General overview

Ds-Workflows
Ds-workflows (dsw) is an extension to data platform (dp). Dsw is a collection of tools (fetch, transform, deliver & manager) allowing user to define how data should be understood and transfeered between systems. Interaction with Dsw, from a user perspective, by using Dsw APIs.

Resources

Postman Collection

LucidChart

Teams / SharePoint

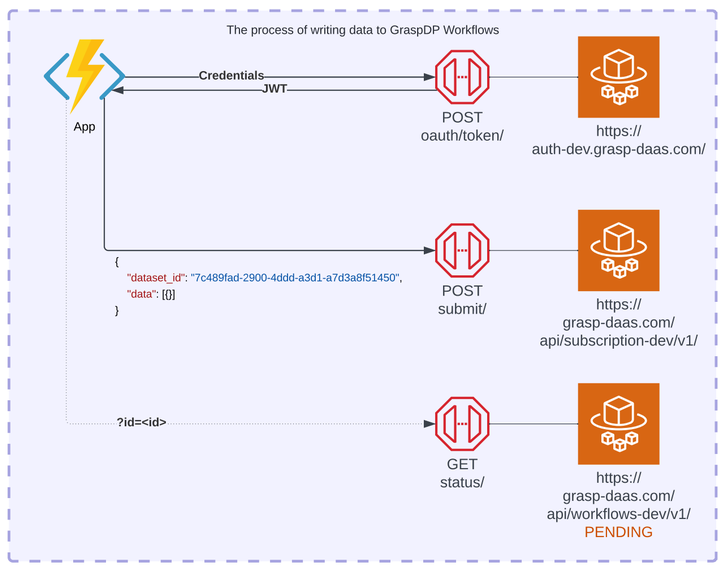
GitHub

Submid data Flow

Authorization using Oauth2.0 Credentials Flow

Credentials	
clientId	...
clientSecret	...

Credentials are shared using Tokenized url (valid 2 hours)



Data models

Customer Dataset Schemas

addCustomers (Company, Xledger)			
Requirement	**Name**	**Type**	**Description**
Mandatory	code	string	
Mandatory	companyNumber	string	
Optional	companyName	string	
Optional	description	string	
Optional	name	string	
Mandatory	address_streetAddress	string	
Mandatory	address_zipCode	string	
Mandatory	address_place	string	

addCustomers (Person, Xledger)			
Requirement	**Name**	**Type**	**Description**
Mandatory	code	string	
Mandatory	socialSec	string	
Optional	companyName	string	
Optional	description	string	
Optional	name	string	
Mandatory	address_streetAddress	string	
Mandatory	address_zipCode	string	
Mandatory	address_place	string	

InvoiceBaseItems Dataset Schemas

addInvoiceBaseItems (Xledger)			
Requirement	**Name**	**Type**	**Description**
Mandatory	subledger_code	string	
Mandatory	product_code	string	
Mandatory	amount	MoneyString	
Mandatory	glObject1_code	string	
Mandatory	glObject2_code	string	
Mandatory	glObject3_code	string	
Optional	glObject5_code	string	

Code samples

Code example

C# dotNet

```
using System;
using System.Collections.Generic;
using System.Net.Http;
using System.Text;
using System.Threading.Tasks;
using Newtonsoft.Json;

public class Program
{
    public static async Task<Dictionary<string, string>> GetBasicAuthHeaderAsync(string clientId, string clientSecret)
    {
        var userPass = $"{clientId}:{clientSecret}";
        var authString = Convert.ToBase64String(Encoding.UTF8.GetBytes(userPass));
        var authHeaders = new Dictionary<string, string>
        {
            { "Authorization", "Basic " + authString }
        };
        return authHeaders;
    }

    public static async Task<Dictionary<string, string>> GetAuthHeaderAsync(string url, string clientId, string clientSecret)
    {
        var authHeaders = await GetBasicAuthHeaderAsync(clientId, clientSecret);

        var tokenRequestData = new Dictionary<string, string>
        {
            { "grant_type", "client_credentials" },
        };

        using var client = new HttpClient();

        var request = new HttpRequestMessage(HttpMethod.Post, url)
        {
            Content = new FormUrlEncodedContent(tokenRequestData),
        };

        foreach (var header in authHeaders)
        {
            request.Headers.Add(header.Key, header.Value);
        }

        var response = await client.SendAsync(request);

        if (!response.IsSuccessStatusCode)
        {
            throw new Exception($"Failed to get JWT token: {await response.Content.ReadAsStringAsync()}");
        }

        Console.WriteLine("Authentication successful");

        var responseContent = await response.Content.ReadAsStringAsync();
        var identityPayload = JsonConvert.DeserializeObject<Dictionary<string, string>>(responseContent);

        return new Dictionary<string, string> { { "Authorization", $"Bearer {identityPayload["access_token"]}" } };
    }

    public static async Task<HttpResponseMessage> SubmitDataAsync(
        string url,
        Dictionary<string, string> headers,
        Dictionary<string, string> data)
    {
        using var client = new HttpClient();

        var jsonData = JsonConvert.SerializeObject(data);
        var content = new StringContent(jsonData, Encoding.UTF8, "application/json"); // Correct placement of 'Content-Type'

        foreach (var header in headers)
        {
            client.DefaultRequestHeaders.Add(header.Key, header.Value); // Only request headers here
        }

        var response = await client.PostAsync(url, content); // Content with correct 'Content-Type'
        return response;
    }

    public static async Task Main(string[] args)
    {
        string clientId = Environment.GetEnvironmentVariable("CLIENT_ID");
        string clientSecret = Environment.GetEnvironmentVariable("CLIENT_SECRET");
        string loginUrl = "https://auth-dev.grasp-daas.com/oauth/token/";
        string submitUrl = "https://grasp-daas.com/api/subscription-dev/v1/submit/";

        var authHeaders = await GetAuthHeaderAsync(loginUrl, clientId, clientSecret);

        var data = new Dictionary<string, string>
        {
            { "dataset_id", "12345" },
            { "data", ""}
        };

        var headers = new Dictionary<string, string>(authHeaders);

        var response = await SubmitDataAsync(submitUrl, headers, data);

        Console.WriteLine(response.StatusCode);
        Console.WriteLine(await response.Content.ReadAsStringAsync());
    }
}
```

Python Requests

```
import base64
import json
import os
import typing

import requests


def get_basic_auth_header(client_id, client_secret) -> dict:
    """
    Return a dict containing the correct headers to set to make HTTP Basic
    Auth request
    @param client_id: client_id
    @param client_secret: client_secret
    @return: dict
    """
    user_pass = "{0}:{1}".format(client_id, client_secret)
    auth_string = base64.b64encode(user_pass.encode("utf-8"))
    auth_headers = {
        "AUTHORIZATION": "Basic " + auth_string.decode("utf-8"),
    }
    return auth_headers


def get_auth_header(url: str, client_id: str, client_secret: str) -> typing.Dict:
    """
    Function returning JWT token Auth Header OAuth2 client credentials flow.
    @param url: url
    @param client_id: client_id
    @param client_secret: client_secret
    @return: str
    """
    token_request_data = {
        "grant_type": "client_credentials",
    }
    auth_headers = get_basic_auth_header(
        client_id, client_secret
    )

    response = requests.post(
        url, data=token_request_data, headers=auth_headers
    )

    if response.status_code != 200:
        raise PermissionError(f"Failed to get JWT token: {response.content}")

    print("Authentication successful")

    identity_payload = json.loads(response.content.decode("utf-8"))
    return {"Authorization": f"Bearer {identity_payload['access_token']}"}


def submit_data(
    url: str,
    headers: typing.Dict,
    data: typing.Dict,
) -> requests.Response:
    """
    Function to submit data to API.
    """
    return requests.post(url, data=json.dumps(data), headers=headers)


if __name__ == "__main__":

    """
    # When data is invalid, 422 response is returned with instruction on how
    # to fix the data
    <Response [422]> {
      'detail': [
        {
          'loc': ['body', 0],
          'msg': 'Expecting value: line 1 column 1 (char 0)', 'type': 'value_error.jsondecode',
          'ctx': {
            'msg': 'Expecting value',
            'doc': 'dataset_id=',
            'pos': 0,
            'lineno': 1,
            'colno': 1
          }
        }
      ]
    }

    # 404 response is returned when dataset_id is invalid (cannot be found)
    <Response [404]> {
      'detail': 'No dataset definition found: workflows/eae3b2be-f377-445e-86b0-ead33827daae/datasets/.json'
    }

    # 200 response is returned when data is successfully submitted
    <Response [200]> {
      "status": "pending",
      "dataset_id": "<uuid>",
      "session_id": "<uuid>"
    }
    """

    CLIENT_ID = os.environ.get("CLIENT_ID")
    CLIENT_SECRET = os.environ.get("CLIENT_SECRET")
    LOGIN_URL = "https://auth-dev.grasp-daas.com/oauth/token/"
    SUBMIT_URL = "https://grasp-daas.com/api/subscription-dev/v1/submit/"
    DATASET_ID = ""

    data = [{}]
    auth_headers = get_auth_header(
        url=LOGIN_URL,
        client_id=CLIENT_ID,
        client_secret=CLIENT_SECRET,
    )

    extra_headers = {"Content-Type": "application/json"}

    headers = {**auth_headers, **extra_headers}

    payload = {"dataset_id": DATASET_ID, "data": data}

    response = submit_data(
        url=SUBMIT_URL,
        headers=headers,
        data=payload,
    )
    print(response, response.json())
```