INTEGRUM ESG

Integrum ESG API Guide V2.8 Dec 2024

Introduction

This document provides a developer with an overview of the concepts in Integrum ESG data, which should help with understanding the descriptions of the APIs in the reference guides and examples.

It is a low-level API and is intended to serve as a general-purpose machine interface. Later in this document we will describe tools for inspecting the data model and give examples for using the queries in some different technologies.

The companies and metrics web APIs are provided as a GraphQL endpoint. GraphQL is a powerful API query language that allows users to select which data they want.

The API Document Collection

1. API Guide (this document): Describes the core features and main areas of the different Integrum ESG APIs and the mechanisms for querying them.
2. Portfolio Management API Guide: Designed for the integration requirement of uploading portfolios to the Integrum ESG system
3. API Reference Schema: Contains the full metadata/ data dictionary for each Integrum ESG API
4. GraphQL API Manual: This document directs the user to the GraphQL query builder and gives an example of building a step-by-step complete extract for a single company. It also gives examples of building sentiment queries and bulk extract.
5. ERD: An entity-relational diagram, for users who want to populate a data warehouse using our bulk API.
6. All metrics and available data: An Excel file containing every data point that can be pulled via the API.

Designing the integration

Some integration use cases might be for ad-hoc ESG data queries, either on a company or portfolio basis. For portfolio cases, see the Portfolio Management API guide on uploading portfolios, or contact us for options on automatic extracts from portfolio management systems.

Other, more analytic cases may call for the construction of a local database and a bulk extract to feed it. In that case, using individual queries would be inefficient, so we recommend using a set of different API top levels.

In general, we prefer fewer, larger queries rather than a lot of small ones. Our system dynamically calculates scores and there is a per query overhead.

The resulting large responses can also be compressed - and this is discussed in the bulk API examples, found in the GraphQL API Examples document.

GraphQL not REST

For the metrics and sentiment data, we have multiple levels which might want to be queried in collections, as individual endpoints would become cumbersome.

As different customers' integration needs are different it makes more sense to use a query to specify the data required – and a good standard for web API-based queries is GraphQL. https://GraphQL.org/

POST not GET

That means that a query is required to access the data and that query is provided to the API in a POST request, not a GET request. You can still experiment with the API with standard tools like Postman – but you have to follow the GraphQL standard.

GraphiQL not Swagger

Although you can use standard tools like Postman (and some curl examples are given in the examples document), GraphQL is designed to serve its own metadata so we do not have static documentation to describe the schema.

Instead, it is easier to browse the metadata in an interactive tool like GraphiQL. How to use the GraphiQL tool is in this document, and a full worked example is given in the GraphQL API Examples document.

A tabular version of the schema is still mildly useful for searching so it is included as an appendix to the API reference – along with a script which can generate it from the live metadata.

Metric Structure

We strongly suggest reviewing the Integrum ESG public dashboard or Company Level Dashboard PDF to see how the metrics and scores appear in practice before attempting to pull the data. This will help you better understand the names and meanings of the data, as the dashboard provides a clearer representation of the information.

There are videos and PDFs on our website to explain the key functionality: https://www.integrumesg.com/tutorial-video-hub

Materiality

Integrum ESG is a licensee of the SASB/IFRS materiality map. This proposes the most important sustainability issues for each global sub-sector. Therefore, not all metrics will be relevant for each company. The sub-sector metric mappings are given in detail in the API reference document.

This can be an issue when we want to calculate scores and metrics for some key measures like emissions. Despite the interest in carbon emissions, it is not always material for the sustainability score.

So if we are looking for metric-level data for comparisons across diverse companies, it will be better to use the Impact metrics (see below).

Impact Metrics are universal

The Impact metrics are measured across six themes which are highly representative of a company's contribution to the United Nations Sustainable Development Goals (SDG).

The Impact metrics are present for all companies and include important measures on ecosystems and resources. They are slightly less obvious as they have abstract names from the SDGs.

The latest API introduces a metric description field which makes this easier to see.

Impact Metric	**Description**
Wellbeing	Tax contributed to all governments
Resource Security	Waste generation
Decent Work	Number of jobs created by the company
Climate Stability	CO_2 (and other GHG) emissions
Healthy Ecosystems	Water consumption

Values, Scores and Grades

Our scores are built up in a hierarchy in a very transparent and configurable way. You can read about our scoring methodology here: Scoring Logic

Metric scores and grades

At the bottom level of the scoring, we have metrics.

A metric has two scores :

1. **Awareness:** The awareness score is an absolute score and is defined on a per-metric basis by our scoring logic. Typically it will involve looking at the policies around the topic, disclosure of data, and setting meaningful targets. The reason for scoring each metric is described in the dashboard and is available as the "awareness_comment" field in the metric APIs (see the API reference document)
2. **Performance:** For a quantitative metric this is a relative score of quartiles 1-4 in the company's global sub-sector with zero for non-disclosure.

A quantitative metric will have a performance value which is the actual numeric value, it also has a standard unit that it is conformed to. Where appropriate, there is also a scaled performance value which is the performance value per million dollars of revenue which allows for comparison between companies of different sizes.

Awareness and performance scores are scored from 0-4 and are then directly converted to E-A grades.

Excerpts

A metric also has excerpts which are short pieces of text from the company disclosures which influenced our scoring. You can pull the relevant excerpts from company disclosures that support every score at the sub-metric level.

We can also a derive a url to retrieve the screenshot of the page where we found the excerpt.The screenshots themselves are not served through the api.

See the 'Detailed metric example' in the examples document for more information

Dimension scores

The low-level metrics are grouped into higher-level dimensions.

Some sustainability dimensions (e.g. Product Quality and Safety) and many Governance dimensions have multiple sub-metrics.

The dimension score is the average of the metric scores.

Issue scores

The dimension scores are then aggregated into an issue score (an issue being the high-level topic such as Sustainability, Impact and Governance). As a default, all metrics are equally weighted, as SASB has determined that the metrics mapped against a sub-sector are all material - each dimension can be given a customised weight in the profile.

The Sustainability and Governance issues are then aggregated into the overall company score.

Grades

The dimension, issue and company levels A-E grades are determined by taking the score and its distance in standard deviations (for that type of score) from 2.

Score lower	Score upper	Grade
> 2 + 1.5 sd		Grade A
2 + 0.5 sd	2 + 1.5 sd	Grade B
2 - 0.5 sd	2 + 0.5 sd	Grade C

2 - 1.5 sd	2 - 0.5 sd	Grade D
	< 2 - 1.5 sd	Grade E

This is why, on our scoring system, C is a fair grade rather than a bad one.

Connectivity and Authentication

The production endpoint is: https://dashboard.integrumesg.com/graphqlapi/v1/graphql

Authentication is by API key, being passed as '`Api-Key XXX`' in the '`Authorization`' HTTP header. (Note that '`Api-Key`' is case sensitive)

The API key generation is self-service and it is obtained from the user options on the Integrum ESG dashboard. To find the API key, please follow the steps below:

1. Log in to the Integrum ESG Dashboard: https://dashboard.integrumesg.com/dashboard/
2. Click your initials in the top right-hand corner
3. Click "Account"
4. Click "Generate API Key"

Requests are made in the usual GraphQL way of posting the query and receiving the results as a JSON payload

Rate Limiting

Given complex and nested queries can be expensive, we do apply rate limiting. The system estimates the complexity of a query and applies a limit to the number that can be applied in a given period.

Initially, it applies back pressure by delaying responses. However, if you are making a lot of queries and the load continues, then it refuses authorization and an "element not found" error can be returned.

Therefore, it can be better to design queries which return as much of what you want as possible, rather than making a larger number of smaller queries.

It is also better not to try to make a lot of requests in parallel as it overcomes the effect of backpressure and increases the chances of requests being rejected.

V1 and V2 Root Elements

The V1 API packages up the most important fields as displayed on the dashboard and makes them available programmatically. It combines the current and previous year scores along with sentiment and SFDR compliance.

The V2 API is suitable for more complex and bulk API integration scenarios. It unbundles the separate data elements allowing much more control - at the cost of requiring a deeper understanding of the structure and meaning of the data.

V1 API - Root Elements

The root elements:

`company_api`

`country_api`

are the simplest entry points for company and country scoring data, and they contain relationships to the individual sub-metrics and the dimensions which group the sub-metrics (Those objects being `dimension_api`, `company_dimension_api`, `metric_api` and `country_metric_api`)

There is also supporting metadata for sectors and the standard GraphQL aggregation objects for each of the main objects.

Queries on the `company_api` and `country_api` should be constrained to a profile_id as calculating the scores across all of the profiles is very expensive, and will usually cause the query to time out.

The value –1 is a distinguished profile_id for the equal-weighted profile.

The `company_api` element is also nested under the `portfolio_api` element. See the examples document for an example of retrieving all of the ESG scores for a portfolio.

V2 API - Root Elements

The main v2 root elements is: `company_api_v2`

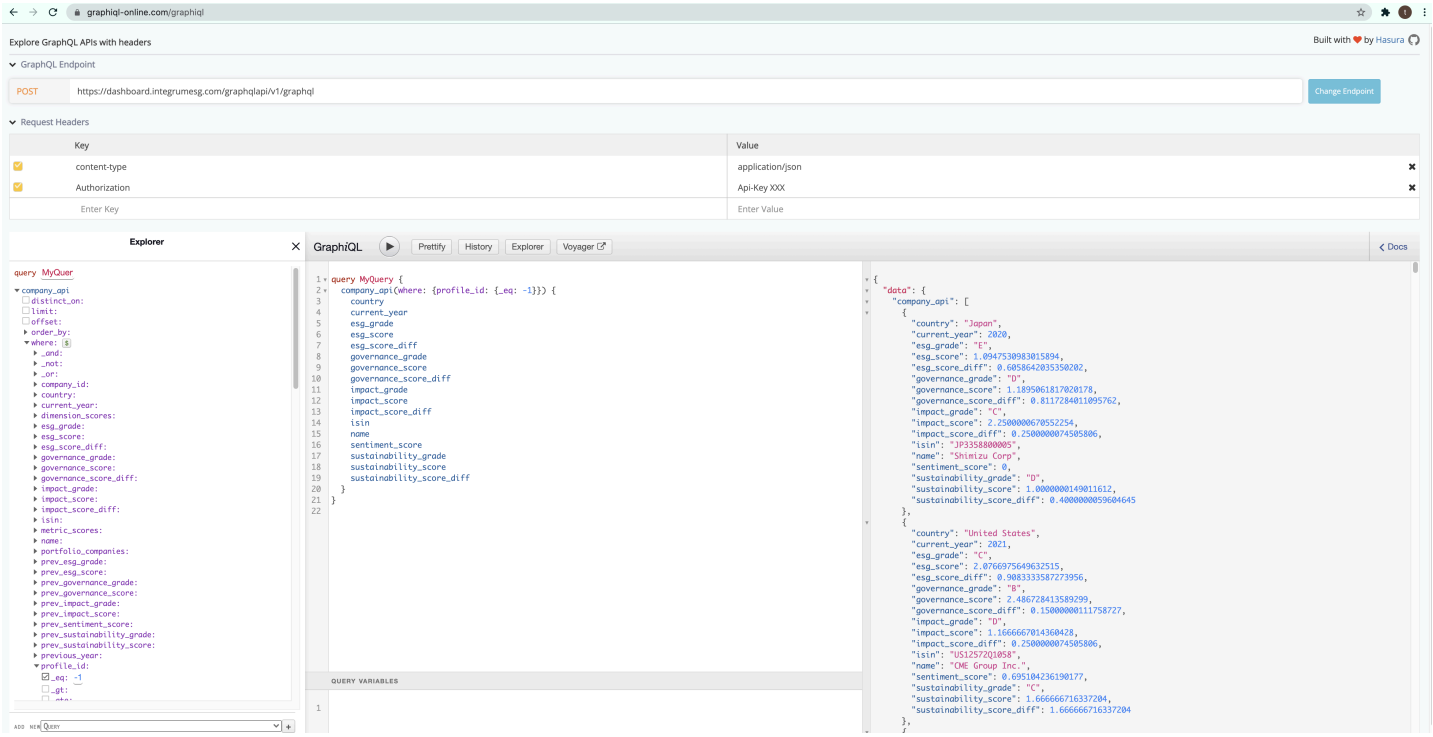
There is also a `company_scores_api_v2` but that is largely intended to be nested under the `company_api_v2` where it appears as `company_sores`.

Using GraphiQL to browse and generate queries

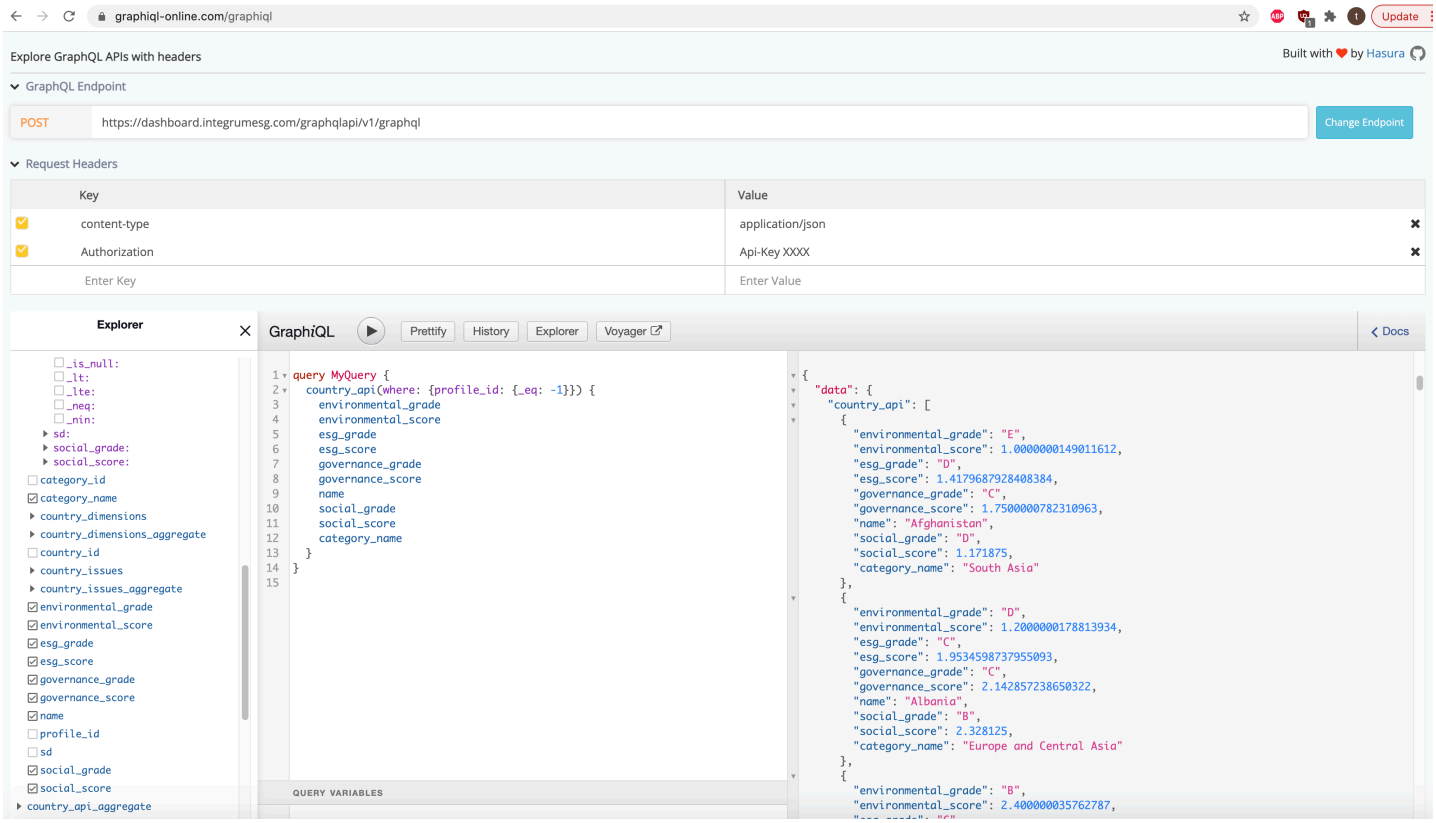
GraphiQL is an interactive tool for exploring the schema and generating queries.

This example shows using the free online version https://graphiql-online.com/graphiql, but apps and plugins for development environments are available as well.

The screenshot shows the explorer view (left pane) selecting data from the `company_api` and generating and fetching the data.



Similarly, for countries:



Tips on structuring GraphQL queries

In GraphQL, the structure of the query is designed as the shape of the json that you would like to see returned.

Each nested element may have its own where clauses, there are examples of that in the examples document where the scores elements can be nested in portfolio elements.

Another example would be getting specific metrics instead of all of them.

A nested sub-element will relate to its parent. For example, the `metric_scores` element of the `company_api` will contain the scores for that particular company. In that sense the nesting takes the place of joins in a relational data model.

There are sometimes multiple nestings as different queries with different roots can benefit from those relationships - so there may be multiple different queries which could solve a problem in different structures.

Metric Flag

The `metric_flag` field has been introduced in the `metric_api` and `metric_api_with_proxies` to provide additional contextual data for the **Independent Directors %** metric, the `metric_flag` field includes the chair's name and their independence status.

In this example:

```
query MyQuery () {
        metric_api(where: {metric_flag: {_is_null: false}, profile_id: {_eq: -1}}) {
          company_id
          metric_flag
          name
        }
      }
    }
```

The response structure will be:

```
    "metric_api": [
      {
        "company_id": 1,
        "metric_flag": {
```

```
        "chair": "Doe, John",
        "is_independent": "Yes/No"
      },
      "name": "Independent directors %"
    },
```

ISIN and ISINs

Investors often use an ISIN to identify the instruments that they have invested in and we map those ISINs onto the issuing company. Note that the company for ESG monitoring purposes may not be the exact same legal entity as the issuer. For example, bonds may be issued by a finance subsidiary but the ESG reporting is done at the group level.

Even for equity instruments, there may be multiple subsidiaries and multiple issues.

The single ISIN available in the company API is a 'primary' ISIN, usually the equity instrument issued in the country of headquarters.

Using variables to query a list of ISINs

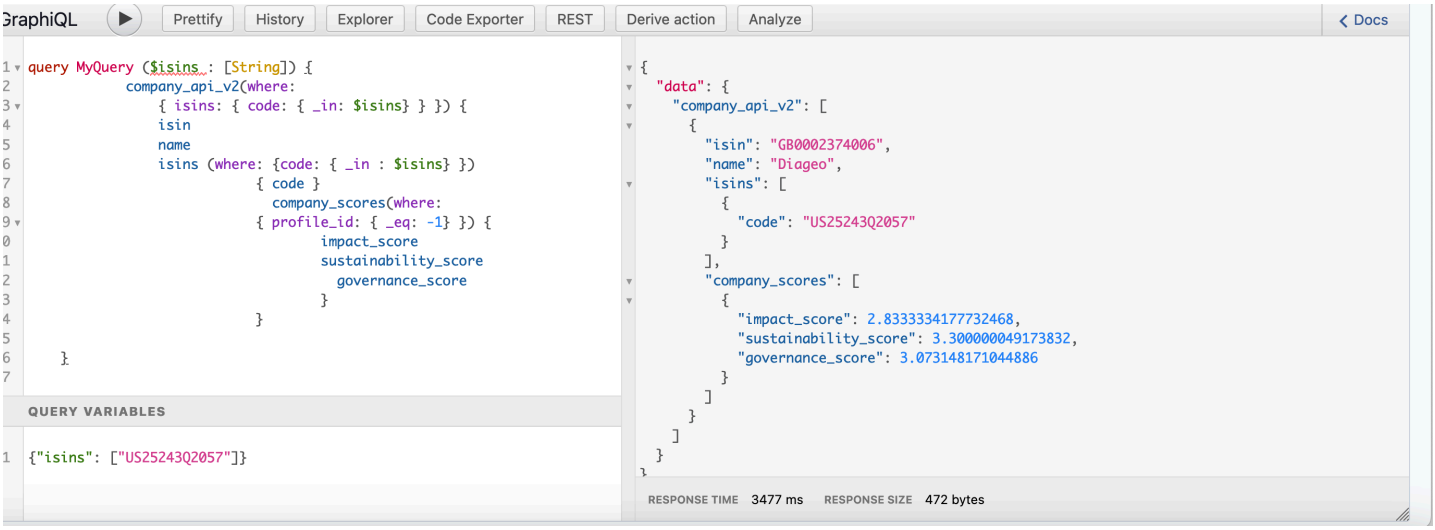
In this example:

```
query MyQuery ($isins : [String]) {
            company_api_v2(where:
                { isins: { code: { _in: $isins} } }) {
                isin
                name
                isins (where: {code: { _in : $isins} })
                            { code }
                              company_scores(where:
                            { profile_id: { _eq: -1} }) {
                                    impact_score
                                    sustainability_score
                                    governance_score
                                }
                            }

    }
```

The returned `isin` is the primary isin from the company. The incoming isin in the query can be used both in the condition in the where clause and the returned elements by use of a variable.

The isin is in the `code` field of the `isins` element.



Proxy coverage

Where we do not have coverage of a company from company reports, we can derive a "proxy" score using the average results for the sector and geographic region. In the dashboard, we have the option to enable proxies for a specific portfolio, and this selection is respected by the portfolio API without having to make any change.

It does this by using the company_api_with_proxies root element for scores which is also available directly. There is also an equivalent company_api_v2_with_proxies for providing finer grain detail - and can be used to get proxy coveraage

Note that there are a lot of companies potentially returned from these elements as we create a proxy for every ISIN issuer for which we know a sector. This is at least an order of magnitude more than covered companies.

These are scoring APIs and like all of the other scoring APIs must be called with a where clause of a profile_id as otherwise it will try to compute scores for hundreds of profiles and time out.

Proxy performance values

The scaled_performance value is the average for the sector and region, and is used to provide a company-specific estimate of the performance value if the proxy company's revenue is known (i.e. we have been able to map a ticker onto a company that we have financial data for). If we do not have the revenue data for the proxy company the sector/region average performance value is returned - and is therefore not company specific. The returned metric data contains a `scaled_flag` element which is set to false in this case.

Proxies for missing data

The metric_api_with_proxies will return a proxy value a metric in an FCM company where the data is not disclosed. This is metric-level only - and does not contribute to the scoring system.

Proxies for ICM covered companies

We are unable to expose ICM companies data via the API for data licensing reasons, so for these companies we can expose proxy metric-level data. The current implementation does not give company scores so these companies are not available from the 'V1' api and is only available from the company_api_v2_with_proxies, where we can nest the metric_api_with_proxies without having to compute a company level score.

Proxies without companies

For people wanting to use proxies for other purposes and doing the allocation to companies outside of a portfolio a new root element `proxy_cube_api` is available.

This is a similar shape of API as the `company_cube_api` in that it has the same four scoring layers. It's not a history API so the record_date is actually the time now.

For an extract to your database outside integrum it's probably simplest to do this in two steps using both apis. Get all of the proxy company names, with country and sector using `company_api_with_proxies` and all of the scores with `proxy_cube_api`.

Examples are in the examples document.

Metadata

All of the metadata is available for querying via the standard GraphQL `__schema` object (and indeed is used by tools like GraphiQL as shown above to generate the explorer view – which is one of the easier ways of exploring the API and constructing queries).

Bulk Extract Examples

For use cases where you want to load the ESG data into a system of your own, we provide some bulk mechanisms. Many data providers find it more convenient to provide CSV content rather than provide a full set of content through the API. But with the use of some specialist root elements, it is perfectly possible to do this directly in GraphQL.

Reporting Oriented Flat Structure

The `company_cube_api` provides a non-nested structure of the four layers of company score

- company level

- issue
- dimension
- metric

It is denormalized into a single record structure which means that some of the elements are optional if non-releveant at that level.

It provides the same data as the dashboard scoring with the scores for the current year and the previous year.

History for the default equally weighted profile

The `company_cube_history` contains a daily dump of the `company_cube_api` for the default equally weighted profile. The scores are quite slow-moving so we only keep the start of each month and the last 30 days.

Multiple years history - company_cube_history_all_years

The `company_cube_history` element above provides the denormalized view of the same scores that are in the dashboard. In particular, we show the current year and previous year for each company as at the record date. If a company is due to report but is slower than another company they may have different current years - but they're still shown together as that the best information we have for each.

For multiple years history this makes less sense. We don't want this years reporting date to affect the score from several years ago. It is better to have scores for companies aligned for the same financial year.

So that is the scoring basis for the `company_cube_history_all_years` which has the same denormalized structure - but the peer groups are all at the same financial year - going back to 2019 if we had coverage of the company.

The coverage increased from around 2000 for financial year 2019 to over 3000 for financial year 2022.

It has a single set of entries for the financial year so it does not need a `yeardiff` element and is queried by `financial_year` rather than `record_date`.

Using a query from a system

This section shows making GraphQL requests from different language clients.

The examples will either be standalone (for running in GraphiQL) or Python, but the core query should be able to be used from any language capable of making and processing web requests.

Command line:

```
q1="
query MyQuery {
  company_api(where: {profile_id: {_eq: -1}})  {
    company_id
    country
    isin
    name
    sector_id
  }
}
"

eq=`echo "$q1" | tr '\n' ' '`


curl -H "Authorization:Api-Key XXXX" -H "Content-Type: application/json" -X POST -d "{\"query\":
\"$eq\"}" https://dashboard.integrumesg.com/graphqlapi/v1/graphql
```

Javascript:

```
/*

This is an example snippet - you should consider tailoring it
```

```
to your service.

*/

async function fetchGraphQL(operationsDoc, operationName, variables) {
  const result = await fetch("undefined", {
    method: "POST",
    headers: {'Authorization': 'Api-Key XXXX'},
    body: JSON.stringify({
      query: operationsDoc,
      variables: variables,
      operationName: operationName,
    }),
  });

  return await result.json();
}

const operationsDoc = `
  query MyQuery {
    company_api(where: { profile_id: { _eq: -1 } }) {
      name
      country
      summary {
        esg_score
        governance_score
        impact_score
        sustainability_score
      }
    }
  }
`;

function fetchMyQuery() {
  return fetchGraphQL(operationsDoc, "MyQuery", {});
}

async function startFetchMyQuery() {
  const { errors, data } = await fetchMyQuery();

  if (errors) {
    // handle those errors like a pro
    console.error(errors);
  }

  // do something great with this precious data
  console.log(data);
}

startFetchMyQuery();
```

Python:

```
import requests
import json
import pandas as pd


q2 = """
query MyQuery {
  company_api(where: { profile_id: { _eq: -1 } }) {
    company_id
```

```
    country
    current_year
    esg_score
    esg_score_diff
    governance_score
    governance_score_diff
    impact_score
    impact_score_diff
    isin
    name
    prev_esg_score
    prev_governance_score
    prev_impact_score
    prev_sentiment_score
    prev_sustainability_score
    previous_year
    profile_id
    sector_id
    sentiment_score
    sustainability_score
    sustainability_score_diff
    usd_revenue
    metric_scores {
      name
      performance_score
    }
  }
}
"""

url = 'https://dashboard.integrumesg.com/graphqlapi/v1/graphql'
r = requests.post(
    url, headers={'Authorization': 'Api-Key XXXX'}, json={'query': q2})
print(r.status_code)
json_data = json.loads(r.text)
# print(json_data)
df = pd.json_normalize(json_data, record_path=['data', 'company_api'])
print(df)
```

Java

```
package com.integrumesg.app;

/**
 * Hello world!
 *
 */

import okhttp3.MediaType;
import okhttp3.OkHttpClient;
import okhttp3.Request;
import okhttp3.RequestBody;
import okhttp3.Response;

import com.google.gson.JsonObject;
import com.google.gson.JsonParser;


public class App
{

    public static JsonObject example()
```

```
    {
        String apiUrl = "https://dashboard.integrumesg.com/graphqlapi/v1/graphql";
        String apiToken = "XXX"; // Replace with your API token or credentials

        // Define the GraphQL query
        String GraphQLQuery = "{\"query\": \"query MyQuery { " +
            "company_api(where: {name: {_eq: \\\"Alphabet\\\"}, profile_id: {_eq: -1}}) { " +
            "esg_grade esg_score sustainability_grade sustainability_score " +
            "governance_grade governance_score impact_grade impact_score " +
            "name " +
            "metric_scores { description dimension_type name performance_value " +
            "performance_score performance_grade awareness_grade awareness_score unit } " +
            "pais { pai_name text score } " +
            "company_sfdr_article { article } " +
            "}}\" " +
            "}";

        System.out.println( GraphQLQuery );

        // Create an OkHttpClient
        OkHttpClient client = new OkHttpClient();

        // Create a request body with the GraphQL query
        RequestBody requestBody = RequestBody.create(MediaType.get("application/json"),
GraphQLQuery);

        // Create the HTTP request
        Request request = new Request.Builder().url(apiUrl)
            .post(requestBody)
            .header("Authorization", "Api-Key " + apiToken) // Set your authorization header here
            .build();

        try {
            // Send the request and get the response
            Response response = client.newCall(request)
                .execute();

            // Check if the request was successful
            if (response.isSuccessful()) {
                // Parse the response JSON
                String responseBody = response.body()
                    .string();
                JsonObject jsonResponse = JsonParser.parseString(responseBody)
                    .getAsJsonObject();

                // Handle the JSON response as needed
                return jsonResponse;
            } else {
                // Handle the error
                System.err.println("Request failed with code: " + response.code());
                System.err.println(response.body()
                    .string());
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }
    public static void main( String[] args )
    {
        JsonObject resp = example();
```

```
        System.out.println( resp );
    }
}
```

Microsoft Powerquery formula language (M)

```
// This script shows how to use M language (Power Query Formula Language)
// to read data from Integrum ESG GraphQL API using a POST request.
// This can come in handy when building PowerBI reports that utilize GraphQL endpoints for loading
data.

let
    vUrl = "https://dashboard.integrumesg.com/graphqlapi/v1/graphql",
    vHeaders =[
      #"Method"="POST",
      #"Content-Type"="application/json",
      #"Authorization"="Api-Key XXX"
    ],
    // Notice the quote escaping here
    vContent=Text.ToBinary("{""query"": ""query MyQuery {
  company_api(where: {profile_id: {_eq: -1}}) {
    company_id
    country
    current_year
    esg_score
    esg_score_diff
    governance_score
    governance_score_diff
    impact_score
    impact_score_diff
    isin
    name
    prev_esg_score
    prev_governance_score
    prev_impact_score
    prev_sentiment_score
    prev_sustainability_score
    previous_year
    profile_id
    sector_id
    sentiment_score
    sustainability_score
    sustainability_score_diff
    usd_revenue
  }
}
    ""}"),
    Source = Web.Contents(vUrl, [Headers=vHeaders, Content=vContent]),
    #"JSON" = Json.Document(Source),
    data = JSON[data],
    company_api = data[company_api],
    #"Converted to Table" = Table.FromList(company_api, Splitter.SplitByNothing(), null, null,
ExtraValues.Error),
    #"Expanded Column1" = Table.ExpandRecordColumn(#"Converted to Table", "Column1", {"company_id",
"country", "current_year", "esg_score", "esg_score_diff", "governance_score",
"governance_score_diff", "impact_score", "impact_score_diff", "isin", "name", "prev_esg_score",
"prev_governance_score", "prev_impact_score", "prev_sentiment_score", "prev_sustainability_score",
"previous_year", "profile_id", "sector_id", "sentiment_score", "sustainability_score",
"sustainability_score_diff", "usd_revenue"}, {"Column1.company_id", "Column1.country",
"Column1.current_year", "Column1.esg_score", "Column1.esg_score_diff", "Column1.governance_score",
"Column1.governance_score_diff", "Column1.impact_score", "Column1.impact_score_diff", "Column1.isin",
"Column1.name", "Column1.prev_esg_score", "Column1.prev_governance_score",
```

```
"Column1.prev_impact_score", "Column1.prev_sentiment_score", "Column1.prev_sustainability_score",
"Column1.previous_year", "Column1.profile_id", "Column1.sector_id", "Column1.sentiment_score",
"Column1.sustainability_score", "Column1.sustainability_score_diff", "Column1.usd_revenue"})
in
    #"Expanded Column1"
```