

Semantix



Semantix[®]

All about data

Table of contents:

- [Automation](#)
 - [What is Automation?](#)
- [Low code](#)
 - [Access to the responses of the steps action, code block and repeat](#)
- [Trigger Event](#)
 - [Action \(API REST, Database ou Webservice SOAP\)](#)
 - [Webhook](#)
- [Action](#)
- [Conditional](#)
- [Repeat](#)
- [Code Block](#)
- [Stop](#)
- [Trace](#)
- [Dictionary](#)
 - [Get](#)
 - [Set](#)
- [High code](#)
- [High code - Creating a High Code](#)
 - [Creating a High Code](#)
- [High code - Development](#)
- [Trigger](#)
 - [Trigger Properties](#)
- [Templates - Overview](#)
 - [Intro](#)
 - [Creating a New Template](#)
- [Templates - Using a Template](#)
 - [Intro](#)
 - [Using the Template](#)
 - [Review Information](#)
 - [Components](#)
 - [Automation Name](#)
 - [Credentials](#)
 - [Triggers](#)
- [About Packages](#)
- [Package - Private Component](#)
 - [Creating a Private Component](#)
 - [New Component and Component Info](#)
 - [Authentication](#)
 - [Global Settings](#)
 - [Resources](#)
 - [Response](#)
 - [Code](#)
- [Package - REST](#)

- [Package - REST](#)
 - [Creating a REST Component](#)
 - [Information](#)
 - [Authentication](#)
 - [Global Settings](#)
 - [Resources](#)
- [Package - MongoDB](#)
 - [Creating a MongoDB Component](#)
- [Package - SOAP](#)
 - [Creating a MongoDB Component](#)
 - [Information](#)
 - [Authentication](#)
 - [Resources](#)
- [Package - SQL Database](#)
 - [Creating a SQL Database component](#)
 - [Information](#)
 - [Authentication](#)
 - [Resources](#)
 - ["General" tab](#)
 - ["Script" tab](#)
 - ["Response" tab](#)
 - ["Code" tab](#)
- [About API Gateways](#)
 - [Intro](#)
- [API Gateways - Global Settings](#)
 - [Intro](#)
 - [General](#)
 - [Global flow](#)
- [Plugin - About](#)
- [Plugin - Rate Limit](#)
 - [Intro](#)
 - [Creating a Rate Limit](#)
- [Plugin - Schema Validator](#)
 - [Intro](#)
 - [Creating a Schema Validator](#)
- [Plugin - IP Restriction](#)
 - [Intro](#)
 - [Creating an IP Restriction](#)
- [Plugin - Cache](#)
 - [Intro](#)
 - [Creating a Cache](#)
- [Monetization - API Gateways](#)
 - [Intro](#)
 - [Creating a Monetization](#)
 - [Details about your monetizations](#)
- [API Gateway - Resources](#)

- [API Gateways - Resources](#)
 - [How to create a resource](#)
 - ["General" tab](#)
 - ["Request" tab](#)
 - ["Response" tab](#)
 - ["Flow" tab](#)
 - ["Test" tab](#)
- [API Gateways - Documentation](#)
- [Developer Portal - About](#)
 - [Intro](#)
- [Developer Portal - Settings](#)
 - [Intro](#)
- [Developer Portal - Permissions](#)
 - [Intro](#)
 - [Waiting for Approval](#)
 - [Approved Users](#)
- [Developer Portal - Login](#)
 - [Intro](#)
 - [Request access](#)
 - [Request a password reset](#)
- [Developer Portal - Documentation](#)
 - [Intro](#)
 - [Authentication](#)
 - [Endpoint](#)
- [API Gateways - Granting API Access](#)
- [API Gateways - Api Key Authentication](#)
- [API Gateways - OAuth2 Authentication](#)
- [API Gateways - Basic Authentication](#)
- [API Gateways - None Authentication](#)
- [Management - Client](#)
 - [Intro](#)
 - [Authentication](#)
 - [Credentials](#)
- [Management - Tenant](#)
 - [Intro](#)
 - [Creating a Tenant](#)
- [Management - Overview](#)
 - [Intro](#)
 - [Creating a credential](#)
- [Management - OAuth2 Credentials](#)
 - [Intro](#)
 - [Obtaining the Access Token](#)
 - [Using Custom Names](#)
 - [Using Default Values](#)
- [Management - Dictionary](#)
 - [Intro](#)

- [intro](#)
- [Creating and editing a dictionary group](#)
- [Visualizing and managing group keys](#)
- [Management - Users](#)
 - [Intro](#)
 - [Creating an user](#)
- [SDK - Parser](#)
 - [JSON](#)
 - [XML](#)
 - [CSV](#)
- [SDK - Logger](#)
 - [Log](#)
- [SDK - Component](#)
- [SDK - Dictionary](#)
 - [Get](#)
 - [Set](#)
- [SDK - MongoDB](#)
- [SDK - MySQL](#)
- [SDK - SQL Server](#)
- [SDK - Oracle DB](#)
- [SDK - Redshift](#)
- [SDK - PostgreSQL](#)
- [SDK - SOAP](#)
- [SDK - Request](#)
 - [Simple HTTP call](#)
- [SDK - Parallel](#)
- [SDK - FTP](#)
 - [FTP](#)
- [SDK - Third Party Libs](#)
 - [lodash](#)
 - [axios](#)
 - [decimal.js](#)
 - [IMPORTANT](#)
- [CLI - Getting Started](#)
 - [About our CLI](#)
 - [Setup](#)
 - [How to get the latest version?](#)
- [CLI - Projects](#)
 - [Create a project](#)
 - [Clone a project](#)
 - [Manage dependencies](#)
 - [Delete a project](#)
 - [Generate template files](#)
 - [Save changes](#)
 - [Discard changes](#)
 - [Publish changes](#)

- Publish changes
- Sync Files
- Declare Project as Webhook
- CLI - Triggers
 - Create a trigger
 - Start a trigger
 - Delete a trigger
- IP whitelists
- Creating a ticket request inside LinkApi
- Support tickets SLA

Automation

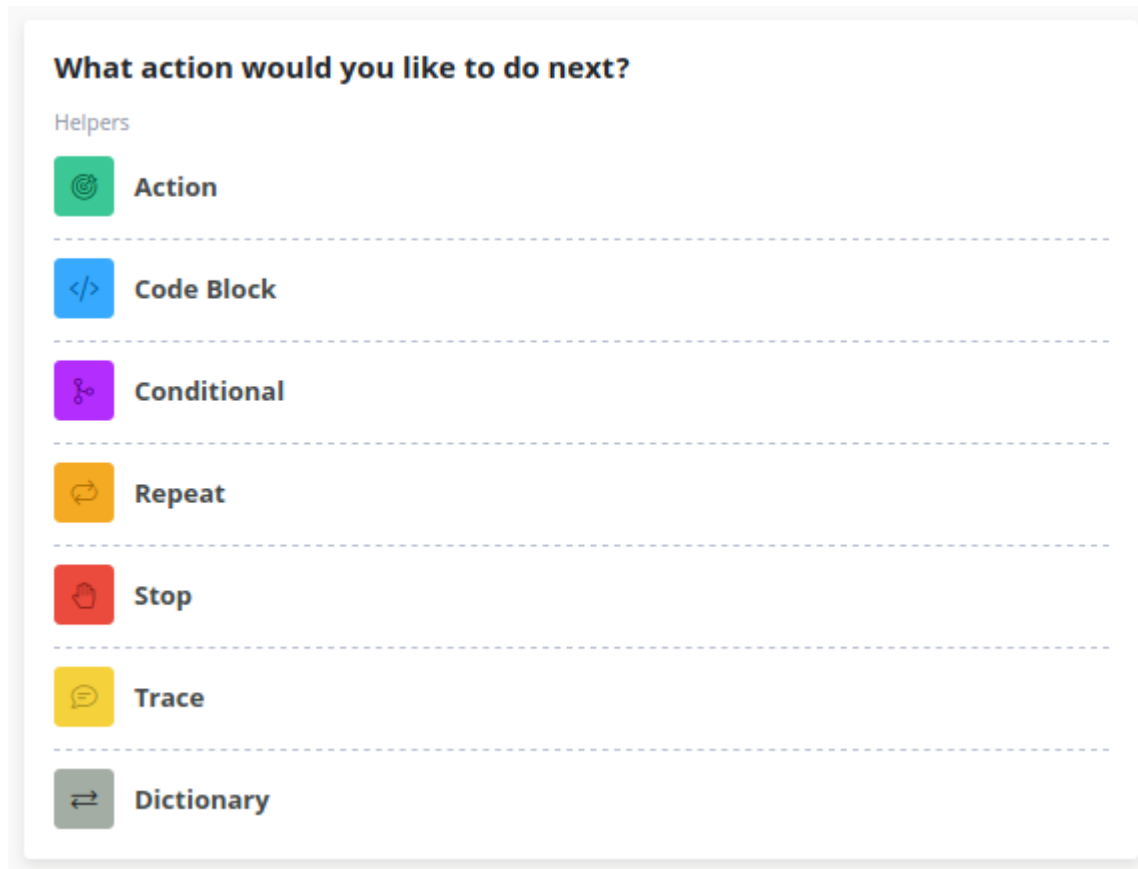
What is Automation?

A simple API integration that will run on scheduled Jobs.

Low code

Low code automations aim to build integrations through fewer code interactions and more screen configurations.

To build your automation, Low code has the following configurations:



Action

Run a call to a REST, SOAP, or DATABASE component.

Code Block

Perform some treatments with JS code in the flow

Conditional

Perform a condition in the flow

Repeat

Iterate through some flow data

Stop

Stop automation

Trace

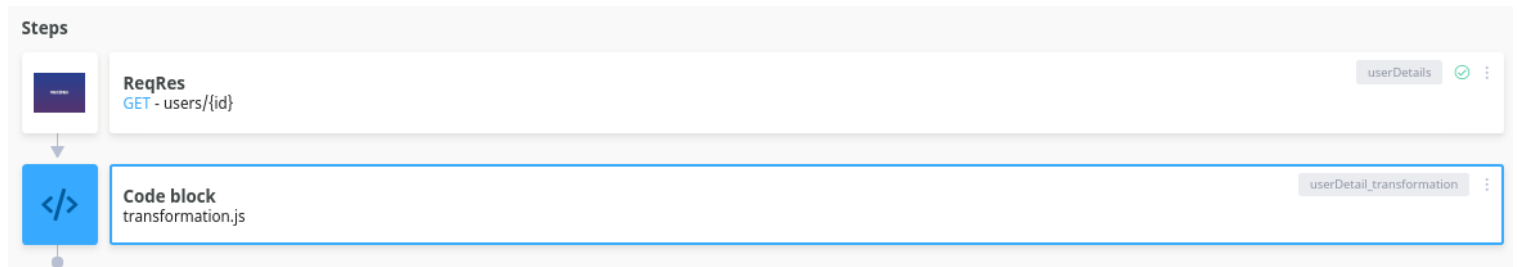
Log some step of the flow

Dictionary

Store and retrieve values

Access to the responses of the steps action, code block and repeat

Example: Suppose you have an action that searches for an user by id, then you will use an action that you named as userDetails. After this search you need a code to transform userDetails into a new property, then you add a code block step and to get the data that came in userDetails in the code block function you will receive a data that inside this object will have a property userDetails with the information from the previous action.



Code Block

Step name

userDetail_transformation

Code

File name *

userDetail_transformation.js



File extension: .js

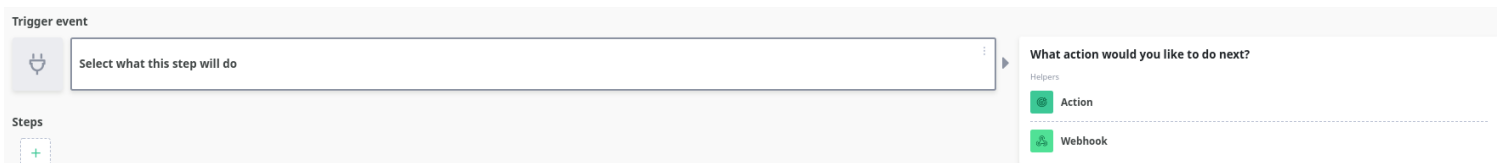
Expand code

```
1 module.exports = async (data) => {  
2   const userDetails = data.userDetails;  
3  
4 };
```

Finish

Trigger Event

This configuration has the objective of selecting a trigger to start the Automation.



The screenshot shows a configuration interface for a trigger event. On the left, there is a section labeled "Trigger event" with a dropdown menu containing the text "Select what this step will do". Below this is a "Steps" section with a plus sign icon. On the right, there is a section titled "What action would you like to do next?" with a "Helpers" sub-section. Under "Helpers", there are two options: "Action" and "Webhook", each with a green square icon.

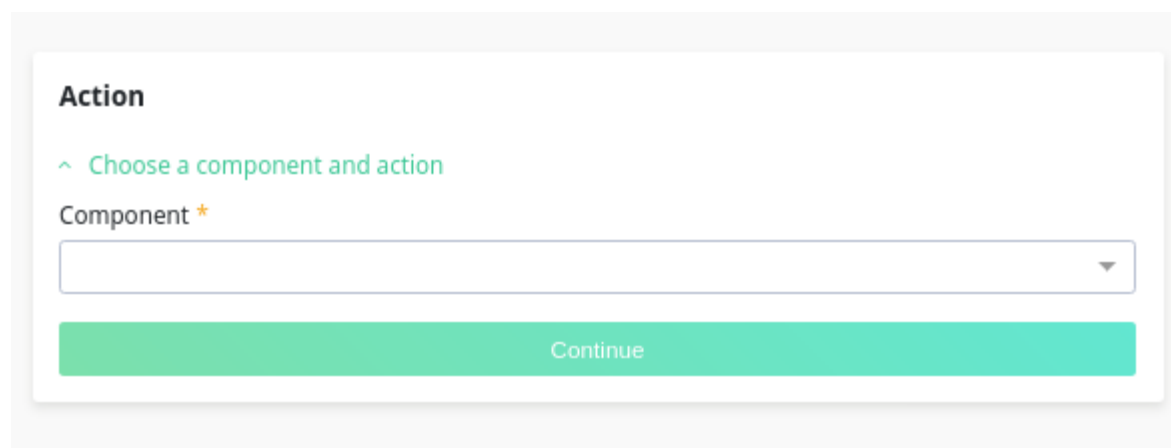
To start an automation, a trigger is needed. This can be a search in an API, DATABASE or Webhook, it can also be through a Webhook notification.

Action (API REST, Database ou Webservice SOAP)



Action

First select a component



The screenshot shows the "Action" configuration interface. It has a title "Action" and a sub-section "Choose a component and action". Below this is a "Component" label with an asterisk, followed by a dropdown menu. At the bottom, there is a green "Continue" button.

Choose an action

Action

^ Choose a component and action

Component *

Microsoft SQL Server ▼

Script *

OBRCAD
delete
PRODUCTS
select updated products

Fill out the selected component credentials fields

Action

∨ Choose a component and action ✓

∧ Authentication ✓

LinkApi needs access to your account.

server *

user *

password *

database *

port *

LinkApi will not keep this info neither modify any original API data but we need authentication to search for your endpoints.

[Continue](#)

Now you can set the general settings.

Result Path

All trigger action must return a list. However if the action response not return a list you can use the result path to indicate the field that has the list.

Example: Suppose that in the resource response the list is inside the data property, so the result path becomes data

Action

✓ Choose a component and action ✓

✓ Authentication ✓

^ General settings

Result path (Optional)

Allow bulk data



Continue

Allow bulk data

Instead of rotating each item in the list in the steps you can choose to rotate the complete list.

And lastly you can pass some configurations to execute this resource.

Action

▼ Choose a component and action ✓

▼ Authentication ✓

▼ General settings

^ Data transformation

Would you like to see request and response samples of the current or previous steps?

```
Expand code ↗  
1  module.exports = async (data) => {  
2    return {  
3      headers: {},  
4      body: {},  
5      queryString: {},  
6      urlParams: {}  
7    };  
8  };
```

Finish

Webhook

Select Webhook option.

 Webhook

First you need inform a payload that will be received at webhook request.

Webhook

^ JSON

```
1 {  
2   "productId": "231",  
3   "stock": 5,  
4   "price": 20.99|  
5 }
```

Convert JSON and continue

Now you can set the general settings.

Unique Keys

In case you want to assign a unique key to any payload key just select it as uniqueKey.

Webhook

^ JSON

^ General settings

Unique keys

productId
stock
price

Allow bulk data

Instead of rotating each item in the list in the steps you can choose to rotate the complete list.

Action

This step is responsible for making calls to the REST, SOAP and DATABASE components packages built on the LinkApi platform.



First you must select a private or prebuilt component.

A screenshot of a web interface titled "Action". Below the title is a green link that says "Choose a component and action" with a small upward-pointing arrow. Underneath is a label "Component *" followed by a dropdown menu. The dropdown menu is open, showing a list of options: "Private", "CodeK.Connector", "ReqRes", "RabbitMQ", "MySQL", "MongoDB", and "Rest". The "CodeK.Connector" option is currently selected and highlighted with a light blue background.

After choose a component you need select a resource.

Action

^ Choose a component and action

Component *

MongoDB

Resource *

Resource

customers

findOne - customers/

find - customers/

insertMany - customers/

insertOne - customers/

deleteMany - customers/

updateOne - customers/

Now you need to fill in the credential fields for the selected component

Action

^ Choose a component and action



^ Authentication



LinkApi needs access to your account.

uri *

mongodb://<user>:<password>@<host>:27017/<database>

LinkApi will not keep this info neither modify any original API data but we need authentication to search for your endpoints.

Save

^ Step name

To save the answer to the action step, you need to give it a name.

Action

✓ Choose a component and action



✓ Authentication



^ Step name

Declare a name to this step to have access to the output in other steps.

Step name *

Save and continue

To finish the configuration of the step action, it is necessary to inform the settings that will be used in the consumption of the resource

Action

✓ Choose a component and action

✓ Authentication

Step name

customers

^ Data transformation

Would you like to see request and response samples of the current or previous steps?

```
Expand code ↗  
1  module.exports = async (data) => {  
2    return {  
3      headers: {},  
4      body: {},  
5      query: {},  
6      queryValues: {},  
7      queryString: {},  
8      urlParams: {}  
9    };  
10 };
```

Finish

Remember if you need to access the result of this step in the steps below, just use the variable 'data.stepName'

```
module.exports = async (data) => {  
  
  const customersData = data.customers;  
  
  return {  
    headers: {},  
    body: {  
      customers: customersData  
    },  
    query: {},  
    queryValues: {},  
  };  
};
```

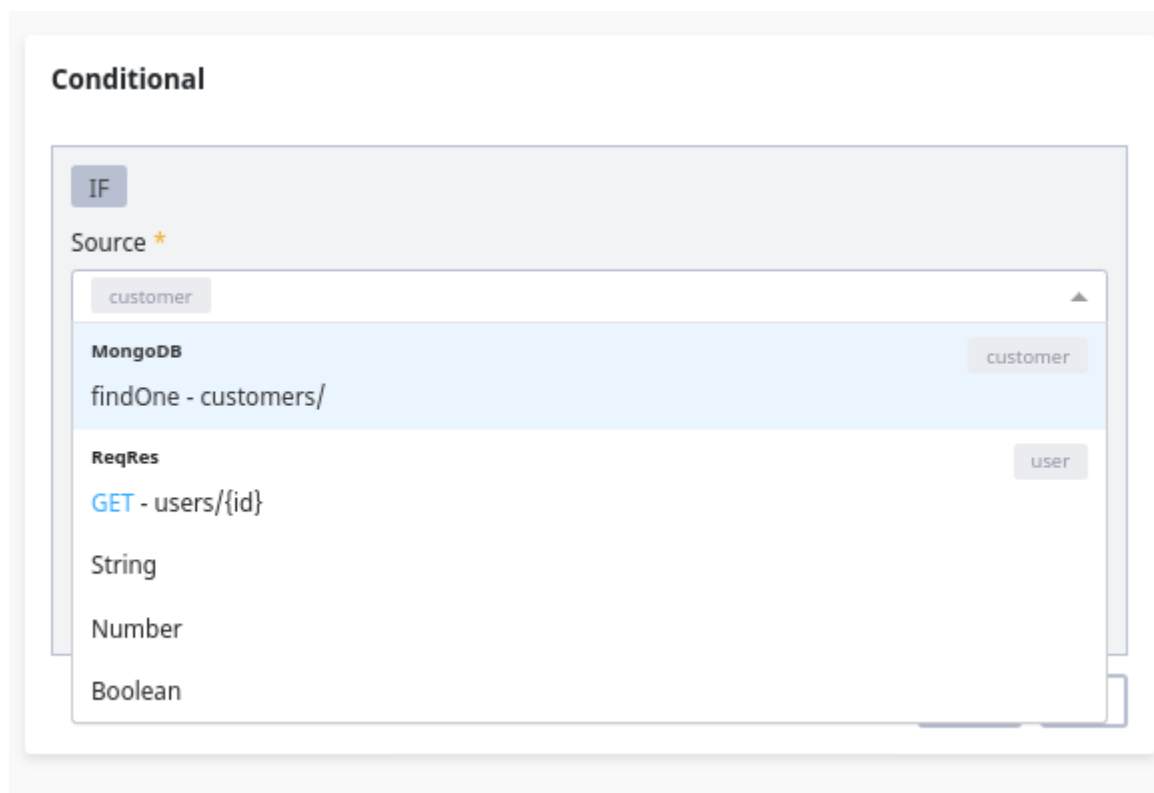
```
queryString: {},  
urlParams: {}  
};  
};
```

Conditional

To perform a condition in your automation flow, select the conditional step



First, select a source to make the condition. You can choose a previous step or primitive data like String, Number, or Boolean.



After select a source you can fill a data field.

If the source selected were a previous step, you can pass the name the field name.

Example: You choose a previous step called customer and you want compare a firstName field inside on the customer step. So the data field will be 'firstName'.

Conditional

IF

Source *

customer

Data field (Optional)

firstName

If you want to directly use the output of the selected source leave this input blank

Comparasion Operator

Add new conditional:

AND

OR

On two or more access levels in the object, pass the access path through '.', Continuing in the customer example, let's assume that you need to compare the street field but this field is within address so the data field will be equal to 'address.street'

Conditional

IF

Source *

customer

Data field (Optional)

address.street

If you want to directly use the output of the selected source leave this input blank

Comparasion Operator

Add new conditional:

AND

OR

After filled out data field, you must choose a Comparasion Operator

Conditional

IF firstName equals

IF

Source *

customer

Data field (Optional)

firstName

If you want to directly use the output of the selected source leave this input blank

Comparasion Operator

equals

equals

not equals

contains

doesn't contains

exists

doesn't exists

greater

And finally, select another source to perform the comparison.

Conditional

IF firstName equals LinkApi Solutions

IF

Source *

customer

Data field (Optional)

firstName

If you want to directly use the output of the selected source leave this input blank

Comparasion Operator

equals

Source *

String

Value (Optional)

LinkApi Solutions

Add new conditional:

AND

OR

Repeat

If you want to iterate over some data from a previous step, you must use this configuration.



Repeat

First, it is necessary to name the step. Remember that this name will have saved the value of each iterated item.

Repeat

^ Step name

Declare a name to this step to have access to the output in other steps.

Step name *

Save and continue

After, you must select a source to make the iteration. Remember, origins are earlier steps.

Repeat

v Step name

customer

^ For configuration

FOR EACH

Source *

customers

ReqRes

beginData

GET - users/

MongoDB

customers

find - customers

FINISH

After select a source you can fill a input list.

Example: You choose a previous step called customers, If the list is at the root of customers it is not necessary to pass the input list. But if the list is inside customer in the data parameter then put the date in the input list field

Repeat

Step name customer

For configuration

FOR EACH item in do

Source *

Input list (Optional)

If you want to iterate directly the output of the selected source leave this input blank

Finish

On two or more access levels in the object, pass the access path through '.', Continuing in the customers example, Let's assume that your list is in the list property within the data property that is within customers then the input list field 'data.list'

Code Block

This configuration is to perform some transformation or work on conversions.

 **Code Block**

Give the step a name, and then perform its function. Remember that what you return in this function you can take in the steps ahead.

Stop

If you want stop the automation at some condition or debug the automation you must use this step.

Select a stop configuration.

After choose a status to stop

Trace

This configuration has the objective of presenting logs in the flow

First of a trace name

Then select the status of the trace

Select the source you would like to log in. If you want you can log the entire step, just do not put the data field, but if you just want to log a certain field, just put the field name in the data field.

Dictionary

To use Dictionary actions in your automation flow, select the Dictionary step

Store and retrieve values in the key/value format for various purposes. Values are stored in groups according to the tenant in the context. Groups are a way to organize your dictionary data as you desire, like a database entity. Filling the **Group** field when using the dictionary step will only get/set values within the defined group.

First you need to select a name for this step, remember that this name will save the returned value of the step.

Then, define which dictionary group will you get/set values from and select the command between 'GET' and 'SET'. Here you can define an existing group or create a new one by typing its name.

Now after selecting which command to use, see more details about each of them below.

Get

Retrieve stored values using a unique key identifier, it can be a string retrieved from a previous step or manually inputed. See the example below, we are getting a value from the "bling-intelipost" example group:

When selecting a string from another step you can optionally set the exact data field, leaving it blank will set the direct output of the selected step.

Set

Store or update values on the defined group using a unique key identifier and its value. Both fields can be selected from the result of a previous step or a manually inputed string. See the example below, note that here we are setting a new value on the same "bling-intelipost" example group, if the **Group** field was filled with a group that still doesn't exist it would be created upon setting the value.

Select the "Allow update key" if you wish to overwrite existing dictionary data with the selected key.

High code

High code is an Advanced Code Automation. You should clone your project into your machine, you can use LinkApi's [SDK](#) and [CLI \(Command Line Interface\)](#) to develop and manage this project.

Guess what? You can use your **IDE or Text Editor favorite**. Like the **vscode** or others.

High code - Creating a High Code

Creating a High Code

To create a high code, follow the way: "Integration" > "Automation". Next clicks on "New automation", select 'High code' and filled the automation's name.

The next step is clone this automation in your machine. To do it, follow the instructions at platform.

High code - Development

After clone and open your `high code` into machine, you will have full power to develop it.

The project there are five folders, each one with its responsibility: **automations**, **data-transformations**, **functions**, **mocks**, **tests**. Below, there are details:

automations

Here your can create your automations, but always keeping the pattern below:

```
class Automation {
  async run(ctx) {

    // Your beautiful code goes here

  }
}
```

data-transformations

Here your can create your **data-transformations**, it helps you do transformations of the data. For example, you receive an object with the properties 'name' and 'lastname', but you need of the property with full name, then you can create a `data-transformation` to resolve this.

Bellow there is an example:

```
module.exports = async (user) => {
  const { name, lastName } = user;

  return { fullName: `${name} ${lastName}` };
}
```

functions

Here your can **create functions** to use in your projects.

Bellow there is an example:

```
module.exports = (items) => {
  const itemsFiltered = items.filter(item => item.status === 'ready');
```

```
return itemsFiltered;
}
```

mocks

Here you can create a JSON file to 'mock' a response.

Bellow there is an example:

```
{
  "name": "John",
  "age": 31,
  "city": "New York"
}
```

tests

Here you can create your tests to ensure quality in your project. The project is configured to use 'JEST' to help you.

Bellow there is an example:

```
const exampleAutomation = require('../automations/example-automation');

describe('Automation test', () => {

  it('should return status SUCCESS', async () => {
    const result = await exampleAutomation.run();
    expect(result).toEqual({ status: 'SUCCESS' });
  });

});
```

To run your test, you should run the command: `npm test` or `yarn test` .

Don't forget install your packages, you can run `npm install` or `yarn install`

Trigger

Triggers are used to search or receive events and perform automations based on the retrieved data. In the LinkApi portal it is allowed to create two types of triggers.

Polling

A poll trigger constantly executes automation based on time scheduled or time intervals.

Webhook

A webhook trigger is executed when a user fires a request for a generated URL when registering a webhook trigger informing a body to execute the automation linked to that trigger

To execute this trigger just copy the webhook url generated.

An perform a request.

Trigger Properties

- Tenant - Indicates which tenant this trigger will be for
- Automation - Indicates that this trigger will be for that automation
- Allow Notification - Allows notification through registered tags
- Status - Indicates whether the trigger is active or not
- Allow duplicated transaction - It is used when a single key-based transaction is allowed to happen if successful more than once

Templates - Overview

Intro

A template is a way to create new automations, using another automation as a base. The process will create a new automation with the same settings as the base automation, adapting the dependencies of the automation to a new environment.

Creating a New Template

To setup a new template, you must have an automation properly configured. After that, you need to access the "Templates" section on the "Integrations" menu and click on "New Template" on the upper right corner of the screen.

This will open a window where you can fill in the information needed for your template to work.

Name

This will be the name of your template. Set up a descriptive name, as this will be the main introduction to your template

Description

This is the description of your template. When someone tries to use your template, they will be able to see this description to better understand what your template is trying to accomplish.

Automation

This field is where you should select the automation used as a base for your template.

Privacy

The privacy of your template can be public or private. A private template will only be available in your account, and a public one will be available to everyone.

Category

This field is based on categorizing your template to make it easier for people to find it. You can search templates by these categories so setting up the right category will make it easier for others to find what they need.

Stage

The stage is used to define a template's status. A template with the "Stable" or "Beta" stage will be available to everyone, but a template with the "Under Construction" stage is only available internally on LinkApi.

Templates - Using a Template

Intro

After creating a template, you can use it multiple times to create a new automation based on the one you selected as the base for that template. The procedure to use the template is divided by steps, which we will cover below.

Using the Template

To begin the process of using a template, you must click on the "Try Now" button on the template card.

Review Information

After selecting your template and clicking on the button, you will be shown the basic information of the template, such as the name, the description and the automation flow. Notice that the automation flow will not be shown if you're using an High Code Automation.

Components

If you have any private components being used in your base automation, you will need to clone those components into your subscriber to ensure that your new automation has all the dependencies necessary in its environment. Notice that you will only be able to setup clones for private components, as the public components are always available to your automation. Simply define a display name and an internal name and press "Next" and the components will be cloned.

Automation Name

After dealing with the private components, you'll need to set up a name for your new automation. Be aware that clicking on "Next" here will create your new automation automatically.

Credentials

Now that you created your new automation, there are a few optional steps to customize your experience with it. The first one is the Credentials step, where you can create credentials for each component in the automation. Simply fill the fields and press "Next". If you wish to skip it, just click in the "Skip" button.

Triggers

If you are using a Low Code Automation as base for your template, you will be shown one last optional step to create a trigger for your new automation. If you wish to do so, fill the fields and hit next. If you don't want to do it now, you can simply click on "Skip" and end the process.

About Packages

The Packages is where you have an overview of abstractions of APIs and databases that can be used to build your projects. Within the Packages, you can find **Pre-Built Components**, **Private Components** and **Functions**.

All files inside Packages can be easily implemented inside your projects as a Dependency . You can select a **Pre-Built Component**, or clone to edit them, simply hovering over them, adding new Resources and your business rules. This will turn it into a **Private Component**.

If none of the Pre-Built Components fits you, you can also create a Private Component to connect with any endpoint online. Check how to create in section "Creating a Private Component".

You can easily start a Project in the Packages list, selecting Components and Functions as a Dependency.

Package - Private Component

Creating a Private Component

Build your own endpoints rapidly the way you need

Components are "encapsulated" REST APIs, databases, Web Services and what else you need to connect when integrating or exposing APIs.

So, how should you create and set a new Component on LinkApi?

New Component and Component Info

Click on "New component" in header on the right side and start filling the information fields, then click in "create" button.

Then you'll see the first fields that should be filled.

The **internal name**'s purpose is to label this Component when you use it in your project with an individual ID. As each API has its own peculiarities, we highly recommend consulting the documentation of the Component you're building for information like the base URL, authentication, resource names and others.

Component internal name	How your Component will be named inside the Catalog directory. Cannot contain spaces or special characters.
Name	The official name of your Component.
Logo	Paste an image URL that represents your Component.
Component description	Describe your Component in a few words. This is how the world will see your Component, so make it comprehensible.

Component internal name	How your Component will be named inside the Catalog directory. Cannot contain spaces or special characters.
Base URL	Your Component's base URL that will be reflected in each Resource. You can explicit the version of your Component at the end, for example: https://www.base-url.com/v1 . This field also accepts custom authentication params inside the URL using braces, like that: https://www.base-url.{myECommerce}.com/v1

Authentication

The second configuration tab is about Authentication types. So, whether it is a querystring, a header, a basic or a custom one, this is where you set them. Again, these are information that are found in the API's documentation. You can customize or add fields using the buttons on the upper right side, if necessary.

Authentication Type	Description
NO AUTH	Use it when your Component doesn't need an authentication login.
BASIC	Basic authentication only requires Username and Password fields.
HEADER	Authentication params that need to be sent via header.
QUERYSTRING	Authentication params that need to be sent via querystring.
OAUTH2	Authentication model that allows the user to give access to their data without exposing their password. For example: Facebook login. More info about OAUTH2 Components.
CUSTOM	Create any fields your Component requires and customize its code. This authentication type will allow you to access the Auth code tab, where you can customize your authentication even more. Clicking "Customize" on any Auth type will activate the Custom authentication.

Global Settings

The next step is defining what are going to be the default parameters in your Component. This step is optional. These are the permanent information within your requests, for instance, some querystring pattern or content type. The API's documentation will provide those.

The Default Params will be set to every Resource you build, so you don't need to register it in every Resource. Just set a default parameter and it will be replicated. You can choose between Querystring or Headers params.

By default, the results are returned in JSON format, however, if you need to receive them in XML, you can set this by adding a header type, choose Accept as the name and component/xml as a value, as shown below:

Resources

The final step is when you will define which Resources you are going to use. The LinkApi platform gives you the possible methods for each one of them (get, patch, delete, etc.) to reduce complexities. In order to execute the request, you have to set the credentials (in some cases, the API Key and the Component Key).

Response

A component maybe has a `Response block`, this section is responsible to help you understand response of the component or a resource.

You should write a json to represents the response and you can transform the json in visual fields.

Code

In "Code", you may view the code the behavior of the resource or component, edit and test it. You can for example, use the our powerful [SDK](#) to convert data or anything else.

In "**Request**" block you can data to the function, like `queryValues` to the variables.

Package - REST

Creating a REST Component

How can you create REST component on LinkApi's platform? Here, we will detail the step by step to a new REST component.

Click on "New component" in header on the right side and start filling the information fields. In this step, is that you should select REST on the type field and fill in the other fields, like name, description, api host, etc. Once that's done, you can move on to the next step

Information

The **Information** tab is about initial configurations. You may complete informations, like an image to component's logo or change any information.

Authentication

For **Authentication** help, access the Authentication article.

Global Settings

For **Global Settings** help, access the Global Settings article.

Resources

The **Resources** tab you should specify the URI name and select the action type, like `GET` , `POST` , `PUT` , `DELETE` or `PATCH` .

If you click in a resource, you may fill others configurations to your resource created:

"General" tab

In "General" tab, there are the setup general options for this resource.

"Request" tab

You can setup possible request entries for this resource. Set `querystring` and `url params` , for example.

"Response" tab

In "Response" tab, you can setup possible responses for this resource. More informations, [here](#)

"Code" tab

In "Code", you may view the code the behavior of the resource, edit and test it. More informations, [here](#)

When you're done, hit Create and your Component will be ready to use.

Package - MongoDB

Creating a MongoDB Component

How can you create Database component on LinkApi's platform? Here, we will detail the step by step to a new MongoDB component.

Click on "New component" in header on the right side and start filling the information fields. In this step, the only difference from creating different types of components, is that you should select "Database" on the type field, and MongoDB in the database type. Once that's done, you can move on to the next step.

The **Information** tab is about initial configurations. You may complete informations, like an image to component's logo or change any information.

The **Authentication** tab is about MongoDB authentication. MongoDB comes by default with "uri" authentication, so for this example, there is nothing to be altered.

- The **Resources** tab you specify which collections (In MongoDB's case) you need to consume and its actions, like `find`, `insertMany`, `agregate` and others.

If you click in a resource, you may fill others configurations to your resource created. In "General" tab, there are the setup general options for this resource. In "Schema", you may create a schema. In "Code", you may view the code the behavior of the resource, edit and test it.

In the same way when creating a REST component, the database Components also allow you to handle the Code, so you can specify some parameters, filters, or other changes that is convenient for what you need on your received payloads.

Package - SOAP

Creating a MongoDB Component

SOAP (Simple Object Access Protocol) is a messaging protocol specification for Web Services.

Inside LinkApi, you can easily create SOAP Components. When creating a Component, simply change its type to SOAP. Some changes will occur.

Click on "New component" in header on the right side and start filling the information fields. In this step, the only difference from creating different types of components, is that you should select "SOAP" on the type field. Once that's done, you can move on to the next step.

Information

The "Information" tab is about initial configurations. You may complete informations, like an image to component's logo or change any information.

Authentication

For Authentication help, access the [Authentication article](#).

Resources

In the Resources tab, the Alias represents the action you'll consume in the web service. You can create a new resource and should configure them clicking it.

"General" tab

In `General` tab: the Path will request the URL Action, and others informations, like title and description.

"Request" tab

You can setup possible request entries for this resource. Set `querystring` and `url params`, for example.

"Response" tab

In "Response" tab, you can setup possible responses for this resource. More informations, [here](#)

"Code" tab

In `Code` tab there are the code responsible to the action, you can change and test it.

You may read more, [here](#).

When you're done, hit Create and your Component will be ready to use.

Package - SQL Database

Creating a SQL Database component

How can you create Database component on LinkApi's platform? Here, we will detail the step by step to a new SQL Database component.

You can create those options: **MySQL, PostgreSQL, Oracle, SQL Server, Redshift.**

Click on "New component" in header on the right side and start filling the information fields. In this step, the only difference from creating different types of components, is that you should select "Database" on the type field, and MySQL in the database type. Once that's done, you can move on to the next step.

Information

The **Information** tab is about initial configurations. You may complete informations, like an image to component's logo or change any information.

Authentication

You should configure the authentication like your database require.

For **Authentication** help, access the [Authentication article](#).

Resources

The **Resources** tab you can create your queries, named them by alias and group them by something and settings.

You can create a resource clicking on green button with 'plus' icon.

If you click in a resource, you may fill others configurations to your resource created. Like informations in "**General**" tab, write the SQL in "**Script**" tab, write its response in "**Response**" tab and handler the code behavior of the resource in "**Code**" tab. Bellow this section, there are more details about each one.

"General" tab

In "General" tab, there are the setup general options for this resource, like alias, title and description.

"Script" tab

In "Script" tab, you should write the SQL of the resource. If you need variables in your SQL, you should create its with "@" and you can convert the variables to visual fields clicking on the button.

"Response" tab

There are all information about "Response", [here](#).

"Code" tab

In "Code" tab, you may view the code the behavior of the resource, edit and test it. You can for example, use the our powerful [SDK](#) to convert data or anything else.

In "Request" block you can data to the function, like `queryValues` to the SQL variables.

About API Gateways

Intro

With LinkApi, you can create projects where you can expose entire Automation flows or Components Resources through HTTPs APIs. To create an API Gateways. access "APIS" > "API Gateways", then clicks on "New API Gateways" and fill the fields.

API Gateways - Global Settings

Intro

The "Global Settings" tab is responsible to configure your API Gateways. You can configure informations, authentication, middlewares to each request or response, and others settings. Below, there are more details.

General

Here you can configure informations about the API Gateways, like choice a logo, write a descriptions, configure the authentication, view the name and Gateway URL

Global flow

Here you can configure global settings to apply in all requests or responses in your API Gateways, like a middleware to traffic management. For example, you can configure **IP Restrictions**, **Rate Limit**, **Schema Validator** to all requests. Or configure anything to the responses, like **cache** to help your API performance.

Observations:

- The steps configured follow the direction of the arrow (top to bottom).

Plugin - About

You can use plugins built by Link API in `Request` and `Response` flow. There are many plugins to help you. Its possible use is in `Global Flow` or in a specific `resource` .

To use these plugins, you should click on button with 'plus' icon in 'request' or 'response' section.

To use a plugin simply select it and fill the fields. To more details, you can see the plugin documentation.

Plugin - Rate Limit

Intro

Rate limit is used to manage the limit of requests in a time interval in your API Gateway. This plugin is used in `request flow`.

Creating a Rate Limit

To create a **Rate Limit**, select the Rate Limit card and fill the fields to create the plugin.

More details about the fields bellow:

Name: The plugin name

Requests Limit: The limit of API's requests.

Interval: The time interval in which requests can be sent.

Tags: Define by tags which groups will be affected by the rate limit.

Example: If you filled '5' in 'Requests Limit' and filled '10' in 'Interval', then we can send at most 5 requests to the API in an interval of 10 seconds. When we reach this request limit, we should wait for the interval time to finish to get it again.

Plugin - Schema Validator

Intro

Schema Validator is used to validate data sent by request or in a response.

For example, you created a API's resource to create an user in your database and the fields 'name' and 'email' are required. Then you can use a `schema validator` to validate if the requests has 'name' and 'email' in body request.

Creating a Schema Validator

To create a Schema Validator plugin, select the Schema Validator card and fill the fields to create the plugin.

More details about the fields bellow:

Name:

The plugin name

Parameter Context:

The context of the data you want to validate. If the data are in `Querystring` , `body` or `headers` .

Properties Context:

The properties that you want validate. For example, check if in body request has the field 'email'. To insert many properties, you should insert each properties separated by comma, like we usually creating an array.

Plugin - IP Restriction

Intro

IP Restriction is used to restrict your API Gateways to specific IPs.

Creating an IP Restriction

To create a IP Restriction plugin, select its card and fill the fields to create the plugin.

More details about the fields bellow:

Name:

The plugin name

IPS:

The IPs with access to your API. Only machines with IP filled in this field will be able to send requests.

You should insert each IP separated by comma, like we usually creating an array.

Plugin - Cache

Intro

Cache is used to save a response in cache and if the resource receives others requests into time interval configured, then the resource returns a response from cache.

Creating a Cache

To create a **Cache** plugin select the Cache card and fill the fields to create the plugin.

More details about the fields bellow:

Name:

The plugin name.

Duration:

The life cycle duration of the cache in seconds.

Monetization - API Gateways

Intro

The **monetization plugin** is used to monetize your API for each request. Example, you can define a request price to a specific endpoint and client, then you can handle the monetization by your clients. This plugin is used in `response flow` .

Creating a Monetization

To create a Monetization, click on the 'use' button and fill the fields to create the plugin.

Bellow, there are more details about the fields:

Name:

The plugin name.

Value per request:

The price for each request.

Tags:

Your tags can help you.

Details about your monetizations

To access the details about your api monetizations, you should access 'APIs > Monetization' in menu.

API Gateways - Resources

How to create a resource

You should click on green button with 'plus' icon and fill all fields.

The fields are **http request types** (`GET` , `POST` , `PUT` , `DELETE` and `PATCH`) and **URI name**.

When you click on a specific resource, you can configure the resource. Below there are details about all configurations you can do.

"General" tab

Here you can define the title and description of the resource.

"Request" tab

Here you can configure the pattern of the possible fields to send a request. This functionality helps you to document and view all fields.

"Response" tab

Here you can register all possible response pattern. To create a new response, you should click in green button with "plus" icon and fill the fields, like http status, response name and a color. The format to create the response is `JSON` and you can click on green button to convert fields.

"Flow" tab

Here you should define resource flow.

Request

In **Request**, you can define rules to requests in this resource, like `rate limit` or `validator` .

Runtime

In **Runtime**, you can define steps to functioning of this resource, like a `controller` . In each one step you can include a `JavaScript Code` or a `Component` .

Response

In **Response**, you can define steps to resource response, like a cache to improve the resource performance.

There is a flag called **Apply global flow**, if it's activated, then all steps/rules defined in [Global Settings](#) is activated to this resource.

"Test" tab

Here you can test your resource sending data, like `queryString` , `body` and `headers` . And expecting a response.

You can test clicking on button `Test` in top-right from code blocks.

API Gateways - Documentation

LinkApi will generate your API documentation automatically. In the action bar, select `Save` and then `Deploy`. In a few moments, your documentation will be published and can be accessed by anyone.

The link will be available in the action bar, next the buttons `save` and `deploy`. Each project will have a separate documentation. When a project is edited and published, the API Documentation will be updated in a few moments.

Developer Portal - About

Intro

With LinkApi, you can create developer portals, where you can group your gateways and set who can manage them. To create a developer portal, access "APIs" > "Developer Portals", then click on "New developer portal" and fill the fields.

Developer Portal - Settings

Intro

The "**Settings**" tab is responsible to configure your Developer Portals. You can configure the portal's basic information and select the APIs to be exposed in your developer portal.

Developer Portal - Permissions

Intro

The "Permission" tab is where you will select who has permission to access your developer portal. You can approve or reject users waiting for approval or revoke access for users already approved.

Waiting for Approval

Here you can manage users who requested access to your developer portal. You can approve their request, which will allow them access into the portal, or you can reject their request, which will delist it.

Approved Users

This is where you can check the list of users that have access to your developer portal. If you need, you can revoke access by clicking on the red button.

Developer Portal - Login

Intro

When you first access your developer portal page, you will enter the login screen. Here you can access the system using your credentials, ask for permission or request a new password if you forgot yours.

Request access

Requesting access to a portal is fairly easy, you need to click "Request Access" in the login screen, fill the form with your information and you can submit your request.

Request a password reset

If you forgot your password, you can click "" fill the field with your e-mail and you will receive an e-mail with a link where you can change your password to a new one.

Developer Portal - Documentation

Intro

Here you can check the documentation of the selected API. At the top, you'll see basic informations about it like it's name, description, base URL and the authentication.

Authentication

In this section you can set the authentication method for your API. This will affect every request you send through this page.

Endpoint

Here you can check the information of a specific endpoint. You can see the HTTP method, description, response example, the querystring, the response model and you can even test your endpoint with the "Test" button.

API Gateways - Granting API Access

If you define your API Gateways with authentications, then you should grant API access. To grant access, you should create a `client` to the projects.

To more details, [click here](#).

API Gateways - Api Key Authentication

This authentication can be configured in the settings screen which can be accessed through the kog icon in the left menu. An API key is a token that a client provides when making API calls. The key can be sent in the query string:

```
GET /something?api_key=abcdef12345
```

Note: It is important to ensure that there is no access with the same key value

API Gateways - OAuth2 Authentication

OAuth2 is an authorization standard that gives an API client limited data access. Our OAuth2 implementation relies on credentials to generate new temporary access.

To generate an access token, send a **POST** request to the endpoint `/token` of your API Gateway, with a JSON as body with the following content:

```
// POST https://<your api gateway URL>.gateway.linkapi.solutions/v1/token
{
  "clientId": "id",
  "clientSecret": "secret"
}
```

If the credentials are correct, an answer containing the `access_token` should be returned, as follows:

```
// Successful auth response
{
  "expires_in": 3600,
  "data": "token",
  "access_token": "token"
}
```

The access token expires after an hour that has been generated.

To use other endpoints, use the `access_token` generated as a header, in the following format:

```
// HTTP request header
{
  "Authorization": "Bearer <access_token>"
}
```

Keep in mind that the `/token` endpoint only accepts the credentials in the format described above, other authentication flows where the credentials are sent in different ways are not supported yet.

API Gateways - Basic Authentication

Basic Authentication is the most common authentication system for the HTTP protocol. It is included in the HTTP request header like this:

```
Authorization: Basic {base 64 credentials in the format user:password}
```

Note: The use of Base 64 is due to the MIME standard.

API Gateways - None Authentication

This configuration is not recommended as the API is exposed to anyone. To use any feature of the API there is no need to authenticate. To consume the API just call the desired route as in the example below:

```
GET /something
```

Note: When using this configuration, it will not be possible to work with the context of the tenants.

Management - Client

Intro

To granting API Gateways' access, you must create a **client**. Access the follow: "Management" > "Clients". Next clicks on "New client" and fill the fields.

Authentication

After created your client, you can add your APIs into the client. To add an API, you should click on 'edit'. In tab 'Authentication', then clicks on the button 'Add access', search your api and click on 'create' button. After that, you can copy your credentials.

Credentials

You can also create new credentials for your Components based on the client. To do that, you must access the "Credentials" tab and click on the "New Credentials".

This will open a new window where you can select a component and create a new credential for it. To do that, select a component and then fill the fields that show up.

Management - Tenant

Intro

Tenant is like a group or environment that running specific `triggers`. You can use it to your custom, department, store or anything group. For example: You have a marketing platform with many customers. You customers can use 'triggers' that your builded to help them, like a 'trigger' to get all new sales from CRM to will send to a ERP or anything else. Then you can use 'tenants' to separeted per customers, creating a tenant named with customer name for example(Customer X, Customer Y).

Creating a Tenant

To create a Tenant. access "Management" > "Tenants", then clicks on "New Tenant" and fill the fields. When you will create it, you can attach in a 'trigger'.

Management - Overview

Intro

You can create a credentials to a specific component. When you don't passing a credentials by high code, then will use the credentials created.

To more more details about component credentials, [read here](#).

Creating a credential

To create a component's credentials. access "Management" > "Credentials", then clicks on "New credential" and fill the fields.

Management - OAuth2 Credentials

Intro

If you need to create a credential using OAuth2, you can select an component with the "OAuth2" authorization type and you'll need to fill the fields.

Obtaining the Access Token

After filling the fields, you can click on Save and you will be redirected to your selected Auth URL, where you'll need to grant access to your credential. After that, your access token will be generated automatically based on the Access token URL provided.

Using Custom Names

If your integration uses a different name on any of the fields, you can customize it on the authentication options of your component.

When you use a custom name, not only it will change the display of that field on the credentials screen, but it will also change the generated URL when used on fields such as "client_id" or "client_secret". In the example below, those fields will be treated as "customer_key" and "customer_secret"

Using Default Values

If you need to use constant values on your component's credentials, you can set your fields as default, which will allow you to set up a default value on those fields on the "Default Values" tab.

That field will be set with that default value when you create any credential based on that component.

Management - Dictionary

Intro

Dictionary is a LinkApi feature to store data in key/value pairs for multiple purposes. These keys are stored inside groups that are saved by tenant on your account. You can save and get values on both Low code and High code automations. The Dictionary page's purpose is to visually manage the groups of keys you have stored.

Read more details about using dictionary on automations:

- [Low code](#)
- [High code](#)

Creating and editing a dictionary group

To create a new group you can simply click on the "New dictionary" button on the top right corner and fill the creation modal with a group name, tenant and one or more key/value pairs. See the example below:

You can also edit the group name if necessary, click the "Edit group name" button on the group row to do so:

To delete a group click the "Delete group" button on the group row:

Important: Deleting a group will also delete all its keys and it wont be possible to restore it after.

Visualizing and managing group keys

By clicking on the "View keys" button of any group you will be taken to a page where you can search, add, edit and delete keys inside that group. See the example:

Use the button "New key" to create new key/value pairs inside the selected group, a modal just like the group creation one will be shown. To edit an existing key, click the "edit" button on the key row, you can only edit its value:

To delete a key you simply click the "delete" button on the selected key. Just like deleting entire groups, keys also cannot be recovered after deletion.

Management - Users

Intro

You can create new users to so they can access Link API's Platform.

Creating an user

To create an user, you should access the follow: "Management" > "Users". Next clicks on "New User" and fill the fields.

If you want to remove the user's access, you must click in 'lock' icon.

SDK - Parser

Used to convert data such as XML to JSON. or JSON to CSV.

JSON

JSON to XML

To convert JSON to XML use the following example:

```
const { json } = require('@linkapi.solutions/nodejs-sdk/parser');

const jsonToParse = {
  name: 'John'
};

const xml = await json.toXML(jsonToParse);
```

JSON to CSV

To convert JSON to CSV use the following example:

```
const { json } = require('@linkapi.solutions/nodejs-sdk/parser');

const jsonToParse = {
  name: 'John'
};

const csv = await json.toCSV(jsonToParse);
```

XML

XML to JSON

To convert XML to JSON use the following example:

```
const { xml } = require('@linkapi.solutions/nodejs-sdk/parser');

const xmlToParse = `
<note>
```

```
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
`;

const jsonParsed = await xml.toJSON(xmlToParse);
```

CSV

CSV to JSON

To convert XML to JSON use the following example:

```
const { csv } = require('@linkapi.solutions/nodejs-sdk/parser');

const csvToParse = `
a,b,c
1,2,3
4,5,6
`;

const jsonParsed = await csv.toJSON(csvToParse);
```

SDK - Logger

Used to create logs on the platform and provide visibility of what is happening within the flows.

Log

Method used to create logs, to create a log follow the example below:

```
const logger = require('@linkapi.solutions/nodejs-sdk/logger');

logger.log({
  data: {},
  name: 'Log',
  uniqueKey: '-',
  tags: [],
  status: 'SUCCESS',
  finalLog: false
});
```

You can view logs from your automations on Integrations > Logs:

SDK - Component

Used to consume Components and other entities.

Request

To consume a Component Resource use the following example:

```
const Component = require('@linkapi.solutions/nodejs-sdk/component');

const defaultOptions = {
  headers: {},
  queryString: {}
};
const myComponent = new Component('myComponent', defaultOptions);

const resourceMethod = 'GET'; // 'POST' | 'PUT' | 'DELETE' | 'PATCH'
const resourceName = 'products/{id}';

const result = await myComponent.request(resourceMethod, resourceName, {
  body: {},
  headers: {},
  queryString: {},
  urlParams: {
    id: '1'
  }
})
```

SDK - Dictionary

Store and retrieve values in the key/value format for various purposes. Values are stored in groups according to the tenant in the context.

Get

Retrieve stored values using a unique identifier and a group name, use the following example:

```
const dictionary = require('@linkapi.solutions/nodejs-sdk/dictionary');

const keyName = 'example';
const groupName = 'example-group';
const value = await dictionary.get(keyName, groupName);
```

Set

Store or update values belonging to tenants using a unique identifier and a group name, use the following example:

```
const dictionary = require('@linkapi.solutions/nodejs-sdk/dictionary');

const dataToStore = {
  name: 'John'
};

const options = {
  allowUpdateKey: false // when enabled update a value if key already exists
}

const keyName = 'userName';
const groupName = 'example-group';

await dictionary.set(keyName, dataToStore, groupName, options);
```


SDK - MongoDB

Used to perform operations on the MongoDB database management system

find

Selects documents in collection by a condition

```
const MongoDBService = require('@linkapi.solutions/nodejs-sdk/mongodb');

const findQuery = {};
const findOptions = {
  skip: 0,
  limit: 20,
  sort: { codigo: -1 },
  projection: { email: 1, codigo: 1 }
};
const collectionName = 'customers';
const connectionOptions = {
  uri: 'mongodb://localhost:27017/databaseName',
  // dbName: 'databaseName' // If you choose not to pass the database name in the connection string you can pass this
  // parameter
};

const collection = new MongoDBService(collectionName, connectionOptions);
const result = await collection.find(findQuery, findOptions);
```

findOne

Returns one document that satisfies the specified query criteria on the collection

```
const MongoDBService = require('@linkapi.solutions/nodejs-sdk/mongodb');

const findOneQuery = { email: 'product@domain' };
const findOneOptions = {
  projection: { email: 1, codigo: 1 }
};
const collectionName = 'customers';
const connectionOptions = {
  uri: 'mongodb://localhost:27017/databaseName',
  // dbName: 'databaseName' // If you choose not to pass the database name in the connection string you can pass this
  // parameter
};

const collection = new MongoDBService(collectionName, connectionOptions);
const result = await collection.findOne(findOneQuery, findOneOptions);
```

insertOne

Inserts a document into a collection.

```
const MongoDBService = require('@linkapi.solutions/nodejs-sdk/mongodb');
const document = {
  codigo: 7,
  email: 'name@domain.com',
  name: 'Name'
};

const collectionName = 'customers';
const connectionOptions = {
  uri: 'mongodb://localhost:27017/databaseName',
  // dbName: 'databaseName' // If you choose not to pass the database name in the connection string you can pass this
  // parameter
};

const collection = new MongoDBService(collectionName, connectionOptions);
const result = await collection.insertOne(document);
```

insertMany

Inserts multiple documents into a collection.

```
const MongoDBService = require('@linkapi.solutions/nodejs-sdk/mongodb');
const documents = [
  {
    codigo: 1,
    email: 'customer@domain.com',
    name: 'Customer'
  },
  {
    codigo: 2,
    email: 'customer2@domain.com',
    name: 'Customer 2'
  }
];

const collectionName = 'customers';
const connectionOptions = {
  uri: 'mongodb://localhost:27017/databaseName',
  // dbName: 'databaseName' // If you choose not to pass the database name in the connection string you can pass this
  // parameter
};

const collection = new MongoDBService(collectionName, connectionOptions);
const result = await collection.insertMany(documents);
```

updateOne

Updates a single document within the collection based on the query.

```
const MongoDBService = require('@linkapi.solutions/nodejs-sdk/mongodb');

const updateOneQuery = { email: 'myname@domain.com' };
```

```

const updateOneAction = { $set: { customDomain: true, name: 'MyName' } }; // In this parameter you can inform any action
parameter of the mongodb update operation
const updateOneOptions = {
  // upsert: <boolean>,
  // writeConcern: <document>,
  // collation: <document>,
  // arrayFilters: [ <filterdocument1>, ... ],
  // hint: <document|string> // Available starting in MongoDB 4.2.1
};

const collectionName = 'customers';
const connectionOptions = {
  uri: 'mongodb://localhost:27017/databaseName',
  // dbName: 'databaseName' // If you choose not to pass the database name in the connection string you can pass this
parameter
};

const collection = new MongoDBService(collectionName, connectionOptions);
const result = await collection.updateOne(updateOneQuery, updateOneAction, updateOneOptions);

```

updateMany

Updates all documents that match the specified query for a collection.

```

const MongoDBService = require('@linkapi.solutions/nodejs-sdk/mongodb');

const updateManyQuery = { email: /.*@domain.*/ };
const updateManyAction = { $set: { customDomain: true } }; // In this parameter you can inform any action parameter of
the mongodb update operation
const updateManyOptions = {
  // upsert: <boolean>,
  // writeConcern: <document>,
  // collation: <document>,
  // arrayFilters: [ <filterdocument1>, ... ],
  // hint: <document|string> // Available starting in MongoDB 4.2.1
};

const collectionName = 'customers';
const connectionOptions = {
  uri: 'mongodb://localhost:27017/databaseName',
  // dbName: 'databaseName' // If you choose not to pass the database name in the connection string you can pass this
parameter
};

const collection = new MongoDBService(collectionName, connectionOptions);
const result = await collection.updateMany(updateManyQuery, updateManyAction, updateManyOptions);

```

deleteOne

Removes a single document from a collection.

```

const MongoDBService = require('@linkapi.solutions/nodejs-sdk/mongodb');

const deleteOneQuery = { email: 'customer@domain.com' };
const collectionName = 'customers';

```

```

const connectionOptions = {
  uri: 'mongodb://localhost:27017/databaseName',
  // dbName: 'databaseName' // If you choose not to pass the database name in the connection string you can pass this
  // parameter
};

const collection = new MongoDBService(collectionName, connectionOptions);
const result = await collection.deleteOne(deleteOneQuery);

```

deleteMany

Removes all documents that match the query from a collection.

```

const MongoDBService = require('@linkapi.solutions/nodejs-sdk/mongodb');

const deleteManyQuery = { customDomain: true };
const collectionName = 'customers';
const connectionOptions = {
  uri: 'mongodb://localhost:27017/databaseName',
  // dbName: 'databaseName' // If you choose not to pass the database name in the connection string you can pass this
  // parameter
};

const collection = new MongoDBService(collectionName, connectionOptions);
const result = await collection.deleteMany(deleteManyQuery);

```

aggregate

Calculates aggregate values for the data in a collection

```

const MongoDBService = require('@linkapi.solutions/nodejs-sdk/mongodb');
const aggregatePipeline = [
  {
    $match: {
      customDomain: false
    }
  },
  {
    $lookup: {
      from: 'customeraddresses',
      localField: 'codigo',
      foreignField: 'customer',
      as: 'addresses'
    }
  }
];

const collectionName = 'customers';
const connectionOptions = {
  uri: 'mongodb://localhost:27017/databaseName',
  // dbName: 'databaseName' // If you choose not to pass the database name in the connection string you can pass this
  // parameter
};

const collection = new MongoDBService(collectionName, connectionOptions);
const result = await collection.aggregate(aggregatePipeline);

```

SDK - MySQL

Used to perform operations on the MySQL database management system

execute

Execute a query

```
const mysql = require('@linkapi.solutions/nodejs-sdk/mysql');

const params = {
  query: `SELECT * FROM users WHERE id = @userId`,
  queryValues: {
    userId: 82 // This field will replace the query's @userId
  }
};

// For a complete list of options see the npm mysql2 documentation.
const connectionOptions = {
  host: 'localhost',
  user: 'admin',
  password: 'admin',
  database: 'databaseName',
  port: '3306'
};

const response = await mysql.execute(params, connectionOptions);
```

SDK - SQL Server

Used to perform operations on the SQL Server database management system

execute

Execute a query

```
const sqlserver = require('@linkapi.solutions/sqlserver');

const params = {
  query: `SELECT * FROM users WHERE id = @userId`,
  queryValues: {
    userId: 3 // This field will replace the query's @userId
  },
  credentials: {
    database: "mydb",
    port: "1433",
    password: "pass",
    username: "user",
    host: "host",
    options: { // For a complete list of options see the npm mssql documentation.
      enableArithAbort: true,
      encrypt: true
    }
  }
};

const response = await sqlServer.execute(params);
```

SDK - Oracle DB

Used to perform operations on the Oracle DB database management system

execute

Execute an Oracle DB query

```
const oracledb = require('@linkapi.solutions/nodejs-sdk/oracledb');

const queryParams = {
  query: 'SELECT * FROM MYTABLE WHERE condition = @condition',
  queryValues: {
    condition: true // This field will replace the query's @condition
  }
};

// For a complete list of options see the npm oracledb documentation.
const queryOptions = {
  // autoCommit: true,
  // batchErrors: true, // continue processing even if there are data errors
  // extendedMetadata: true, // get extra metadata
  // prefetchRows: 100, // internal buffer allocation size for tuning
  // fetchArraySize: 100 // internal buffer allocation size for tuning
};

const connectionOptions = {
  user: 'oracleUser',
  password: 'oraclePassword',
  connectionString: 'oracleConnectionString'
};

const result = await oracledb.execute(queryParams, connectionOptions, queryOptions);
```

SDK - Redshift

Used to perform operations on the Redshift database management system

execute

Execute a query

```
const redshift = require('@linkapi.solutions/nodejs-sdk/redshift');

const params = {
  query: `SELECT * FROM users WHERE id = @userId`,
  queryValues: {
    userId: 3 // This field will replace the query's @userId
  }
};
// For a complete list of options see the npm `node-redshift` documentation.
const connectionOptions = {
  host: 'localhost',
  user: 'userName',
  password: 'pwd',
  port: 5439,
  database: 'databaseName'
};

const response = await redshift.execute(params, connectionOptions);
```


SDK - PostgreSQL

Used to perform operations on the PostgreSQL database management system

execute

Execute a query

```
const postgresql = require('@linkapi.solutions/nodejs-sdk/postgresql');

const params = {
  query: `SELECT * FROM users WHERE id = @userId`,
  queryValues: {
    userId: 3 // This field will replace the query's @userId
  }
};

const queryOptions = {
  rawResult: true // optional, returns full query result when true - default: false
};

// For a complete list of options see the npm `node-redshift` documentation.
const connectionOptions = {
  host: 'localhost',
  user: 'userName',
  password: 'pwd',
  port: 5439,
  database: 'databaseName'
};

const response = await postgresql.execute(params, connectionOptions, queryOptions);
```

SDK - SOAP

Used to perform SOAP requests

Request

To consume a SOAP webservice use the following example:

```
const soap = require('@linkapi.solutions/nodejs-sdk/soap');

const connectionParams = {
  wsdl: 'https://apps.correios.com.br/SigepMasterJPA/AtendeClienteService/AtendeCliente?wsdl',
  // username: 'username', If the webservice has Basic auth authentication you must enter this field
  // password: 'password' If the webservice has Basic auth authentication you must enter this field
};

const connectionOptions = {
  // disableCache: boolean, don't cache WSDL files, request them every time.
  // endpoint: string, override the SOAP service's host specified in the .wsdl file.
  // envelopeKey: string, set specific key instead of <pre>< soap: Body></soap: Body ></pre >.
  // overridePromiseSuffix: string, if your wsdl operations contains names with Async suffix, you will need to override
  the default promise suffix to a custom one, default: Async.
};

const client = await soap.connect(connectionParams, connectionOptions);
// If the webservice has Basic auth authentication you must connect at your client using the method below
// const client = soap.connectBasic(connectionParams, connectionOptions);

const group = 'AtendeClienteService';
const servicePath = 'AtendeClientePort.consultaCEP';
const body = {
  cep: '00000000'
};

const result = await soap.request(client, group, servicePath, body);
```

As seen above the XML Envelope content is sent in the "body" object. If you prefer, it's possible to send de XML Envelope as a string in the body using the property "\$xml":

```
const body = {
  $xml: '<cep>00000000</cep>'
}
```

The image below shows how it was done to get the group and servicePath parameters used in the code above.

Note that the consultaCEP service is within the AtendeClientePort topic, so the servicePath is 'AtendeClientePort.consultaCEP'

SDK - Request

Through this library it is possible to make HTTP and HTTPS calls. The library is an abstraction of the axios module, all parameters passed are compatible with the module. If you need any information that was not found here, visit: <https://github.com/axios/axios>

Simple HTTP call

```
const request = require('@linkapi.solutions/nodejs-sdk/request');

const options = {
  method: 'GET', // 'POST' | 'PUT' | 'PATCH' | 'DELETE'
  url: 'https://api.com',
  queryString: {
    page: 2
  },
  headers: {
    key: 'value'
  },
  body: {
    some: 'data'
  },
  generateException: false, // optional - generate an exception if the response status code is not between 200-299 - default true
  timeout: 1000, // optional - timeout in ms
  rejectUnauthorized: false // optional - returns an error if certificate validation fails - default true
  fetchWithFullResponse: false // optional - returns only the response body - default true,
};

const response = await request(options);

// output
// {
//   "request": {
//     "queryString": {},
//     "body": {},
//     "url": "",
//     "method": "get",
//     "params": {},
//     "data": "{}",
//     "headers": { }
//   },
//   "response": {
//     "body": {
//     },
//     "statusCode": 200,
//     "headers": { }
//   }
// }
```

SDK - Parallel

Run functions in parallel from an array of data and configure transformation to handle the array more easily.

Parallel

To run functions in parallel from an array use the following example:

```
const parallelExec = require('@linkapi.solutions/nodejs-sdk/parallel');

const array = []; // required

async function callback(item, uniqueKey) { // required
  try {
    // do stuff
  } catch(err) {
    // handle err
  }
}

const options = { // optional
  parallelExecutions: 2, // number of parallel executions, default: 2, max: 100
  uniqueKeyPath: 'id', // unique key path for each element of the array
  filterDuplicates: true, // filter duplicate elements from the array through the unique key, default: false
  interval: 1000 // interval between parallel executions in milliseconds, default: 200
};

await parallelExec(array, callback, options);
```

SDK - FTP

FTP

Used to communicate with FTP (File Transfer Protocol)

####Code structure

The code structure will always follow the same pattern, and you should only change the method line according to what you want to do.

```
const FTP = require('@linkapi.solutions/nodejs-sdk/ftp');

module.exports = async ctx => {
  try {
    // For a complete list of options see the npm ftp documentation.
    const connectionOptions = {
      host: 'localhost', // The hostname or IP address of the FTP server
      user: 'user', // Username for authentication
      password: 'password', // Password for authentication
      port: 21 // The port of the FTP server
    };

    const ftp = new FTP(connectionOptions);
    const result = await ftp.readFile(filePath); // Method Line

    return result;
  } catch (err) {
    throw err;
  }
};
```

####Methods ####Read File Retrieves a file at path from the server.

Method: `readFile(filePath)`

```
// Change on the method line
const result = await ftp.readFile(filePath);
```

####Write File Insert data into a server file.

Method: `writeFile(filePath, fileData)`

```
// Change on the method line
const result = await ftp.writeFile(filePath, fileData);
```

####Delete File Deletes a file from the server

Method: `deleteFile(filePath)`

```
// Change on the method line  
const result = await ftp.deleteFile(filePath);
```

####Copy File Copies a file from one folder to another

Method: `copyFile(fromPath, toPath)`

```
// Change on the method line  
const result = await ftp.copyFile(fromPath, toPath);
```

####Read Directory Retrieves the directory listing of path.

Method: `readDirectory(directory)`

```
// Change on the method line  
const result = await ftp.readDirectory(directoryPath);
```

SDK - Third Party Libs

Packages available to use when developing your project automations.

Lodash

Lodash is a lib that includes many methods that make data handling easier and faster.

Some example methods:

filter

```
const _ = require('lodash') // underscore (_) is a common import name for Lodash, to enable easy use, like jQuery ($)

var users = [
  { 'user': 'Kapi', 'age': 36, 'active': true },
  { 'user': 'John', 'age': 40, 'active': false }
];

_.filter(users, obj => { return obj.active; }); // Return all users where the property "active" is truthy
// => { 'user': 'Kapi', 'age': 36, 'active': true }
```

get

```
const _ = require('lodash')

var object = { 'a': [{ 'b': { 'c': 3 } }] };

_.get(object, 'a[0].b.c');
// => 3

_.get(object, ['a', '0', 'b', 'c']); // Same as object.a[0].b.c
// => 3

_.get(object, 'a.b.c', 'default'); // If path to desired property is invalid, fallback value "default" is return
// => 'default'
```

.get is a useful method in cases where property doesn't exist, it will not throw an error and runtime isn't stopped

random

```
const _ = require('lodash')

const randomNumber = _.random(0, 999999) // Returns a random number between 0 and 999999
```


For more methods, check out lodash's official [documentation](#)

axios

A simple lib that allows easy HTTP Requests

Example

```
const axios = require('axios');

axios.get('https://linkapi.solutions')
  .then(response => {
    return response.data
  })
  .catch(error => {
    // Do something with your failure :)
    return error
  })

// Or with async/await
async function makeRequest () {
  try {
    const request = await axios.get('https://linkapi.solutions')
    return request.dat
  } catch (error) {
    // Another error?
    return error
  }
}
```

For more details and params, check axios' official [documentation](#)

decimal.js

An arbitrary-precision Decimal type for JavaScript.

Examples

```
const Decimal = require('decimal.js')

let x = new Decimal(123.4567)
let y = new Decimal('123456.7e-3')
console.log(x === y) // true

let a = new Decimal('0xff.f') // '255.9375'
let b = new Decimal('0b10101100') // '172'
```

For more details on methods and how decimal.js can help, check their official [documentation](#)

IMPORTANT

Please note that other libs not included on this list are not available for use in LinkApi projects, if your use case needs a different library, please [open a support ticket telling us that](#)

CLI - Getting Started

About our CLI

With our CLI you can create your own catalog of components like components, functions and data-transformations. You can also consume the global components provided by LinkApi.

Our CLI (Command Line Interface) offers you a series of tools dedicated to developing and managing your projects on LinkApi.

Setup

Before installing LinkApi CLI, you'll need to install Node.js(8.12.0+). Having it installed, you can use NPM, which is Node package manager, to install our CLI executing the following command:

```
npm i -g @linkapi.solutions/cli
```

Now you have a linkapi global command, which can be executed on any terminal window. After installing the CLI, you'll need to login on your Google account, using:

```
lkp login
```

This command allows your local machine to connect to your LinkApi account, as well as access to your projects. In order to test if the automation worked, you can execute linkapi project list to see your whole project list. It needs to have the same projects that you can see on you LinkApi's portal.

How to get the latest version?

To check if your CLI is updated, you have to execute the installation command once again:

```
npm i -g @linkapi.solutions/cli
```

CLI - Projects

Create a project

To create a project via CLI, you should use the following command:

```
lkp project create
```

Shortcut version:

```
lkp p create
```

After executing the command, answer the questions on the form and at the end the project will be automatically cloned.

Clone a project

The first thing you need is a created project (via CLI or Portal). Now, to clone a project via CLI, you should use the following command:

```
lkp project list
```

Shortcut version:

```
lkp p l
```

After executing the command, a list of available projects will be displayed, just select the desired project and press enter. After selecting the project, select the Clone option. It is only possible to clone automation-type projects.

Manage dependencies

The first thing you need is a created project (via CLI or Portal). Now, to manage a project dependencies via CLI, you should use the following command:

```
lkp project list
```

Shortcut version:

```
lkp p l
```

After executing the command, a list of available projects will be displayed, just select the desired project and press enter. After selecting the project, select the Manage Dependencies option. A list will be displayed with the available components, just select the desired ones by pressing SPACE and press ENTER after selecting all the desired components.

Then a list of functions will be displayed and the selection procedure is the same as that of components.

Delete a project

The first thing you need is a created project (via CLI or Portal). Now, to delete a project via CLI, you should use the following command:

```
lkp project list
```

Shortcut version:

```
lkp p l
```

After executing the command, a list of available projects will be displayed, just select the desired project and press enter. After selecting the project, select the Delete option and confirm.

Generate template files

To generate a file from automation, function or data-transformation template, you should execute the following command:

```
lkp generate <type> <name>
```

Shortcut version:

```
lkp g <type> <name>
```

Valid values: automation, function or data-transformation

After executing this command, the file will be created from a template according to the declared type. Example:

```
lkp project generate automation test
```

Save changes

To save the changes you made into a project, you can use the following command:

```
lkp project commit "[message]"
```

After executing this command, all of the changes will be saved on the platform. Notice: When committing a project, the message is optional.

Discard changes

To discard all of the changes made on a project at once, you can execute this command:

```
lkp project discard
```

Shortcut version:

```
lkp p d
```

Publish changes

To publish a project via CLI, use this command:

```
lkp project publish
```

Shortcut version:

```
lkp p p
```

Sync Files

To sync changes of a project using CLI, use the following command:

```
lkp project sync
```

After executing this command, your project will be synced and all of the changes will be merged in its directory

Declare Project as Webhook

```
lkp project set --isWebHook true
```

After executing this command, you can create a Webhook trigger.

CLI - Triggers

Create a trigger

To create a trigger via CLI, you should use the following command:

```
lkp trigger create
```

Shortcut version:

```
lkp t c
```

After executing the command, answer the questions on the form and at the end the trigger will be created

Start a trigger

To start a trigger via CLI, you should use the following command:

```
lkp trigger list
```

Shortcut version:

```
lkp t l
```

After executing the command, a list of available triggers will be displayed, just select the desired trigger and press enter. After selecting the trigger, select the Run option.

Delete a trigger

The first thing you need is a created trigger (via CLI or Portal). Now, to delete a trigger via CLI, you should use the following command:

```
lkp trigger list
```


Shortcut version:

```
lsp t l
```

After executing the command, a list of available triggers will be displayed, just select the desired trigger and press enter. After selecting the trigger, select the Delete option and confirm

IP whitelists

Traffic from LinkApi

IP whitelisting allows you to ensure traffic to/from LinkApi.

- 35.174.51.251
- 52.21.113.85
- 54.209.233.223

You can add these IP addresses to your application/firewall whitelist. Add all three IP addresses to the whitelist to ensure continuous access.

Creating a ticket request inside LinkApi

Looking for improvements in every step of our experience, we have updated the experience on how to create and follow up tickets inside our platform in partnership with Atlassian. This article will show the step-by-step on how to create tickets inside LinkApi.

ATTENTION

Create tickets to get help only with LinkApi features and pages. If you need help with other APIs, authentication and connections, we recommend contacting directly the desired company.

First, click on the help menu and go to Tickets. You'll be redirected to company's ticket list page like the one below:

Here is where you can create tickets. You simply click the button "Open ticket" to open the creation modal. There you can select a category from Suggestion, Report a problem and Doubts. Choose either one and start describing your request.

Fill the fields above with the information you need help with. Make sure to be clear and direct in the subject when describing your problem. You can also upload screenshots to help. Just hit send when you're ready.

After sending your request you will be taken back to the tickets list page and your newest ticket will be there along with any previous requests. Click on the ticket row to go to see its details and comments.

On the ticket details page you will see all the initial request information and be able to follow along to responses like seen above. Use this page to reply and send new attachments if necessary. You will also receive email notifications when one of our agents responds your request.

Viewing older tickets opened through Jira Service Desk

If you previously sent requests to our support team using Jira Service Desk you will still be able to see these requests on your tickets list. Differently than requests opened through our platform, you will be redirected to the request's page on Jira when clicking to visualize its details.

We hope you liked our new tickets page!

Have any feedback or suggestions? Why don't you try creating a ticket with your submission? 😊

Support tickets SLA

Each ticket opened in our platform will be responded and handled according to its priority. Our team have defined SLA rules to identify a request's priority. See the table below to understand how we organize and handle each request: