

Televic Education

translationQ REST API

04 March 2021

Created by Sylvia Joos



Table of contents

1	Getting started	3
1.1	About the translationQ REST API	3
1.2	Your connection details	3
2	Authentication and security	4
3	Using the translationQ REST API.....	5
3.1	Requests	6
3.1.1	Query string options	6
3.2	Responses	7
3.3	Data types	9
3.4	Pagination	9
4	Typical integration workflow	10
4.1	Creation of source text	11
4.2	Ensuring all translators have a user	11
4.3	Ensuring all revisors have a user	12
4.4	Creation of a translation assignment	12
4.5	Import of translations	12
5	Exploring the translationQ REST API using Postman or curl.....	12
5.1	Postman	13
5.1.1	Initial configuration	13
5.1.2	Setting up authorization	15
5.1.3	Adding a GET API call	17
5.1.4	Adding a POST API call	21
5.1.5	Additional information	22
5.2	Curl	22
6	Contact information	23

1 Getting started

translationQ is a professional platform for revising translations. The translationQ platform provides an advanced revision tool with the possibility to store translation errors in a revision memory and reuse them while revising other translations.

translationQ also has a REST API that can be used by third-party programmers to interact with translationQ.

1.1 About the translationQ REST API


The acronym "API" stands for "Application Programming Interface". An API is a defined way for a program to accomplish a task, usually by retrieving and/or modifying data. In the case of translationQ, we provide a REST API interface for the most common actions within the application (managing users, retrieving existing roles, managing source texts, managing translation assignments, importing translations, ...).

Developers can use the translationQ REST API to create applications, websites, and other projects that interact with translationQ. Programs talk to the translationQ REST API over HTTPS, the same protocol that your browser uses to visit and interact with web pages.

The translationQ REST API has been designed based on the OData specification. OData (Open Data Protocol) is an [ISO/IEC approved, OASIS standard](#) that defines a set of best practices for building and consuming RESTful APIs. More information on OData is available through the OData website: <https://www.odata.org/>. Next to this, REST defines a set of architectural principles by which you can design Web services that focus on a system's resources, including how resource states are addressed and transferred over HTTPS by a wide range of clients written in different languages. If measured by the number of Web services that use it, REST has emerged in the last few years alone as a predominant Web service design model. In fact, REST has had such a large impact on the Web that it has mostly displaced SOAP- and WSDL-based interface design because it's a considerably simpler style to use.

It is possible to integrate with the translationQ demo or production environment. The demo environment is a staging environment and can be used to test and setup the integration.

1.2 Your connection details

 Important note: The information provided in this section is strictly personal and bound to your translationQ environment/channel. Please do not share this information.

In order to start using the translationQ REST API, the following information will be needed for accessing your channel **CHANNEL_NAME** on the translationQ **DEMO|PRODUCTION** environment.

Channel base URL =

CHANNEL BASE URL

OAuth 2.0 Access Token URL =

ACCESS TOKEN URL

API Endpoint URL =

API ENDPOINT URL

Client ID =

YOUR CLIENT ID

Client Secret =

YOUR CLIENT SECRET

Scope =

translationQApi


More details will be provided in the next sections.

2 Authentication and security

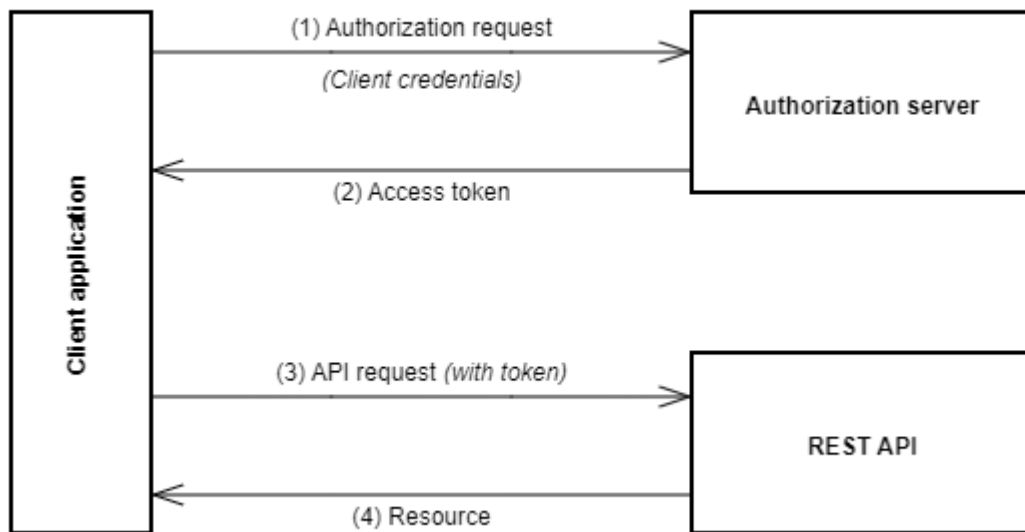
All communication is secure (HTTPS, with certificate) and all calls have to be properly authenticated using OAuth 2.0.

Therefore, prior to calling the translationQ REST API, an OAuth 2.0 access token needs to be obtained and added to each of the request headers. The OAuth 2.0 Access Token URL, Client ID, Client Secret, and Scope listed above must be used for requesting a new access token.

The translationQ REST API support the [Client Credentials Grant Flow](#) for obtaining an access token.

 The requested access token will remain valid for 10 minutes (600 seconds). After that, the token will return unusable and a new token must be requested.

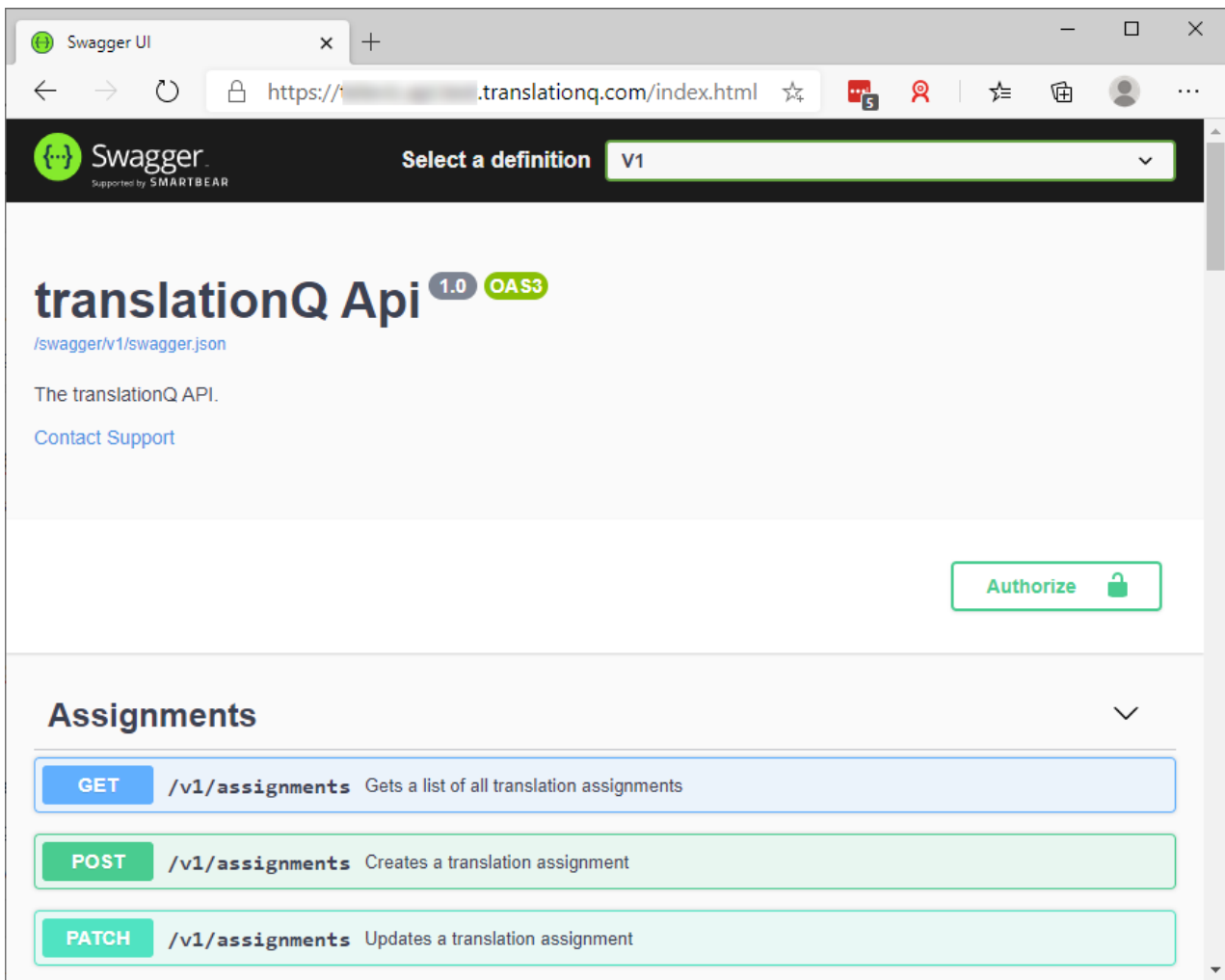
Once a valid access token is obtained, the translationQ REST API can be used. This process is depicted (on the high level) in the picture below.



⚠ The translationQ REST API is an SSL-only API (HTTPS). Therefore, all connections to the translationQ REST API must support the TLS 1.1 or TLS 1.2 protocol and the SNI extension to TLS. Please ensure your technology stack or client library supports TLS1.1/TLS 1.2 and SNI.

3 Using the translationQ REST API

A detailed overview of the available API calls can be retrieved from our automatically generated online documentation and is available by directly browsing to your <API ENDPOINT URL> url:



3.1 Requests

The translationQ REST API is a JSON-based API. Hence, data sent and received will be formatted in JSON.

All JSON properties in the API are case insensitive.

3.1.1 Query string options

A number of OData query string options are supported. These query string options can be added to the API requests and allow controlling the amount, the order or which subset of the requested information is returned.

In the API documentation, an overview is given of the supported query string options for each of the available calls. Typically, translationQ REST API requests allow following query string options: *\$select*, *\$count*, *\$filter*, *\$orderby*, *\$top*, *\$skip*. See below for a number of example queries.

```
/v1/users?$select=email,firstName
```

Will only return the fields 'email' and 'firstName' of the users

```
/v1/users?$count=true
```

Will return the list of all users, including the total number of users (can be used in the case of pagination, see later). The total number of users will be returned in the `@odata.count` property in the resulting JSON object.

```
/v1/users?$filter=email eq 'john.doe@televic.com'
```

Returns the user(s) with e-mail address equal to 'john.doe@televic.com'.

```
/v1/users?$filter=indexof(firstName, 'a') gt -1
```

Returns all the users which have the character 'a' in their first name (case insensitive).

```
/v1/users?$orderby=lastName
```

Returns all the users ordered by their last name (ascending).

```
/v1/users?$orderby=lastName desc
```

Returns all the users ordered by their last name (descending).

```
/v1/users?$top=2
```

Returns the first 2 users in the list

```
/v1/users?$skip=3
```

Skips the first 3 users in the list

Query string options can also be combined. For example, the following request will return only the fields 'email' and 'firstName' of all the users which have the character 'a' in their first name:

```
/v1/users?$select=email,firstName&$filter=indexof(firstName, 'a') gt -1
```

More information on OData query string options can also be read on the following site: <https://www.odata.org/documentation/odata-version-2-0/uri-conventions/>.

Please note that the translationQ REST API does not support all available OData query options and functions.

3.2 Responses

Each response by the translationQ REST API contains an HTTP status code indicating whether the corresponding request could be fulfilled successfully or not. In general, an HTTP status code 200 indicates the request was fulfilled successfully.

In the API documentation, an overview is provided of all possible HTTP status codes for each of the available API calls.

Typical status codes are listed in the table below.

Status code	Reason
200/201/204	<p><i>Success.</i></p> <p>The original request has been processed successfully and the resulting data is returned (if any).</p> <p>HTTP status code 200 is returned in case of a successful GET and PATCH request.</p> <p>HTTP status code 201 is returned in case of a successful POST request.</p> <p>HTTP status code 204 is returned in case of a successful DELETE request.</p>
401	<p><i>Unauthorized.</i></p> <p>The API was called using an invalid OAuth 2.0 access token.</p>
403	<p><i>Forbidden.</i></p> <p>You are trying to perform an action on a channel you have no access to.</p>
404	<p><i>Not found.</i></p> <p>The requested resource cannot be found (e.g. requesting a group with an incorrect ID).</p>
422	<p><i>Unprocessable Entity.</i></p> <p>The original request could not be processed.</p> <p>In the case of an error 422, the response body will contain a field 'reasonCode' and corresponding 'message' field describing the exact error in more details. The content and associated meaning of each 'reasonCode' depends on the API call and is listed in the online documentation.</p> <p>E.g., in the case of creating a new user, following values for the 'reasonCode' and 'message' fields can be returned in case of an HTTP status code 422:</p> <ul style="list-style-type: none"> • reasonCode: 1 message: A user with emailaddress 'email' already exists. • reasonCode: 2 message: The chosen password does not meet length requirements. A minimum length of 8 characters is required. • reasonCode: 3 message: The chosen password does not meet complexity requirements.

Status code	Reason
500	<p><i>Server error.</i></p> <p>Something went wrong processing the original request on the server, an unexpected error occurred. In the case of such error, please contact the Televic Education support desk.</p> <p><u>Note:</u> in the case of an error 500, the response body will contain a 'requestId' field. Please supply this id with your message to the support desk.</p> <p>E.g.:</p> <pre>{ "message": "Internal server error.", "requestId": "38e4fb43-3867-4445-a708-7c72107a448c" }</pre>

3.3 Data types

As the translationQ REST API is JSON-based, accepted data types are:

- string (in double quotes)
- number (integers & doubles)
- boolean (true or false)
- null
- time stamps. Time stamps use UTC time and are formatted as [ISO 8601](#) strings. For example: 2018-11-21T13:07:45.717+01:00

3.4 Pagination

When retrieving data from translationQ through the REST API, a maximum of 100 records/results will be returned. Pagination is used when the result contains more than 100 records/results. In this case, the resulting JSON object will contain the `@odata:nextLink` property. The value of this property contains the URL to the next page of the results.

For example, suppose there are 105 users in translationQ . In this case, the `/v1/users` REST API call will return the first 100 users in translationQ. The resulting JSON data will also contain the `@odata:nextLink` property indicating the URL to the next page of the result:

```
"@odata:nextLink": "<API ENDPOINT URL>/v1/users?$skip=100"
```

(in reality, `<API ENDPOINT URL>` will automatically be replaced by the value specified at the beginning of this document).

Hence, using the value of the `@odata:nextLink` property, calling the `<API ENDPOINT URL>/v1/users?$skip=100` function will return the last 5 users in translationQ. Also, the returning JSON data will now no longer contain the `@odata:nextLink` property.

Instead of using the `@odata:nextLink` property value, it is also possible to cycle through the different pages by using the `$skip` filter options in combination with the `$count` query option.

4 Typical integration workflow

In this section, we recommend a workflow for integrating with translationQ using the API into your own environment (referred to as the '3rd party environment').

Integrating with the translationQ API includes following aspects:

1. User management
2. Retrieving roles
3. Source text management
4. Translation assignment management including the import of translations

For a list of the possible API endpoints we refer to the automatically generated online documentation which is available by directly browsing to your <API ENDPOINT URL> url.

This documentation gives a lot of information about every possible API endpoint by showing or hiding the information in collapsible sections:

Users ∨

GET /v1/users Gets a list of all users

POST /v1/users Creates a user if it does not already exists

Sample request:

```
POST /v1/Users
```

When a HTTP 422 (Unprocessable Entity) occurs, the error message and reason code will indicate the cause. Following reason codes exist:

```
1: A user with emailaddress 'email' already exists.
2: The chosen password does not meet length requirements. A minimum length of 6 characters is required.
3: The chosen password does not meet complexity requirements.
```

Parameters Try it out

No parameters

Request body application/json ∨

This documentation also gives the possibility to test the API calls by clicking on "Try it out".

A typical workflow consists of the following parts:

1. Creation of source text
2. Ensuring all translators have a user
3. Ensuring all revisors have a user
4. Creation of a translation assignment
5. Import of translations

Some parts may be left out, depending on whether or not these are done manually in the application itself.

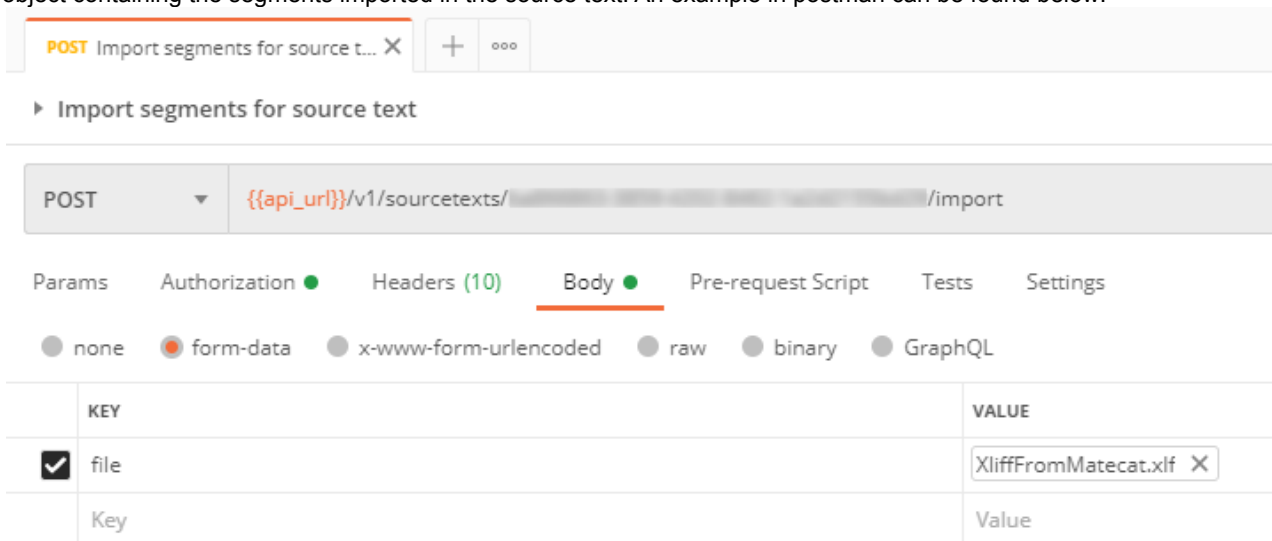
Each part will now be explained in more detail.

4.1 Creation of source text

Optionally a folder can be created first for the source text. The API endpoint **POST /v1/sourcetexts/folder** can be used for this. The result in the response will be a json object containing the identifier "id" of the folder. This "id" field is required in the next step to create the source text.

The API endpoint **POST /v1/sourcetexts** must first be used to create a source text. When creating a source text a language must be specified with the "language" field in the body. The API endpoint **GET /v1/sourcetexts/languages** can be used to retrieve the list of supported languages. Optionally a folder can be specified with the "parentId" field in the body, use the "id" from the previous step. The result in the response will be a json object containing the identifier "id" of the source text. This "id" field is required in the step to add segments to the source text and in the step to create an assignment.

Next, segments must be added to the source text with the API endpoint **POST /v1/sourcetexts/{id}/import**. The placeholder '{id}' must be replaced with the id of the source text obtained in the previous step. An .xliff or .docx file must be attached to the request in a body of type **multipart/form-data** with key **file**. The result in the response will be a json object containing the segments imported in the source text. An example in postman can be found below:



4.2 Ensuring all translators have a user

The API endpoint **GET /v1/users** should be used to verify if each translator already has a user account. An ODATA query string can be used to filter on the "email" field: `/v1/Users?$filter=email eq 'someone@example.com'`.

If a translator does not already have a user account, the API endpoint **POST /v1/users** should be used to create a user. In the body sent in the request, the role "Translator" must be specified using its identifier in the field "roleIds". To get the identifier of the "Translator" role, the API endpoint **GET /v1/roles** should be used. The field "sendMail" will cause an email to be sent upon creation or not. The result in the response will be a json object containing the identifier "id" of the translator. This "id" field is required in the step to add translators to the assignment.

4.3 Ensuring all revisors have a user

This part is identical to the previous part "Ensuring all translators have a user". Keep in mind that in this case the role "Revisor" must be specified in the body instead of the role "Translator". The identifiers of the revisors will be required in the step to create an assignment.

4.4 Creation of a translation assignment

The API endpoint **POST /v1/assignments** should be used to create an assignment. In the body sent in the request the identifier of the source text must be specified in the "sourceTextId" field and the identifiers of the revisors in the "revisorUserIds" field. The target language The result in the response will be a json object containing the identifier "id" of the assignment. This "id" field is required in the next step to add translators to the assignment.

Next, translators must be added to the assignment with the API endpoint **POST /v1/assignments/{assignmentId}/translators/{translatorId}**. The placeholder '{assignmentId}' must be replaced with the identifier of the assignment and the placeholder '{translatorId}' with the identifier of the translator.

4.5 Import of translations

The API endpoint **POST /v1/assignments/{assignmentId}/translators/{translatorId}/translation** should be used to import a translation for an assignment. The placeholder '{assignmentId}' must be replaced with the identifier of the assignment and the placeholder '{translatorId}' with the identifier of the translator. An .xliff or .docx file must be attached to the request in a body of type **multipart/form-data** with key **file**.

5 Exploring the translationQ REST API using Postman or curl

Postman and curl are two of the most widely used tools for testing APIs. With the help of Postman and curl, it is possible to explore the translationQ REST API without the need for writing any line of source code.

In this section, we will provide some basic steps on how to use Postman and curl to start experimenting with the translationQ REST API. We will detail how to obtain an OAuth2.0 access token and use this token to obtain a list of all the users in translationQ.

5.1 Postman

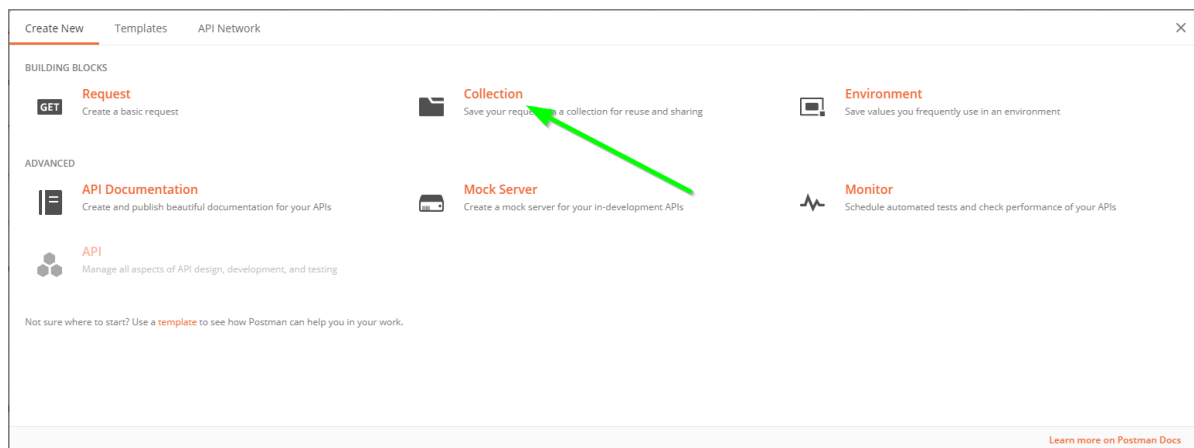
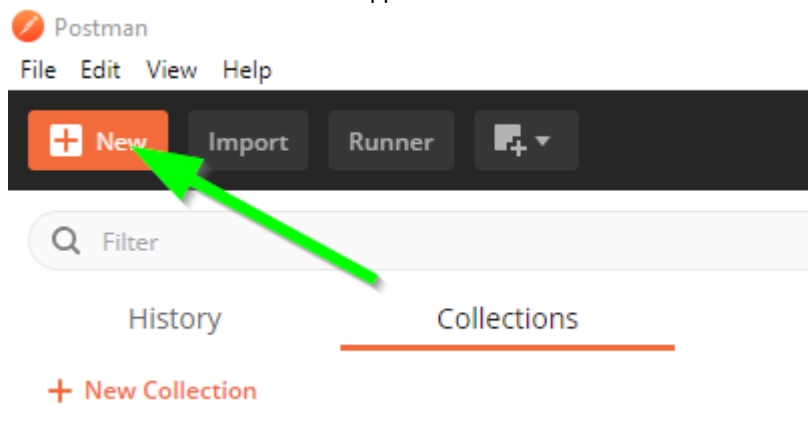
5.1.1 Initial configuration

Postman offers an intuitive UI to test and experiment with the translationQ REST API. The postman software can be downloaded directly from the postman website <https://www.postman.com/> and can be used on Windows, MacOS or Linux.

This documentation is created with version 7.30.1 of the Windows version.

Once you have installed the software, launch Postman and follow the steps below to test the translationQ REST API.

1. First you need to create a collection. This is a logical group of requests that you can organize into folders. So click the '+ New' button in the upper left corner and select 'Collection'.



2. In the 'Create a new collection' dialog, enter a name for the collection (e.g. translationQ REST API) and click "Create".

CREATE A NEW COLLECTION

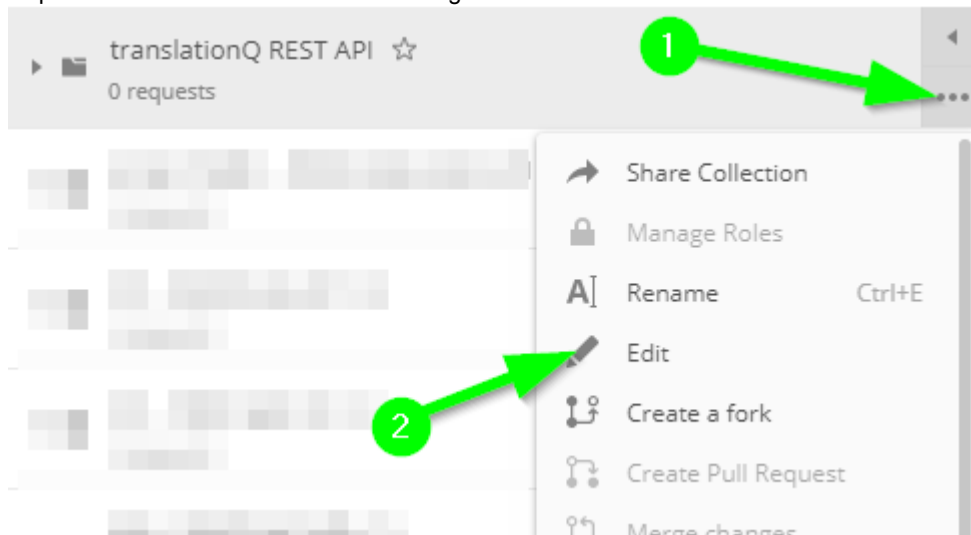
Name

[Description](#)
[Authorization](#)
[Pre-request Scripts](#)
[Tests](#)
[Variables](#)

This description will show in your collection's documentation, along with the descriptions of its f

Make things easier for your teammates with a complete collection description.

- We will now add some variables to the collection which will facilitate authentication and using the translationQ REST API.
 Edit the collection by clicking the three dots next to the newly created collection and selecting 'Edit' from the dropdown menu. the collection and clicking on "Edit".



Click the 'Variables' tab and complete the table as listed below.

EDIT COLLECTION

Name
translationQ REST API

Description Authorization Pre-request Scripts Tests **Variables**

These variables are specific to this collection and its requests. [Learn more about collection variables.](#)

	VARIABLE	INITIAL VALUE ⓘ	CURRENT VALUE ⓘ	...	Persist All	Reset All
<input checked="" type="checkbox"/>	token_url					
<input checked="" type="checkbox"/>	client_id					
<input checked="" type="checkbox"/>	client_secret					
<input checked="" type="checkbox"/>	api_url					
	Add a new variable					

Example values. Check documentation for real values

Replace the placeholder by the data received from Televic Education.

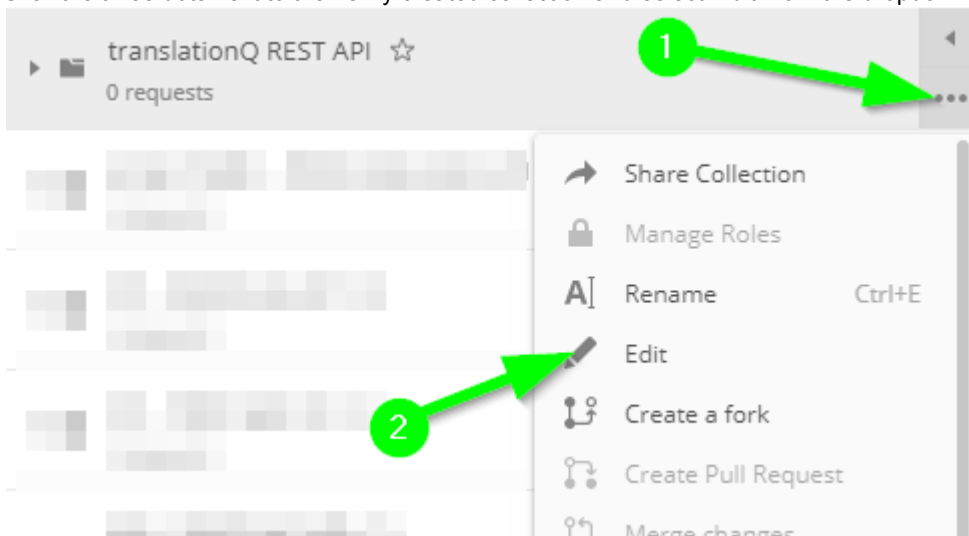
VARIABLE	INITIAL VALUE	CURRENT VALUE
token_url	<OAuth 2.0 Access Token URL>	<OAuth 2.0 Access Token URL>
client_id	<Client ID>	<Client ID>
client_secret	<Client Secret>	<Client Secret>
api_url	<API Endpoint URL>	<API Endpoint URL>

Finish by clicking "Update".

5.1.2 Setting up authorization

Now it's time to request an authorization token.

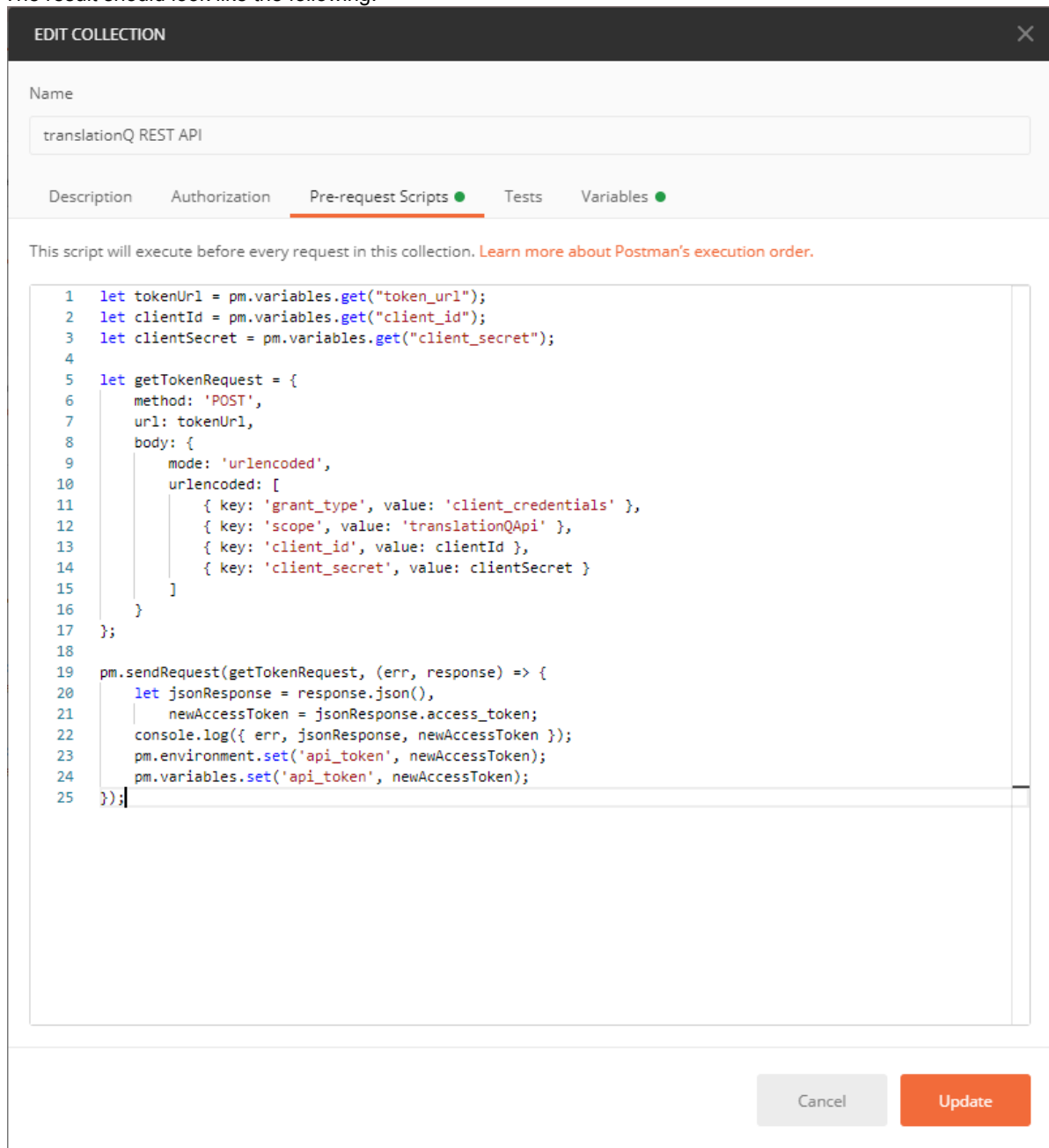
1. Click the three dots next to the newly created collection and select 'Edit' from the dropdown menu.



2. Go to the 'Pre-request Scripts tab' and copy the script below in the editor and click "Update".

```
1 let tokenUrl = pm.variables.get("token_url");
2 let clientId = pm.variables.get("client_id");
3 let clientSecret = pm.variables.get("client_secret");
4
5 let getTokenRequest = {
6   method: 'POST',
7   url: tokenUrl,
8   body: {
9     mode: 'urlencoded',
10    urlencoded: [
11      { key: 'grant_type', value: 'client_credentials' },
12      { key: 'scope', value: 'translationQApi' },
13      { key: 'client_id', value: clientId },
14      { key: 'client_secret', value: clientSecret }
15    ]
16  }
17 };
18
19 pm.sendRequest(getTokenRequest, (err, response) => {
20   let jsonResponse = response.json(),
21       newAccessToken = jsonResponse.access_token;
22   console.log({ err, jsonResponse, newAccessToken });
23   pm.environment.set('api_token', newAccessToken);
24   pm.variables.set('api_token', newAccessToken);
25 });
```


The result should look like the following:



The screenshot shows the 'EDIT COLLECTION' window in Postman. The collection name is 'translationQ REST API'. The 'Pre-request Scripts' tab is selected. A message states: 'This script will execute before every request in this collection. [Learn more about Postman's execution order.](#)'

```
1 let tokenUrl = pm.variables.get("token_url");
2 let clientId = pm.variables.get("client_id");
3 let clientSecret = pm.variables.get("client_secret");
4
5 let getTokenRequest = {
6   method: 'POST',
7   url: tokenUrl,
8   body: {
9     mode: 'urlencoded',
10    urlencoded: [
11      { key: 'grant_type', value: 'client_credentials' },
12      { key: 'scope', value: 'translationQApi' },
13      { key: 'client_id', value: clientId },
14      { key: 'client_secret', value: clientSecret }
15    ]
16  }
17 };
18
19 pm.sendRequest(getTokenRequest, (err, response) => {
20   let jsonResponse = response.json(),
21       newAccessToken = jsonResponse.access_token;
22   console.log({ err, jsonResponse, newAccessToken });
23   pm.environment.set('api_token', newAccessToken);
24   pm.variables.set('api_token', newAccessToken);
25 });
```

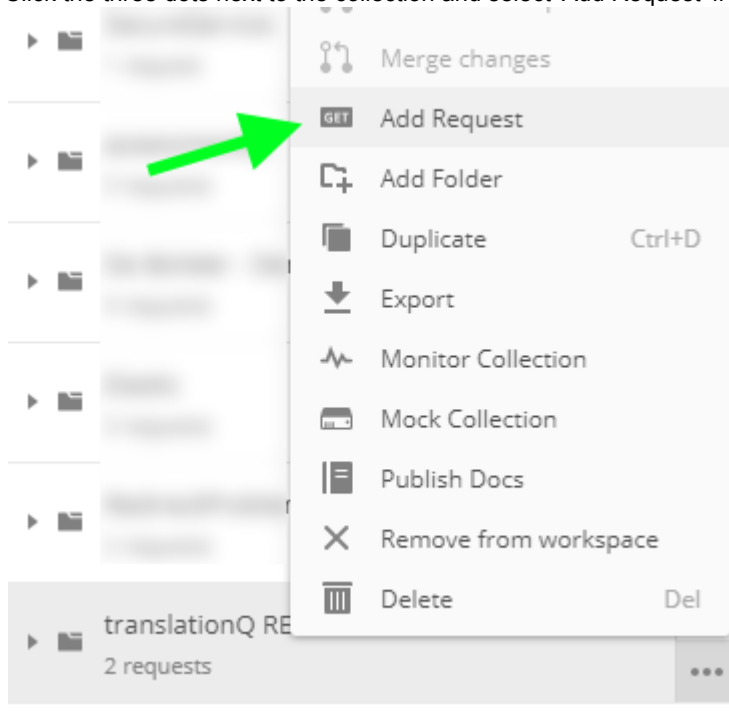
At the bottom right, there are 'Cancel' and 'Update' buttons.

Everything is now setup to add and execute an API call. In the next steps, we will demonstrate how to add an api call.

5.1.3 Adding a GET API call

Let's now add a simple GET call (e.g. /v1/users).

1. Click the three dots next to the collection and select 'Add Request' from the dropdown menu.



2. Enter a name for the request and click the 'Save to ...' button.

SAVE REQUEST

Requests in Postman are saved in collections (a group of requests).
[Learn more about creating collections](#)

Request name

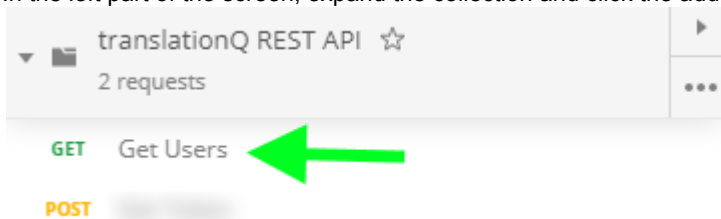
Request description (Optional)

Descriptions support [Markdown](#)

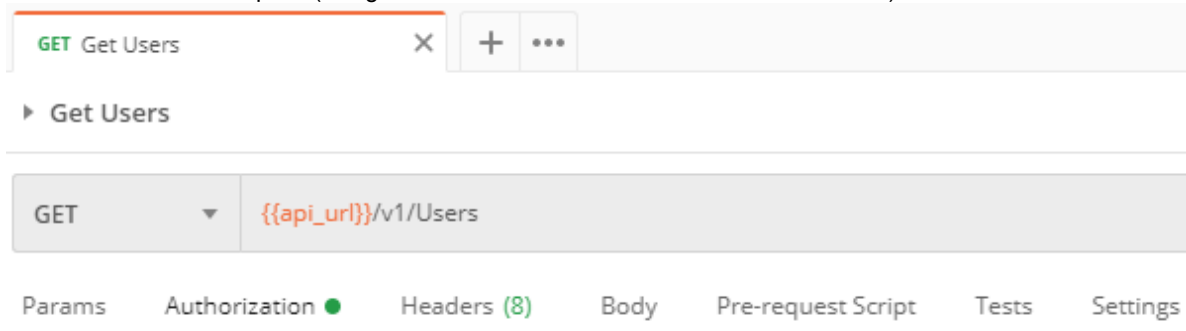
Select a collection or folder to save to:

- translationQ REST API [+ Create Folder](#)
- GET Get Users
- POST Get Token

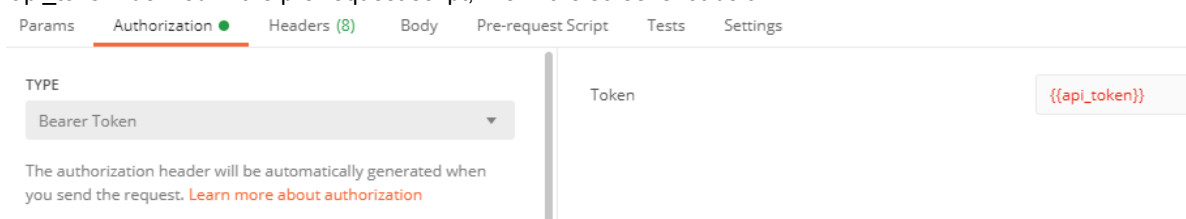
3. In the left part of the screen, expand the collection and click the added request.



- Now we can complete the details of the call in the right side of the window. Enter the URL of the request (using the variables defined in #5.1.1 of this document) like in the screenshot below.

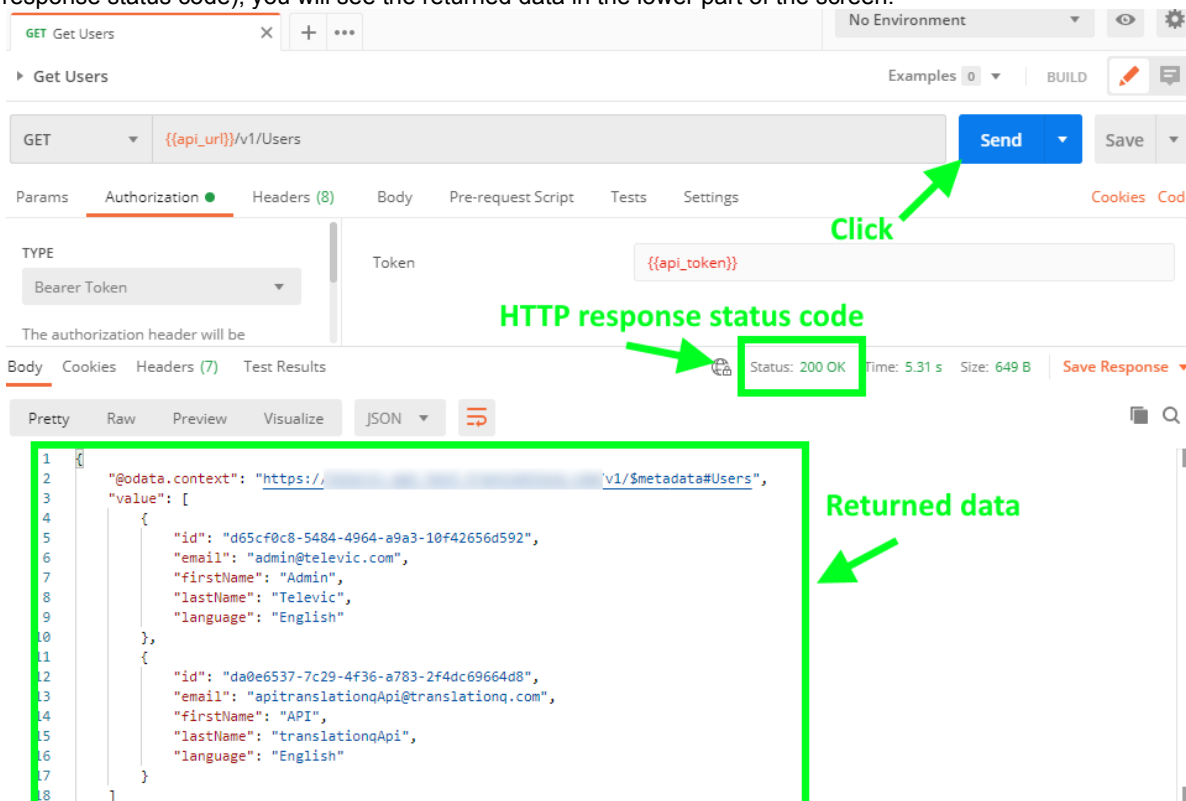


Also, ensure the Authorization type is set to 'Bearer Token' in the 'Authorization' tab and make use of the variable "api_token" defined in the pre-request script, like in the screenshot below.



Finish by clicking the 'Save' button.

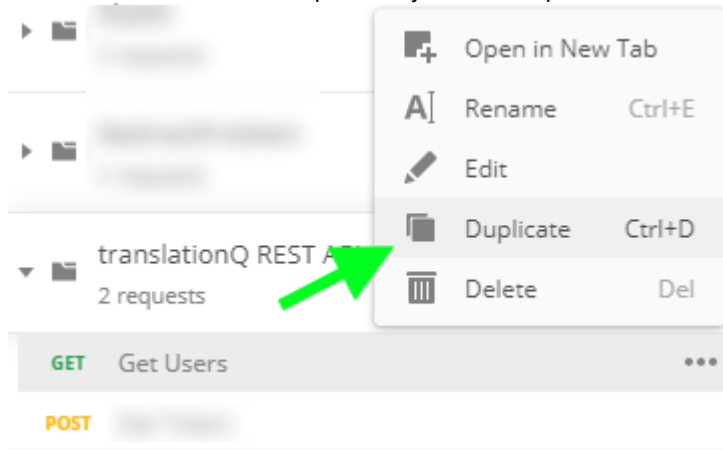
- To execute the created request, click the 'Send' button. If the request completed successfully (check the HTTP response status code), you will see the returned data in the lower part of the screen.



5.1.4 Adding a POST API call

Now we will demonstrate how to create a POST call (pushing data to the translationQ REST API).

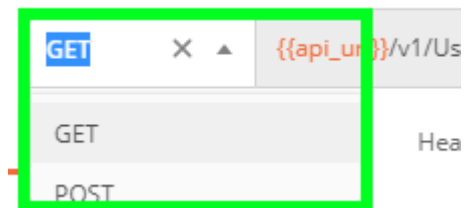
1. Click the three dots next to a previously created request and select 'Duplicate' from the dropdown menu.



2. Change the name by clicking the pencil icon next to the name



3. Change the HTTP method to "POST" by using the dropdown next to the API endpoint url:



4. Go to the tab "Body".
Pick the data type "raw" for the body, and pick "JSON" in the dropdown to the right.
Type a body, a template for the body can be found in the automatically generated online documentation for the API.
Click "Send".
Check the HTTP response status and returned data.

The screenshot shows the Postman interface for a POST request to `{{api_url}}/v1/Users`. The request body is set to 'raw' and contains the following JSON:

```

1 {
2   "email": "Marty.McFly@BackToTheFuture.com",
3   "firstName": "Marty",
4   "lastName": "McFly",
5   "password": "JenniferParker1968",
6   "language": 4,
7   "roleIds": [
8     "373982a2-fe76-4d96-9f88-10f42656d58e"
9   ],
10  "sendMail": false
11 }

```

The response status is `200 OK`. The response body is JSON and contains the following data:

```

1 {
2   "@odata.context": "https://.../v1/$metadata#Users/$entity",
3   "id": "7b2a46a7-ac51-40ce-ae0-c5073a9cef03",
4   "email": "Marty.McFly@BackToTheFuture.com",
5   "firstName": "Marty",
6   "lastName": "McFly",
7   "language": "English"
8 }

```

Annotations in the image indicate the following steps:

- pick "raw"
- pick "JSON"
- type a body
- click Send
- check response status
- check returned data

5.1.5 Additional information

Note that the authentication token remains valid for 10 minutes. Because the request to retrieve an authentication token is stored in the pre-request script of each postman request, this is handled automatically.

For more information on Postman, please check their [Learning Center](#).

5.2 Curl

Curl is a widely used tool on linux to perform API calls. The commands below are tested on Linux. Curl can also be installed on Windows though, or the Windows Subsystem for Linux can be used on Windows to emulate a linux terminal.

To obtain an access token using curl, executing the following command:

```
curl --request POST --data "grant_type=client_credentials&client_id=<YOUR CLIENT ID>&client_secret=<YOUR CLIENT SECRET>&scope=translationQApi" <ACCESS TOKEN URL>
```

Replace `<YOUR CLIENT ID>`, `<YOUR CLIENT SECRET>`, and `<ACCESS TOKEN URL>` by the information provided at the beginning of this document.

If the call completes successfully, an access token will be returned:

```
{
  "access_token": "xyzxyz",
  "expires_in": 600,
  "token_type": "Bearer",
  "scope": "translationQApi"
}
```

The value of the "access_token" property contains the access token that needs to be submitted to the API requests. Note that for security reasons, we changed the access token in the example above. In reality, the access token value will be much longer.

Once the access token has been copied, it can be supplied to the API call for, for example, querying the list of users known in translationQ:

```
curl --request GET --header "Authorization: Bearer <ACCESS TOKEN>" <API ENDPOINT URL>/v1/users
```

Replace <ACCESS TOKEN> by the token returned by the previous command (in this example case xyzxyz) and substitute <API ENDPOINT URL> using the information provided at the beginning of this document.

If the call completes successfully, a JSON formatted string is returned like the following:

```
{
  "@odata.context": "<API ENDPOINT URL>/v1/$metadata#Users",
  "value": [
    {
      "id": "d65cf0c8-5484-4964-a9a3-10f42656d592",
      "email": "admin@televic.com",
      "firstName": "Admin",
      "lastName": "Televic",
      "language": "English"
    },
    {
      "id": "da0e6537-7c29-4f36-a783-2f4dc69664d8",
      "email": "apitranslationqApi@translationq.com",
      "firstName": "API",
      "lastName": "translationqApi",
      "language": "English"
    }
  ]
}
```

6 Contact information

For any further information on the translationQ REST API or in case additional support is needed, please contact the Televic Education helpdesk via support@televic-education.com or +32 51 79 14 98.