Kofax ReadSoft Invoices

Invoices Web API Development Guide

Version: 6.0.0

Date: 2018-04-26

KOFAX

Legal Notice

Table of Contents

Chapter 1

Introduction

This document is not comprehensive and does not cover all of the possibilities available with the Web API. Its purpose is to provide some examples to get you started, and all of the query examples are basically written for the Graphi QL IDE included in the Web API.

The URL to the in-browser IDE for GraphQL is `http://localhost:54050/api/graphql` (replace local host with your web API hostname or IP address). The IDE includes code completion and a documentation explorer that shows everything you can do with the API.

For a more comprehensive tutorial on GrapQL, visit `http://graphql.org/learn/`.

Related documentation

A full set of the documentation for Kofax ReadSoft Invoices 6.0.0 can be found online here: https://docshield.kofax.com/Portal/Products/en_US/RSI/6.0.0-gv9m3oh6jr/RS_Invoices.htm

The ReadSoft Invoices Web API Installation Guide (`ReadSoft_Invoices_WebAPIInstall.pdf`) provides specific information on how to install and configure the Web API application.

Getting help for Kofax products

Kofax regularly updates the Kofax Support site with the latest information about Kofax products.

To access some resources, you must have a valid Support Agreement with an authorized Kofax Reseller/Partner or with Kofax directly.

Use the tools that Kofax provides for researching and identifying issues. For example, use the Kofax Support site to search for answers about messages, keywords, and product issues. To access the Kofax Support page, go to www.kofax.com/support.

The Kofax Support page provides:

- Product information and release news
  Click a product family, select a product, and select a version number.
- Downloadable product documentation
  Click a product family, select a product, and click **Documentation**.
- Access to product knowledge bases
  Click **Knowledge Base**.
- Access to the Kofax Customer Portal (for eligible customers)

Click **Account Management** and log in.

To optimize your use of the portal, go to the Kofax Customer Portal login page and click the link to open the *Guide to the Kofax Support Portal*. This guide describes how to access the support site, what to do before contacting the support team, how to open a new case or view an open case, and what information to collect before opening a case.

- Access to support tools

  Click **Tools** and select the tool to use.

- Information about the support commitment for Kofax products

  Click **Support Details** and select **Kofax Support Commitment**.

Use these tools to find answers to questions that you have, to learn about new functionality, and to research possible solutions to current issues.

Chapter 2

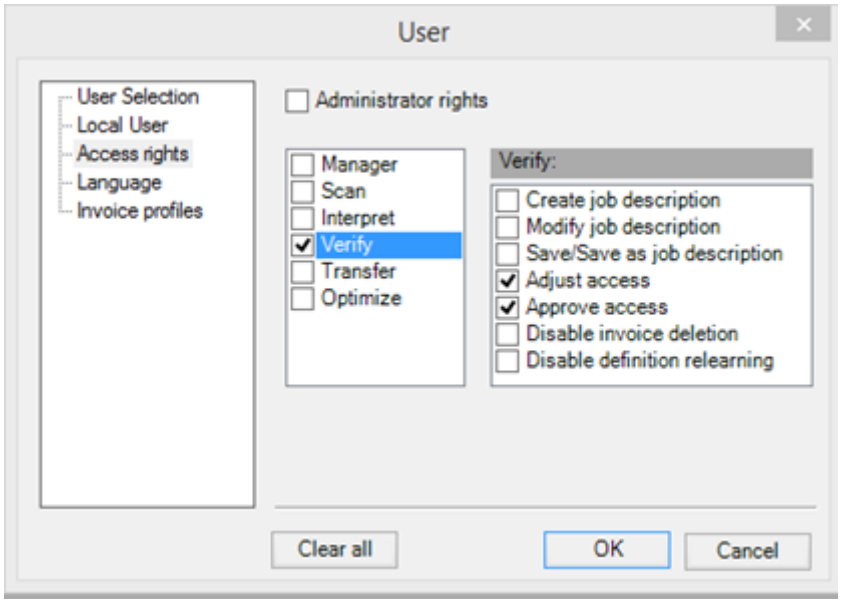
Requirements

A working Kofax ReadSoft Invoices 6.0.0 environment with Kofax ReadSoft Invoices Web API installed and running.

Authorization

A valid access token is required in the header of the http POST request when making calls to the API to ensure that only authorized users can access it. Users receive the token when they successfully log into the GraphQL IDE. In the process, the IDE contacts the IdentityServer to validate the user name and password, and if authentication succeeds, the token is sent and automatically used when the user sends queries and mutations to the API.

The IdentityServer only accepts users that are found in the ReadSoft Invoices database that is connected to the system. Note that users need Verify + adjust + approve permission in ReadSoft Invoices to be able to use the API.



When you create your own client code to access the API, you need to call the IdentityServer to get the token. Following is an example of what the raw post request would look like when calling an IdentityServer end-point at localhost:2101 for the user, *my_user*, and password, *my_user*:

```
POST /connect/token HTTP/1.1 Host: localhost:2101 Content-Type:
 application/x-www-form-urlencoded Cache-Control: no-cache
 client_id=webverify&client_secret=secret&grant_type=password&username=my_user&password=my_user
```

The returned token can then be used in the header of the POST requests made to the API. If the API receives an invalid token, it returns a 401 http error.

The user name included with the token determines which ReadSoft Invoices inbox folders the API can access for that user (if authorization roles are used in the inbox configuration in ReadSoft Invoices).

Note Due to security reasons, the default password ("secret") for the admin user in ReadSoft Invoices is not allowed for the admin user in the API. If you want to use the admin user with the API, you need to change the default password in ReadSoft Invoices Manager first.

Example token

Below is an example of a GraphQL POST request with an Authorization header. The string after "bearer " is an example of the access token you get from the IdentityServer after a successful login.

```
POST /api/graphql HTTP/1.1
Host: localhost:54050
Authorization: bearer
 eyJhbGciOiJSUzI1NiIsImtpZCI6IjI0MkIzQkY4QTkyQTczN0I2RUNBRkI5QjlGODEzMTA2RkM4QUQwREYiLCJ0eXAiOiJK
qGXCrz3J0TLob4gkUC_z3SGgyskeoVW610kYdhzO1quzTNDs85zaBfxT82SiKROUnm7mOSU7GNqTAnzaOdpYr41BwJuA_TQ1B
u6nyoLNY_Y0aq5vApMOy-AuUvXsc5kFw4CGJgiQMfXuNMGd7_njFsVLAA
Content-Type: application/json

{"query":"{\n  invoices {\n    allInvoices {\n      totalCount\n    }\n  }\n}\n
\n","variables":"null","operationName":null}
```

Chapter 3

Queries

GraphQL queries are written in Json, and all of the following examples can be executed from the GraphQL IDE. Queries fetch data from the server side.

List inbox / section criteria

User criteria in ReadSoft Invoices (*userCriteria* in the query) corresponds to the set of inbox folders to which the user who is currently logged in has access.

Doing a general query search in this way provides the information needed to make the search more specific as is seen in the next sections.

The following example fetches all of the user criteria specified, namely, *id*, *fullName*, and *name* for each of the folders the user has access to. The user only gets results from the folders that have been configured for them in the ReadSoft Invoices Manager module.

Query

```
{
  criteria
  {
    userCriteria {
      criteria {id, fullName, name}
    }
  }
}
```

Response

The response below shows that the current user has access to three inbox folders: *gbr_p02*, *gbr_po*, and *Uncertain separation*, and the information includes the *id* and *fullName* for each as was specified in the query.

```
{
  "data": {
    "criteria": {
      "userCriteria": {
        "criteria": [
          {
            "id": "6",
            "fullName": "Unknown buyer/gbr_p02",
            "name": "gbr_p02"
          },
          {
            "id": "7",
            "fullName": "Unknown buyer/gbr_po",
```

```
            "name": "gbr_po"
          },
          {
            "id": "22",
            "fullName": "Uncertain separation",
            "name": "Uncertain separation"
          }
        ]
      }
    }
  }
}
```

Query

The *userCriteria* query can take an argument and return the results for one specific inbox folder. This is done by adding the *criteriaId* parameter to the query.

```
{
  criteria {
    userCriteria(criteriaId: 6) {
      criteria {
        id
        name
        fullName
      }
    }
  }
}
```

Response

```
{
  "data": {
    "criteria": {
      "userCriteria": {
        "criteria": [
          {
            "id": "6",
            "name": "gbr_p02",
            "fullName": "Unknown buyer/gbr_p02"
          }
        ]
      }
    }
  }
}
```

List invoices

You can list all of the invoices the current user has access to. The invoices a user can access is configured in the Inbox configuration in the ReadSoft Invoices Manager module.

The following example returns the *id* and *filename* properties for all relevant invoices.

Query

```
{
```

```
  invoices {
    allInvoices {
      items {
        id
        fileName
      }
    }
  }
}
```

Response

```
{
  "data": {
    "invoices": {
      "allInvoices": {
        "items": [
          {
            "id": "5",
            "fileName": "axfood.tif"
          },
          {
            "id": "9",
            "fileName": "BEVEGO2.tif"
          }
        ]
      }
    }
  }
}
```

Get invoice data

A large amount of data on the invoice is accessible from the API. You can get everything, or you can get subsets.

The following example returns the *id* and *filename* properties for all relevant invoices.

Values and status

The following example shows how to get values and statuses from header and footer tables.

Query

```
{
  invoices {
    invoice(id: "546") {
      id,
      headerFields {
        name
        value
      statusStr}
      tables {
        rows {cells {
          name
          value
        }}
      }
    }
```

```
  }
}
```

Response (truncated)

```
{
  "data": {
    "invoices": {
      "invoice": {
        "id": "546_",
        "headerFields": [
          {
            "name": "Debit/Credit",
            "value": "Debit",
            "statusStr": "Complete"
          },
          {
            "name": "InvoiceNumber",
            "value": "510870",
            "statusStr": "Complete"
          }
          . . . .
        ],
        "tables": [
          {
            "rows": [
              {
                "cells": [
                  {
                    "name": "LI_OrderLine",
                    "value": ""
                  },
       . . . .
                  {
                    "name": "LI_Description",
                    "value": ""
                  }
                ]
              },
              . . . .
              {
                "cells": [
                  {
                    "name": "LI_OrderLine",
                    "value": ""
                  },
                  . . . .
                  {
                    "name": "LI_Description",
                    "value": ""
                  }
                ]
              }
                ]
              }
   . . . .
              }
            ]
          }
        ]
      }
    }
  }
}
```

Pages and OCR data

The following example returns the pages and ocr data for a single invoice (with id 546).

Query

The following examlpe shows how to get values and statuses from header and footer tables.

```
{
  invoices {
    invoice(id: "546") {
      id
      pages {
        imageFileName
        ocrWords {
          text
          rect {
            x
            y
            width
            height
          }
        }
      }
    }
  }
}
```

Response (truncated)

```
{
  "data": {
    "invoices": {
      "invoice": {
        "id": "546_",
        "pages": [
          {
            "imageFileName": "E:\\Test\\import\\Ident\\GBR-PO-A1.000",
            "ocrWords": [
              {
                "text": "ML",
                "rect": {
                  "x": 365,
                  "y": 523,
                  "width": 63,
                  "height": 34
                }
              },
              . . . .
              {
                "text": "order",
                "rect": {
                  "x": 843,
                  "y": 2733,
                  "width": 124,
                  "height": 40
                }
              }
            ]
```

```
            }
          ]
        }
      }
    }
}
```

Chapter 4

Queries with variables

Instead of passing your arguments in the query and modifying your queries every time an argument needs to change, you can use variables instead and keep your predefined query intact.

Query

```
query QueryName($id: Int) {
  criteria {
    userCriteria(criteriaId: $id) {
      criteria {
        id
        name
        fullName
      }
    }
  }
}
```

Variables

```
{"id": 6}
```

Request payload

This is what the actual JSON request payload looks like:

```
{"query":"\nquery QueryName($id: Int) {\n criteria {\n userCriteria(criteriaId: $id)
 {\n criteria {\n id\n name\n fullName\n }\n
}\n }\n}\n","variables":{"id":6},"operationName":"QueryName"}
```

Mutations

GraphQL mutations are used to modify the data on an invoice.

Lock invoices

Before an invoice can be modified, it must be locked by the current user so that no one else can modify it at the same time. Use the *lockInvoice* mutation to lock an invoice:

GraphQL query

```
mutation {
  lockInvoice(input: {id: "1026"}) {
    success
  }
}
```

GraphQL response

```
{
  "data": {
    "lockInvoice": {
      "success": true
    }
  }
}
```

GraphQL response

If the requested invoice is already locked by a different user, you get an error as in the following example:

```
{
  "data": {
    "lockInvoice": null
  },
  "errors": [
    {
      "message": "The INVOICES.MT.Repository.Contracts.ILockInvoice request faulted:
 INVOICES_ERROR:The invoice with id 1026 is already locked by another user"
    }
  ]
}
```

If the current user has already locked the invoice and uses *lockInvoice* on it again, the error does not appear.

Unlock invoices

When you are finished updating the invoice, you should release the lock so that other users can modify it. This is done using the *releaseInvoice* mutation:

GraphQL query

```
mutation {
  releaseInvoice(input: {id: "1026"}) {
    success
  }
}
```

GraphQL response

```
{
  "data": {
    "releaseInvoice": {
      "success": true
    }
  }
}
```

GraphQL response

If an invoice has been locked by a different user, and the current user attempts to release the lock using a query, an error occurs. In the same way, if a user attempts to release the lock on an invoice that is not locked, the error also occurs.

```
{
  "data": {
    "releaseInvoice": null
  },
  "errors": [
    {
      "message": "The INVOICES.MT.Repository.Contracts.IReleaseInvoice request faulted:
 INVOICES_ERROR:The lock for invoice 1002 could not be released"
    }
  ]
}
```

Chapter 5

Modify invoices

Use the *updateInvoice* mutation to modify an invoice. Currently, header field values, header field rectangles, and the invoice status can be modified. You can also add new user remarks.

An invoice can only be modified by the logged-in user if it is currently locked by the same user. See the *lockInvoice* mutation. When you are finished updating the invoice, release the lock using the *releaseInvoice* mutation.

To update field data on an invoice, provide the invoice *id* and a list of *updatedFields* objects. The objects in the list include the name of the field and the field properties/values that should be updated.

Update the field value of one field on an invoice

You can list all of the invoices the current user has access to. The invoices a user can access is configured in the Inbox configuration in the ReadSoft Invoices Manager module.

The following mutation updates the *InvoiceNumber* field with the new value "123456". You can specify to return a subset of the properties for the updated invoice as a response.

GraphQL mutation

```
mutation {
  updateInvoice(input: {id: "1026",
    updatedFields: [
     {fieldName: "InvoiceNumber", changedValues: [
      {key: "value", value: "123456"}
     ]}
    ]
  }
  ) {
    invoice {
      headerFields {
        name
        value
        statusStr
        infoStr
      }
    }
  }
}
```

GraphQL mutation response

```
{
  "data": {
    "updateInvoice": {
```

```
      "invoice": {
        "headerFields": [
          {
            "name": "InvoiceNumber",
            "value": "123456",
            "statusStr": "Complete",
            "infoStr": ""
          }
.
.
.
```

Validation errors

A field is validated in multiple ways when you attempt to update it. If the field receives a validation error in the process, the information is returned in the *statusStr* and *infoStr* properties.

GraphQL mutation

```
mutation {
  updateInvoice(input: {id: "1026",
    updatedFields: [{fieldName: "InvoiceNumber", changedValues: [
      {key: "value", value: "12"}
    ]}]
  }
  ) {
    invoice {
      headerFields {
        name
        value
        statusStr
        infoStr
      }
    }
  }
}
```

GraphQL mutation response

```
{
  "data": {
    "updateInvoice": {
      "invoice": {
        "headerFields": [
          {
            "name": "Debit/Credit",
            "value": "Debit",
            "statusStr": "Complete",
            "infoStr": ""
          },
          {
            "name": "InvoiceNumber",
            "value": "12",
            "statusStr": "Validation error",
            "infoStr": "Does not match format X(3-16)"
          }
.
.
.
```

Update the field value and override validation errors

If you want to set a field value to Complete even though it contains validation errors, you can use the *confirmed* key in the *updatedFields* list to override the errors:

GraphQL mutation

```
mutation {
  updateInvoice(input: {id: "1026",
    updatedFields: [{fieldName: "InvoiceNumber", changedValues: [
      {key: "value", value: "12"},
      {key: "confirmed", value:"true"}
    ]}]
  }
  ) {
    invoice {
      headerFields {
        name
        value
        statusStr
        infoStr
      }
    }
  }
}
```

GraphQL mutation response

```
{
  "data": {
    "updateInvoice": {
      "invoice": {
        "headerFields": [
          {
            "name": "InvoiceNumber",
            "value": "12",
            "statusStr": "Complete",
            "infoStr": null
          }
.
.
.
```

Update multiple fields at the same time

You can update several fields with the same mutation by adding extra objects to the *updatedFields* list. The following example updates the field values of both *InvoiceNumber* and *OrderNumber*:

GraphQL mutation

```
mutation {
  updateInvoice(input: {id: "1026",
    updatedFields: [
     {fieldName: "InvoiceNumber", changedValues: [
      {key: "value", value: "1234"}
```

```
    ]},
     {fieldName: "OrderNumber", changedValues: [
      {key: "value", value: "ABCD"}
     ]}
   ]
  }
  ) {
    invoice {
      headerFields {
        id
        value
        statusStr
        infoStr
      }
    }
  }
}
```

GraphQL mutation response

```
{
  "data": {
    "updateInvoice": {
      "invoice": {
        "headerFields": [
          {
            "id": "InvoiceNumber",
            "name": "InvoiceNumber",
            "value": "1234",
            "statusStr": "Complete",
            "infoStr": ""
          },
          {
            "id": "InvoiceOrderNumber",
            "name": "OrderNumber",
            "value": "ABCD",
            "statusStr": "Complete",
            "infoStr": ""
          },
.
.
.
```

Adjust the area defined on an invoice for a single field

This mutation updates the value rectangle defined for the *InvoiceNumber* field:

GraphQL mutation

```
mutation {
  updateInvoice(input: {id: "1026",
    updatedFields: [
    {fieldName: "InvoiceNumber", index: 0, changedValues: [
      {key: "valueRect", value: "{'x':1862,'y':997,'width':166,'height':26}"}
    ]}]
  }
  ) {
    invoice {
      headerFields {
```

```
        id
        value
        valueRect {x y width height}
        statusStr
        infoStr
      }
    }
  }
}
```

GraphQL mutation response

```
{
  "data": {
    "updateInvoice": {
      "invoice": {
        "headerFields": [
          {
            "id": "InvoiceNumber",
            "value": "1234",
            "valueRect": {
              "x": 1862,
              "y": 997,
              "width": 166,
              "height": 26
            },
            "statusStr": "Complete",
            "infoStr": ""
          },
.
.
.
```

Update the status of an invoice

Update invoice status using the *updatedStatus* property. The following list contains the valid values for this property. The number represents the status and is followed by the status name:

- 1 - Identified
- 2 - Complete
- 3 - Incomplete
- 5 - ValidationError
- 7 - Transferred
- 8 - Unidentified
- 11 - Approved
- 14 - Rejected
- 17 - Scanned

If you use an invalid value for the status, you get an error back from the mutation.

GraphQL query

```
mutation {
  updateInvoice(input: {id: "1002",
```

```
updatedStatus: 2
  }
  ) {
    invoice {
      status
      statusStr
    }
  }
}
```

GraphQL response

```
{
  "data": {
    "updateInvoice": {
      "invoice": {
        "status": 2,
        "statusStr": "Complete"
      }
    }
  }
}
```

Add user remarks to an invoice

You can add user remarks to an invoice by providing a list of messages to the *newUserRemarks* list property. The example below adds two new user remarks. Note that remarks are automatically registered with the current user and time stamp.

GraphQL query

```
mutation {
  updateInvoice(input: {id: "1026",
    newUserRemarks: [{message: "hello!"},{message: "hello again!"}]
  }
  ) {
    invoice {
      userRemarks {
        message
        user
        timeStamp
      }
    }
  }
}
```

GraphQL response

```
{
  "data": {
    "updateInvoice": {
      "invoice": {
        "userRemarks": [
          {
            "message": "hello!",
            "user": "Administrator",
            "timeStamp": "2017-10-06T06:22:08.307Z"
          },
```

```
          {
            "message": "hello again!",
            "user": "Administrator",
            "timeStamp": "2017-10-06T06:22:08.31Z"
          }
        ]
      }
    }
  }
}
```

Chapter 6

Validate invoices

The invoice is validated when it is updated or saved using the *updateInvoice* mutation. If you want to validate one or more fields without saving to the database, you can use the *validateInvoice* mutation. This is useful when validating while a user is typing field values from a thin client or for doing pre-validations before saving the changes.

An invoice does not have to be locked before calling *validateInvoice* since no changes are saved to the invoice.

For the *validateInvoice* mutation, you specify what invoice to validate and provide a list of updated field values. You can specify a response to return the results of the validation if you want to know which fields were affected by the validation, for example. Depending on the requirements of your client code, it can make sense to fetch more properties for the affected fields.

Validate a field value

GraphQL query

```
mutation
{
  validateInvoice(input:
  {
    id : "3",
    updatedFields : [{
        fieldName : "RechnungsDatum",
        changedValues : [{
            key : "value",
            value : "20170930"
}
          ]
       }
      ]
  }
  ) {
    adjustedFields { id name value formattedValue status statusStr infoStr    }
  }
}
```

GraphQL mutation response

```
{
  "data": {
    "validateInvoice": {
      "adjustedFields": [
        {
          "id": "InvoiceDate",
```

```
          "name": "RechnungsDatum",
          "value": "20170930",
          "formattedValue": "20170930",
          "status": 2,
          "statusStr": "Complete",
          "infoStr": ""
        }
      ]
    }
  }
}
```

Validate error information in responses

If field validation fails, a validation error is included in the response.

GraphQL query

```
mutation
{
  validateInvoice(input:
  {
    id : "3",
    updatedFields : [{
        fieldName : "RechnungsDatum",
        changedValues : [{
            key : "value",
            value : "NOT A VALID DATE"
}
          ]
       }
      ]
  }
  ) {
    adjustedFields { id name value formattedValue status statusStr infoStr    }
  }
}
```

GraphQL response

```
{
  "data": {
    "validateInvoice": {
      "adjustedFields": [
        {
          "id": "InvoiceDate",
          "name": "RechnungsDatum",
          "value": "NOT A VALID DATE",
          "formattedValue": "NOT A VALID DATE",
          "status": 5,
          "statusStr": "Validation error",
          "infoStr": "NOT A VALID DATE is not a valid date"
        }
      ]
    }
  }
}
```

Normalized formats

During validation, date fields and amount fields are formatted to a normalized output format.

Dates are returned with the format, YYYY-MM-DD, and amounts are returned using a period (.) as the decimal separator and no separator for thousands (1234.56, for example).

GraphQL query

In this example query, the value for an invoice number is specified as "30 sep 2017". The invoice is connected to a German/DEU profile which has DDMMMYYYY as a valid format, and as a result, the validation is correct.

```
mutation
{
  validateInvoice(input:
  {
    id : "3",
    updatedFields : [{
        fieldName : "RechnungsDatum",
        changedValues : [{
            key : "value",
            value : "30 sep 2017"
}
          ]
       }
      ]
  }
  ) {
    adjustedFields { id name value formattedValue status statusStr infoStr    }
  }
}
```

GraphQL response

In the response, you get the *formattedValue* back as 2017-09-30 since this is the date value formatted against the normalized date format. The value property still contains the original value (not formatted).

```
{
  "data": {
    "validateInvoice": {
      "adjustedFields": [
        {
          "id": "InvoiceDate",
          "name": "RechnungsDatum",
          "value": "30 sep 2017",
          "formattedValue": "2017-09-30",
          "status": 2,
          "statusStr": "Complete",
          "infoStr": ""
        }
      ]
    }
  }
}
```

Chapter 7

Get the page images for an invoice

The images for invoice pages are fetched using http GET requests and are not available via the GraphQL protocol. The image URI to fetch a specific page has the following format:

http://<API uri>/api/image/{invoiceid}_{pagenumber}

For example, the following string connects to the API located at localhost:54050 and fetches the image for page 1 of the invoice with id 3:

http://localhost:54050/api/image/3_1

The response from the request contains the image converted to PNG image file format.

The image URI is authorized using an access token. To get access, you need to specify an Authorization header in the GET request header.

A complete GET request looks like this:

```
GET /api/image/3_1 HTTP/1.1
Host: localhost:54050
Authorization: bearer
 eyJhbGciOiJSUzI1NiIsImtpZCI6IjI0MkIzQkY4QTkyQTczN0I2RUNBRkI5QjlGODEzMTA2RkM4QUQwREYiLCJ0eXAiOiJK
qGXCrz3J0TLob4gkUC_z3SGgyskeoVW610kYdhzO1quzTNDs85zaBfxT82SiKROUnm7mOSU7GNqTAnzaOdpYr41BwJuA_TQ1B
u6nyoLNY_Y0aq5vApMOy-AuUvXsc5kFw4CGJgiQMfXuNMGd7_njFsVLAA
```

The string after "bearer " is the access token received from the IdentityServer after logging in successfully.