master/protocol/openid-connect/token

```
const response = pm.response.json();
pm.globals.set("intelligencia_token", response.access_token);
```

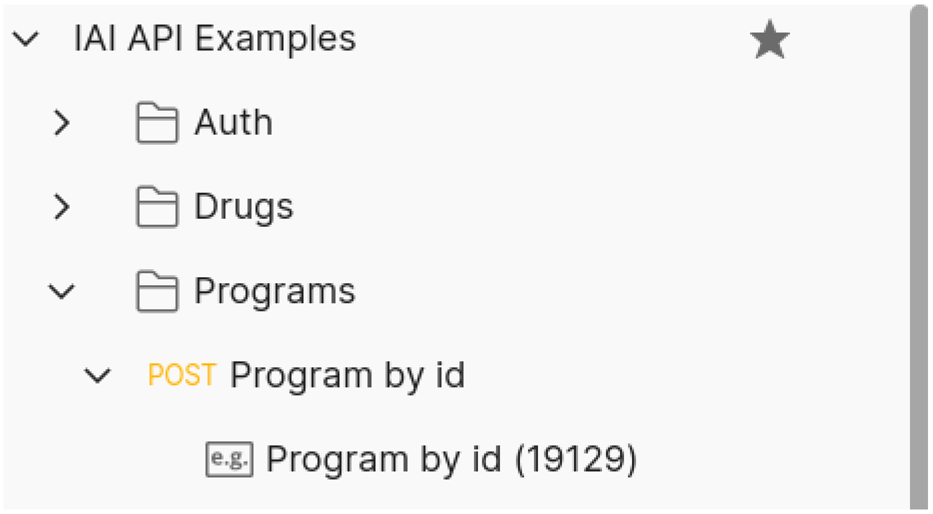
If for some reason you have chosen to create your own environment with the variables discussed earlier, this should be modified as shown below in order to set the variable in the respective environment.


POST {{auth_url}}/auth/realms/master/protocol/openid-connect/token

```
const response = pm.response.json();
pm.environment.set("intelligencia_token", response.access_token);
```

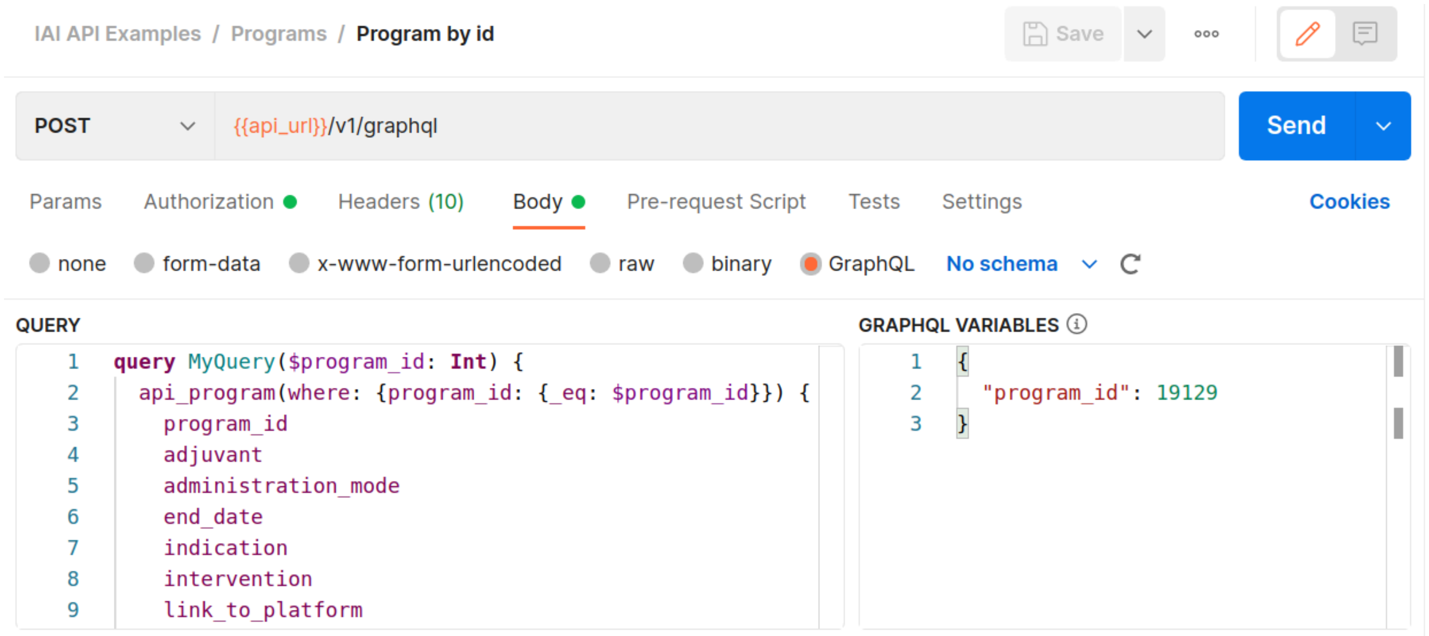
Queries

With the token generated you are able now to execute any of the given query examples, or prepare your own queries. For each given query example you will also find an example request/response within the collection. Click to the arrow next to the example name and then you can access the example request/response.



For instance, above you can see that for the Program by id request there is an example request/response for the program id 19129.

To perform any of the given queries, just select them from the directory list and click the Send button from the bar on the right.

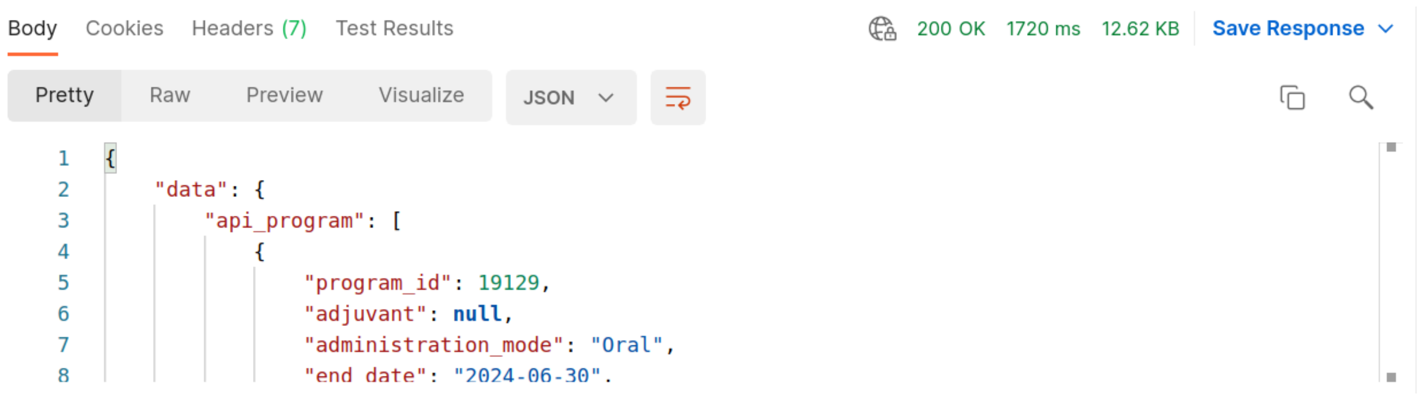


```
{{api_url}}/v1/graphql
```

```
query MyQuery($program_id: Int) {
  api_program(where: {program_id: {_eq: $program_id}}) {
    program_id
    adjuvant
    administration_mode
    end_date
    indication
    intervention
    link_to_platform
```

```
{
  "program_id": 19129
}
```

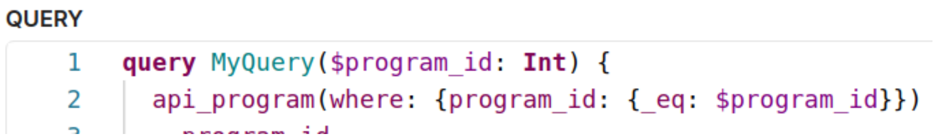
This will return the relevant response to the lower part of the Postman application.



```
{
    "data": {
        "api_program": [
            {
                "program_id": 19129,
                "adjuvant": null,
                "administration_mode": "Oral",
                "end_date": "2024-06-30",
```

Notes on the requests

- You should always pick GraphQL as the type of the request in Postman
- All the requests are performed using the POST method
- Note that the erroneous responses will return status code 200 and they will include in the body of the response a key named errors and a value with an array of those error descriptions. See the “Common Errors” section for more details.
- Most of the requests include some variable as you can see in the screenshot earlier. Feel free to play around with the variables values or the variables themselves to get acquainted with how they work. Although if you add a new variable make sure that you add it in the GraphQL variable object as well along and in the GraphQL query declaration.



```
query MyQuery($program_id: Int) {
  api_program(where: {program_id: {_eq: $program_id}})
```

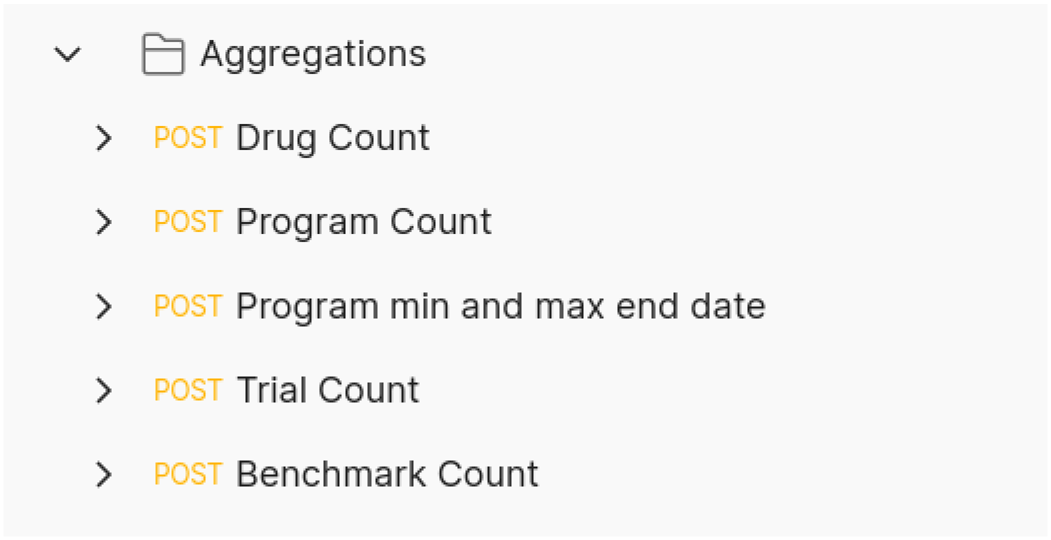
Query operators

Above you saw an example of equality checking for the program id in the where clause. It is important to be aware of the type of the field on which you want to perform filtering. Each of the types has its own operators (several of them intersect), and those are available in the documentation. For example, to see how to query String fields check the String_comparison_exp, for citext the citext_comparison_exp, for jsonb the jsonb_comparison_exp in the https://docs.intelligencia.ai/api/index.html.
If you wish to check for some field’s/object’s existence use the `_is_null` (set to true or false accordingly).

Aggregations

Aggregations are types that allow the user to perform simple aggregations (count, min, max) against a type. The naming convention to use this type is api_*type*_aggregation.

You will find several examples on how to use them in the aggregation directory of the Postman collection. The counts are extremely useful when you are consuming paginated results in order to know how many requests you should perform.

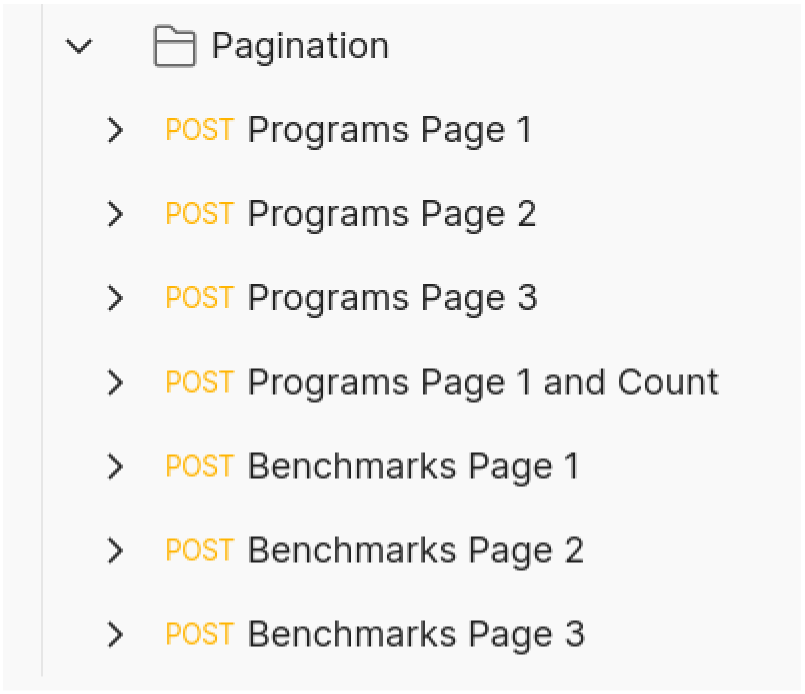


Aggregations are enabled for the core types of the API (api_program, api_trial, api_drug, etc).

Pagination

The operators limit and offset are used for pagination. Limit specifies the number of rows to retain from the result set and offset determines which slice to retain from the results. There are specific limits set to the types exposed from the API. Please check the API documentation for default limits applied per type. These limits will allow the consumer to get the relevant information for one type without forcing him to use nested limits and offsets and also enable good experience by securing good response times.

Limit and offset, as mentioned, are the operators one should use to create paginated responses but it is always needed to use the order_by operator as well. For most of the time you can use the id field (field name convention: *type*_id). For benchmarks, where no id exists, it is recommended to use order_by with indication and/or metric. In the Postman collection you can find examples for both cases (using an id field or using other fields) in the relevant directory.
You can also find examples on how to use aggregation and normal queries in the same request.



Pagination

POST Programs Page 1

POST Programs Page 2

POST Programs Page 3

POST Programs Page 1 and Count

POST Benchmarks Page 1

POST Benchmarks Page 2

POST Benchmarks Page 3

Common errors

In this section we will present some of the common errors you may encounter while working with the API along with suggestions on how to tackle them.

Authentication

Authentication requests will respond with 4xx status codes if something is wrong with the request.

- Attempting authentication with erroneous client id

```
{
   "error": "unauthorized_client",
   "error_description": "INVALID_CREDENTIALS: Invalid client credentials"
}
```

- Attempting authentication with erroneous client secret

```
{
   "error": "unauthorized_client",
   "error_description": "Invalid client secret"
}
```

For both cases please make sure that you are using the correct credentials provided from us.

Query

Queries typically will return a response with a 200 status code even when they are erroneous. We highly recommend that you always check that the response does not contain a key named “errors”. Some common errors you may encounter are:

- Performing a request with an expired token. Response:

```
{
   "errors": [
       {
           "extensions": {
               "path": "$",
                "code": "invalid-jwt"
           },
           "message": "Could not verify JWT: JWTExpired"
       }
   ]
}
```

  In such a case please generate a new access token to use in the following requests.

- Requesting for a field that does not exist in the specific type e.g field my_trial in api_program type.

```
{
   "errors": [
       {
           "extensions": {
               "path": "$.selectionSet.api_program.selectionSet.my_trial",
               "code": "validation-failed"
           },
           "message": "field \"my_trial\" not found in type: 'api_program'"
       }
   ]
}
```

  In this scenario make sure that the list of fields you are requesting for the particular type is valid. Consult the documentation for this. If the field exists in documentation and there are no mistakes in your request contact us.

- Requesting for a type that does not exist e.g api_benchmarkz

```
{
   "errors": [
       {
           "extensions": {
               "path": "$.selectionSet.api_benchmarkz",
               "code": "validation-failed"
           },
           "message": "field \"api_benchmarkz\" not found in type: 'query_root'"
       }
   ]
}
```

In this scenario make sure that the type you are requesting is valid. Consult the documentation for this. If the type exists in documentation and there are no mistakes in your request, contact us.

- Requesting against an endpoint that does not exist e.g api_url/v1/graphql/programs

```
{
   "path": "$",
   "error": "resource does not exist",
   "code": "not-found"
}
```

In this case make sure that you are performing the request against the correct endpoint.

Server Error

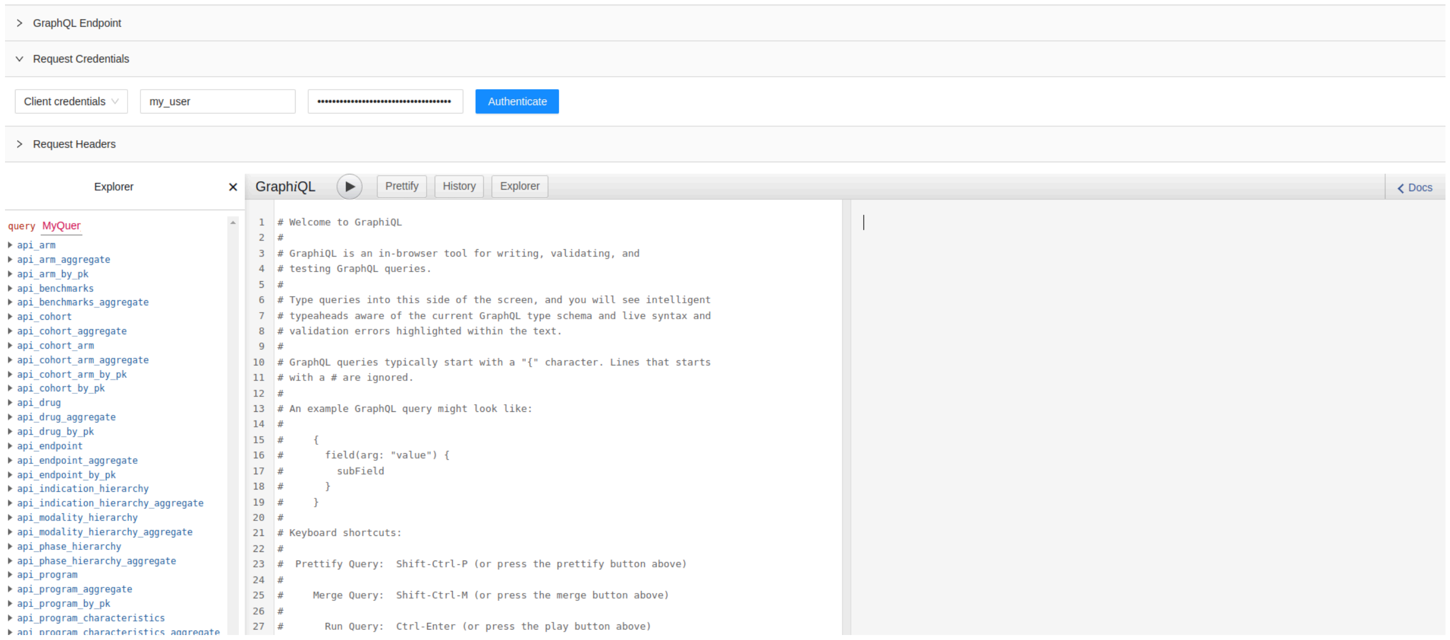
For any request that will result in a status code 5xx (e.g Internal Server Error) please follow the next steps

- Retry after a couple of minutes
- If the problem persists contact us in support@intelligencia.ai

Explorer

This section introduces explorer, a tool that allows direct interaction with our API and the underlying entities, which you can access at https://explorer.intelligencia.ai/.

The tool is based on the open-source GraphiQL explorer (https://github.com/graphql/graphiql). It can be used to create queries by clicking on the desired entities and selecting the fields of interest. In order to use it, you need to provide your client credentials (client id and client secret in their respective fields) and authenticate. After authenticating, all available entities will appear on the left side of the page.



```
query MyQuer
api_arm
api_arm_aggregate
api_arm_by_pk
api_benchmarks
api_benchmarks_aggregate
api_cohort
api_cohort_aggregate
api_cohort_arm
api_cohort_arm_aggregate
api_cohort_arm_by_pk
api_cohort_by_pk
api_drug
api_drug_aggregate
api_drug_by_pk
api_endpoint
api_endpoint_aggregate
api_endpoint_by_pk
api_indication_hierarchy
api_indication_hierarchy_aggregate
api_modality_hierarchy
api_modality_hierarchy_aggregate
api_phase_hierarchy
api_phase_hierarchy_aggregate
api_program
api_program_aggregate
api_program_by_pk
api_program_characteristics
api_program_characteristics_aggregate
```

```
# Welcome to GraphiQL
#
# GraphiQL is an in-browser tool for writing, validating, and
# testing GraphQL queries.
#
# Type queries into this side of the screen, and you will see intelligent
# typeaheads aware of the current GraphQL type schema and live syntax and
# validation errors highlighted within the text.
#
# GraphQL queries typically start with a "{" character. Lines that starts
# with a # are ignored.
#
# An example GraphQL query might look like:
#
#     {
#       field(arg: "value") {
#         subField
#       }
#     }
#
# Keyboard shortcuts:
#
#  Prettify Query:  Shift-Ctrl-P (or press the prettify button above)
#
#     Merge Query:  Shift-Ctrl-M (or press the merge button above)
#
#       Run Query:  Ctrl-Enter (or press the play button above)
```

From this point on, simply point and click on the desired entities, fields, operators, and arguments to explore the schema and generate queries in the query editor section. After generating a query, press the “Execute Query” button (or press Ctrl-Enter) to see the resulting data.

Reporting issues

If you wish to report an issue regarding Intelligencia’s API please contact support@intelligencia.ai and in your mail always include the request, the response (where applicable) -ideally both in their typical format (JSON)-, and the timestamp of the issue along with a short description.