convertigo

Back End
Basic training

Low Code Studio

How to use this tutorial ?

Welcome to your first journey
with Convertigo Low Code Studio.
Let’s explore its many features.

Concepts & Definitions

Convertigo uses many concepts
you may not be familiar with.
Find answers with this icon.

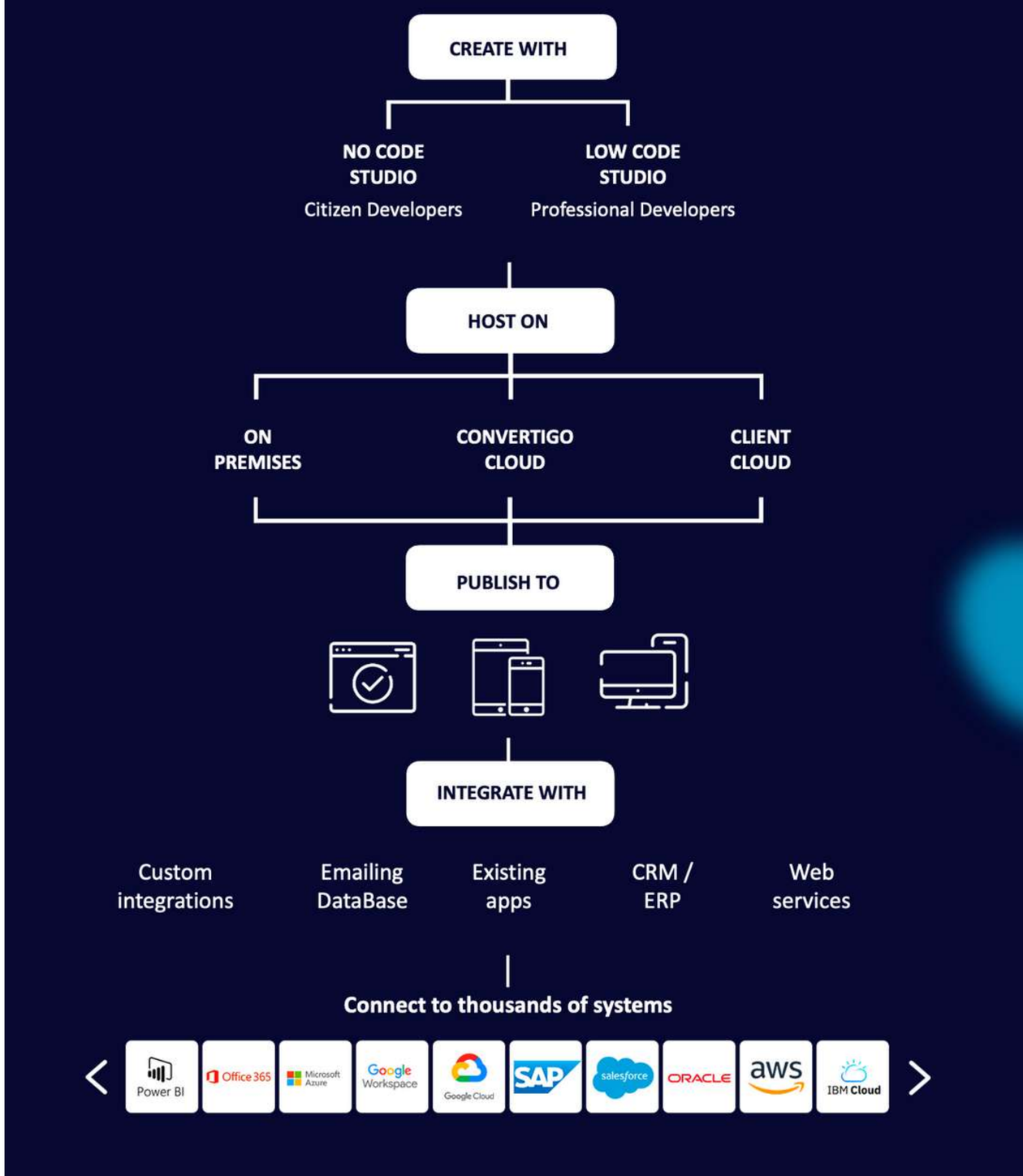
Practice time

You prefer to skip the concepts
and start by practice.
Go straight to this icon.

convertigo

What is Convertigo Low code Platform ?

- Full Stack
- Low Code
- Open Source
- Application Development Platform



What can you do with Convertigo Low code Studio ?

- Connect to back end systems with **Connectors**
- Exchange data with the backend using **Transactions**
- Define backend flows and business logic with **Sequences**
- Create web and mobile user interfaces with **Pages** and **UI Components**
- Create **iOS, Android, Progressive Web Apps** and **Web applications** from the same project
- Define and execute **Test cases**
- Share your projects with **Git versioning**

Table of Contents

1 – INTRODUCTION

Overview of the studio.

2 – GETTING STARTED

How to install and configure the studio.

3 – WEB SERVICES CONNECTORS & TRANSACTIONS

How to consume a Rest API.

4 – SEQUENCES

How to create a flow of actions.

5 – JAVASCRIPT SCOPE

How to handle JavaScript in the studio.

6 – ERROR MANAGEMENT

How to handle errors in the studio.

7 – COLLABORATION WITH GIT

How to share your projects with Git Versioning.

8 – TEST PLATFORM

How to test the backend.

9 – URL MAPPER

How to expose an API REST.

10 – NOCODE DATABASE

How to use the NoCode Database.

11 – LOGS

How to manage logs in the studio.

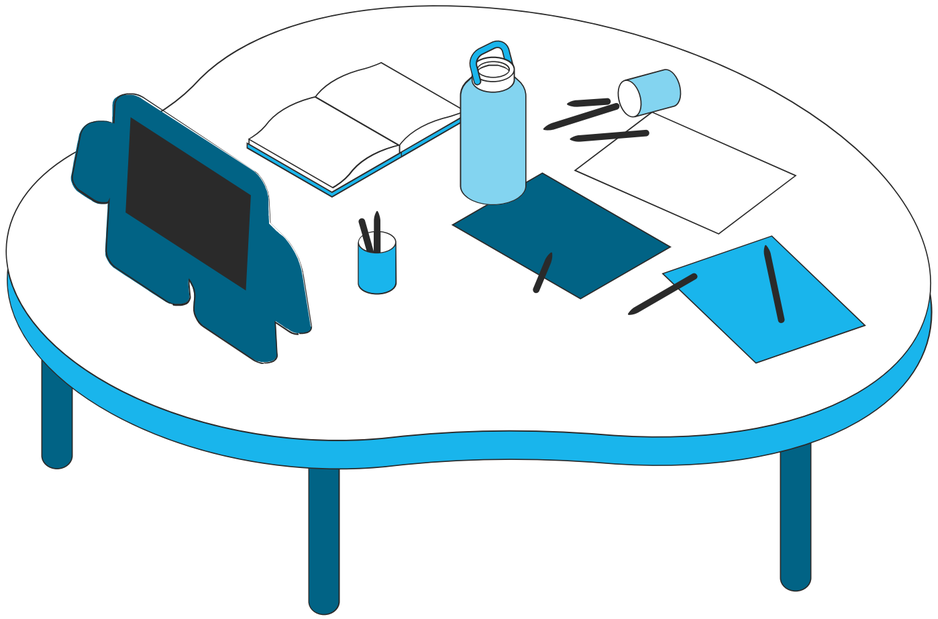
12 – AUTHENTICATION

How to manage authentication in the studio.

APPENDIX

1 – Introduction

Overview of the studio.



1.1 Technical knowledge

1.2 Global architecture

1.3 Convertigo Server

1.4 Objects in Convertigo

1.5 Back-end Objects

1.6 Studio Interface

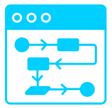
1.7 Panels & Views

1.1 Technical knowledge

The following concepts are necessary for mastering the studio.

WEB TECHNOLOGIES

- XML and XPath
- JavaScript & JSON
- HTTP requests
- REST API



ALGORITHMS

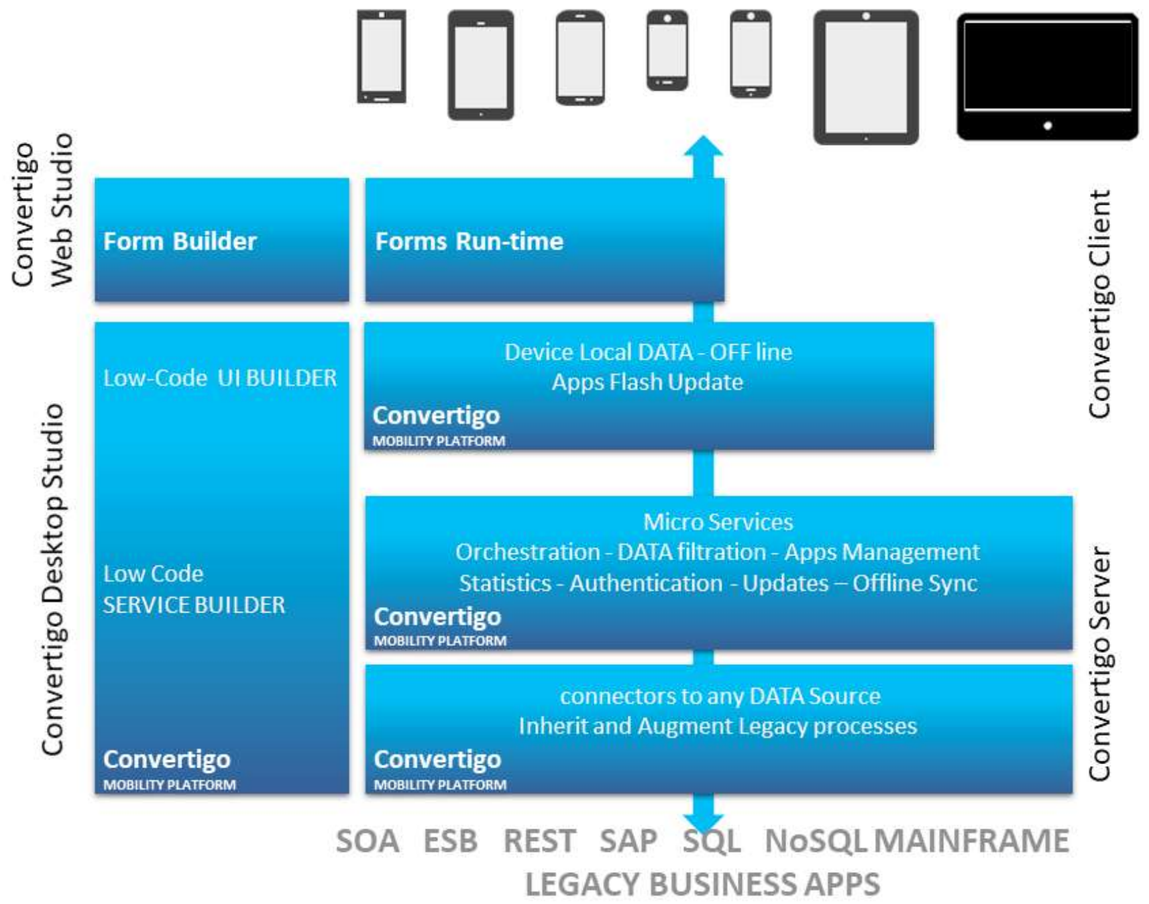
- Pseudocode basics
- Loops, Conditional statements...

DATABASES

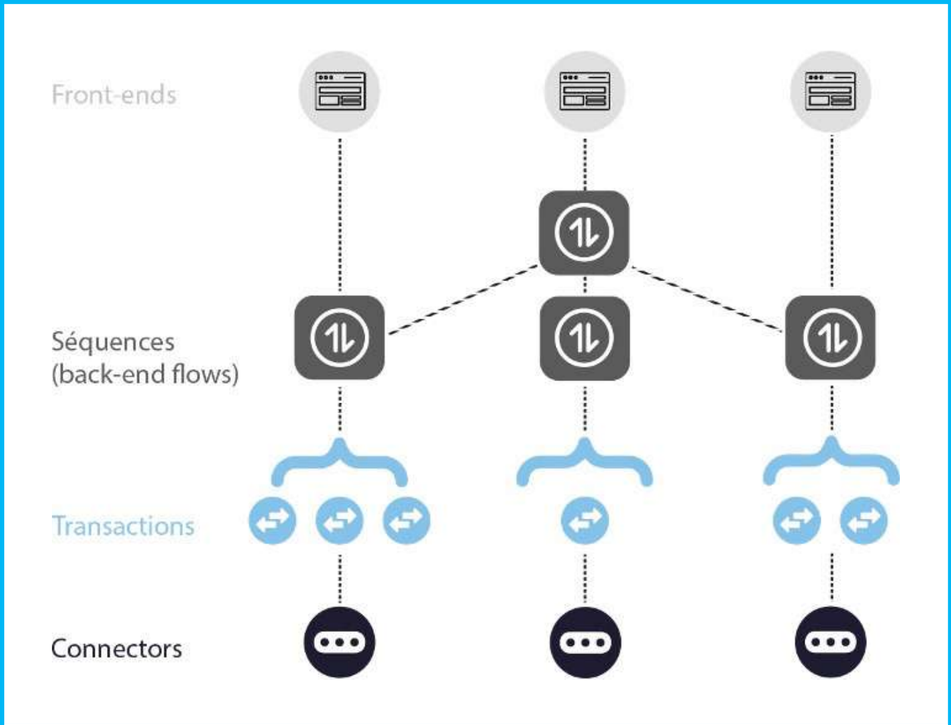
- SQL basics
- NoSQL basics

1.2 Global architecture

- Full Stack Low Code Platform
- Built on Docker Container Micro service Architecture
- Based on Open Standard technologies
  - (Eclipse, Node, Cordova, Angular, Java, Docker, Kubernetes, CouchDB, …)
- Out of the box Connectors to Enterprise Data
- 100% Offline Data sync technology
- Interface with AI & ChatBots
- Includes 100% Web Studio for Form Builder



1.3 Convertigo Server

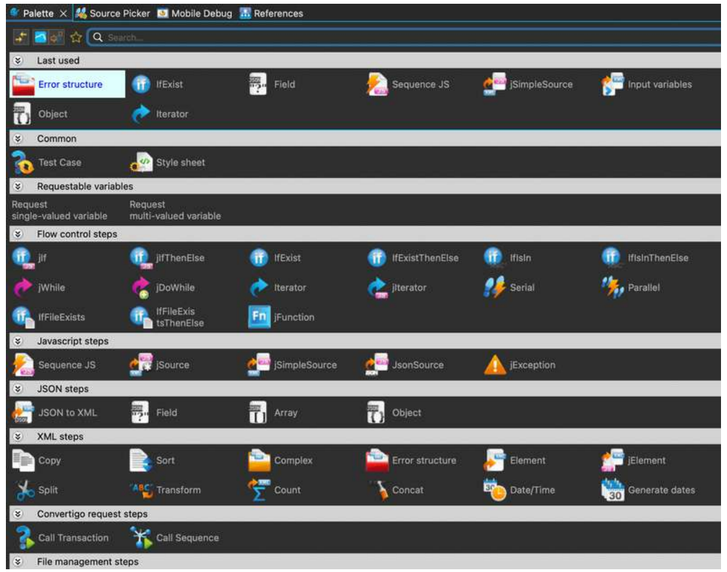


- Runs the **back-end** of the application
- Can be used to **deploy** as many apps as wanted
- Handles data in a **NoSQL database**
- Provides **connectors** to many **data providers**
  (SQL, Web services,
  Legacy apps running on mainframes…)
- Runs in **Docker container platforms**
  as a **Docker Image**
  (Cloud providers, Kubernetes on premises…)

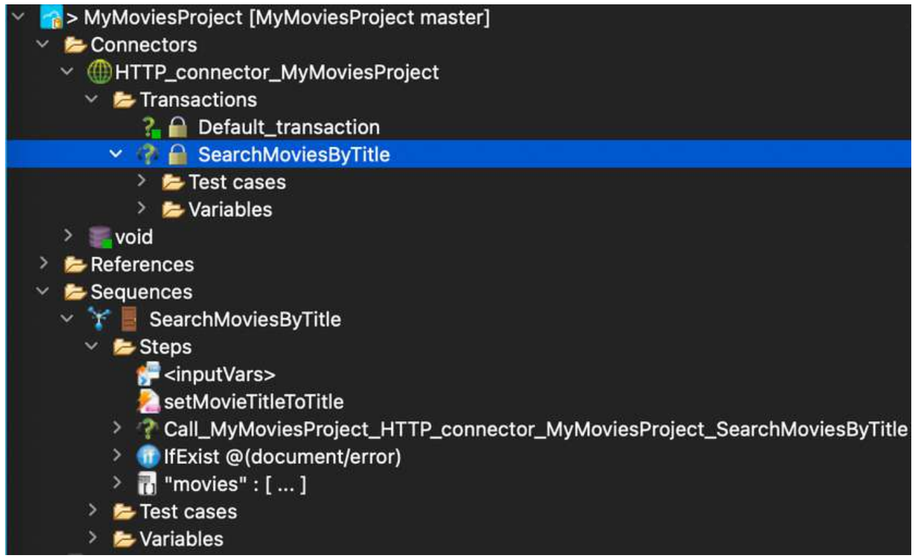
1.4 Objects in Convertigo

In Convertigo, **Objects** refer to **structured components** that **encapsulate data, functions, and properties**.

Objects are used to r**epresent and manipulate various elements** in Convertigo projects.
The objects are available in the **Palette view**.



A Convertigo project is **organized in a treeview**.
where you **drop objects dragged from the palette**.



1.5 Back-end Objects

In a Convertigo project, **back-end objects** handle the **back end processing**.

There are **3 main back-end objects** : **Connector, Transaction, and Sequence**.

Sequences interact with Connectors and Transactions
to read and write data to Databases, WebServices or Third party applications.

Data Source

Connector

Connects
to back end systems

Transaction

Exchanges data
with the backend

Sequence

Defines backend flows
and business logic

1.6 Studio Interface

The studio interface is divided in **5 main panels**. Each one contains **several views**.

PROJECTS PANEL

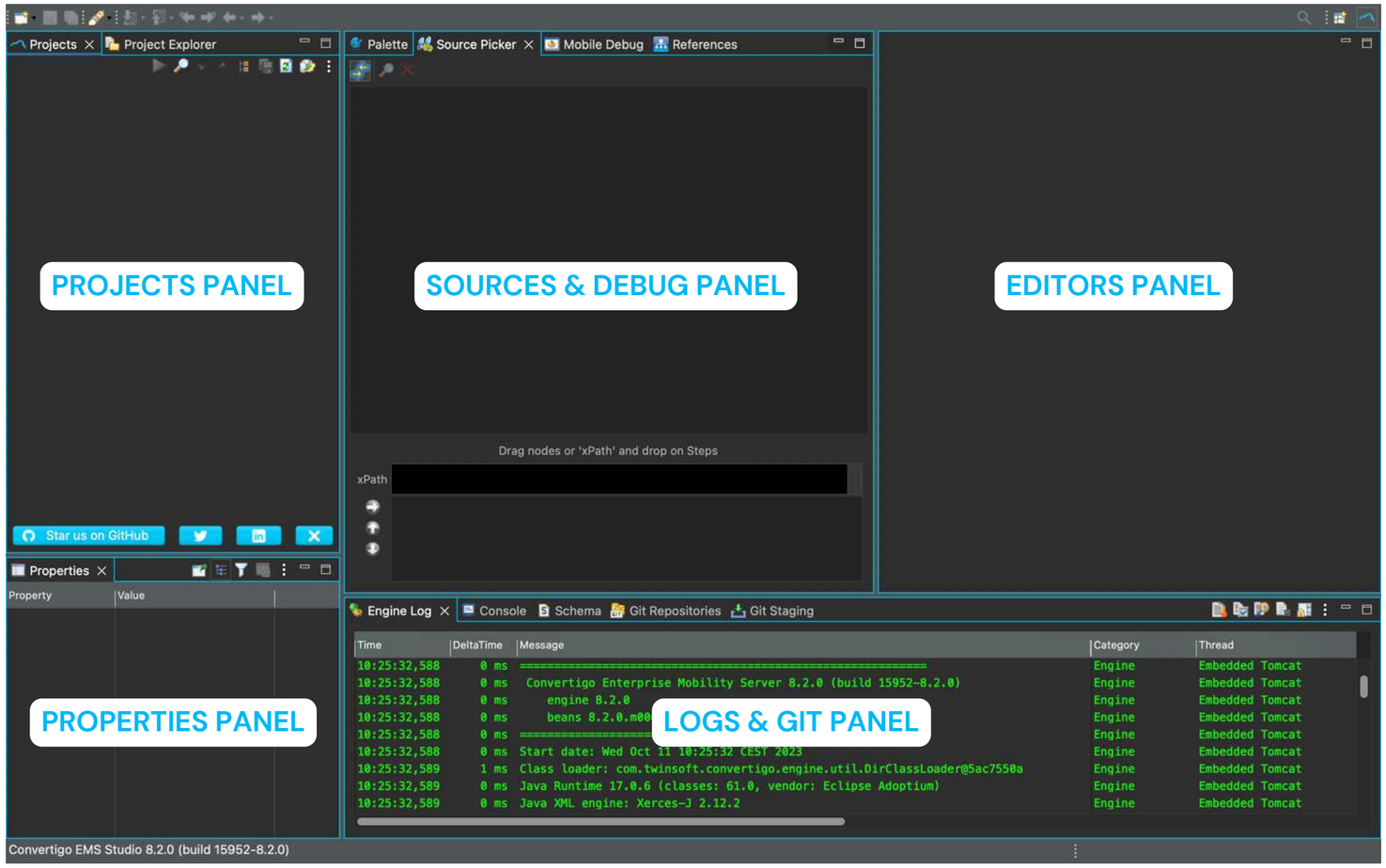
- PROJECTS
- PROJECT EXPLORER

SOURCES & DEBUG PANEL

- PALETTE
- SOURCE PICKER
- REFERENCES
- MOBILE DEBUG

PROPERTIES PANEL

- PROPERTIES



EDITORS PANEL

- VISUAL APP VIEWER
- CODE EDITORS
- CONNECTORS & SEQUENCES RESPONSES

LOGS & GIT PANEL

- ENGINE LOG
- CONSOLE
- SCHEMA
- GIT STAGING
- GIT REPOSITORIES

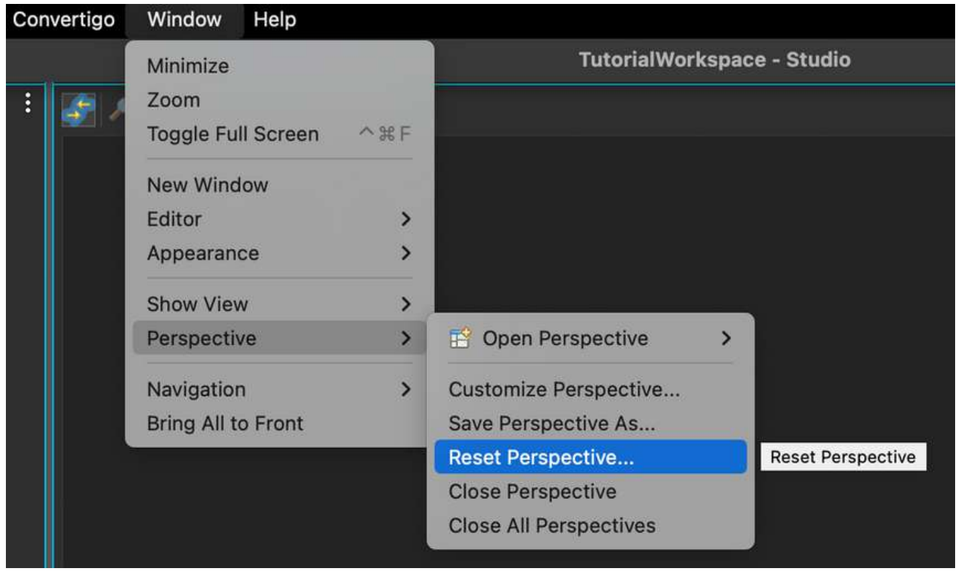
1.6 Studio Interface

The way views are organized is called a **perspective**.

Each view can be moved in other panels.

You can return to the original presentation or perspective

by clicking on **Window**, then selecting **Perspective>**, then selecting **Reset Perspective**.



1.7 Panels & Views

Projects Panel

PROJECTS

Displays the **projects in current workspace** and the **objects that compose them.**

PROJECT EXPLORER

Displays the projects **as files representing project assets, projects definitions as yaml** (for advanced users only)

Properties Panel

PROPERTIES

Displays the **properties of the object** selected in Projects view.

1.7 Panels & Views

Sources & Debug Panel

PALETTE

Displays all Convertigo **backend and frontend objects**.

SOURCE PICKER

Displays the **data sources for data binding** of the selected sequence step.

MOBILE DEBUG

Displays the **debugger for the front end** part.

REFERENCES

Displays **inside and outside project references** of the object selected in the Projects view.

1.7 Panels & Views

Logs & Git Panel

ENGINE LOG

Displays **Convertigo engine execution traces**.

SCHEMA

Displays the **XSD schema** used and/or generated by the project (input and output).

CONSOLE

Displays the **engine execution traces as text**.

1.7 Panels & Views

Logs & Git Panel

GIT REPOSITORIES

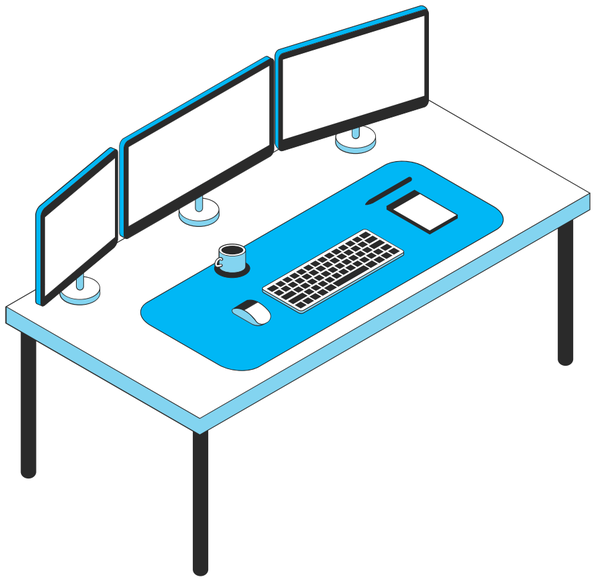
Displays the **Git Repositories** of the projects in your workspace.

GIT STAGING

Displays the files modified since the last commit and **Git management features**.

2 – Getting started

How to install and configure the studio.



2.1 Minimum System Requirements

2.2 Installation

2.3 Workspace & Convertigo archives file

2.4 Configuration

2.5 Create a project

2.6 Export a project

2.7 Import a project

2.1 Minimum system requirements

The following minimum system requirements are necessary for installing the studio.

WINDOWS

- Windows 10
- Windows 11

LINUX

- Ubuntu
  - version 20.04 (LTS)
  - version 22.04 (LTS)
- Debian version 11.0

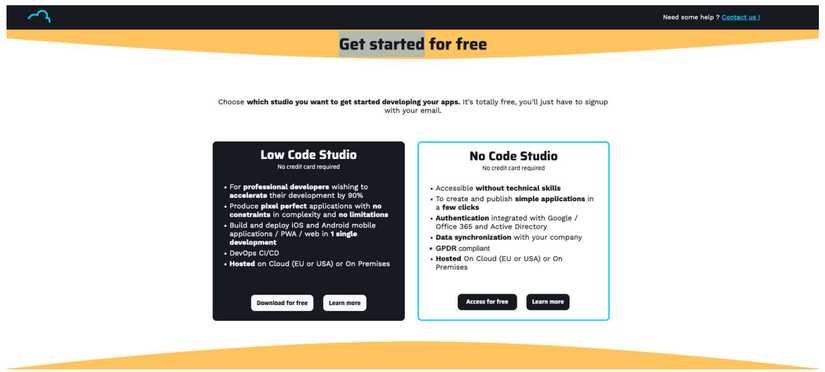
MAC OS

- Mac OS X
  - 10.5 (Leopard)
  - or greater
- Mac ARM

2.2 Installation

Go to the **Get started page** on **https://www.convertigo.com/get-started-page**.

Download the Low Code Studio package file for your operating system (Windows, Linux or Mac OS).



Open the package file and install the studio in a destination directory where you have the required permissions.

The installation package contains

- the Eclipse-based Convertigo Studio
- the embedded Convertigo Server with an Apache Tomcat application server

2.3 Workspace & Convertigo archives file

On first launch, you need to create a workspace for your projects.

A workspace is a directory where are saved

- Studio **configurations**
- Convertigo **projects**
- **Execution logs**

Example : A workspace with a Convertigo project and a .car file

The workspace is **located outside** of the installation directory
to **save your data**
if you need to **uninstall** or **re-install** the studio.

In Convertigo, the **import/export format** is **.car** (**Convertigo archives**) or **.zip**.
The .car is a zip file that contains all your project.

2.4 Configuration

After installation, the Studio needs to be configured on first launch.

CONFIGURATION PROCESS

Step 1 Select a directory as workspace

Step 2 Accept License

Step 3 Complete the workspace creation

Step 4 Configure proxy settings (Optional)

Step 5 Register with Convertigo Cloud Trial

Step 6 Welcome to Convertigo Low Code Studio

2.4 Configuration

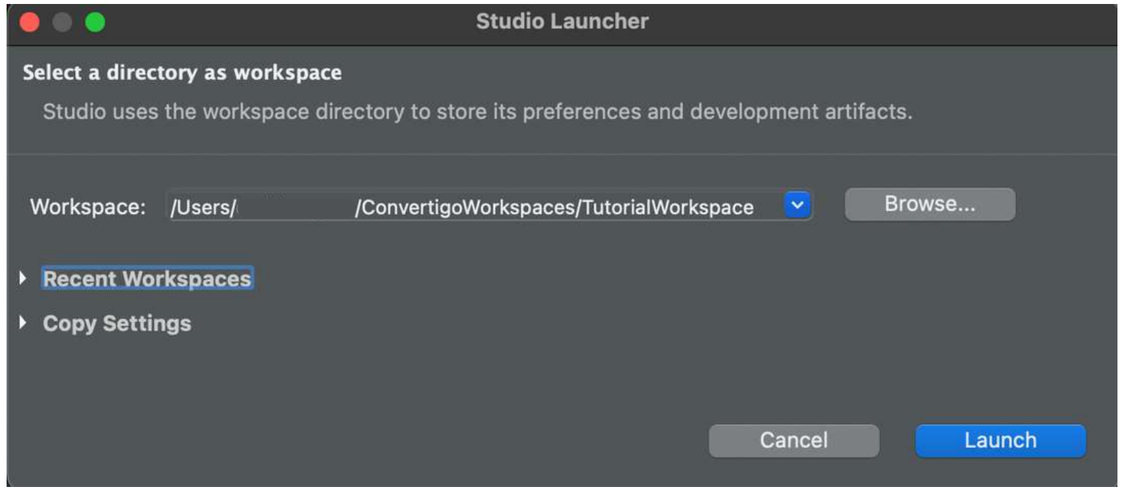
Step 1 – Select a directory as workspace

This is the first time we are going to launch the studio.
Let's start by creating a **workspace** for your projects.

Launch the studio
and **select a folder**
where your **workspace will be created**.

You can

- select an **existing** folder or **create** one.
- create **as many workspaces** as you want
- **wherever you want** on your computer.

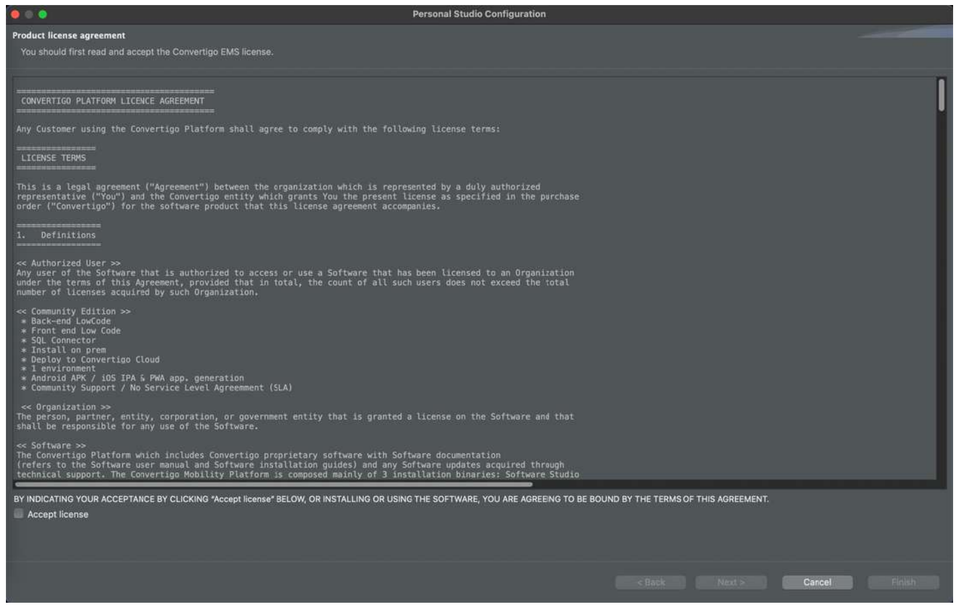


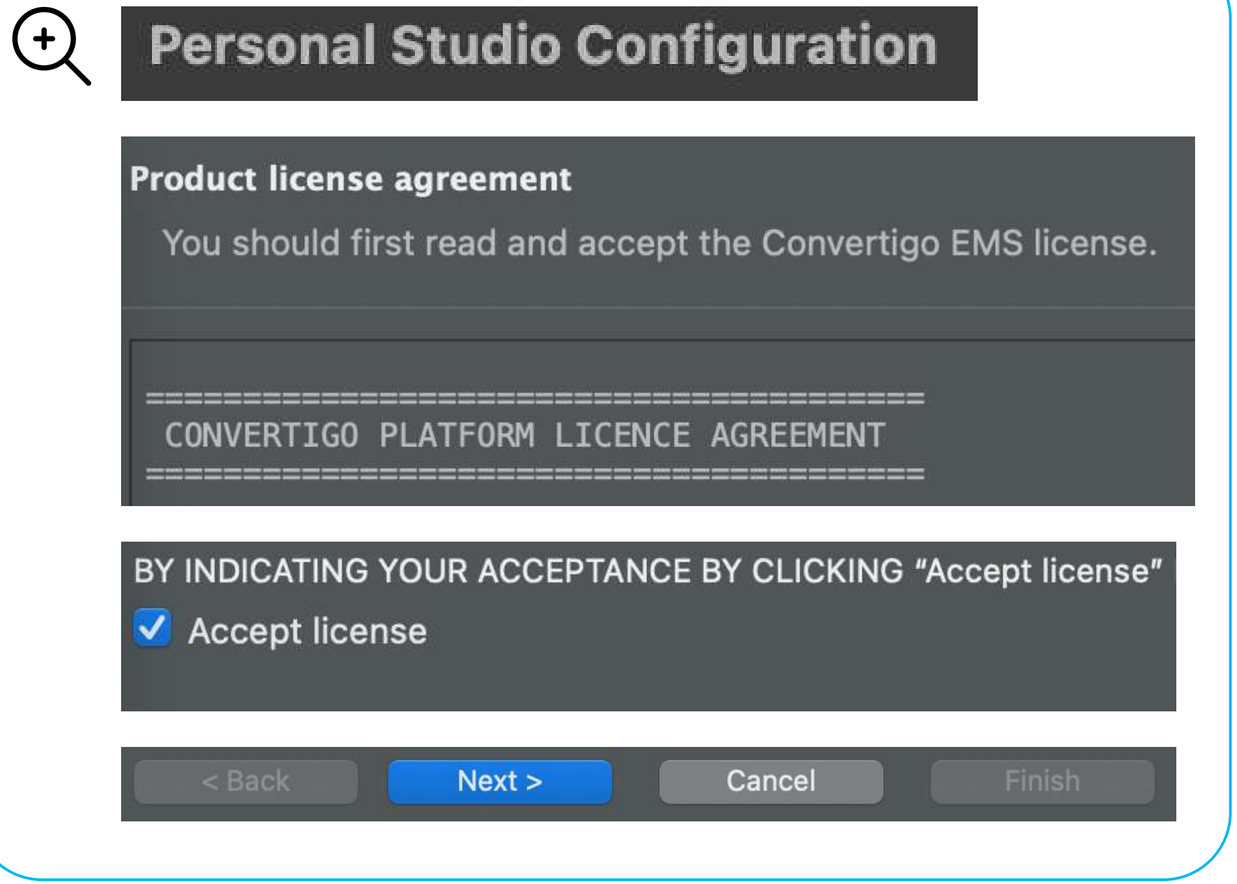
Good practice: create your workspace in your user folder on the same level as the Desktop and the Download folder – BUT NOT INSIDE THEM.

2.4 Configuration

Step 2 – Accept License

In the window **Personal studio Configuration**, **Accept License** and click on **Next >.**

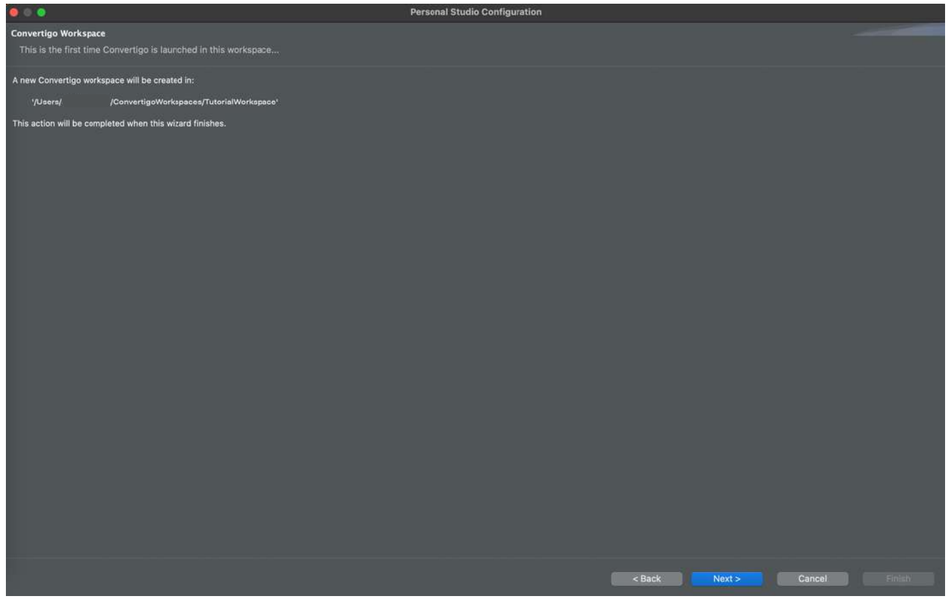


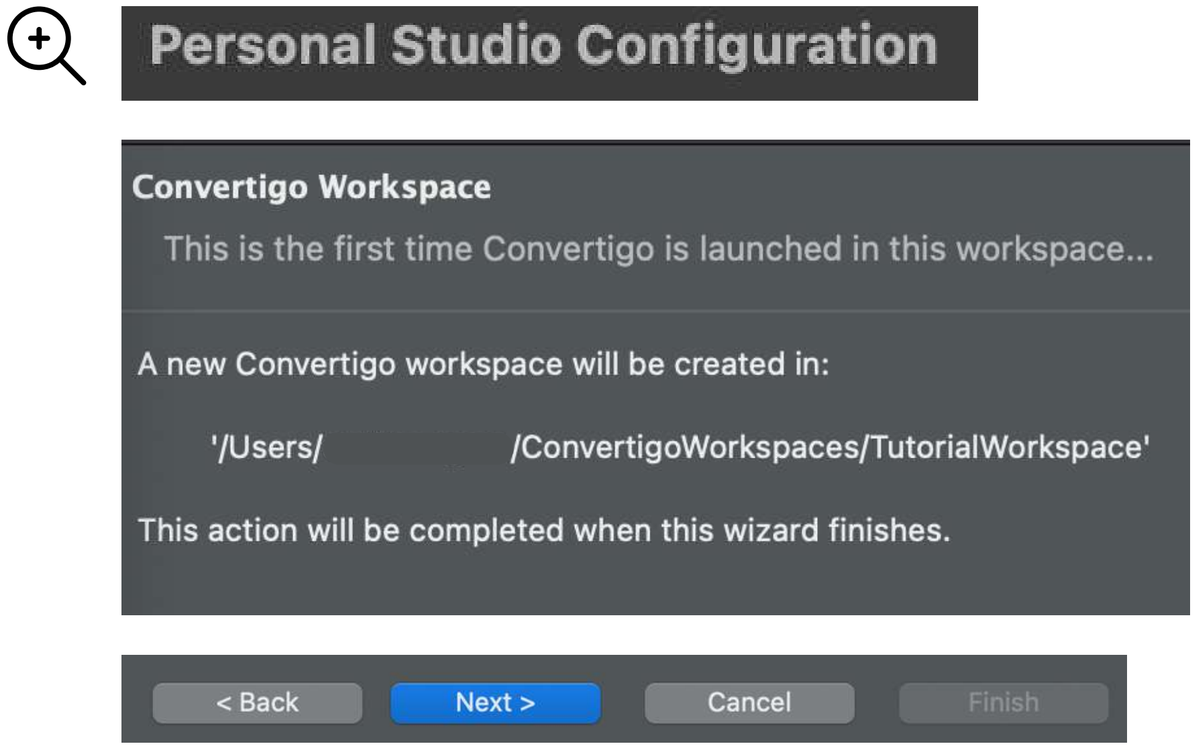


2.4 Configuration

Step 3 – Complete the workspace creation

To complete the creation of your workspace, click on **Next >**

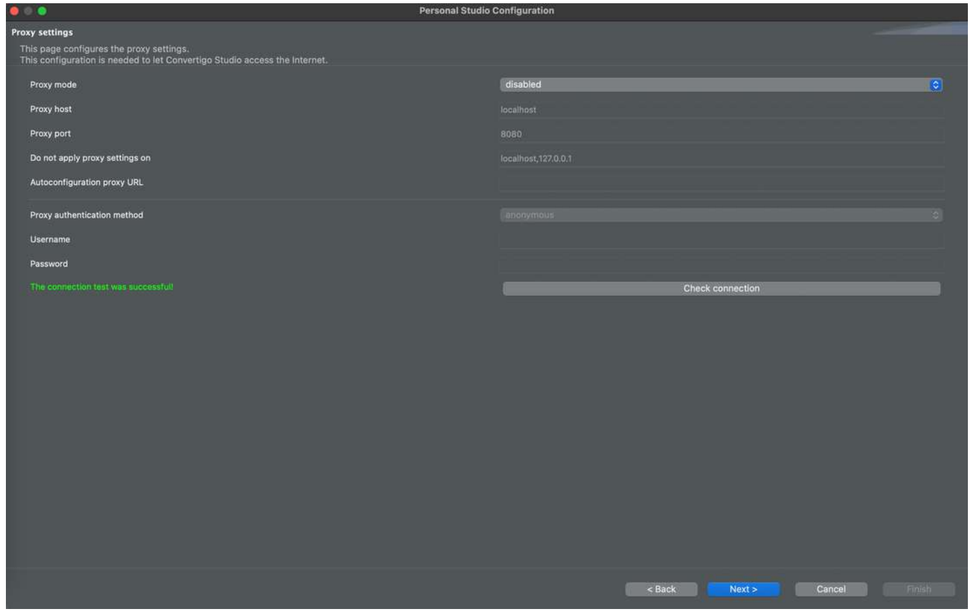


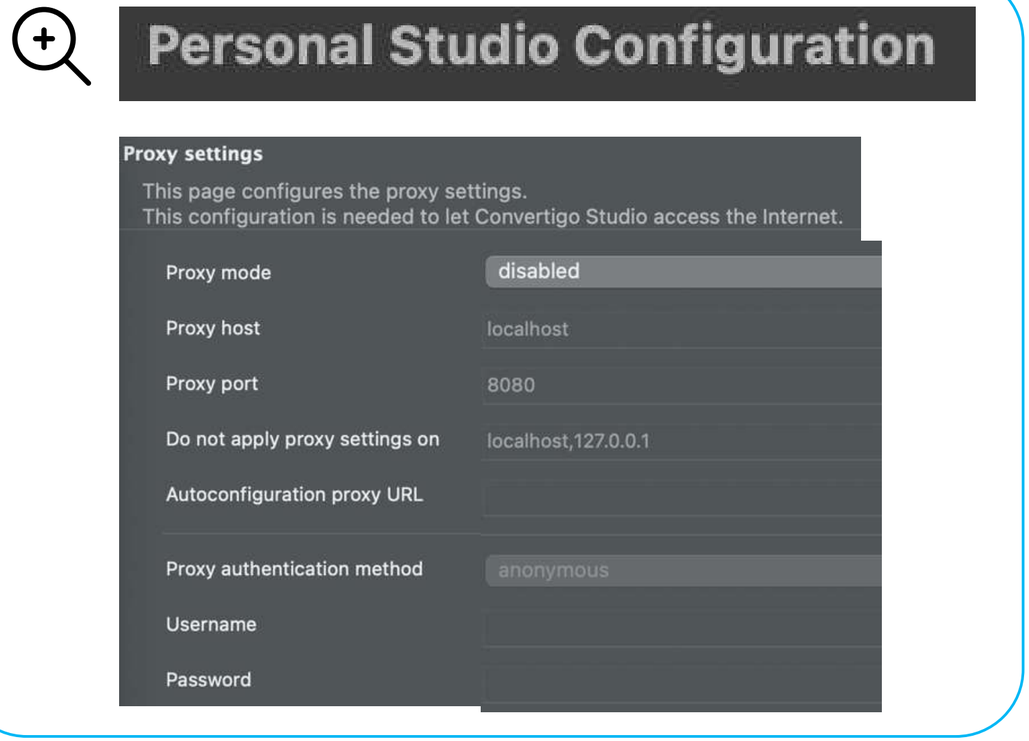


2.4 Configuration

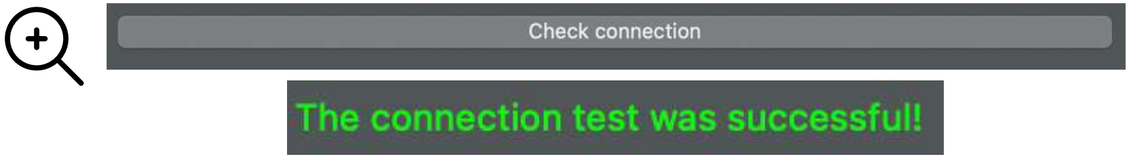
Step 4 – Configure proxy settings

Optional : You can **configure proxy settings** for Convertigo Studio to access the Internet





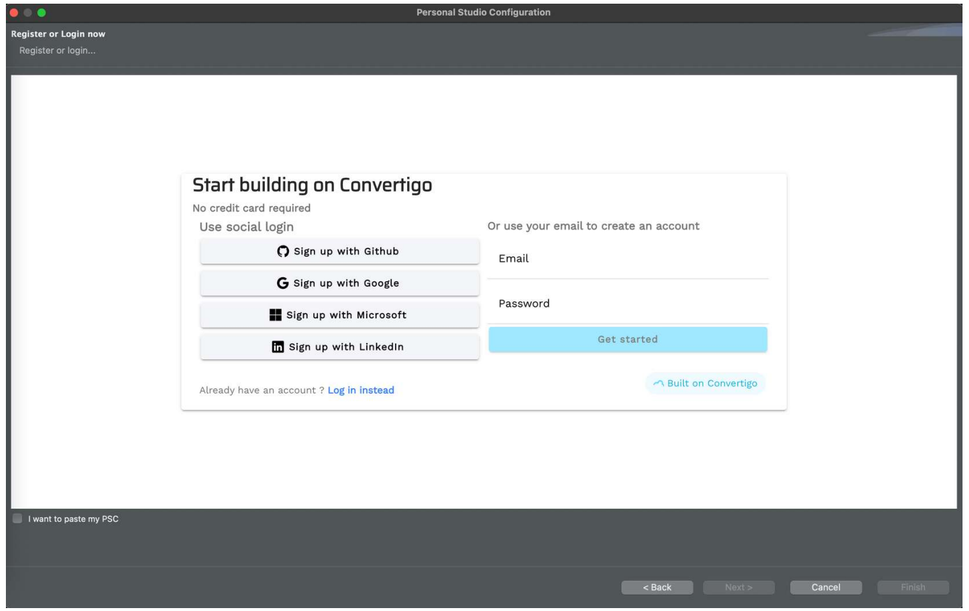
To check the connection, click on **Check connection**

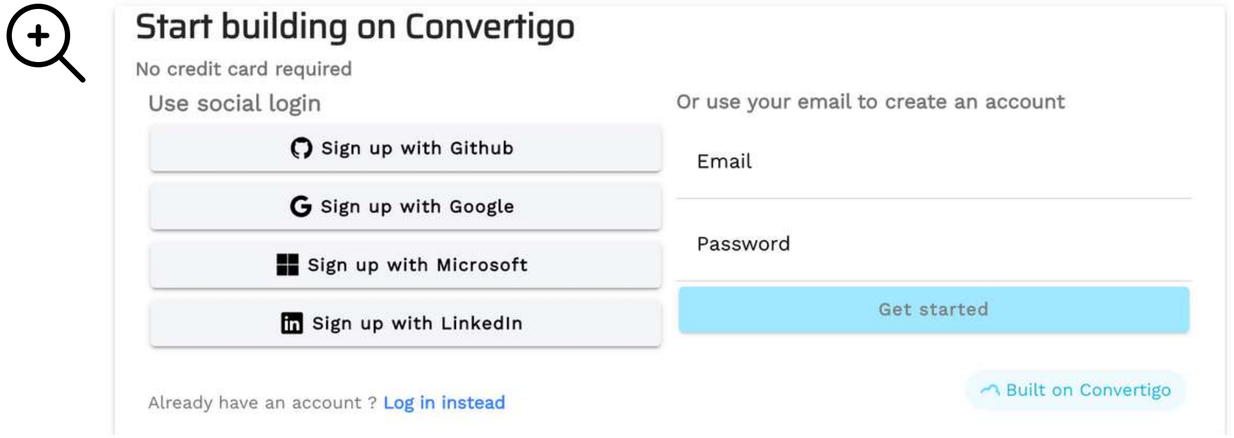


2.4 Configuration

Step 5 – Register with Convertigo Cloud Trial

Complete your registration with **Convertigo Cloud Trial**
by entering your **email** and a **password**, or using a **Credential provider**.
It will create your **account** on **Convertigo Cloud Trial**.





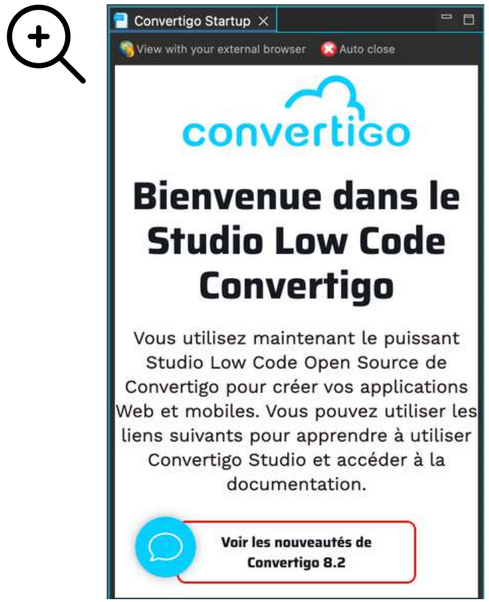
If you **create other workspaces**,
you will just need to **log in to this account**.

2.4 Configuration

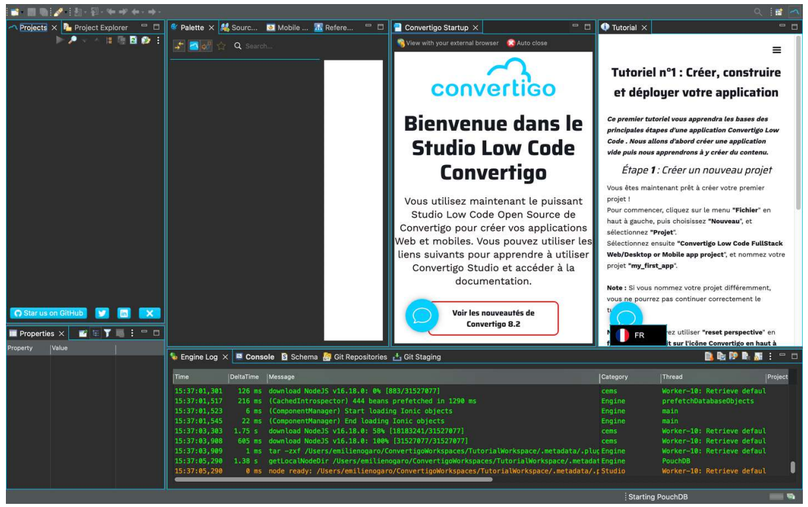
Step 6 – Welcome to Convertigo Low Code Studio

On first lauch, you will find 2 additionnal views :

The **Convertigo Startup view** with a link to **Convertigo's website** and the studio's **documentation**.



The **Tutorial view** featuring **exercises** to help you **getting started** with the studio.

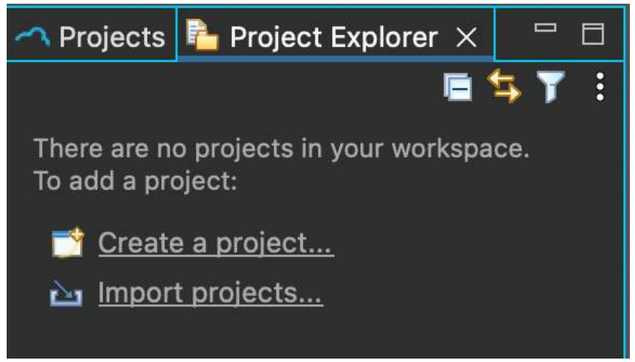


2.5 Create a project

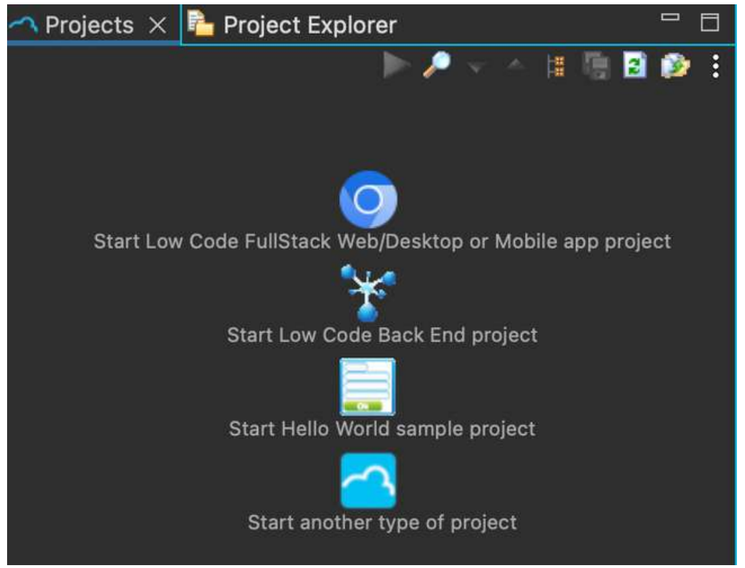
There are several ways to create a project in Convertigo.

When you create a project **for the first time** in a **new workspace**,
you can :

First option :
click on **Create a project**
in the **Project Explorer view**

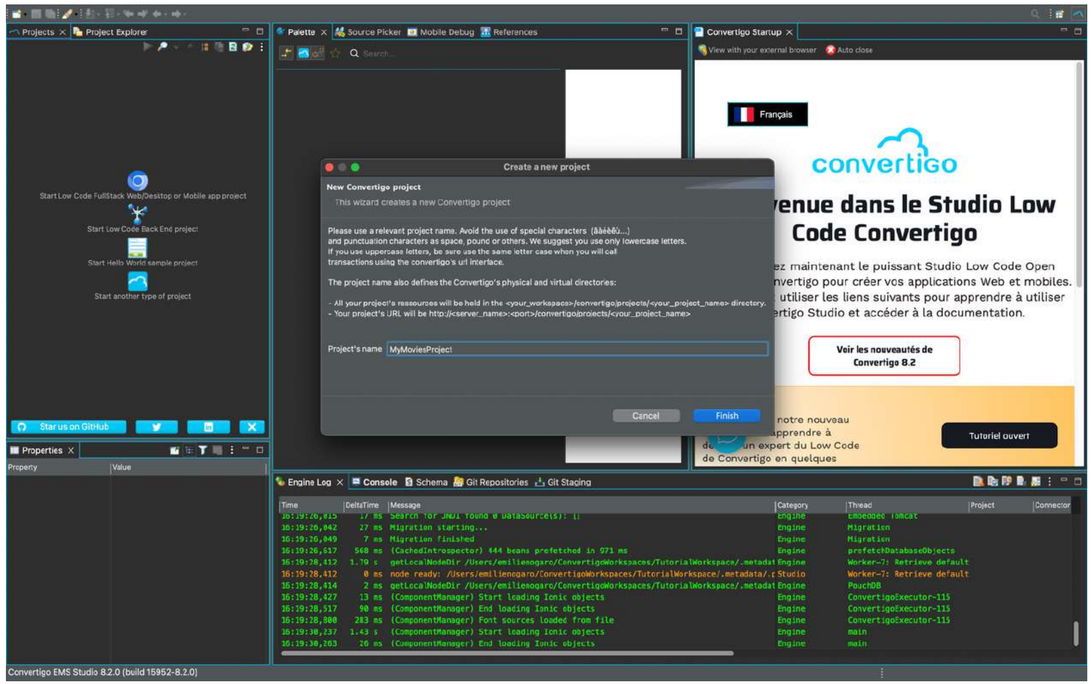


Second option :
click on
Start Low Code Fullstack Web/Desktop
or Mobile app project
in the **Project view**

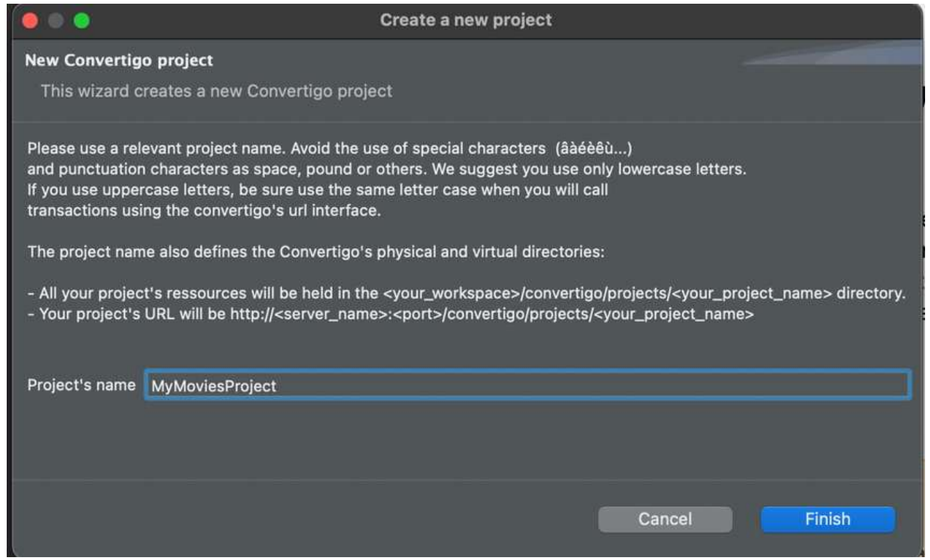


2.5 Create a project

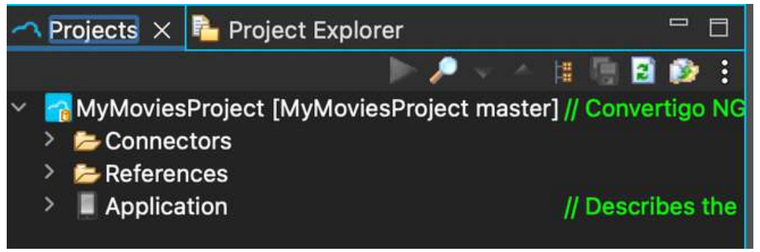
The **Create a new project windows** appears.



Enter a project name, then click on **Finish**.

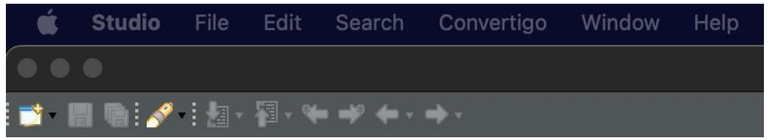


The project appears in the **Project view**.



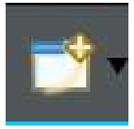
2.5 Create a project

Another way to create a project is to use the toolbar in the **Project view**.



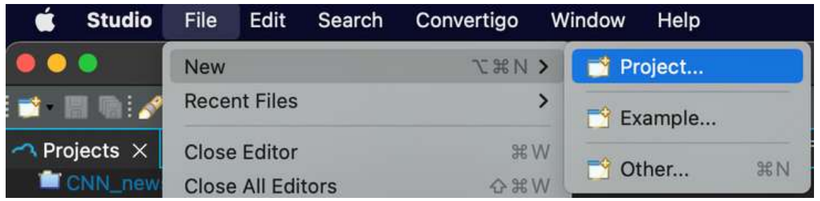
First option :

Click directly on this icon.



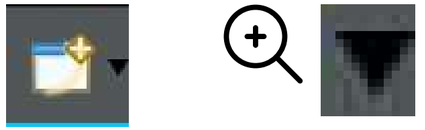
Click on **New>**

then select **Project**.



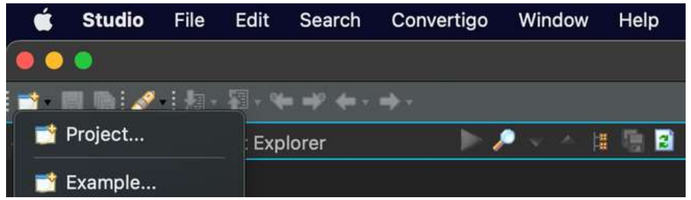
Second option

Click on the arrow on the right of this icon.



A sub menu appears.

Click on **Project** in the menu.

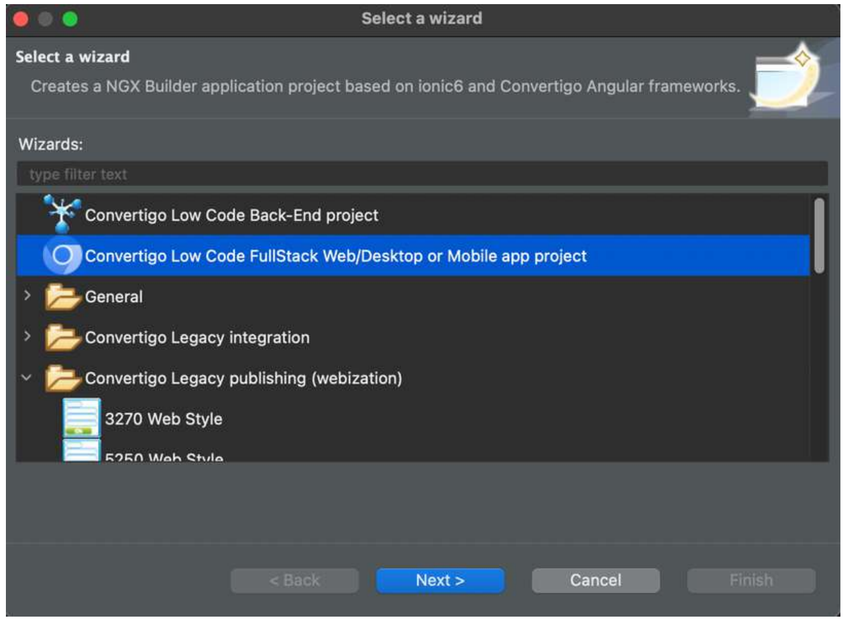


2.5 Create a project

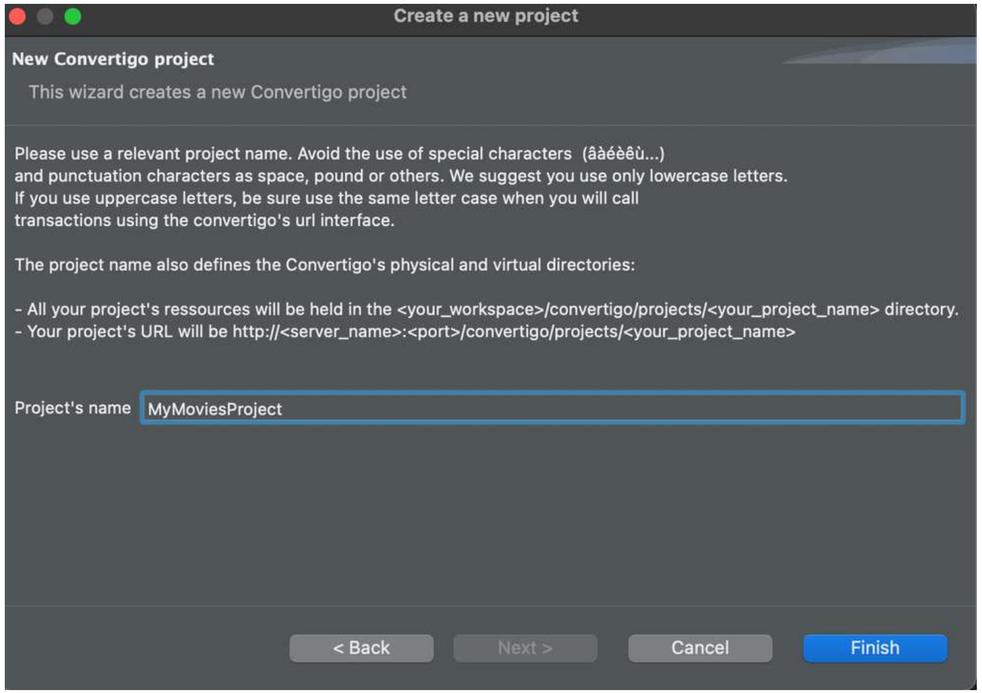
Both options open the **Select a wizard window**.

Select **Convertigo Low Code FullStack Web/Desktop or Mobile app project**

and click on **Next>**.

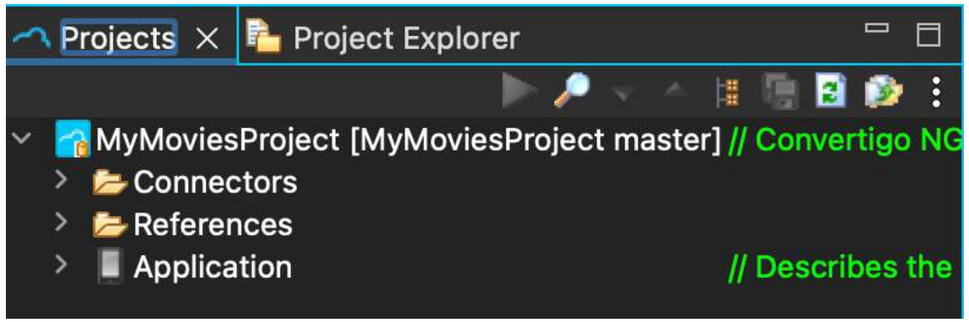


Enter a project name, then click on **Finish**.

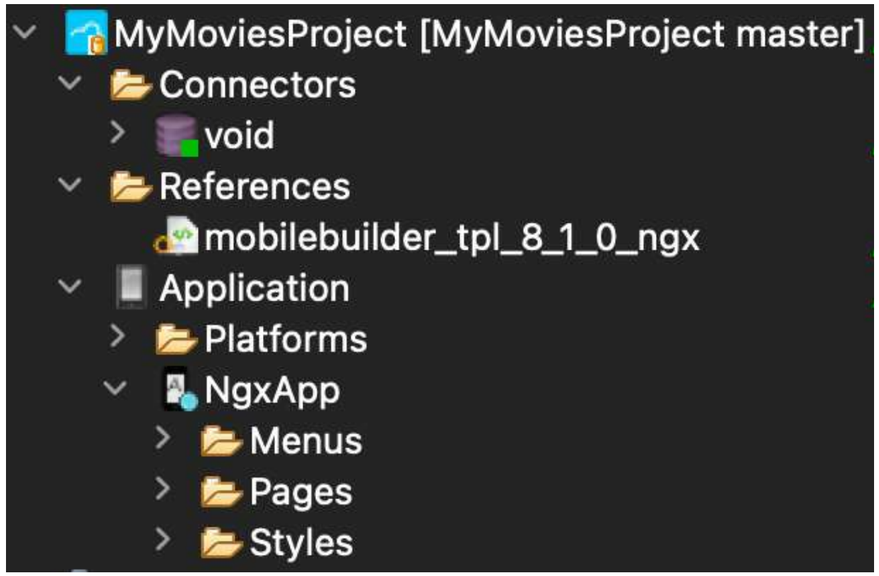


2.5 Create a project

As seen before, the project is created
and appears in the **Project view**.

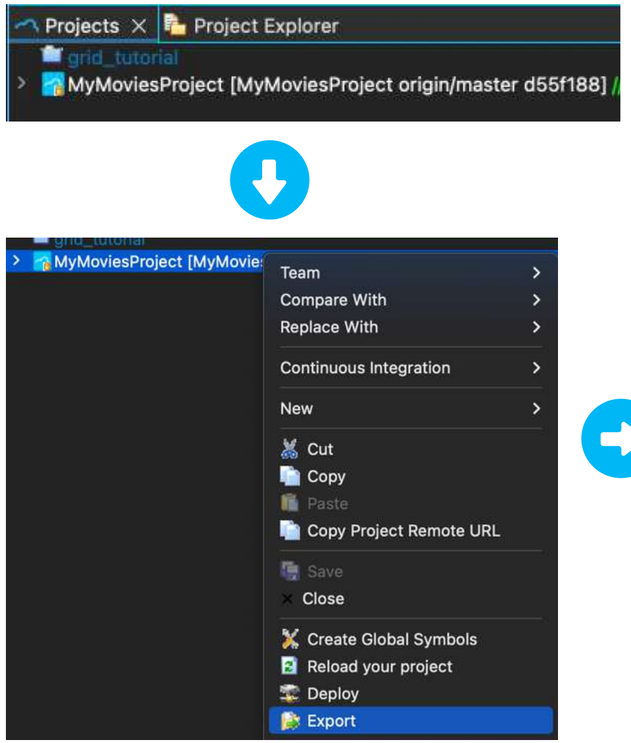


When created,
a project has always the same structure.

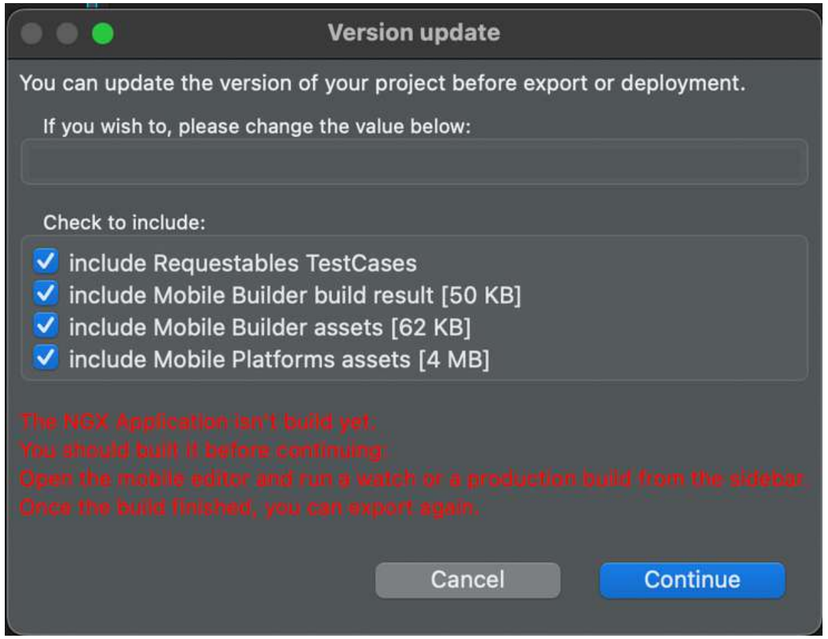


2.6 Export a project

Let's say you want to export a project:
right-click on the **name of your project** in the **Projects view**,
then click on **Export**.



The **Version update window** appears.



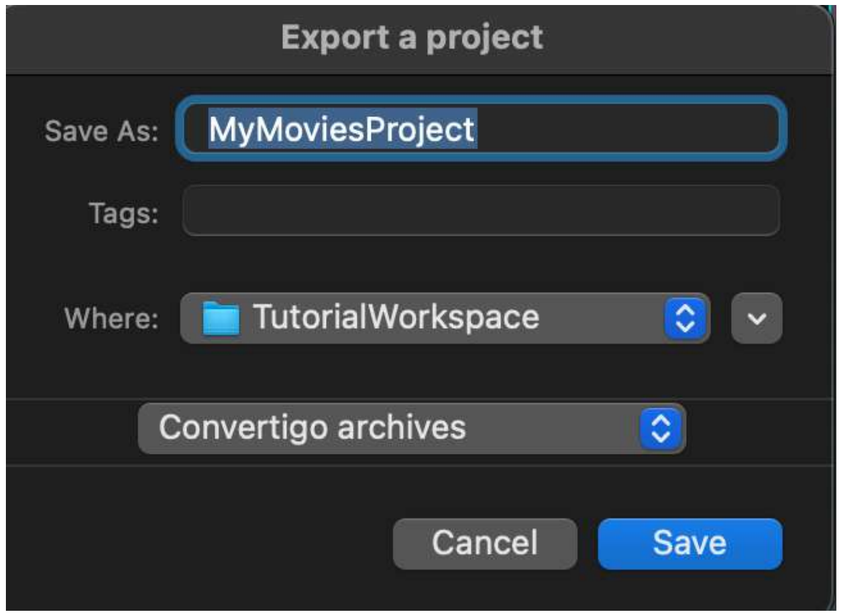
A message in red indicates that you need to run **a watch or a production build** from the **mobile editor.**

Building the project
is necessary **only for the frontend**.

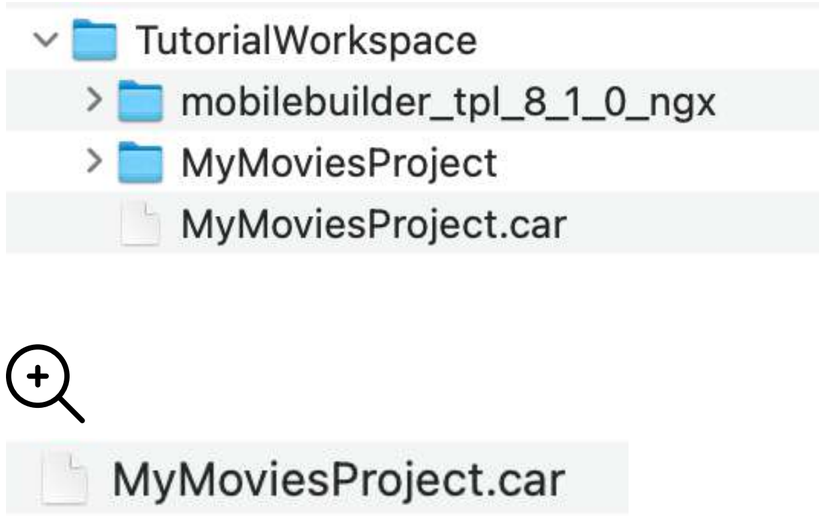
For now, we are **working on the backend**, so we can ignore this message,
and click on **Continue**.

2.6 Export a project

In the **Export a project window**,
you can change the name of the project,
and select where it is saved.

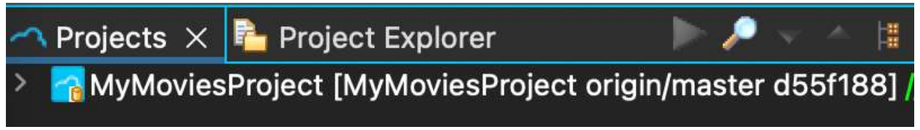


In the folder where it was saved,
the project appears as a **.car file**.

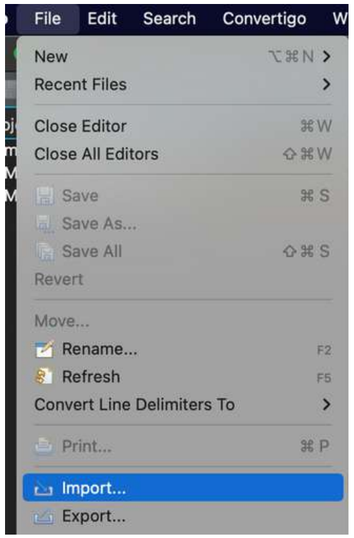


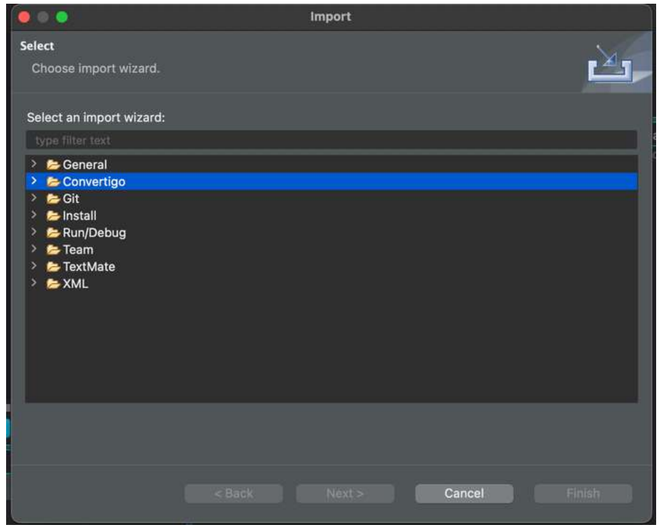
2.7 Import a project

Let’s say you want to import a project from a .car file in your workspace

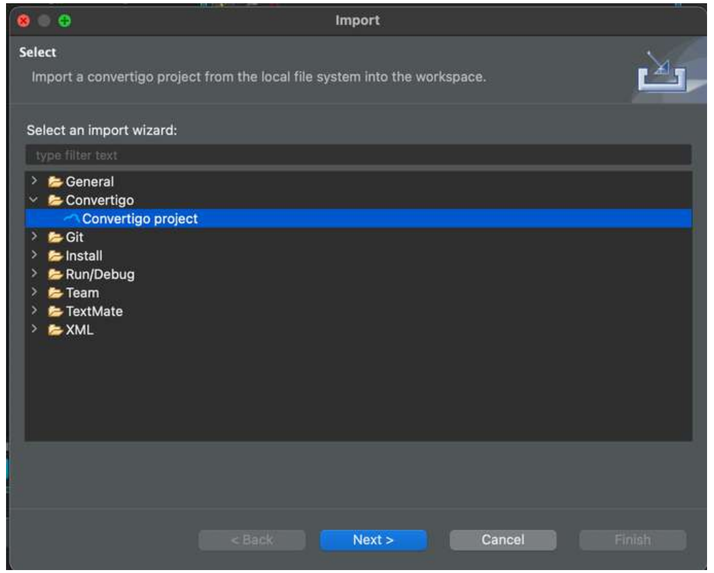


Click on **File**, then **Import** and the **Import windows appears**.



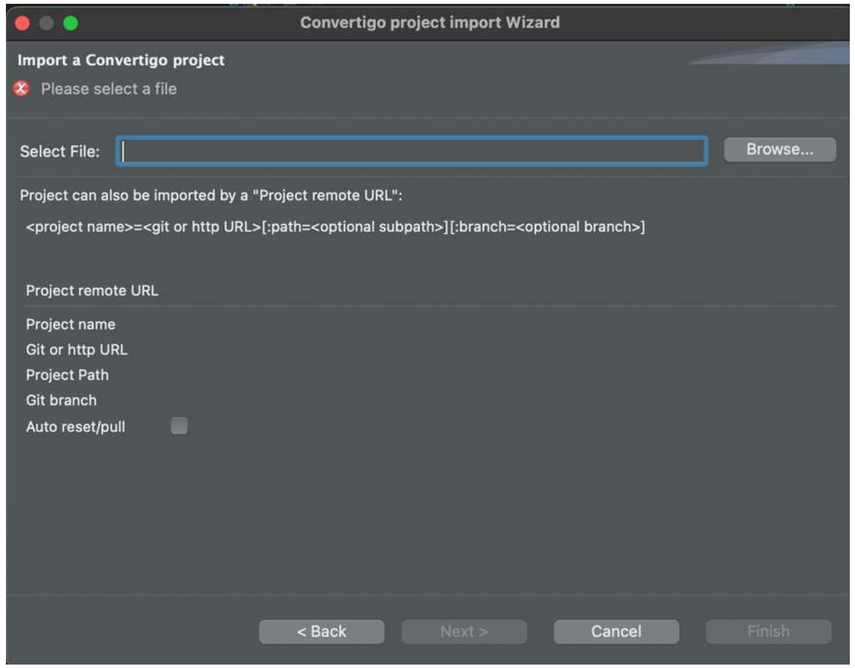


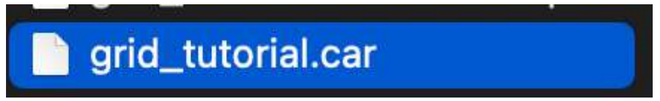
In the **Import windows**, click on **Convertigo**, select **Convertigo project**, then click on **Next>**.



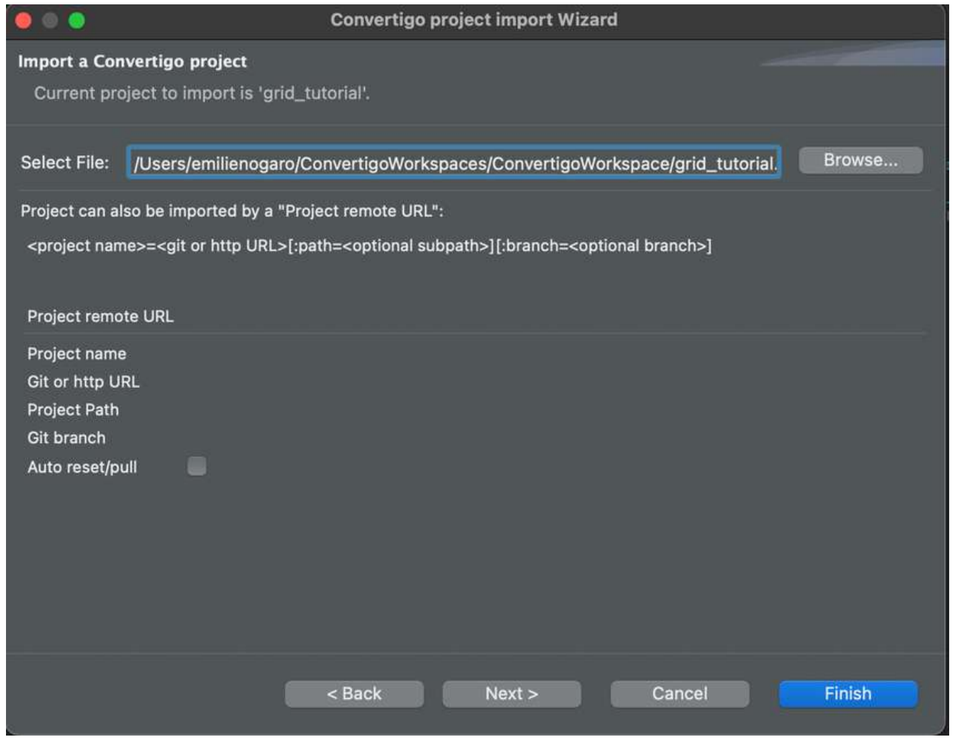
2.7 Import a project

In the **Convertigo Project Import window**,
click on **Browse** to select a file (here grid_tutorial.car)
anywhere in your computer.





Then click on **Next>**.

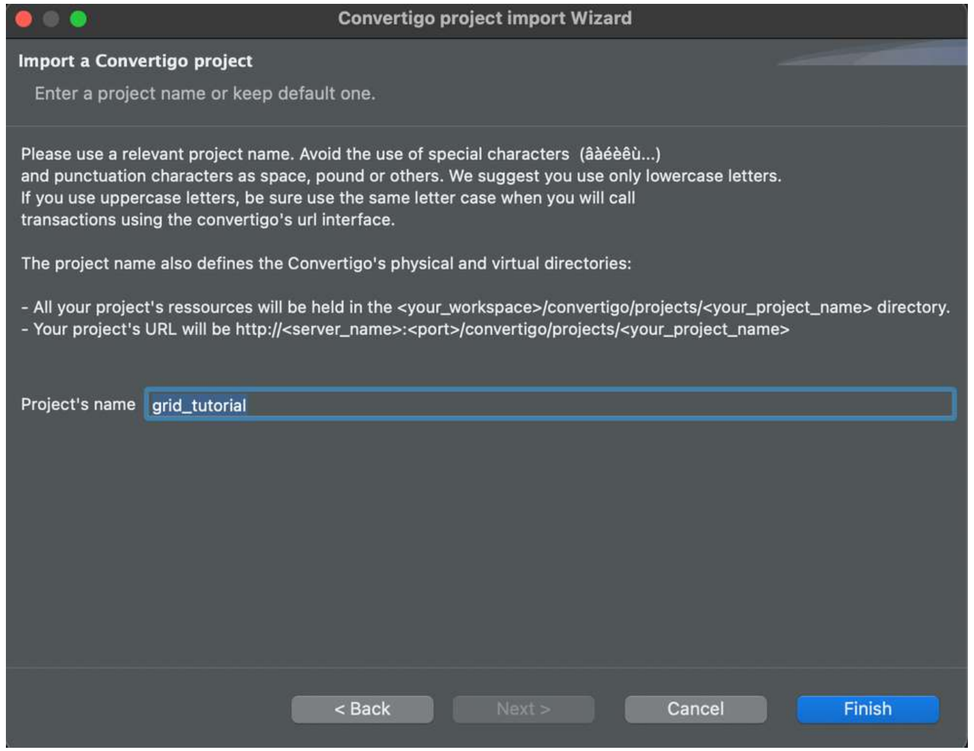


2.7 Import a project

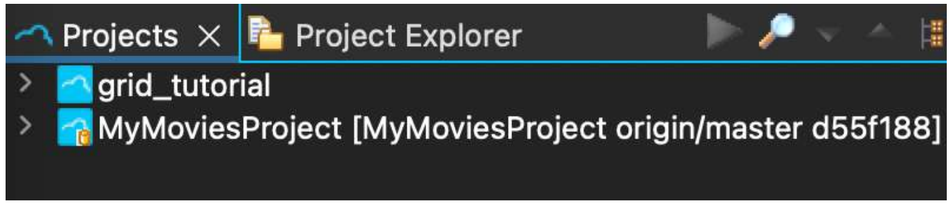
You can rename the project

or keep the .default file name (.car file name).

Then click on **Finish**.



In the **Projects view**, the project appears.



convertiGO

3 – Web services Connectors & Transactions

How to consume a rest API.

3.1 Presentation of the API TMDB

3.2 HTTP connectors & JSON HTTP transactions

3.3 Create an HTTP connector

3.4 Configure the HTTP connector

3.5 Create a transaction

3.6 Add a token

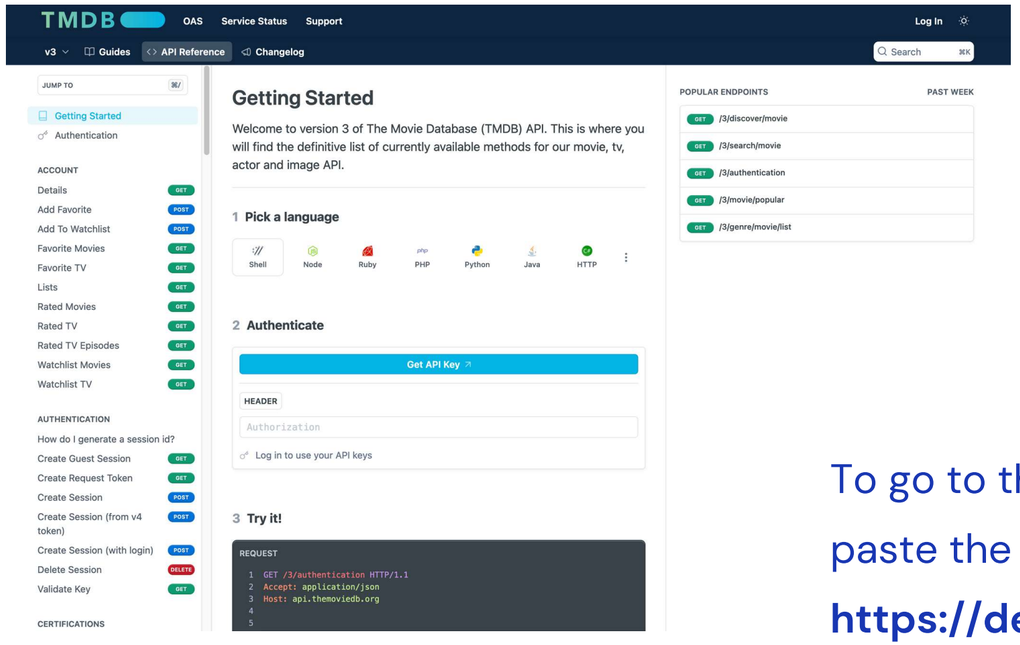
3.7 Edit the request path

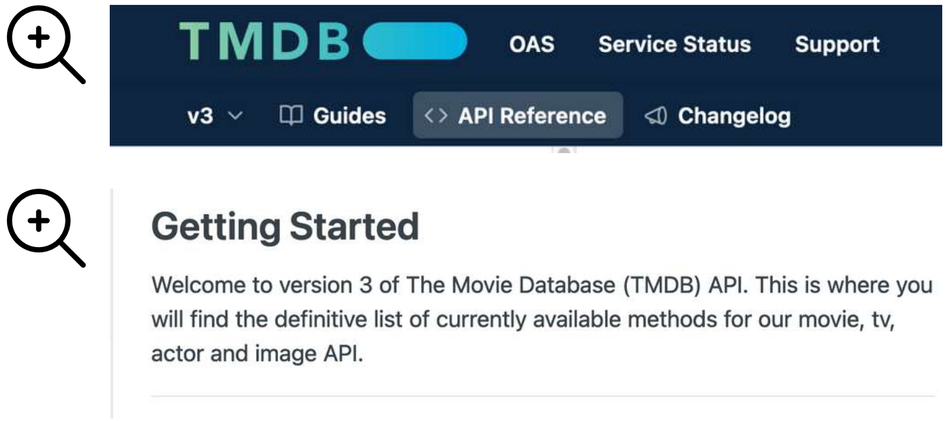
3.8 Test the request

3.1 Presentation of the API TMDB

The Movie Database (TMDb) API provides access to a vast database of information related to movies and television shows.

It is commonly used by developers to integrate movie-related data into their applications, websites, and services





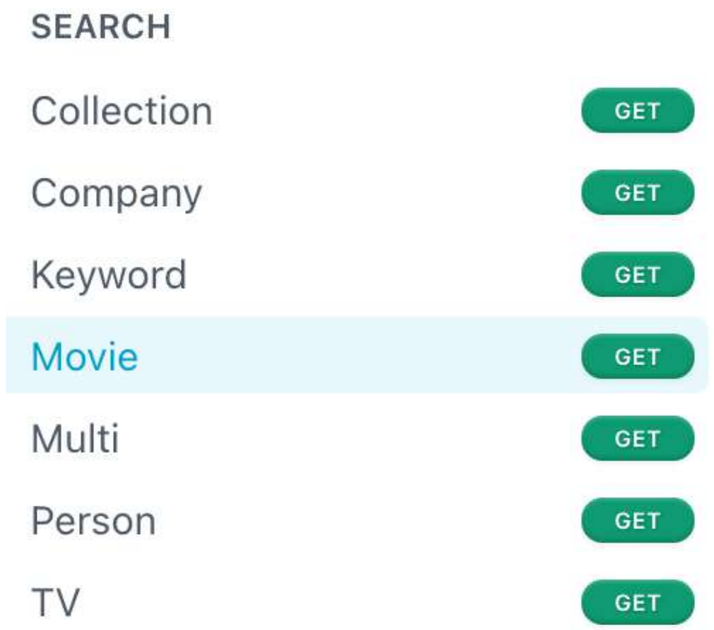
To go to the **Getting started page** of the API, paste the following link in your browser:

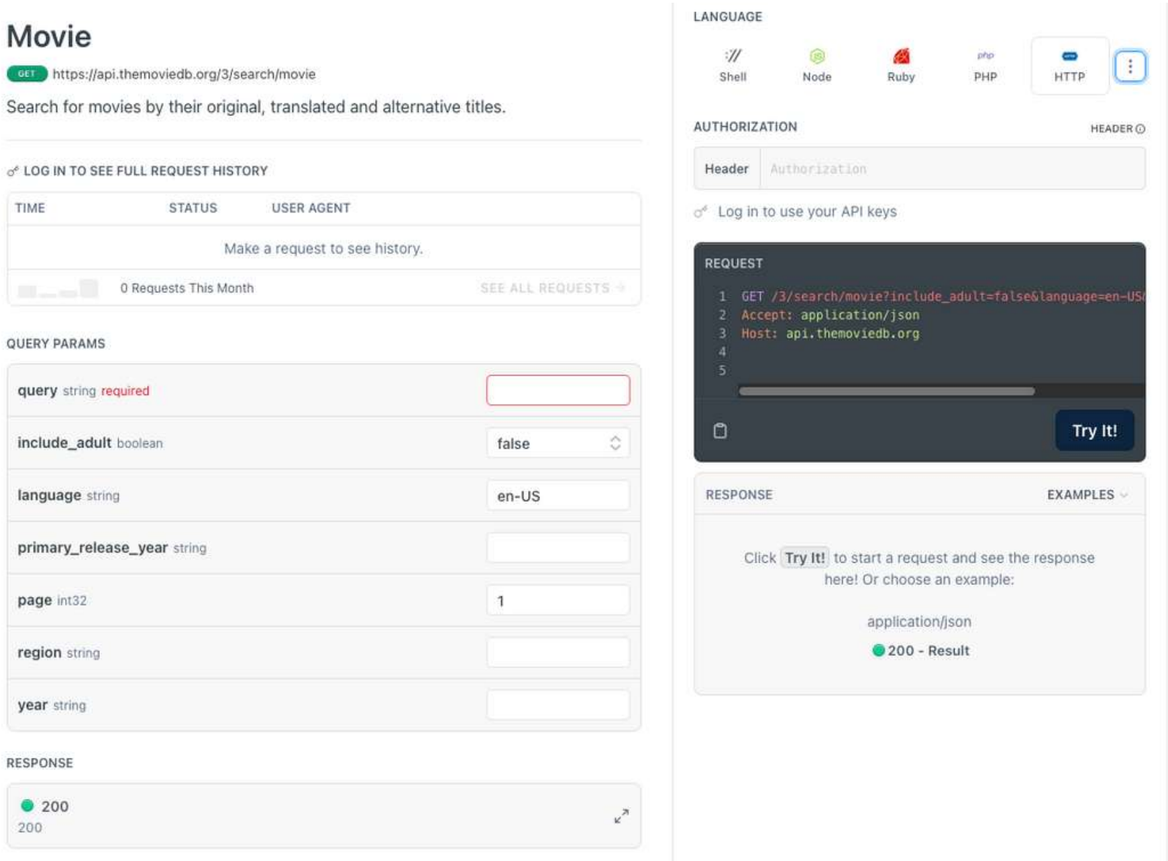
https://developer.themoviedb.org/reference/intro/getting-started

3.1 Presentation of the API TMDB

In the API TMDB documentation, a lot of different requests are available.

Let's go to the **Search Movie page** (https://developer.themoviedb.org/reference/search-movie).





3.1 Presentation of the API TMDB

convertiGO

All the informations you need to write a **Search Movie HTTP REQUEST** are present on the **Search Movie page**.

GET HTTP Request url to search a movie

Movie

GET https://api.themoviedb.org/3/search/movie

Search for movies by their original, translated and alternative titles.

Expected Response code

RESPONSE

200

200

Required and Optional Query params.

QUERY PARAMS

query string required

include_adult boolean

language string

primary_release_year string

page int32

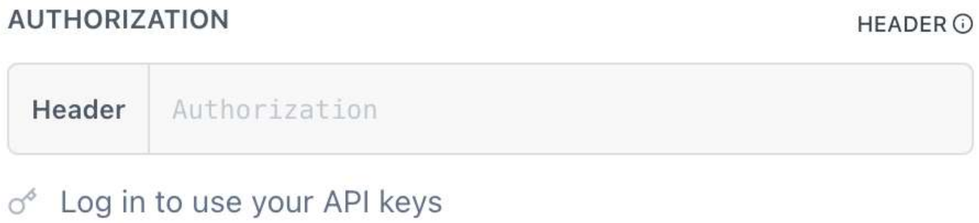
region string

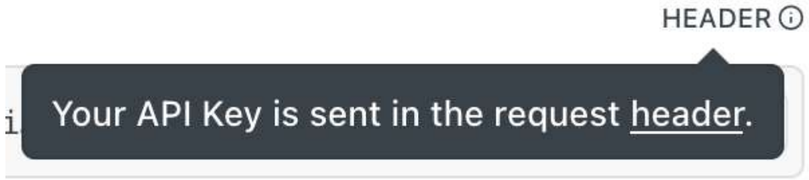
year string

3.1 Presentation of the API TMDB

To use the API TMDB, it is **necessary to create an account**.

Once registered, you will have an **API Key** or **personnal Access Token**.
It will be used in the **request Header** as **Authorization param**.





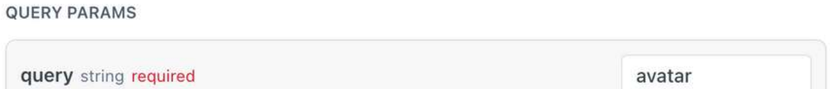
The personnal access token appears automatically when you are logged in.

3.1 Presentation of the API TMDB

Change the **Language to HTTP** to see the request as HTTP

Request url by default

GET /3/search/movie?include_adult=false&language=en-US&page=1



When you add a query param,

the **request url changes** to include it.

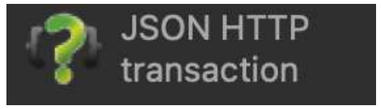
GET /3/search/movie?

query=avatar&include_adult=false&language=en-US&page=1

3.2 HTTP connectors & JSON HTTP transactions

There are **different connectors and transactions** in Convertigo,
used for **different data providers**
(SQL, Web services, Legacy apps running on mainframes...).

For a REST API,
you use the **HTTP connector**.
It allows Convertigo to connect and communicate with **HTTP servers**.
It is used to **consume REST and SOAP web services**,
and retrieve data using the **HTTP protocol**.



To consume a **JSON web service**,
you use a **JSON HTTP transaction**.
It performs the conversion of JSON data from the web service
into XML transaction output.

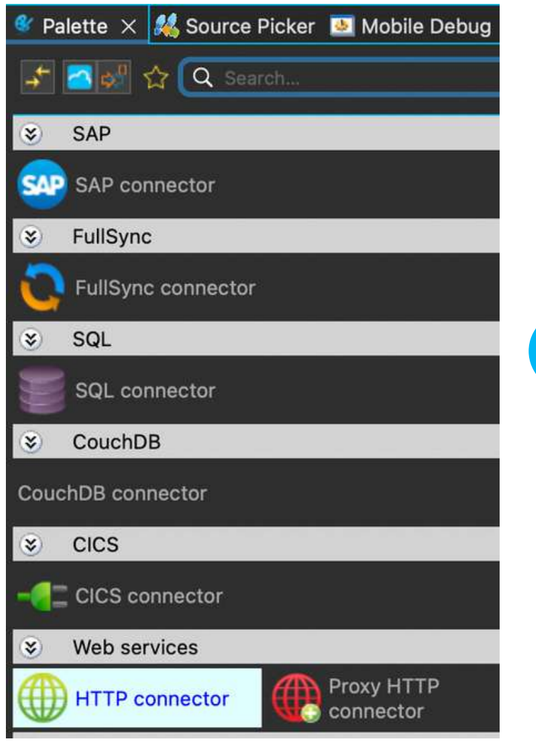
3.3 Create an HTTP connector

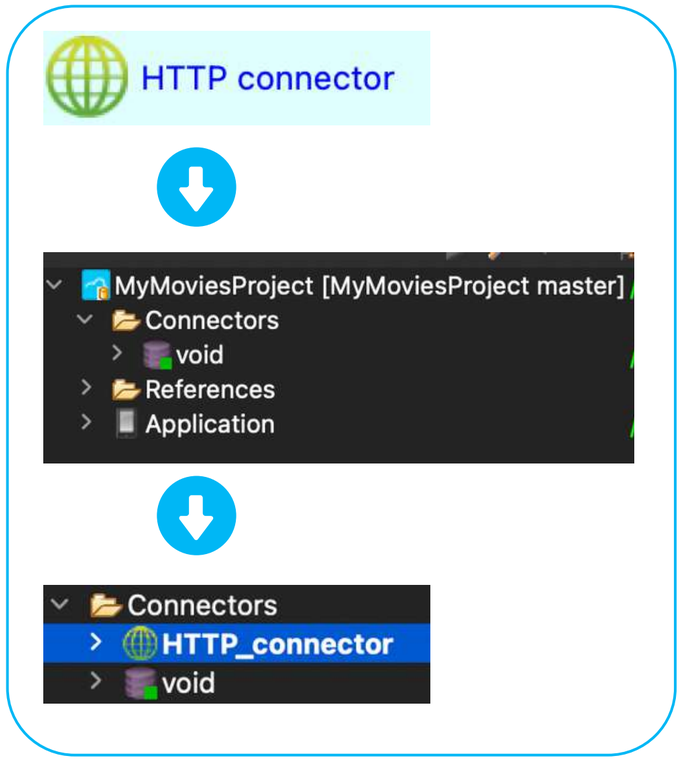
To connect to a **REST API**,

you need to **create an HTTP_connector** in the **Connectors folder**.

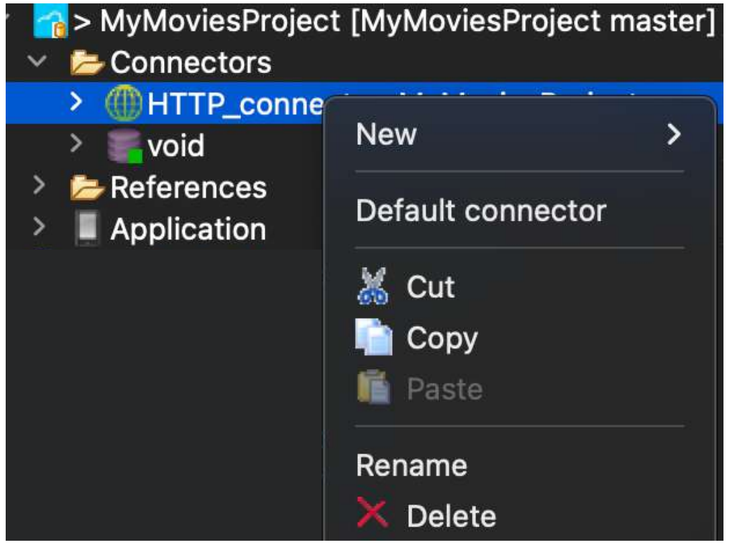
First option:

Drag and drop it from the palette into the folder.





You can then **rename** the connector by right-clicking on it.

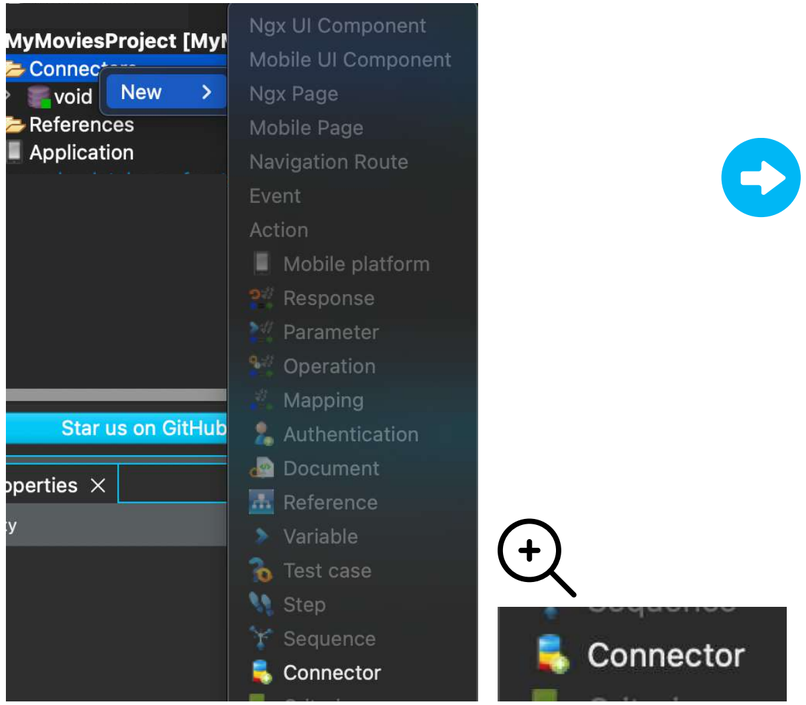


3.3 Create an HTTP connector

Second option:

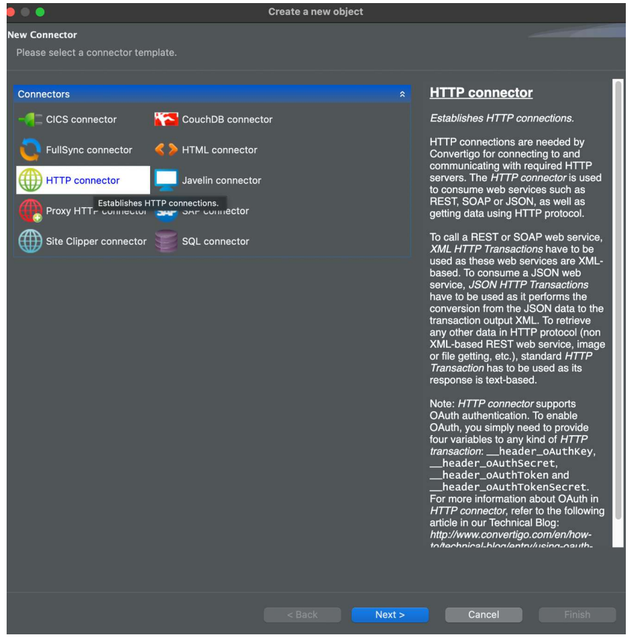
Right-click on the **Connectors** folder,

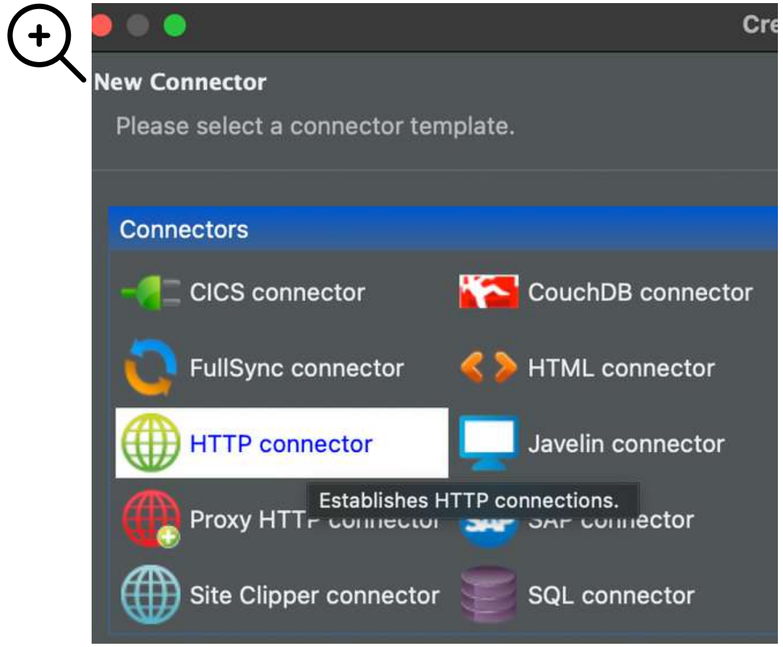
then select **New** and choose **Connector**.



In the **Create a new object** window,

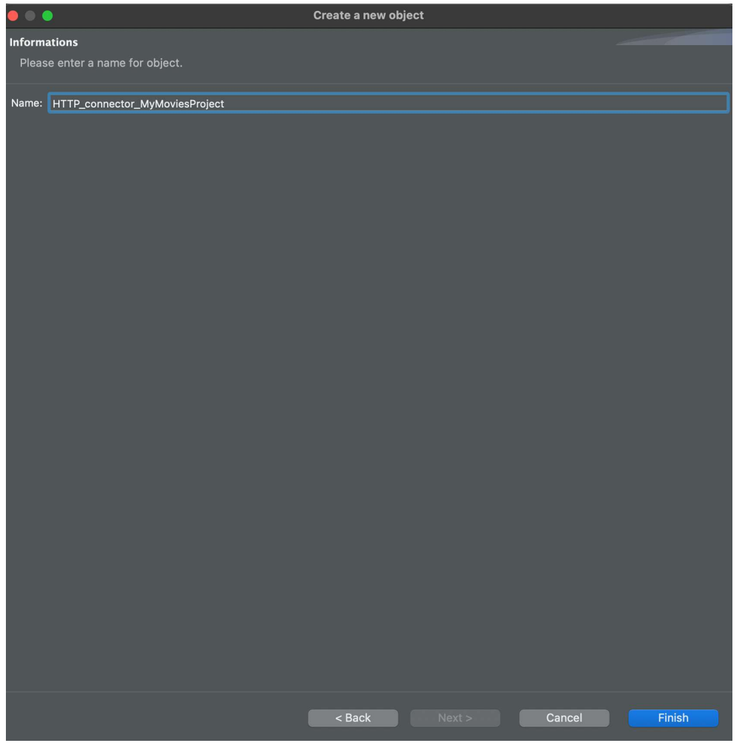
select **HTTP connector** and then click on **Next>**.

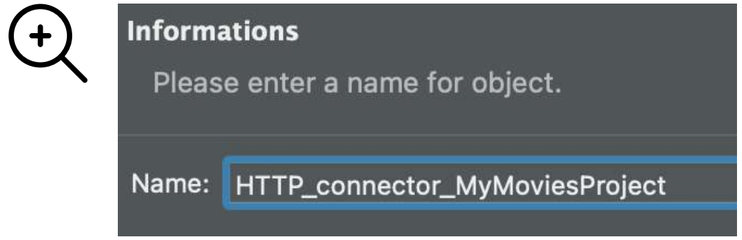




3.3 Create an HTTP connector

Choose a name for the connector, and click on **Finish**.



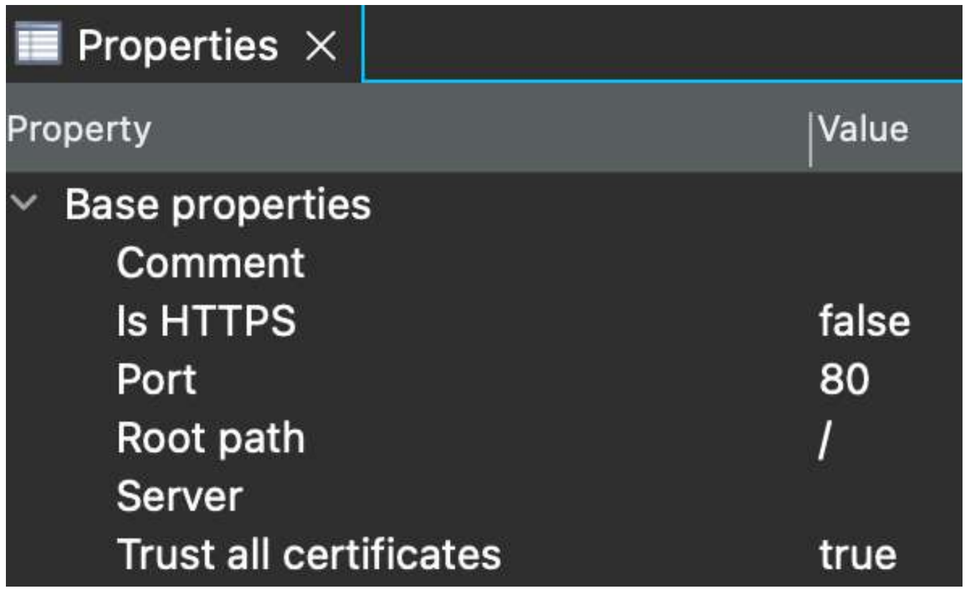


The new connector is created in the **Connectors** folder.

3.4 Configure the HTTP connector

In the **Properties window**,

you will find the **default properties** of the connector.



For **http requests**

- IsHTTPS : false
- Port : 80

For **https requests**

- IsHTTPS : true
- Port : 443

Root path : / (default path)

Server : => enter a server name

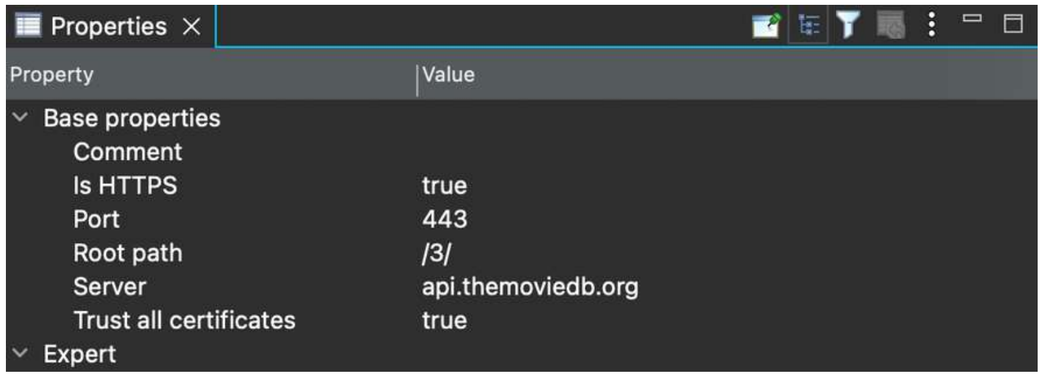
3.4 Configure the HTTP connector

Now, we need to configure the connector
with the informations found in the TMDB API documentation.

GET https://api.themoviedb.org/3/search/movie

```
REQUEST

1  GET /3/search/movie?include_ad
2  Accept: application/json
3  Host: api.themoviedb.org
```



In the TMDB API documentation, we can see that:

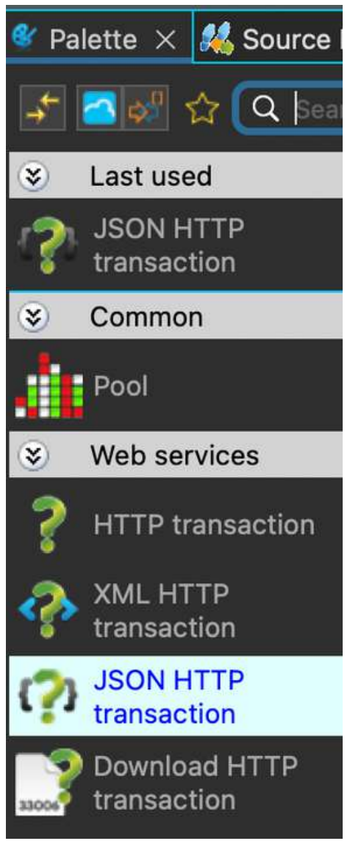
- the request is **https**
- the request has a root path : **/3/**
  (version 3 of the API)
- the domain name is **api.themoviedb.org**

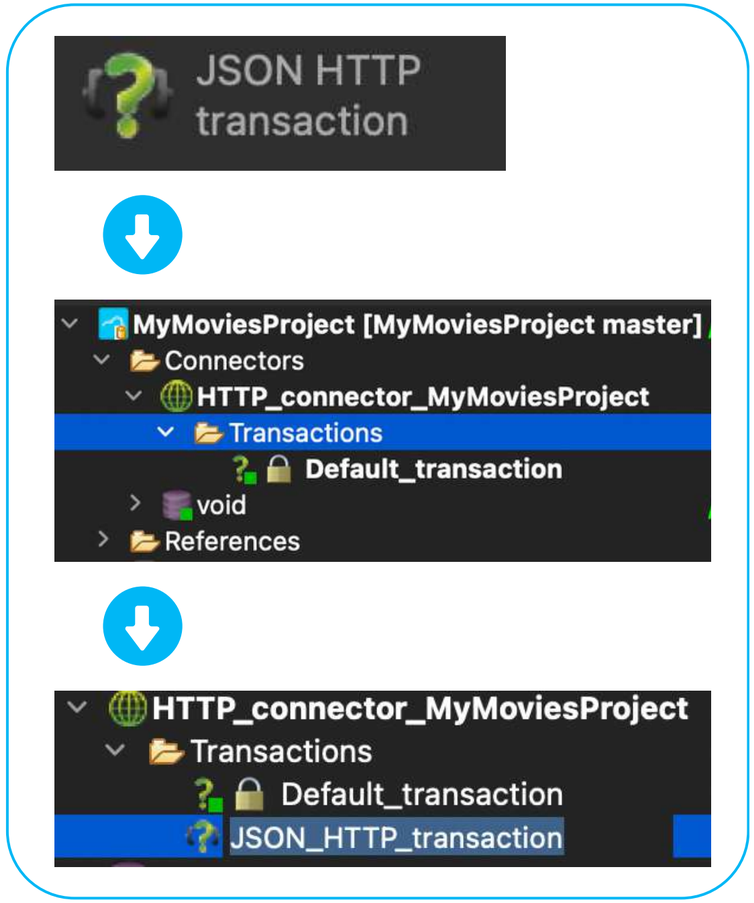
As a result, the **Connector configuration** is

- IsHTTPS : true
- Port : 443
- Root path : /3/
- Server : api.themoviedb.org

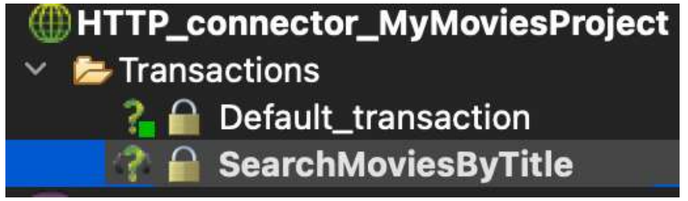
3.5 Create a transaction

First option : Drag and drop a **JSON HTTP transaction** from the palette into the **Connectors folder**.



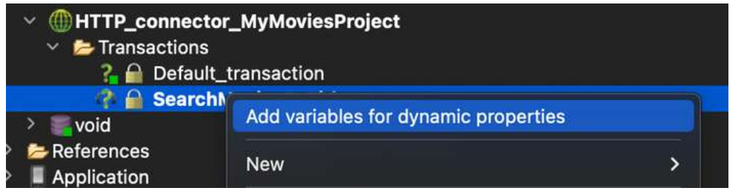


Rename the transaction to **SearchMoviesByTitle**.

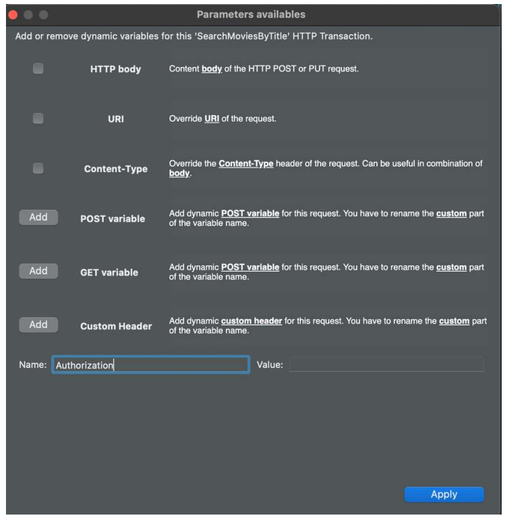


3.5 Create a transaction

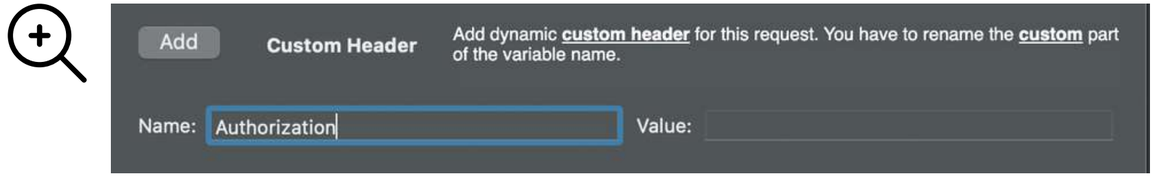
Right-click on the transaction, and select **Add variables for dynamic properties**.



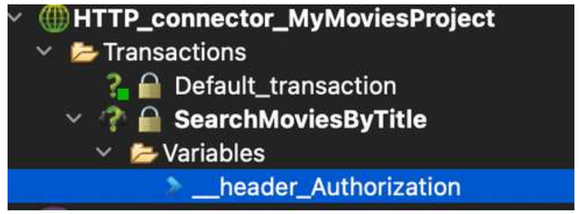
The **Parameters available window** appears.



In the **Parameters available window**,
click on **Add Custom Header** to add the **Authorization** header (which will allow sending the access token),



Then click on **Apply**.



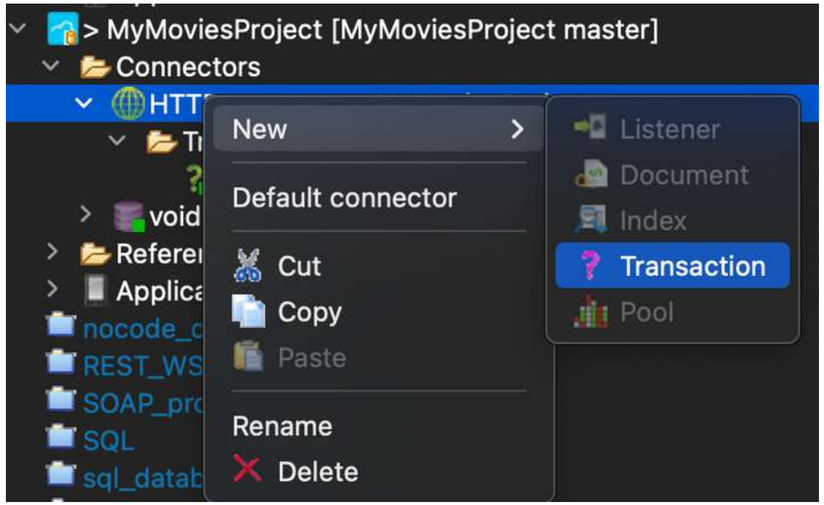
The **Authorization** header will appear in the folder **Variables** of the transaction

3.5 Create a transaction

Second Option :

Right-click on the connector,

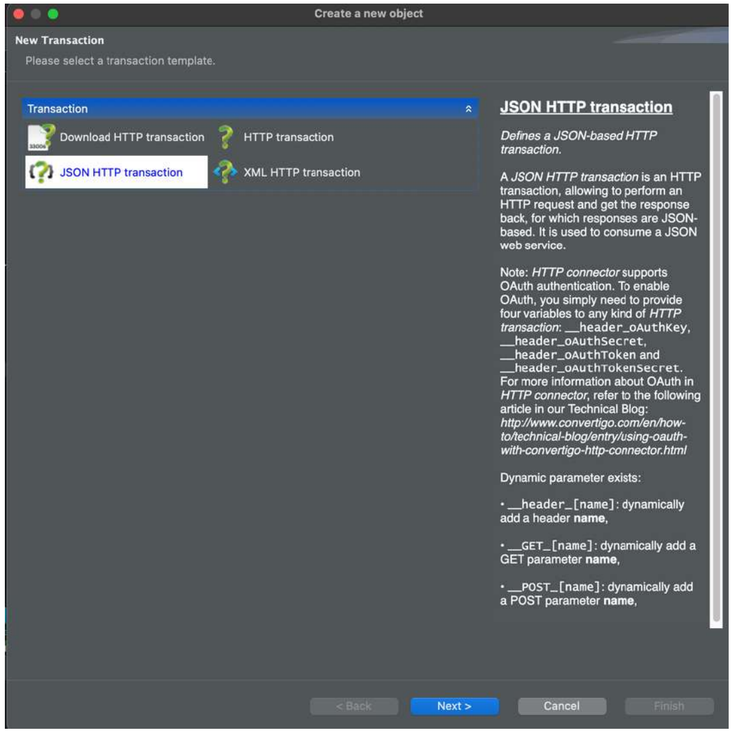
then select **New >**, then select **Transaction**.



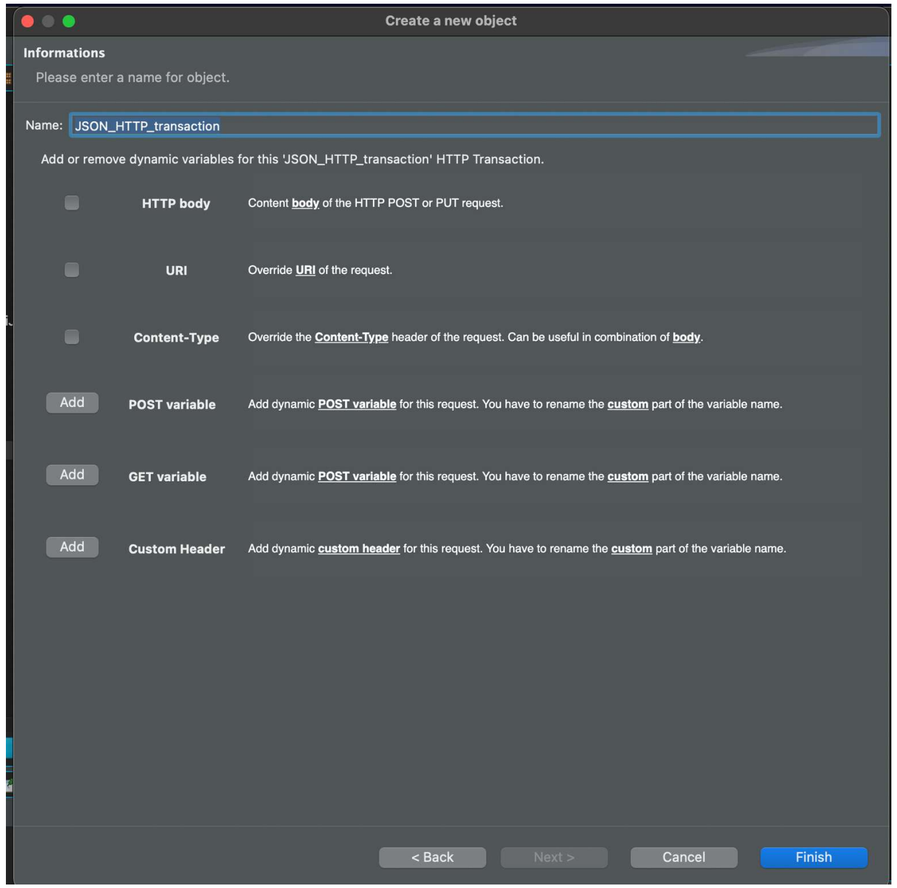
In the **Create a new object window**,

choose **JSON HTTP transaction**,

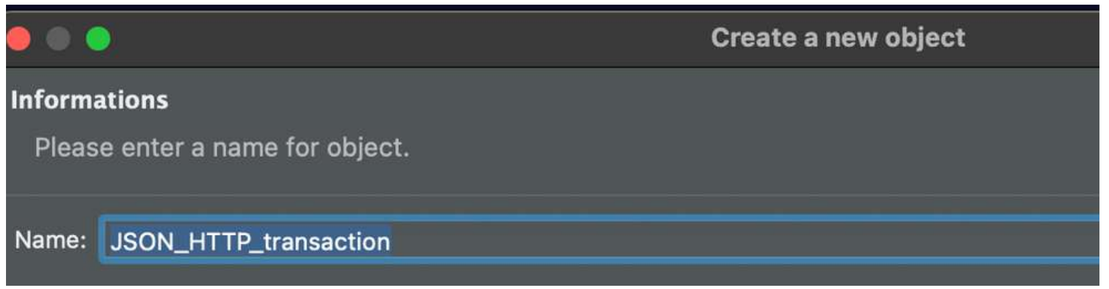
then click on **Next >**.



3.5 Create a transaction



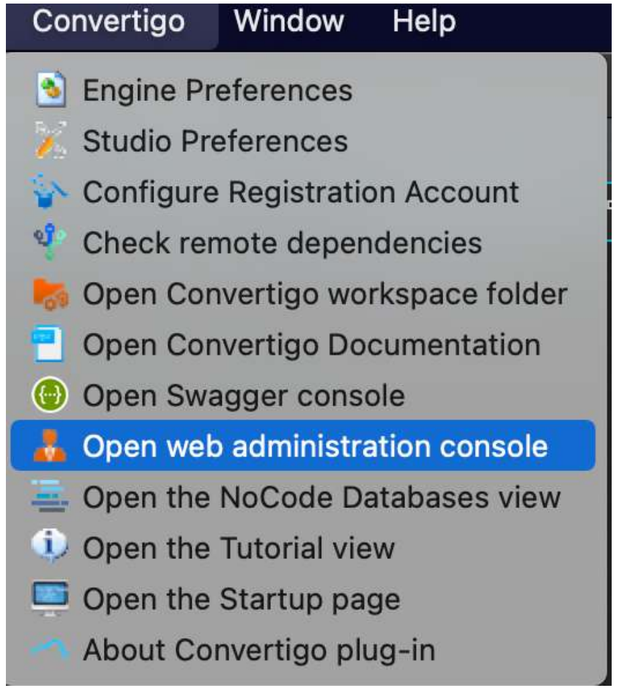
Rename the transaction with the name of the request.

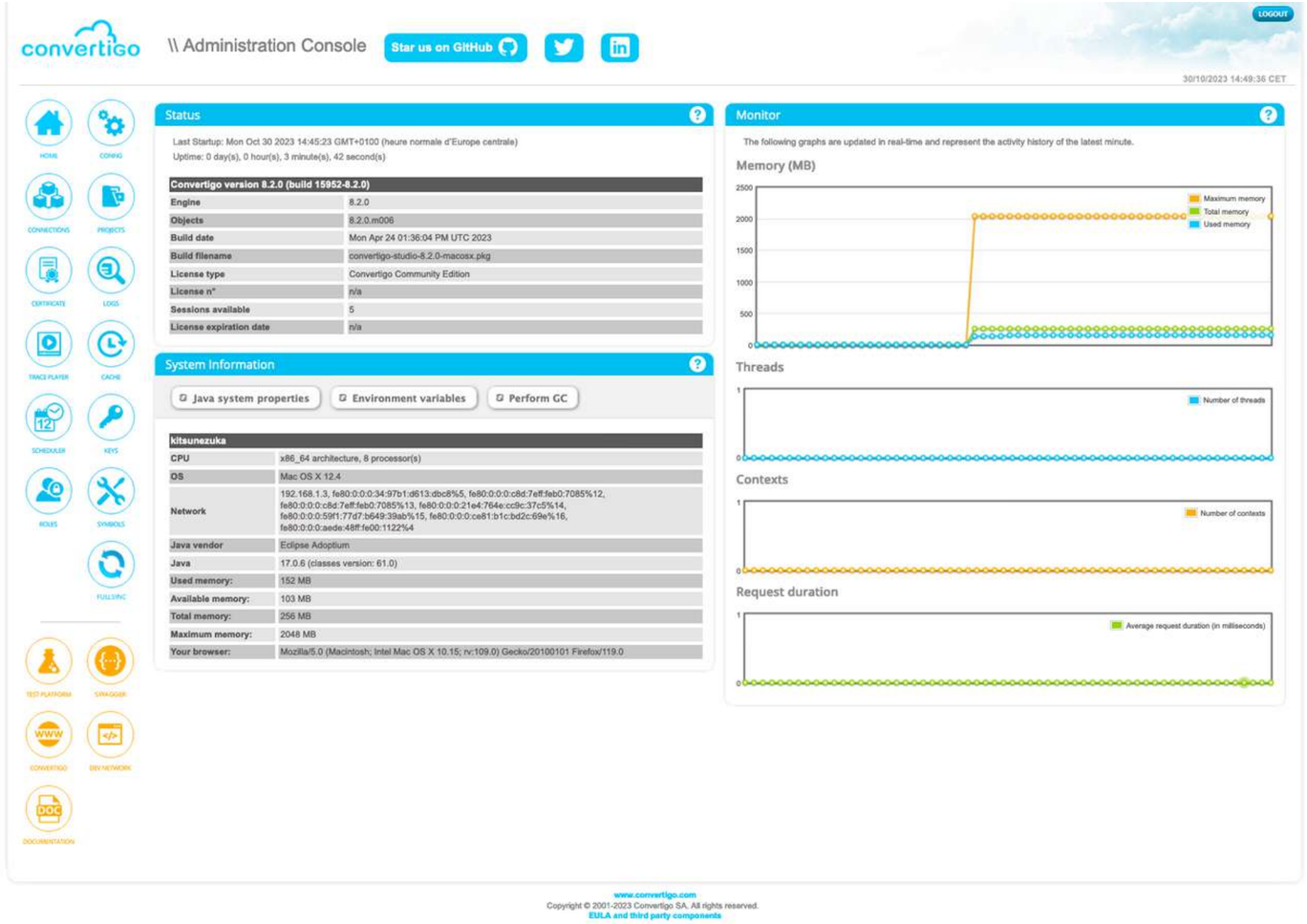


Then, follow the same steps as in the first option.

3.6 Add a token

To open the **web administration console**,
click on **Convertigo**,
then select **Open web administration console**.

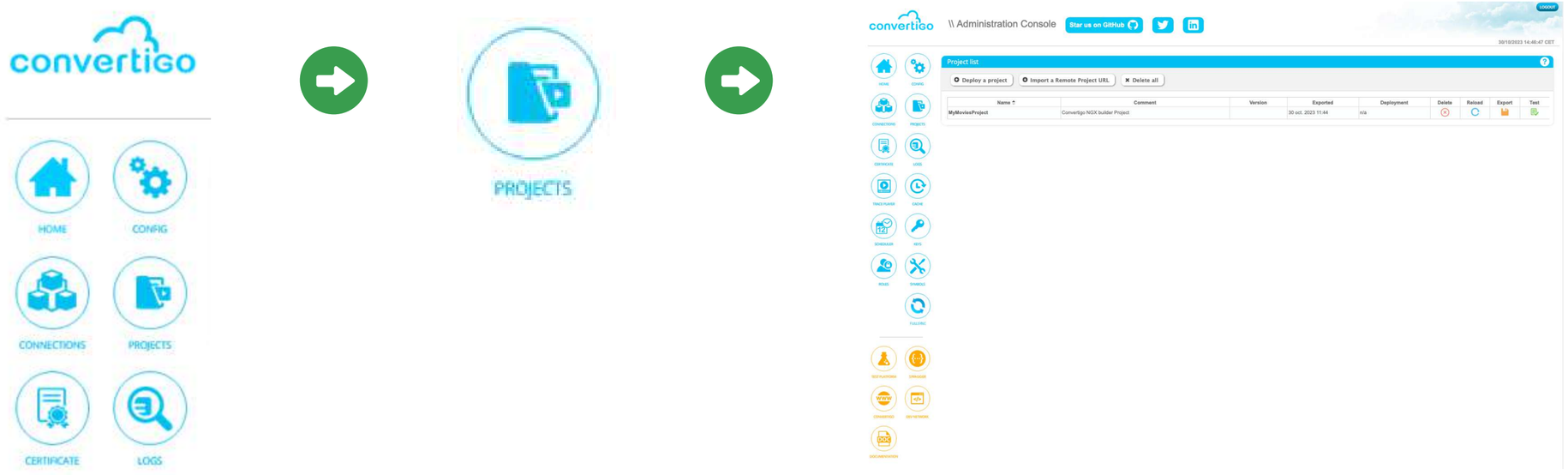




3.6 Add a token

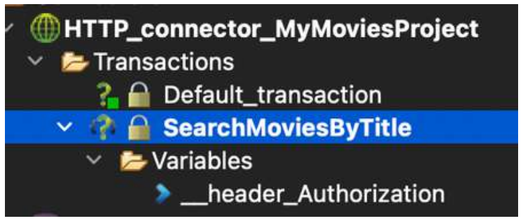
In the **web administration console**,

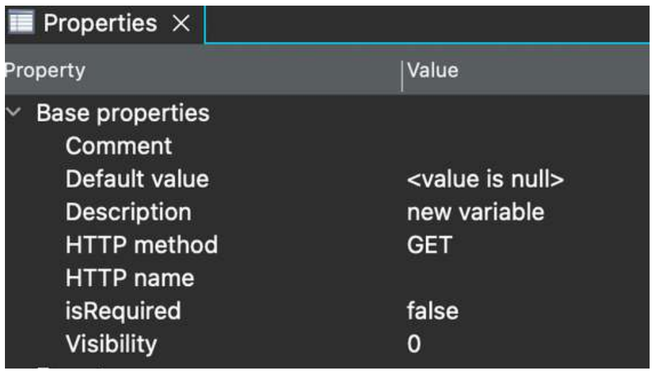
click on the icon **PROJECTS** to view the **projects currently opened** in the studio **workspace**.



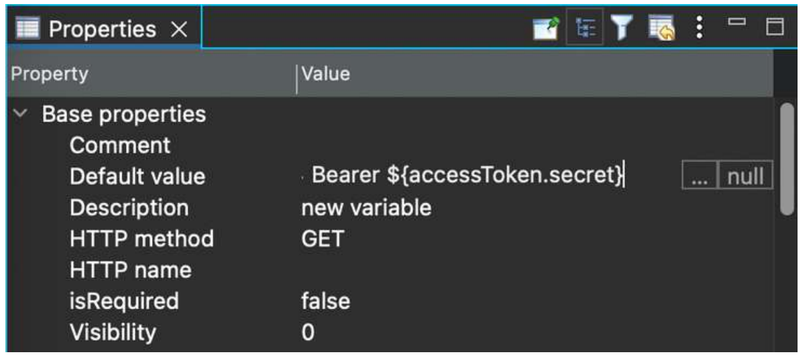
3.6 Add a token

Let’s have a look on the properties of the variable **__header_Authorization.**



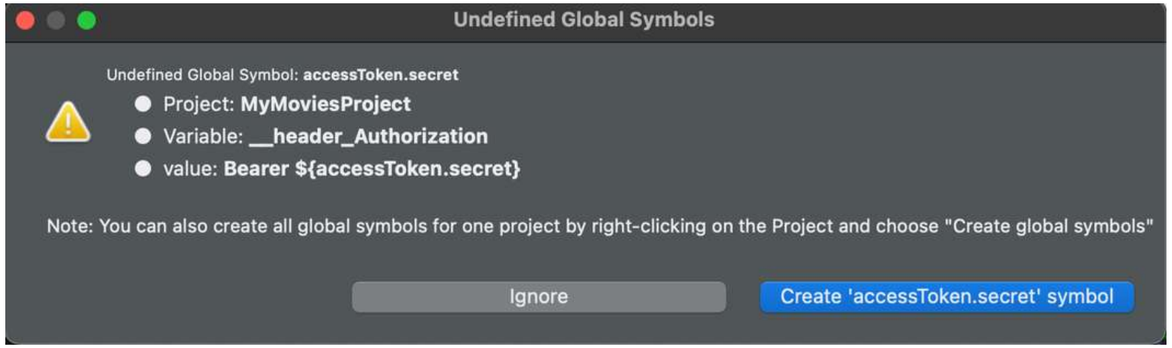


As **Default value**, enter **Bearer ${accessToken.secret}**.



Validating the value will open the **Undefined Global Symbols window**.

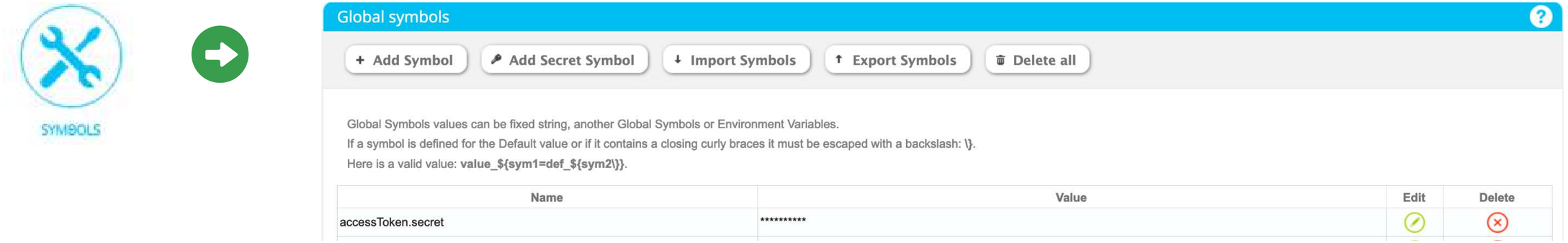
Click on **Create ‘accessToken.secret’ symbol**.

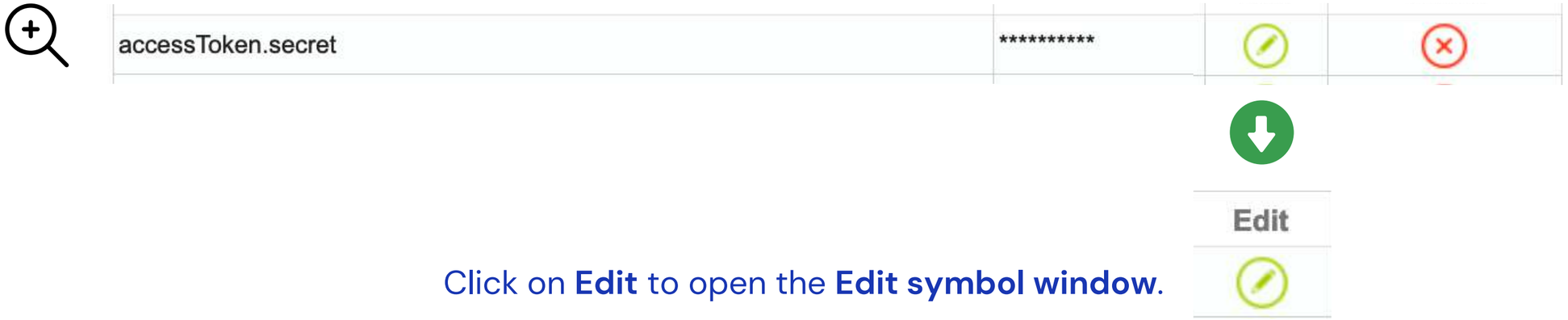


3.6 Add a token

In the web administration console,

click on **Symbols** to access the Global symbols.

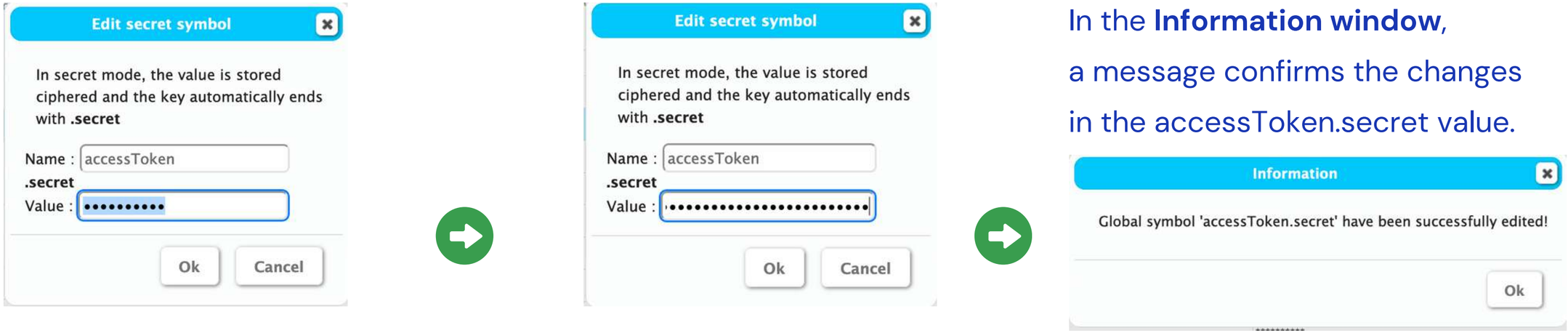




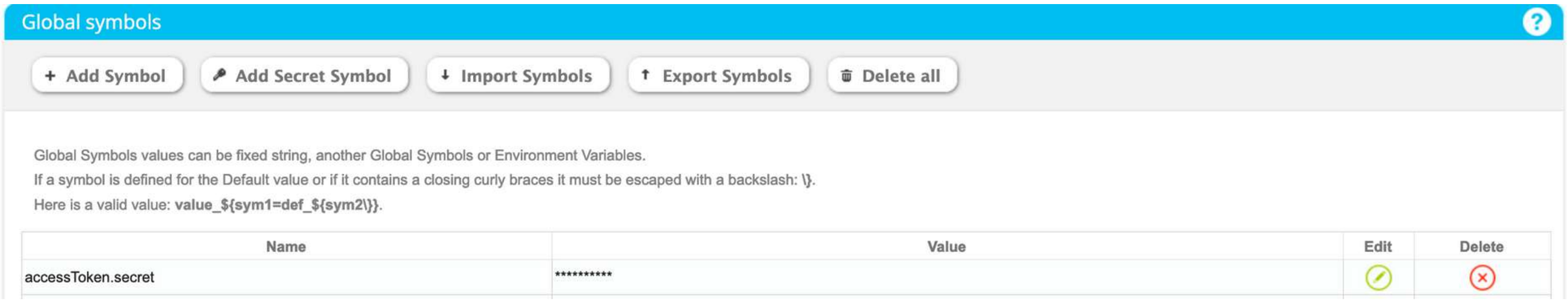
Click on **Edit** to open the **Edit symbol window**.

3.6 Add a token

In the **Edit symbol window**, change the **accessToken.secret** value.

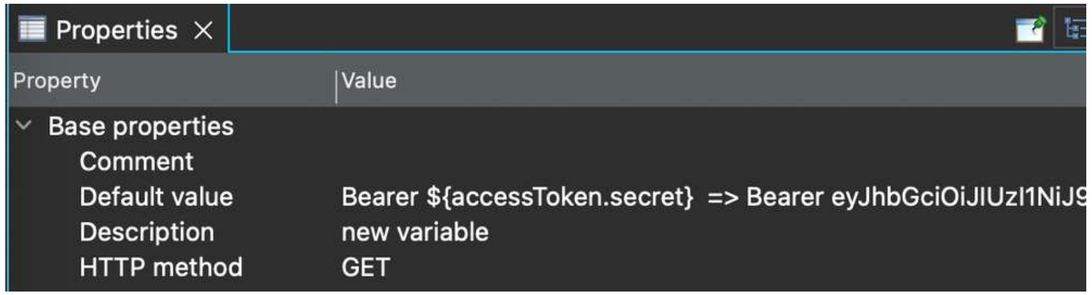


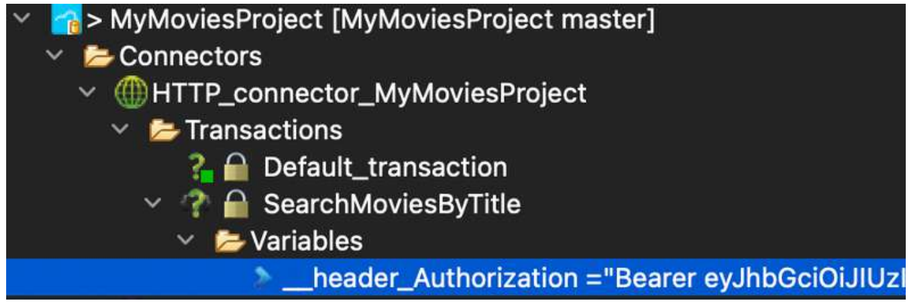
In the **Information window**, a message confirms the changes in the accessToken.secret value.



3.6 Add a token

In the studio, the value of the symbol appears in clear in the Properties of **__header_Authorization** and in the treeview's variable _header_Authorization.





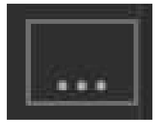
For security purposes,

the value of the symbol MUST BE HIDDEN.

3.6 Add a token

To hide the value of the symbol,

we need to change the **Visibility property**.



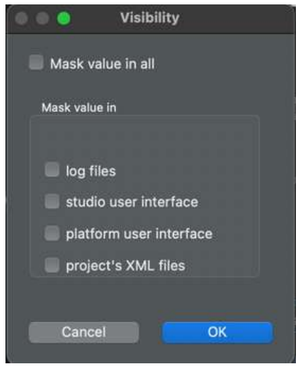
Click on the icon

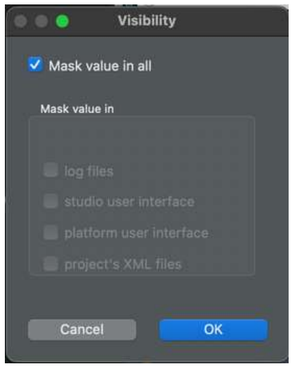
at the end of

the **Visibility property** line.

The **Visibility windows** appears,

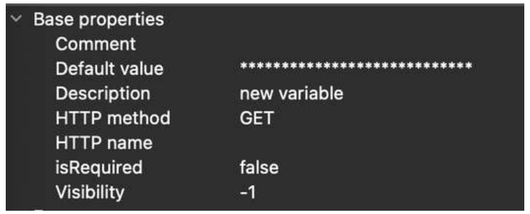
click on **Mask value in all**, then click on **OK**.



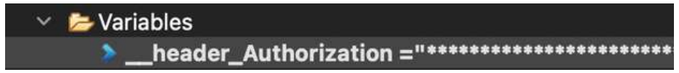


The value of **__header_Authorization** is hidden.

In Properties



And in the variables folder

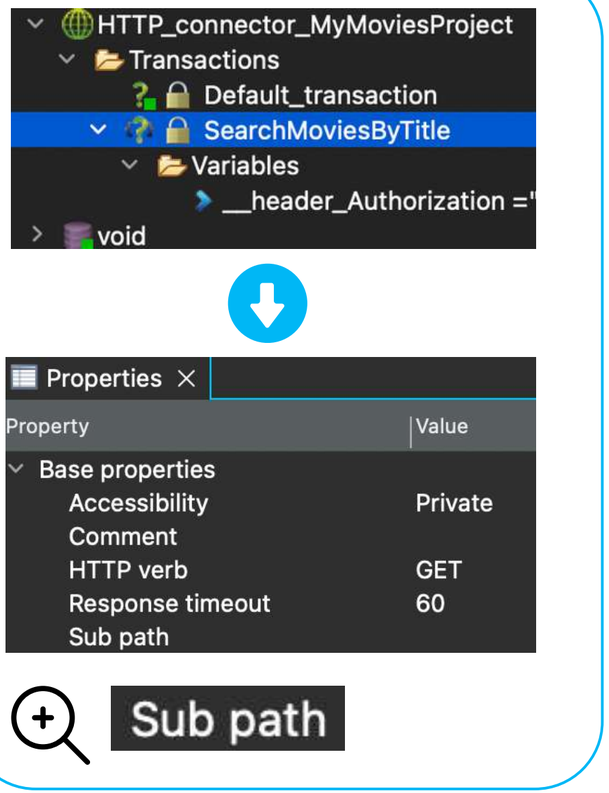


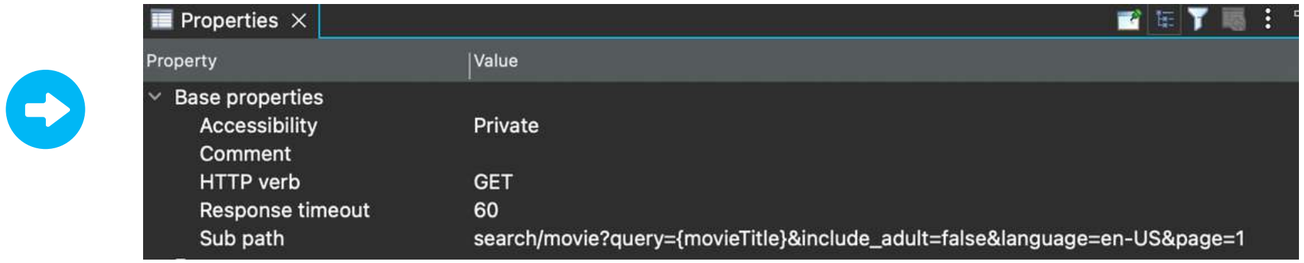
3.7 Edit the request path

In the Properties of the transaction,

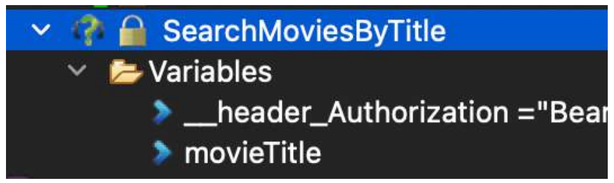
edit the Sub path to include the **request path**:

search/movie?query={movieTitle}&include_adult=false&language=en-US&page=1 (as seen in the TMDB API doc)





To add a **variable part**, enclose it in **curly braces within the path** (in our case, {movieTitle}).

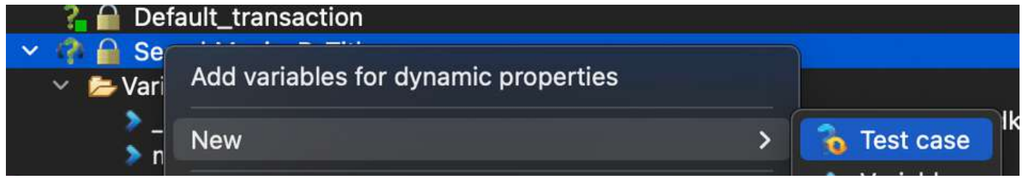


This **automatically adds the variable** to the **Variables folder**.

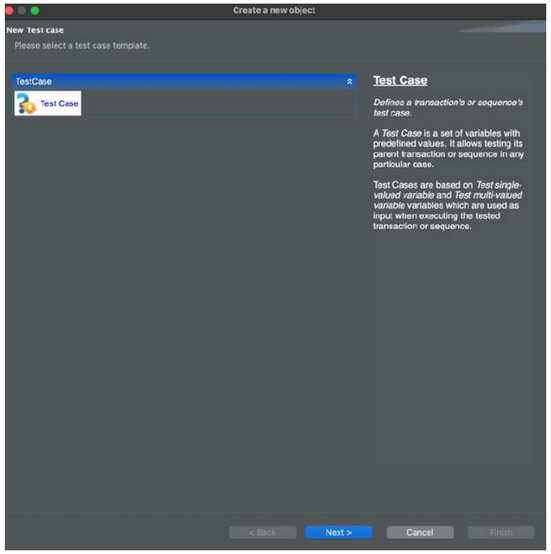
3.8 Test the request

To test the request, you need to create a test case.

Right-click on the transaction,
Select **New >**,
then click on **Test Case**.



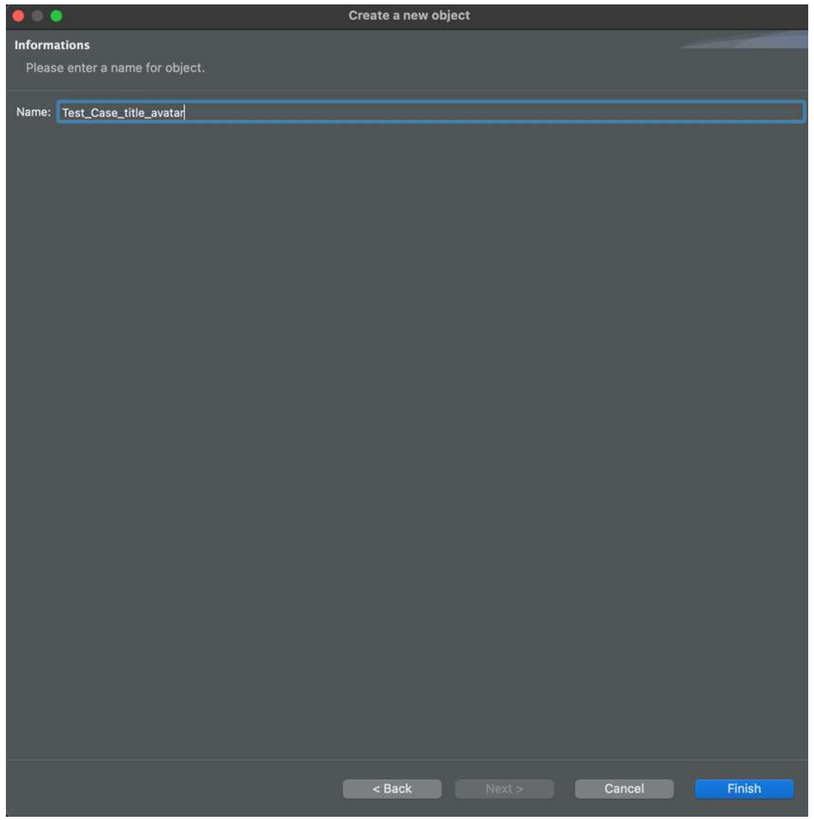
In the **Create a new object window**,
select **Test Case**
and click **Next.**



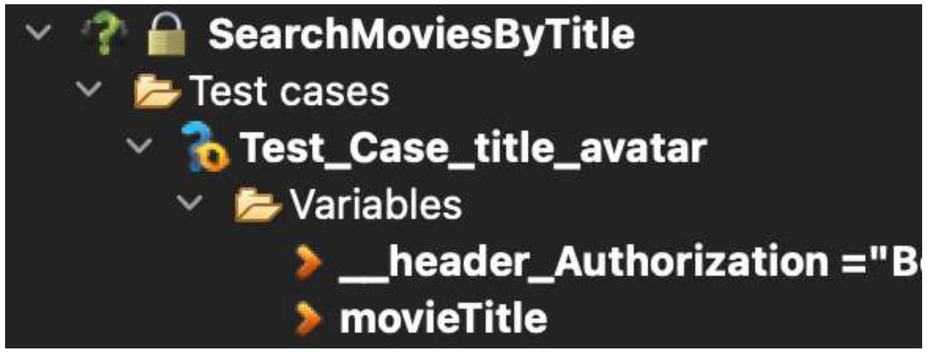
3.8 Test the request

Then, enter a name for the test case,

and click **Finish**.



The test case is created in a **Test Cases folder**.



3.8 Test the request

Select the **variable movieTitle** of the **test case.**

In the Properties, edit the Default Value

to enter a search term (in this case, 'avatar').

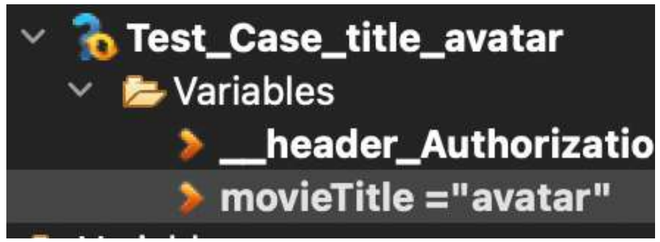
When we **edit** the **Default value**

of the **variable** in **properties**

In the **treeview**,

the **value of the variable** 'movieTitle'

is **automatically modified**.

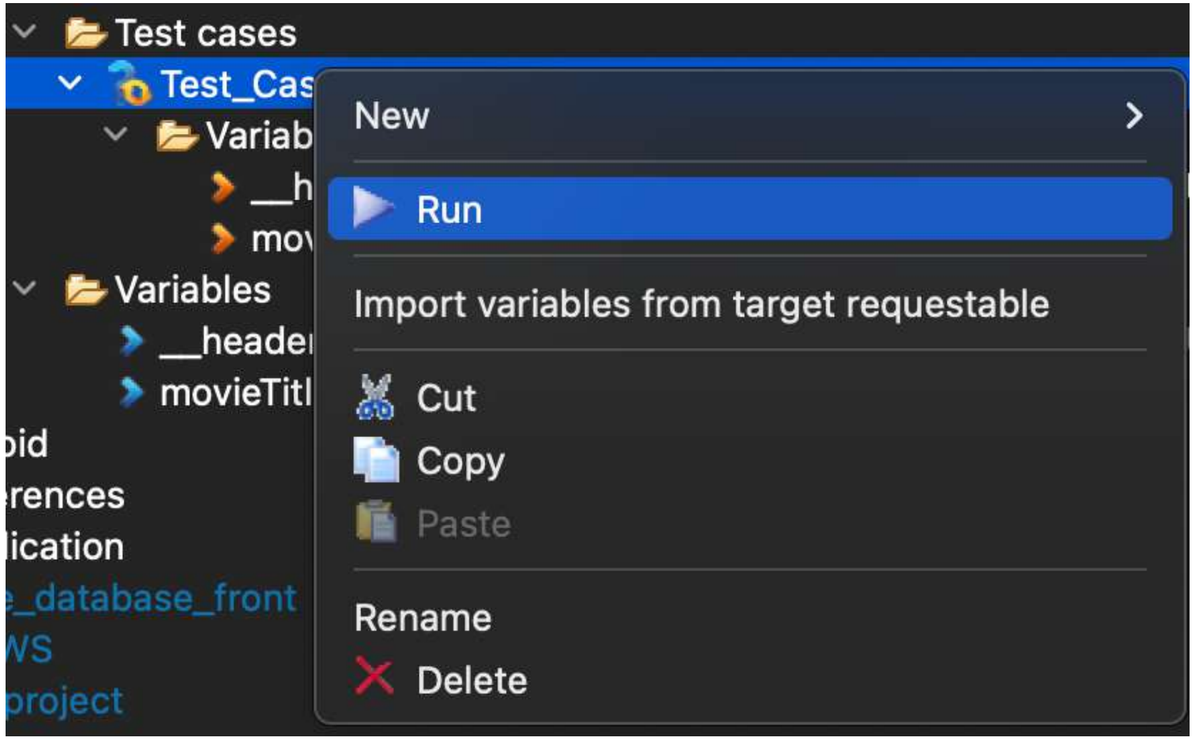


3.8 Test the request

To run the test,

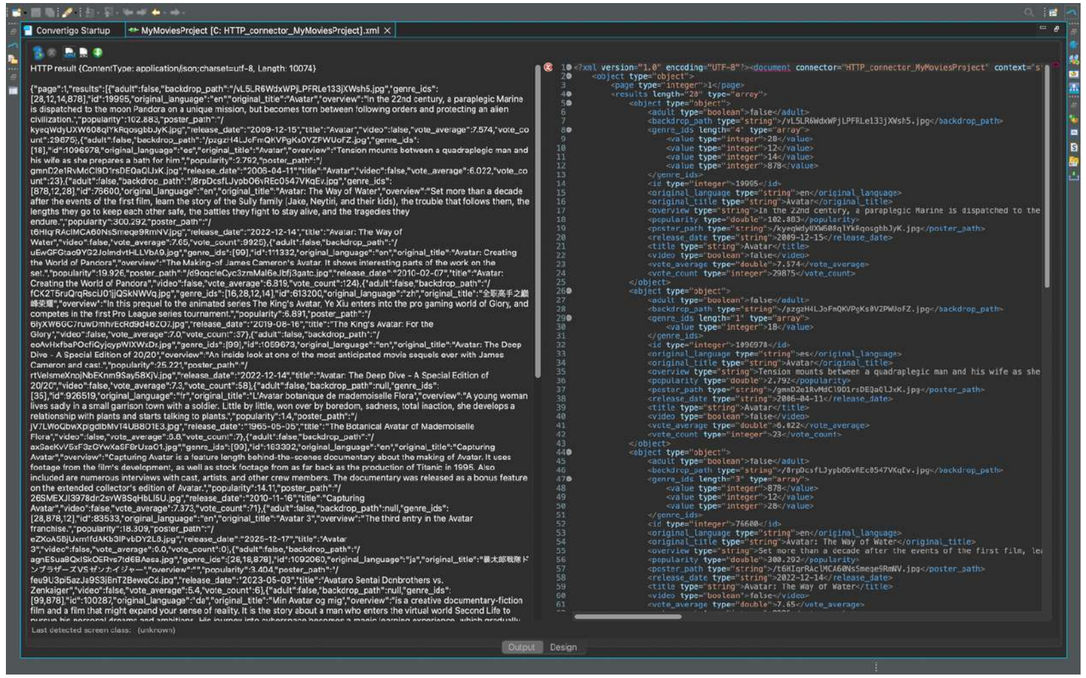
right-click on the test case,

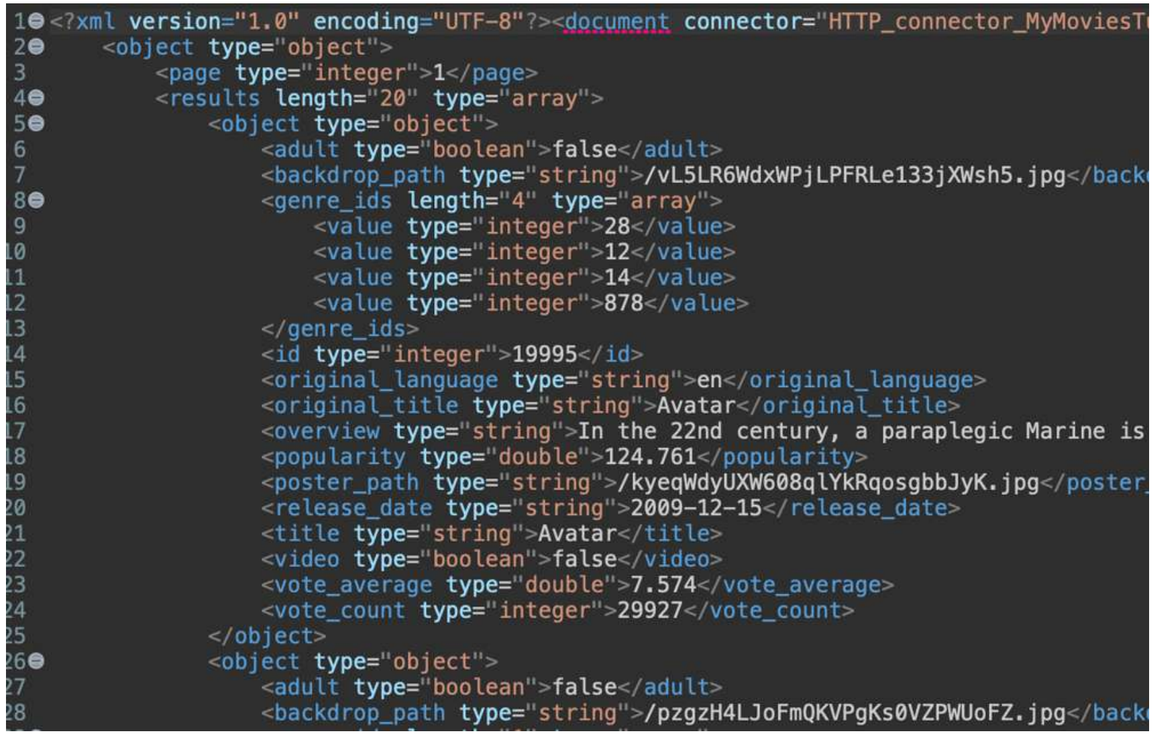
and click on **Run**.



3.8 Test the request

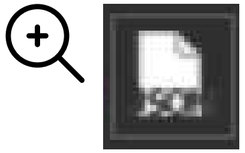
The **API response** is displayed in **XML** by default.

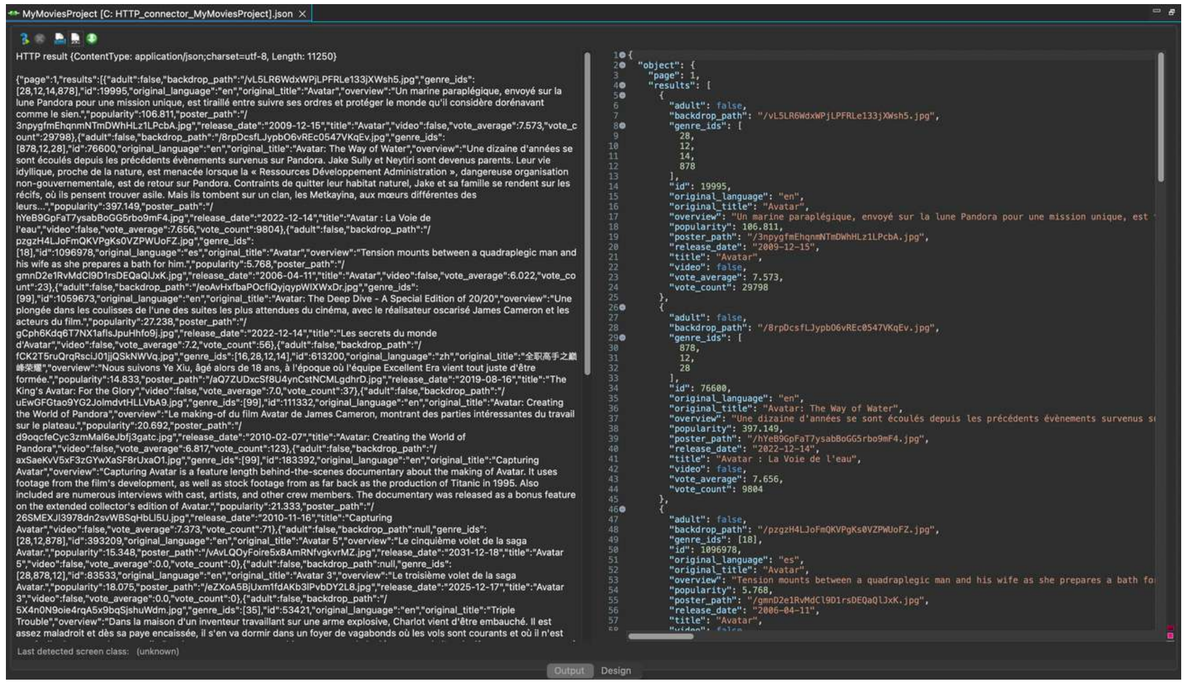




3.8 Test the request

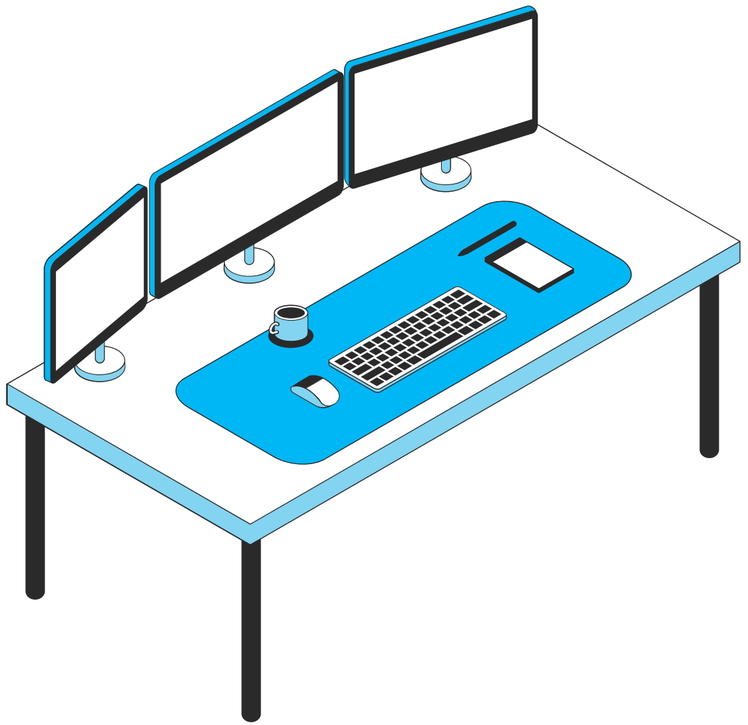
Click on the **JSON button** to display the **API response** in **JSON**.





4 – Sequences

How to create a flow of actions.



4.1 Sequences

4.2 Steps

4.3 XML & XPath

4.4 Source Picker

4.5 Create a sequence

4.6 Call a transaction from a sequence

4.7 Create a custom data structure

4.8 Test the sequence

4.1 Sequences

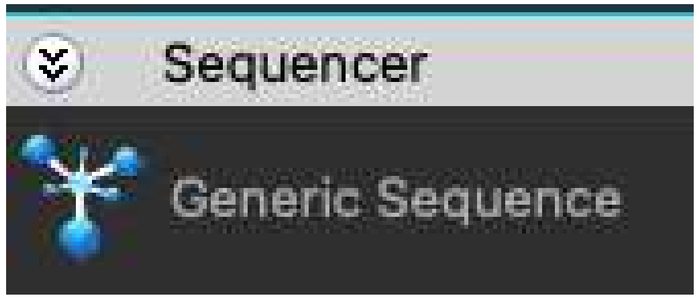
The **Sequence** is a very important **backend object**.
It is labelled as **Generic Sequence** in the palette.

In Convertigo Low Code Studio, **Sequences** are used
to design the **logical flow** and **behavior**
of the **backend** of your application
by specifying **what actions should occur** and **in what order**.

Sequences allows you to

- create **sequences of actions**
- define **conditions and decision points**
- manage **the order** in which these **actions are executed**
- **define and manage the flow of actions**
  with a series of successive steps

Object Sequence in the palette



Sequences folder in a project

4.2 Steps

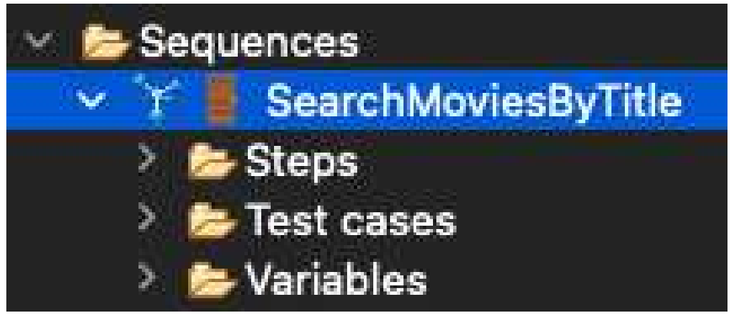
Steps are **back-end objects**.

A step is a **fundamental building block**
used to **define a specific task, action, or operation**
within a sequence.
For example, making an API request, showing a message,
performing data manipulation...

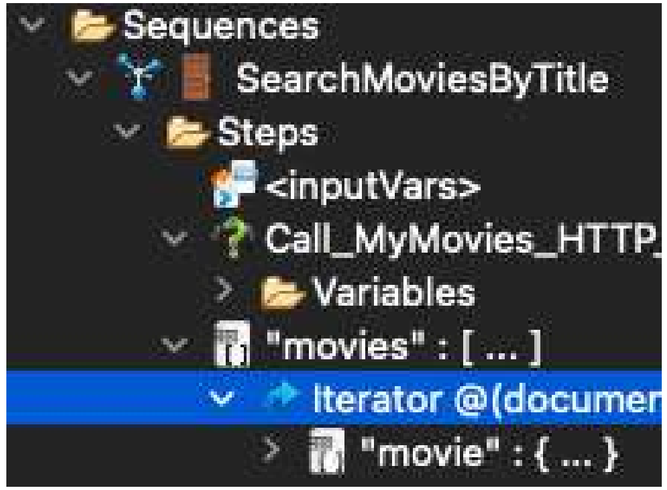
Steps are **organized** to create **a sequence of actions**
that the application should perform
in response to **certain events** or **user interactions**.

It allows developers to **define the logic and behavior**
of the application in a **structured and modular manner**.

Example of a steps folder in a sequence



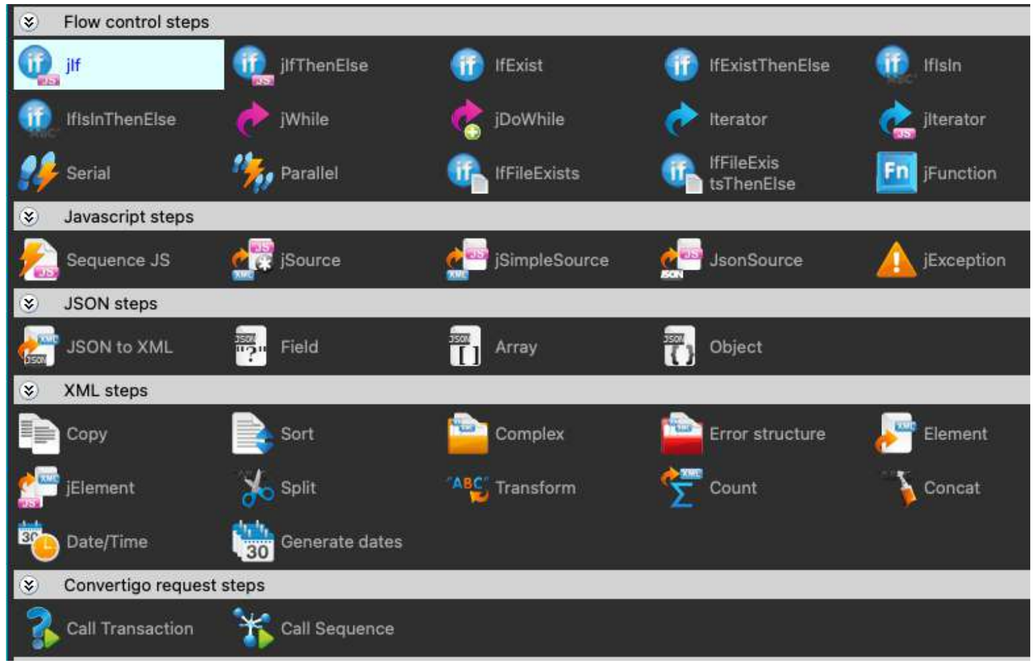
Example of a series of steps in a sequence



4.2 Steps

Categories of Steps

Examples of Steps in the palette



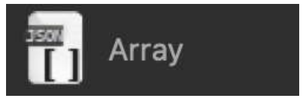
There are different categories of steps :

- **Convertigo request steps** => to call a sequence or transaction
- **Flow Control Steps** => to control the sequence of actions and logic within a sequence
- **Javascript steps** => to incorporate custom JavaScript code in sequences
- **XML steps** => to work with XML data in sequences
- **JSON Steps** => to work with JSON data in sequences
- **HTTP session management** => to manage user sessions in web applications
- **File management steps** => to handle and manipulate files on the local system or server
- Others

4.2 Steps

JSON Steps

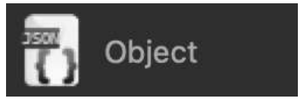
Convertigo provides **JSON steps** to **manipulate and interact with JSON data** in sequences.



Array - JSON step

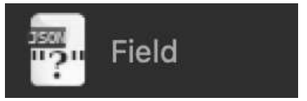
When added to a sequence, this step creates an **XML element** (element node) ready to output a JSON Array

- First, you **drag-and-drop the step into a sequence**
- then, you **drag-and-drop the data**
  you want to manipulate from the Source Picker
  into the step in the sequence.



Object - JSON step

When added to a sequence, this step creates a JSON Object.



Field - JSON Step

When added to a sequence, this step creates a JSON string, number, boolean or null.

4.3 XML & XPath

XML Data Structure

In Convertigo,
the data structure is **based on XML**
regardless of its source.

The XML data structure follows the standard XML format.

It is organized hierarchically in a **tree structure**
with one **root element**, the **document**,
that is the **parent of all other elements**.

Each element has **attributes** and **text content**

Example of XML Data structure in Convertigo

4.3 XML & XPath

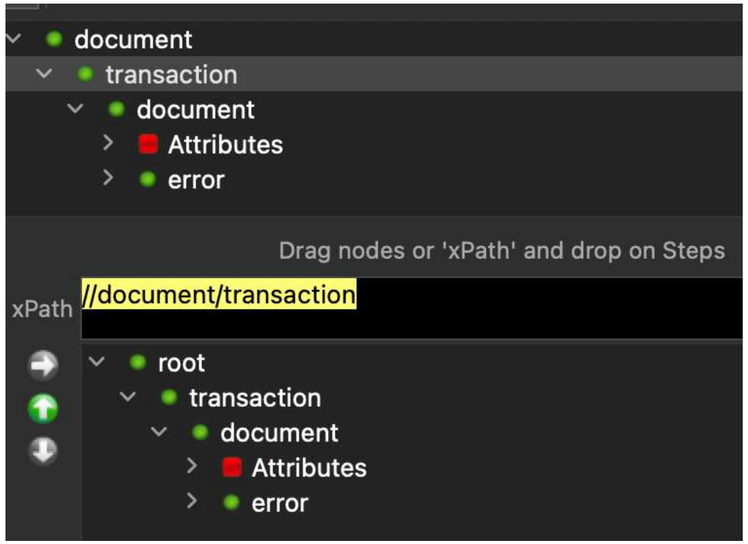
XPath

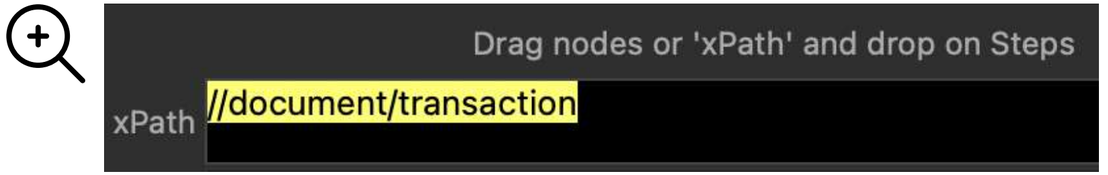
XPath is a **language**
used for **navigating and querying XML documents**

XPath provides a way
to pinpoint **specific elements** and **data**
within an **XML structure**
by using **path expressions** that **define the location of nodes**.

XPath expressions are used
to **identify and traverse these nodes**
within an **XML document**,
allowing for **data extraction and manipulation**.

Example of XML Data structure & XPath in Convertigo





4.3 XML & XPath

Nodes

In XML and XPath,

Nodes are the **individual components of an XML document**.

There are several types of nodes :

- **element nodes** representing XML elements
  -> marked by a green dot in the XML Data structure in Convertigo

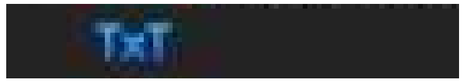
Element node

- **attribute nodes** representing attributes of elements
  -> marked by a red square in the XML Data structure

Attribute node

- **text nodes** containing textual content within elements
  -> marked by **TxT** in the XML Data structure

Text node



4.4 Source Picker

Sources

Each transaction, sequence, and step

- is a **data source** for the next step
- has a property called “output”
- emits data in the source picker

A **source** is defined as a **reference on a step**
previously existing in the parent sequence,
associated with an XPath applied on the step’s result DOM.

At runtime, the **XPath** is applied on the step’s **current execution result XML**
and **extracts a list of XML nodes** resulting from this execution.

4.4 Source Picker

Each transaction, sequence, and step

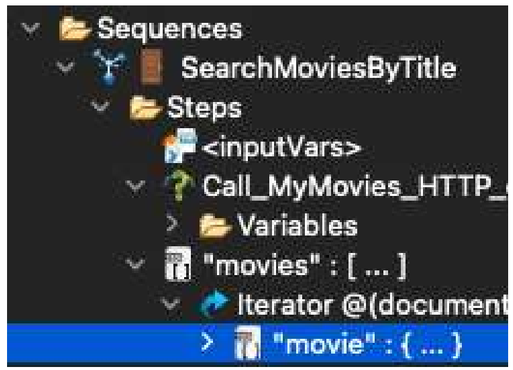
- is a **data source** for the next step
- **emits data** in the **source picker**

The source picker

- displays the **structure of the data** emitted by a step.
- allows you to **select the XPath** without typing it by **dragging and dropping the node directly into a step**.

The **XPath** is used as **data path for accessing data**.

Example : step “movie” in sequence



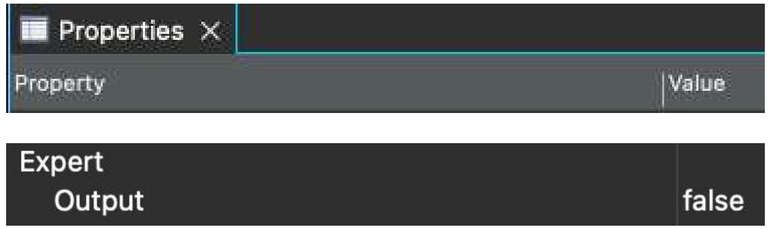
Data structure of step “movie” in source picker

4.4 Source Picker

Output Property

Each transaction, sequence, and step

- has a **property** called “**output**”



The **Output property** defines whether the **XML generated by this step** should be **appended to the resulting XML**.

- Set this property to **true** to add the step’s resulting XML to the **sequence’s output XML** (default value for steps generating XML).

- Set this property to **false** to **prevent the steps’s XML result to appear** in the sequence’s output XML. Setting this property to false **does not prevent** the step’s generated XML from being **used as a source by other steps**.

4.4 Source Picker

Output Property

To handle the **data emitted by a step**,

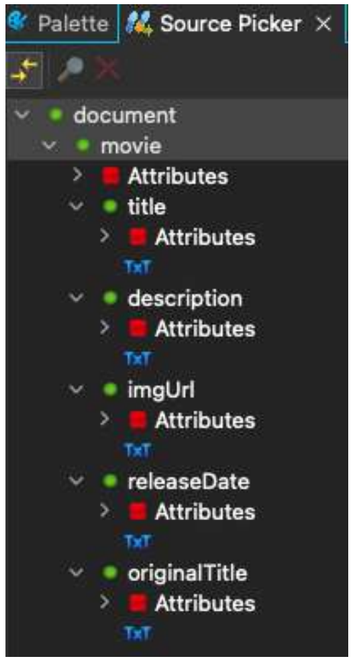
there are 2 options :

First option : If you need the **whole data emitted by a step**

1. Put '**output**' on '**true**'
2. The step emits data in the response

Second option : If you need to **filter the data** and keep only specific data

1. Put '**output**' on '**false**'
2. The step doesn't emit in the response but still **emits in the source picker**.
3. You **select the data** you need **in the source picker** by **drag-and-dropping it in a sourceable step**
4. The following step can connect to this source through it.



4.4 Source Picker

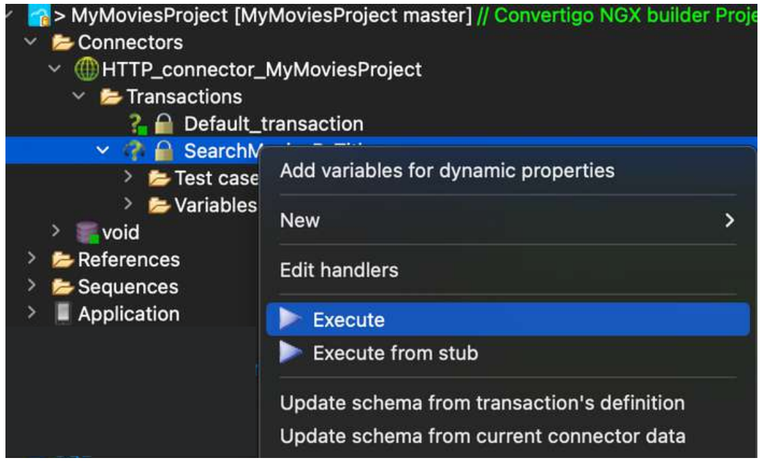
Transaction data structure

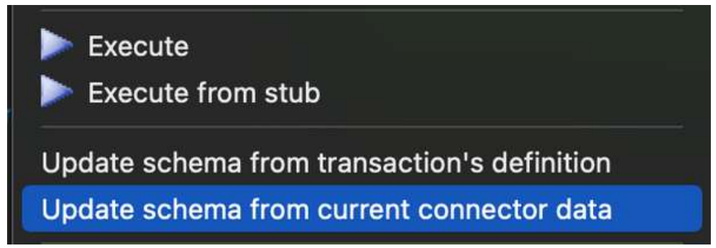
Calling a **transaction** brings back **data**
with a **structure described in the source picker**.

In other steps, the **output structure** is always the same.
In a **transaction call**, the **output structure** is unknown.

→ To discover it, you need to **execute the transaction once**.
→ Then retrieve the structure
with **Import data structure from current connector data**..

As good practice, this should be done
when the transaction is created,
before creating the sequence.



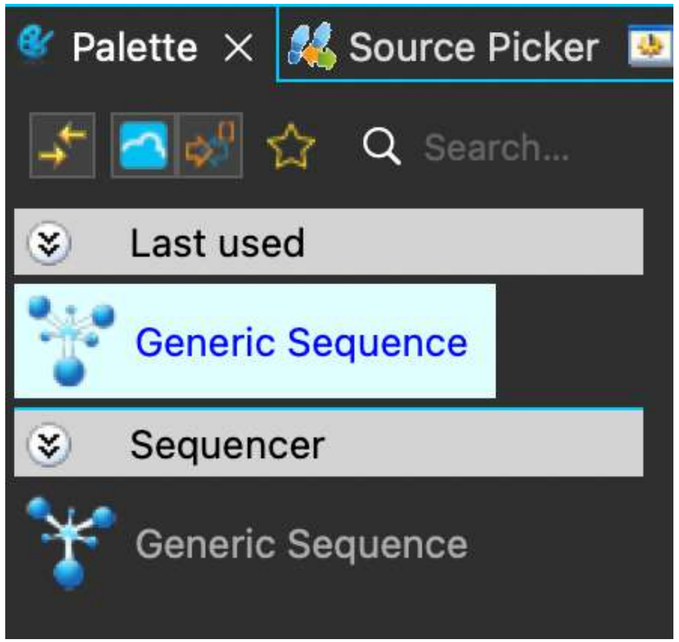


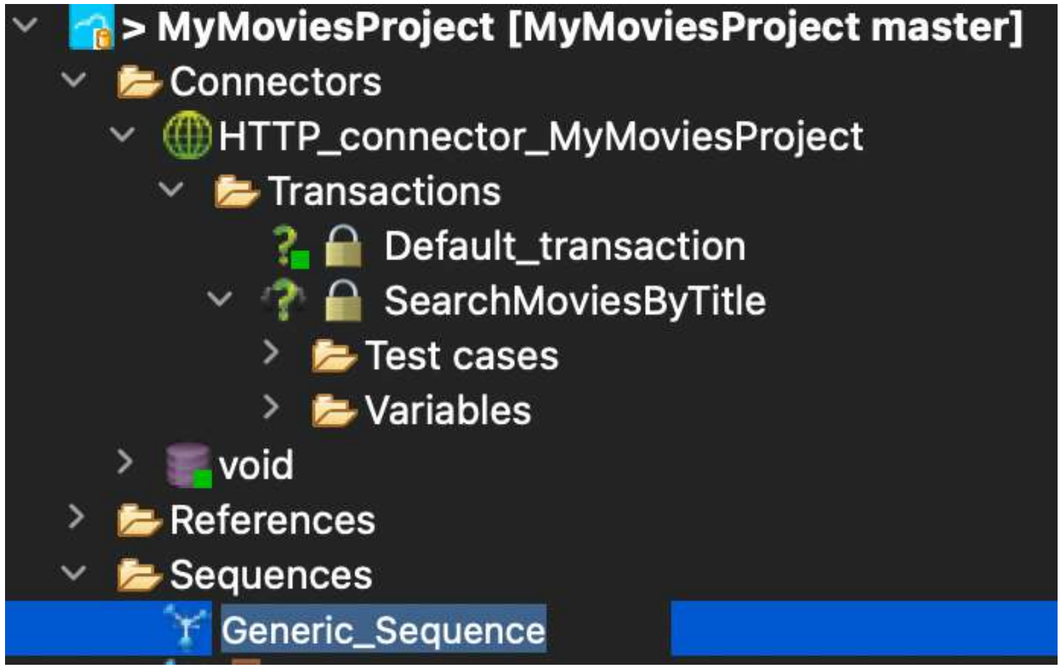
4.5 Create a sequence

To create a sequence, you have several options.

First option:

You can **drag and drop a Generic Sequence** from the palette in the tree structure and rename it.





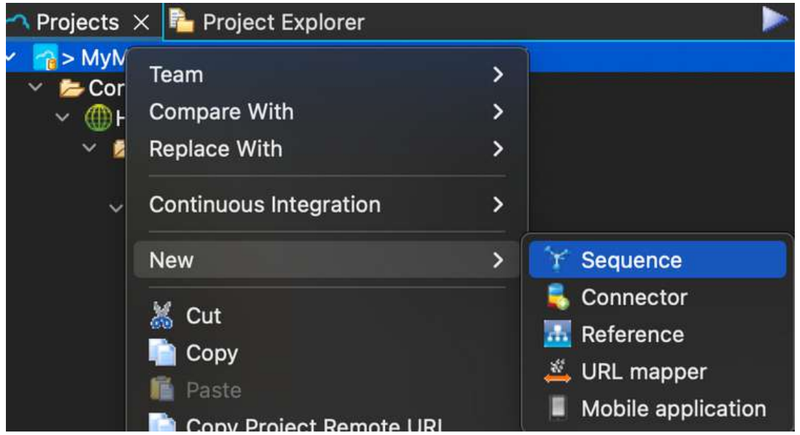
4.5 Create a sequence

Second option:

To create a sequence,

you can **right-click on the project**,

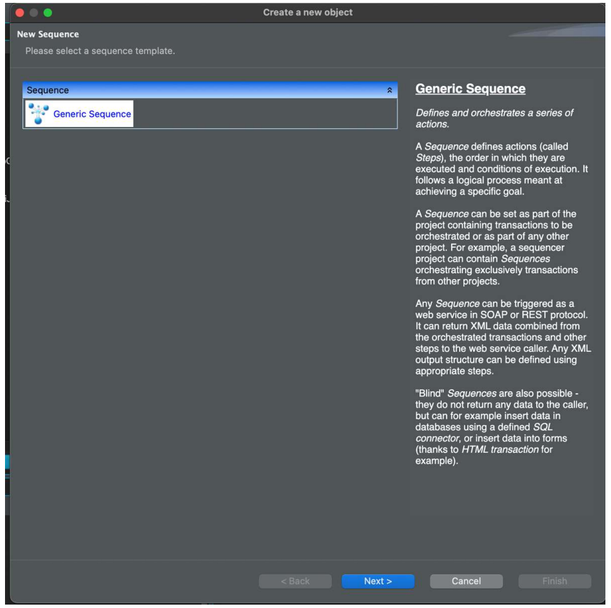
select **New >**, then click on **Sequence**.

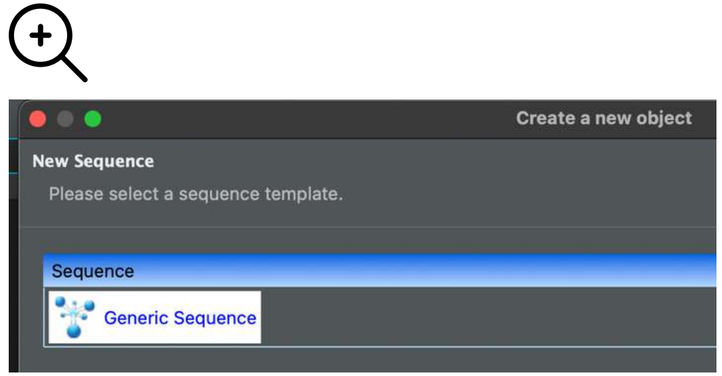


The **Create a new object window** appears.

In the **Create a new object window**,

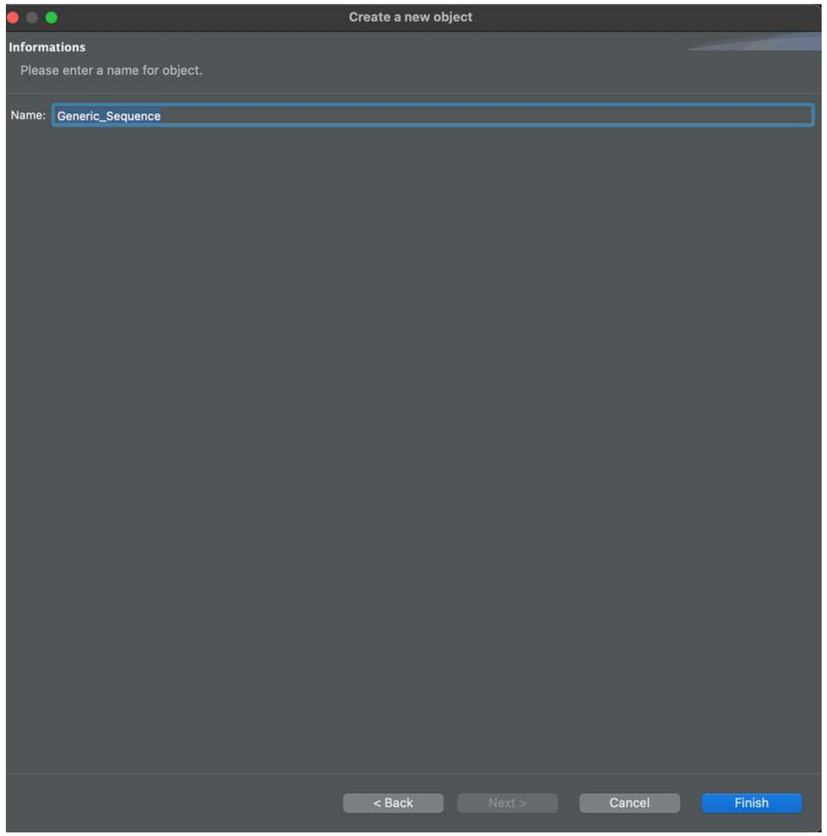
select **Generic Sequence**, then click on **Next >**

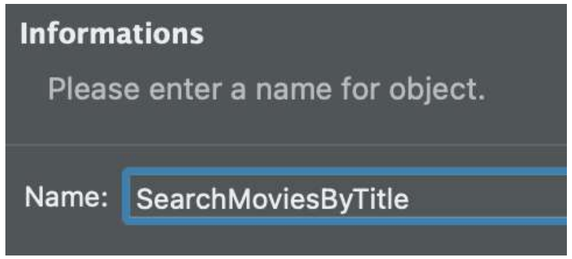




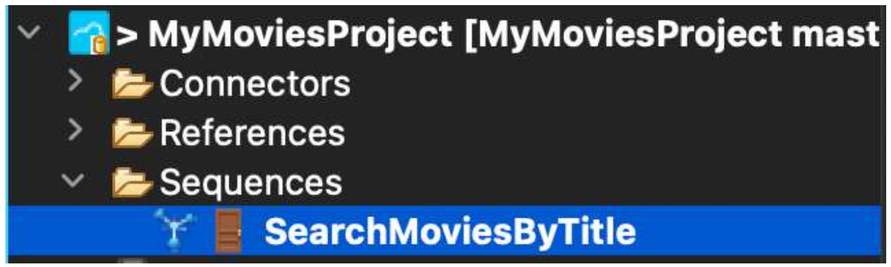
4.5 Create a sequence

In the **Create a new object window**,
rename the sequence and click on **Finish**.





A **Sequences folder** and the **created sequence** appear in the tree structure.



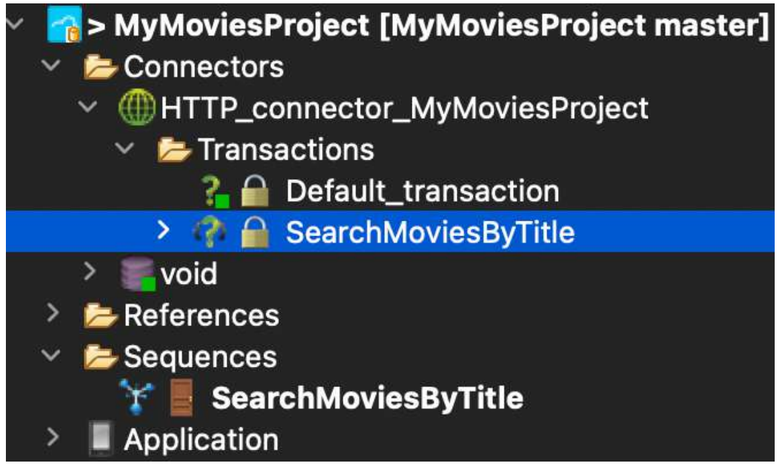
4.6 Call a transaction from a sequence

Import the transaction in the sequence

Once the sequence is created,

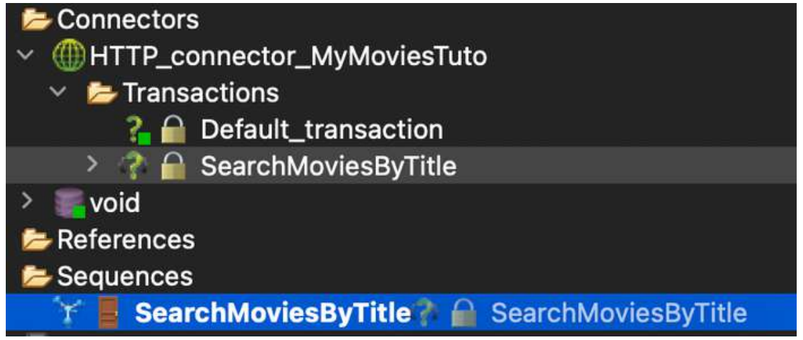
you need to

import the transaction in the sequence.

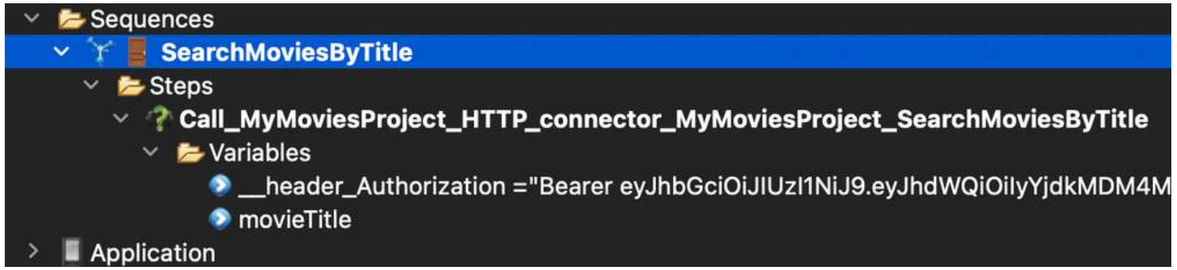


Drag-and-drop the transaction in the sequence

while clicking on **Option** in MacOs or **Control** in Windows.



This creates a **Steps folder** where a **call to the transaction** appears.

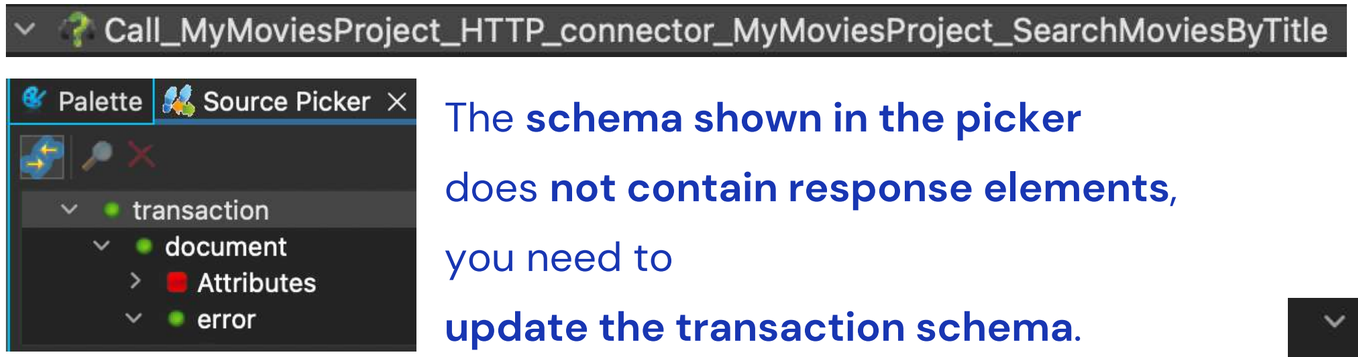


4.6 Call a transaction from a sequence

Update the transaction schema

Reminder : This step can and usually should be done just after creating the transaction.

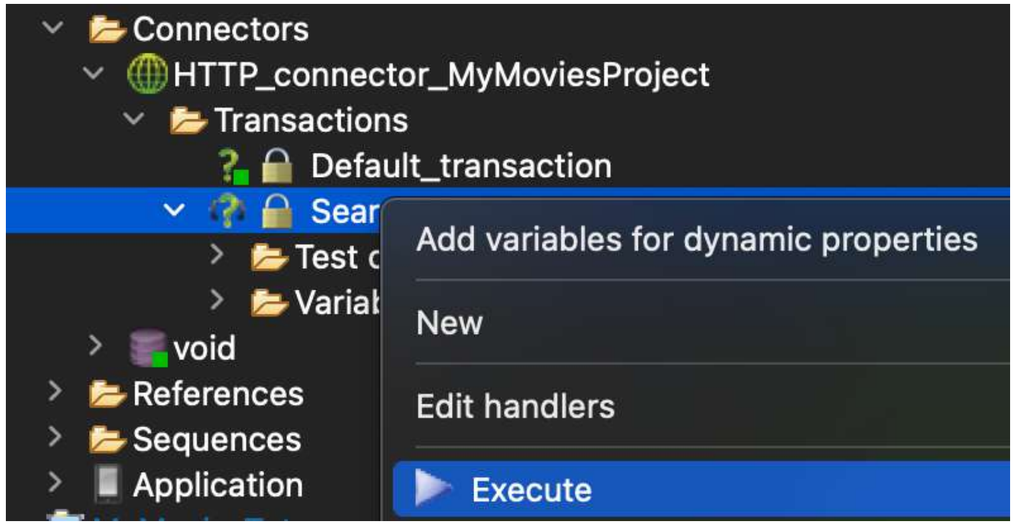
Double-click on the Call Transaction step to display the source picker.



The **schema shown in the picker** does **not contain response elements**, you need to **update the transaction schema**.

There are 2 cases : **a transaction with or without variables**.

If the transaction **doesn't need variables**, you can right-click on the source transaction and choose **Execute** to generate response data.

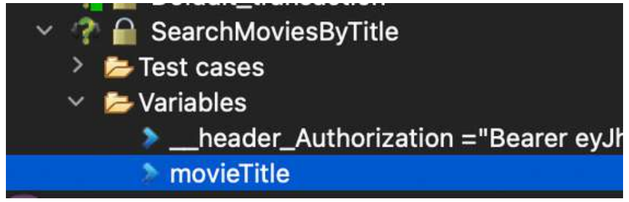


4.6 Call a transaction from a sequence

Update the transaction schema

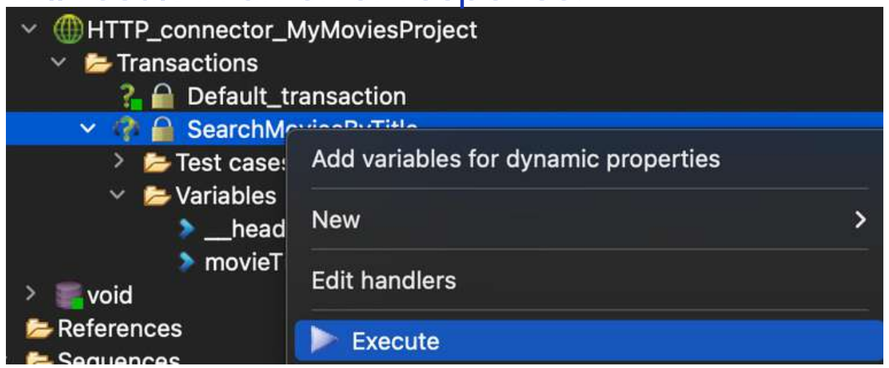
In our case, we have 2 variables :

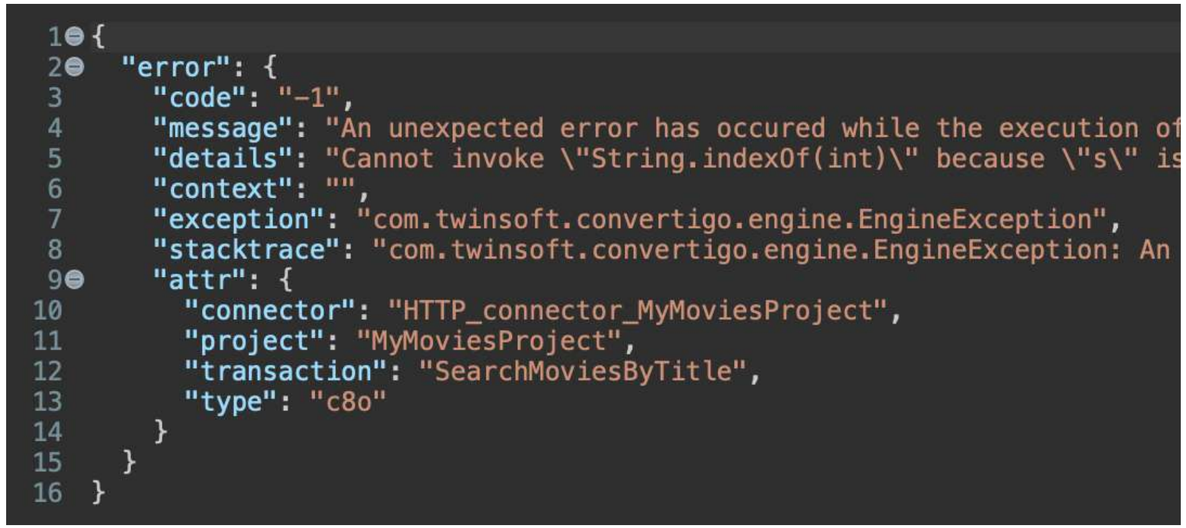
- **_header_Authorization** which has already a value
- **movieTitle** whose value is empty.



Executing the transaction as it is

will result in an error response.





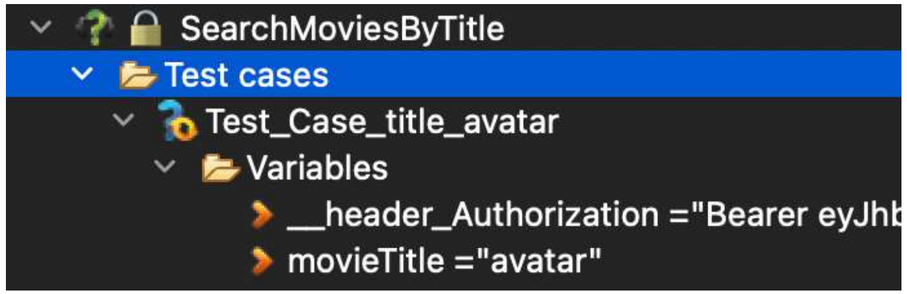
```
{
  "error": {
    "code": "-1",
    "message": "An unexpected error has occured while the execution of
    "details": "Cannot invoke \"String.indexOf(int)\" because \"s\" is
    "context": "",
    "exception": "com.twinsoft.convertigo.engine.EngineException",
    "stacktrace": "com.twinsoft.convertigo.engine.EngineException: An
    "attr": {
      "connector": "HTTP_connector_MyMoviesProject",
      "project": "MyMoviesProject",
      "transaction": "SearchMoviesByTitle",
      "type": "c8o"
    }
  }
}
```

4.6 Call a transaction from a sequence

Update the transaction schema

To get a valid response, you need to use the test case you created before in the transaction SearchMoviesByTitle.
In this test case, the variable **movieTitle** has already a value.

Right-click on the test case.



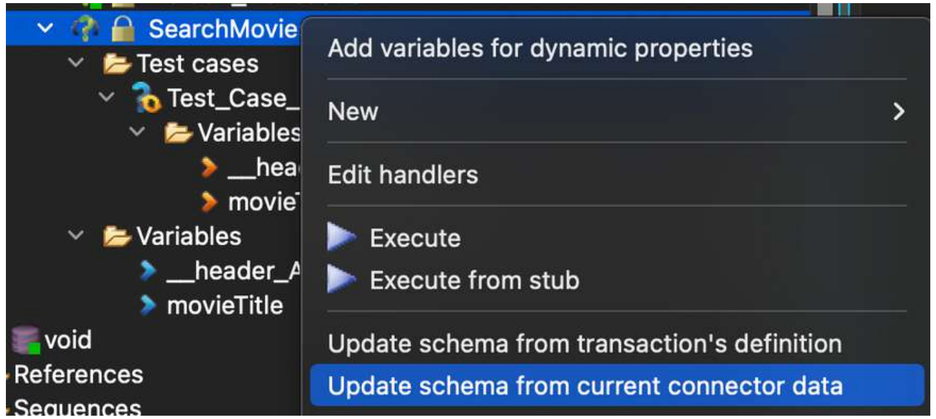
Choose **Run** to generate response data.

The results are displayed in the editors panel.

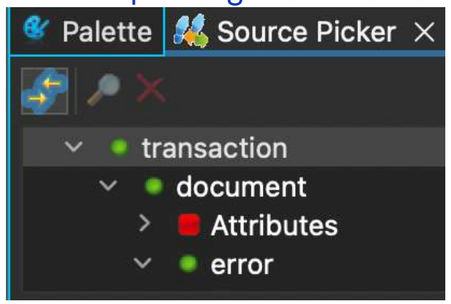
4.6 Call a transaction from a sequence

Update the transaction schema

Right-click on the source transaction **SearchMoviesByTitle** and select **Update schema from current connector data**.

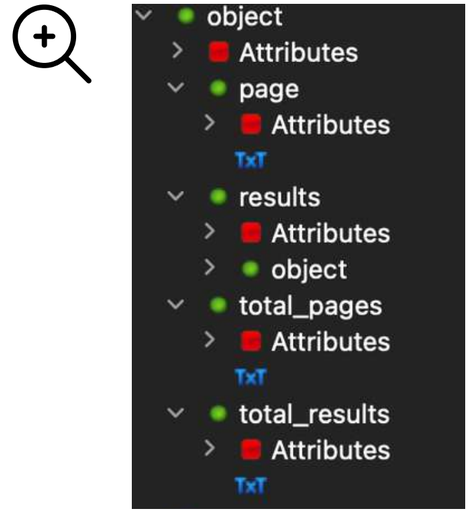


Before updating the schema



Double-click once again on the **Call Transaction Step** to display the **updated transaction schema** in the Source Picker.

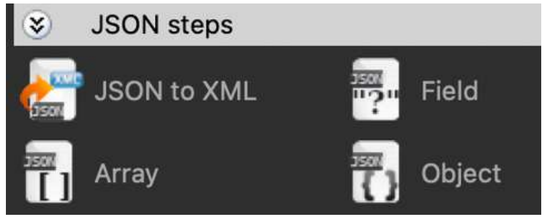
After updating the schema, an **object node** appears in the source picker



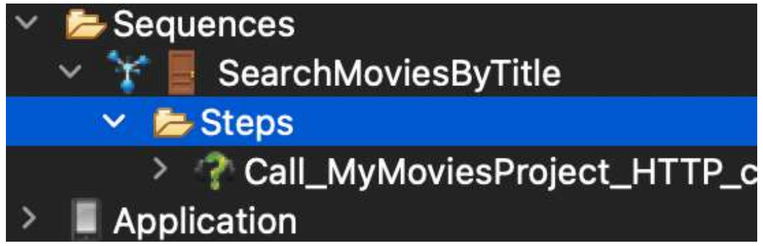
4.7 Create a custom data structure

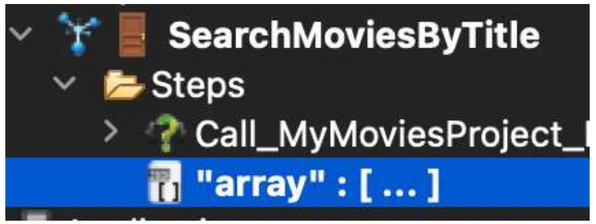
Your sequence is now calling the **transaction SearchMoviesByTitle** which is a **JSON HTTP Transaction**.

To construct our own **response data structure**
from the transaction's r**esponse data**, let's use the Array Step (JSON step).

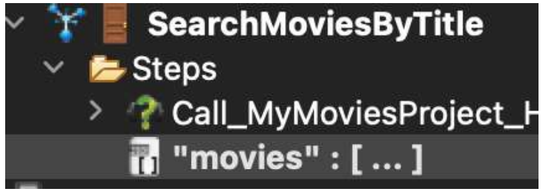


Drag the **Array step** from the palette and drop it
into the **steps folder of your sequence**
after the **transaction call**.

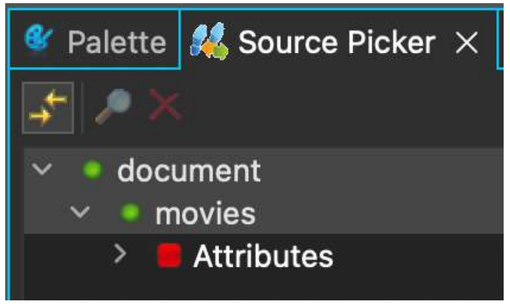




Rename it movies.

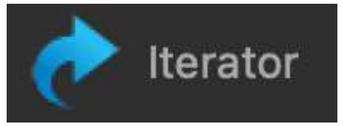


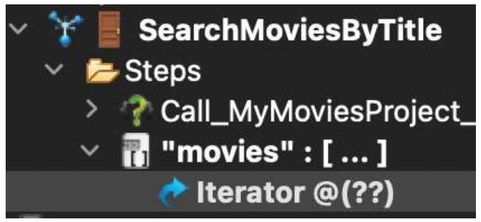
It appears as movies
in the source picker.



4.7 Create a custom data structure

Drag the **Iterator** step from the palette and drop it into the step **movies** in your sequence.



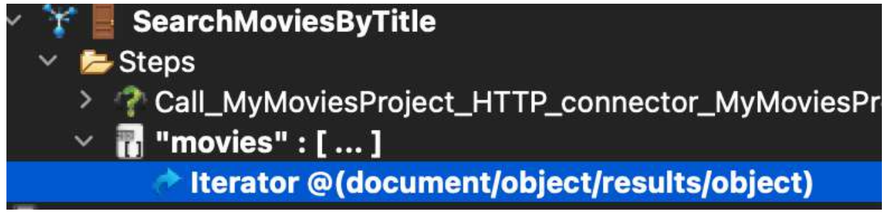


Then, double-click on the transaction call in your sequence to open the source picker.

In the source picker, expand the **results node.**

Then drag and drop the **object node** directly into your iterator.

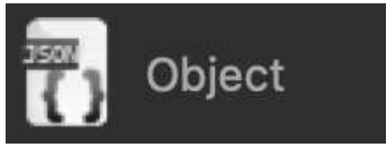
This **object node** provides the information you want in your iterator.

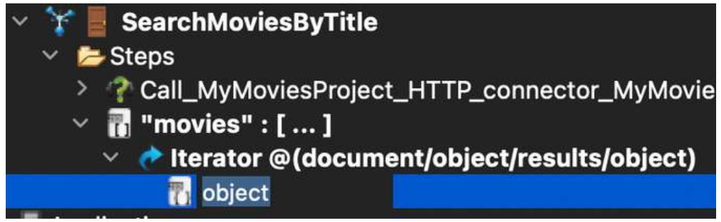


4.7 Create a custom data structure

Drag the **Object step** from the palette

and drop it into the **Iterator step** in your sequence.



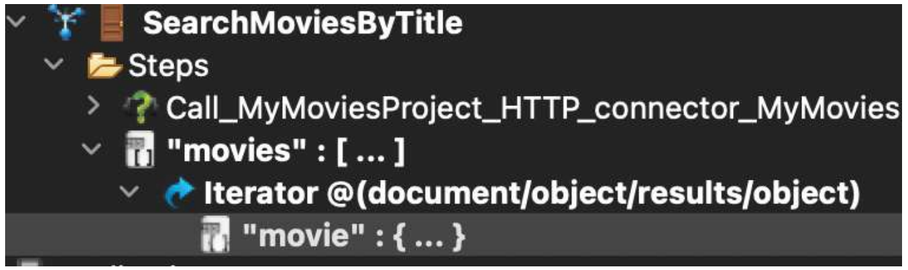


In the source picker,

it appears as **object**.

The Object step is a container for the various elements you'll add to it.

Let's rename it **movie** in the treeview.



Now, in the source picker, it appears as movie.

4.7 Create a custom data structure

In the **response data** from the transaction,

for each item, we receive an object movie with many fields, as shown in the source picker.

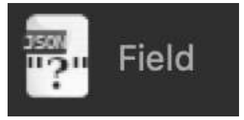
In our application, we only need a few of them and we're going to select the fields that interest us.

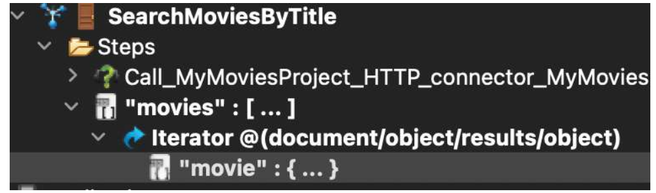
Let's say I want the following fields displayed in the front-end :

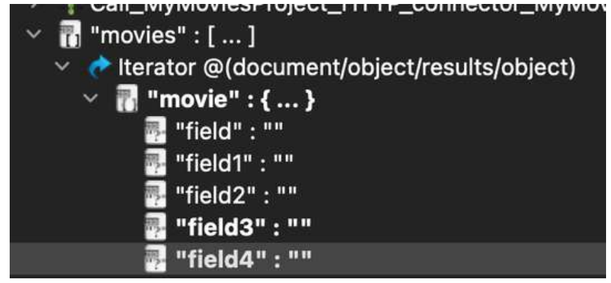
- title
- overview
- poster_path
- release_date
- original_title

Drag the Field step from the palette

and drop it 5 times into the Object step movie.



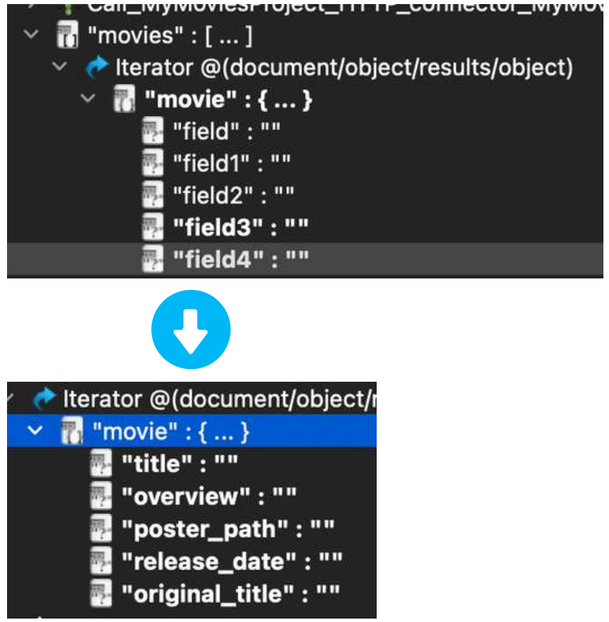




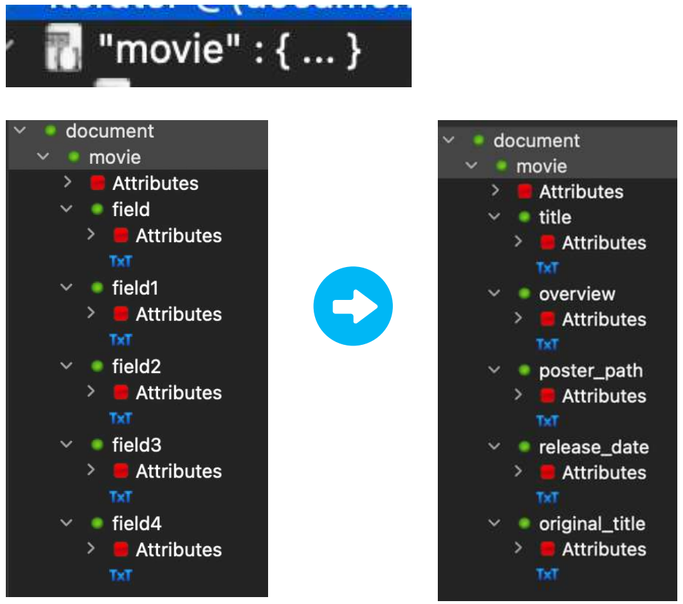
4.7 Create a custom data structure

Click twice on the movie step to display it in the source picker.

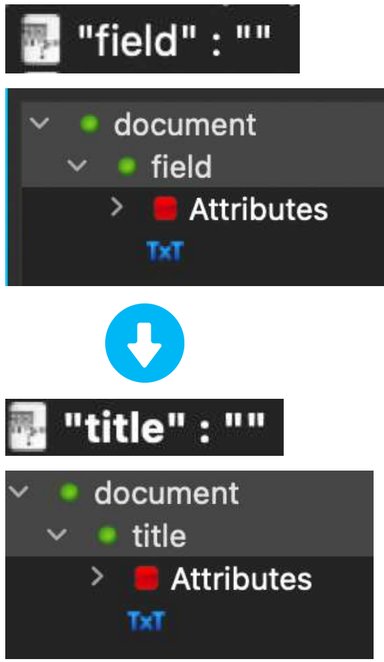
Rename the 5 fields in movie as title, overview, poster_path, release_date, original_title



In the source picker,
you can see that the 5 fields in movie have been renamed as well.



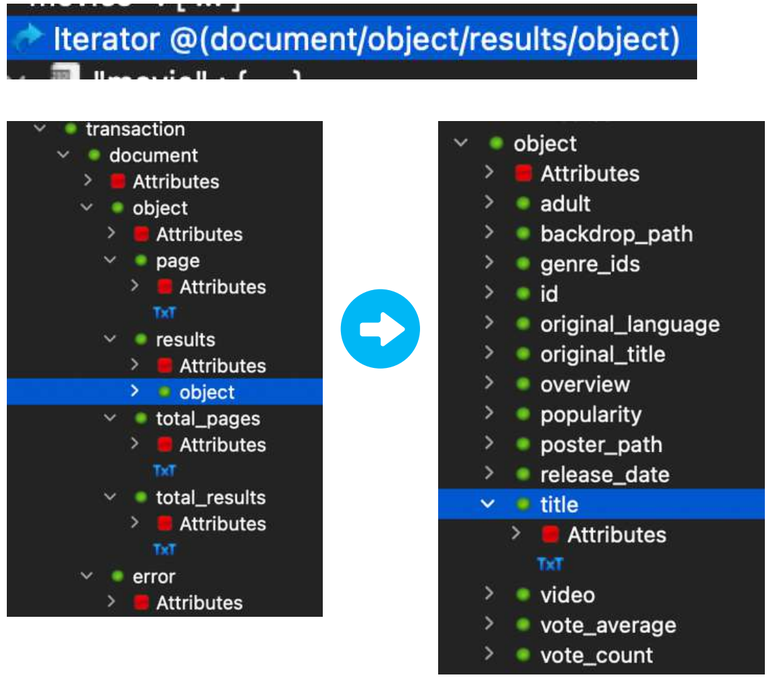
The structure of each field has been renamed



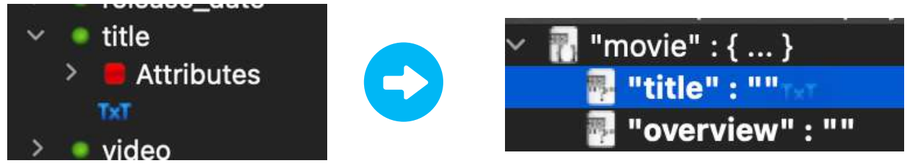
4.7 Create a custom data structure

Now we want to bind these fields to the values of the fields in the iterator.

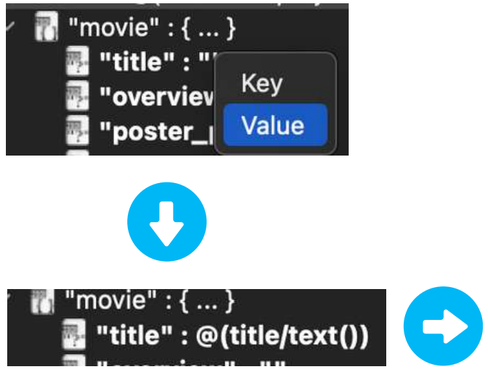
Double-click on the **Iterator step** to display its data in the Source Picker and open the object node.



Drag and drop the **TxT element** corresponding to the required information into the various steps of the element.



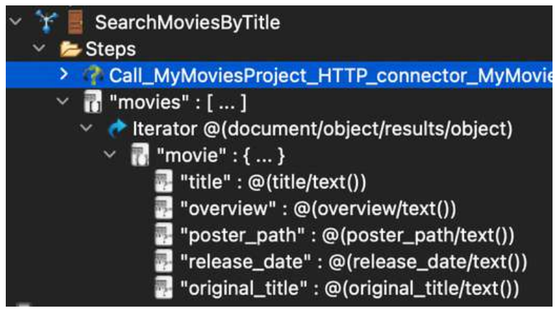
Choose **Value** each time you are prompted to set the value property of the step.



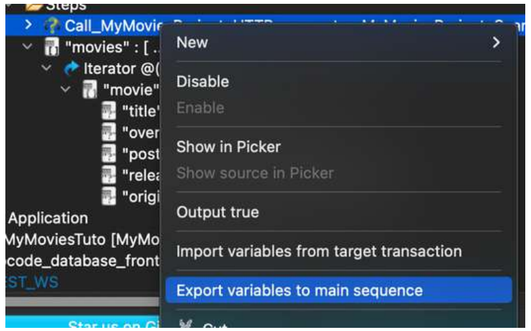
In properties, the value appears as binded.

4.7 Create a custom data structure

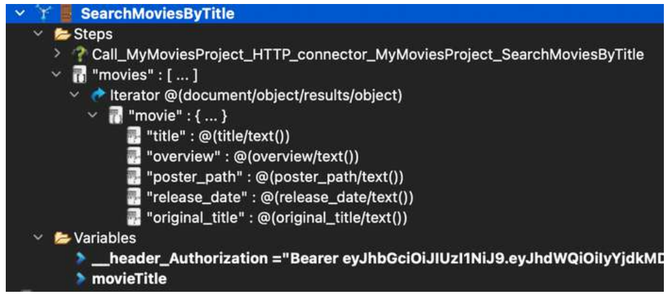
Repeat the same operation for the 5 fields.



Now we want to **import the variables of the transaction into the sequence**,



Right-click on the **transaction call**,
and select **Export variables to main sequences**.

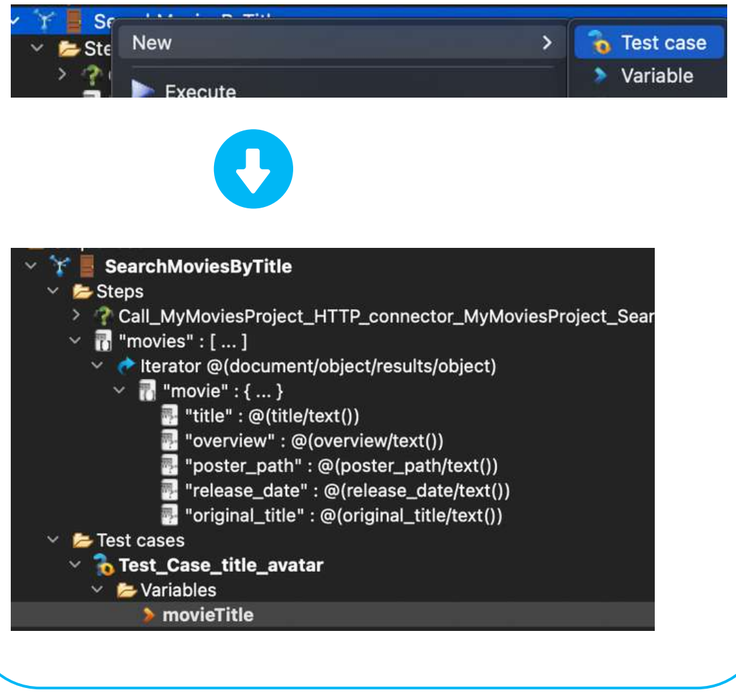


A folder **Variables**
has been added to the sequence.

4.8 Test the sequence

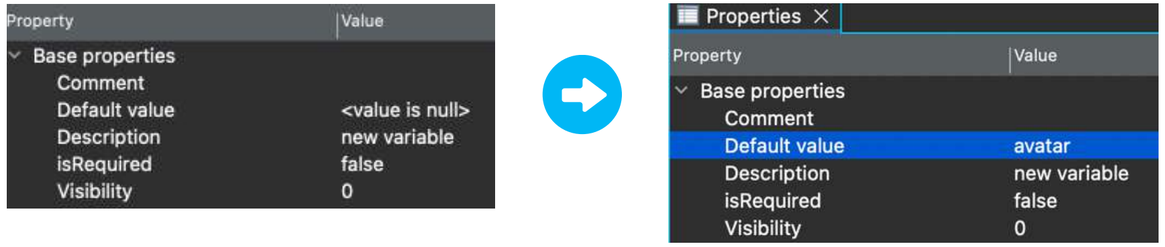
Now, let’s create a test case for the sequence

(as shown in the previous slides for the transaction SearchMoviesByTitle).



Click on the variable movieTitle in the test case.

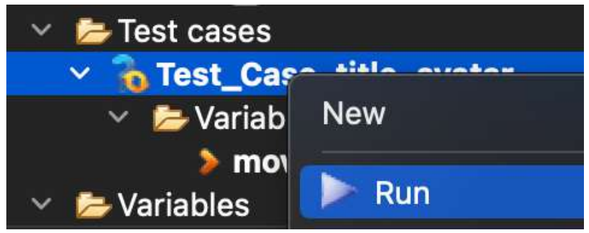
In properties, change the Default value of movieTitle to “avatar“.



The value appears in the treeview.

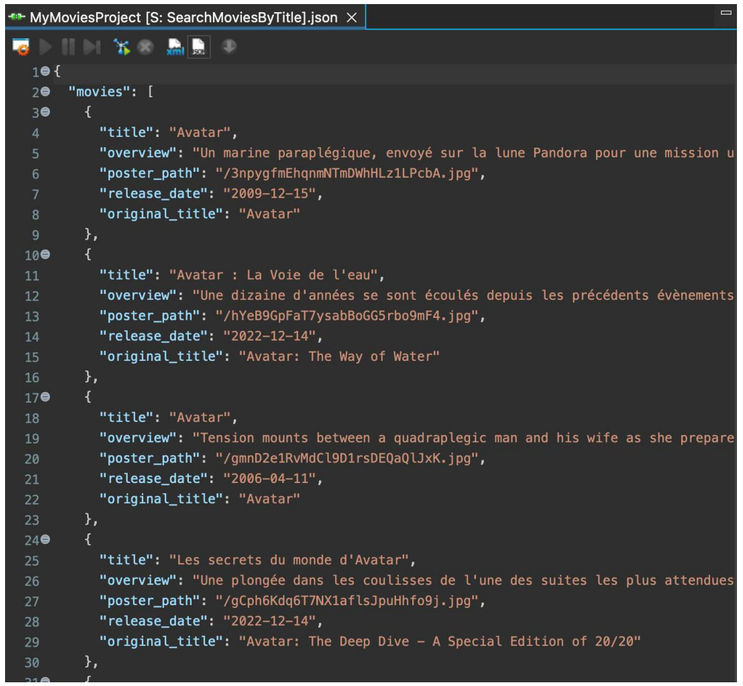
4.8 Test the sequence

Right-click on the test case, choose **Run** to execute it and generate response data.



The results are displayed in the editors panel.

The response data generated by the sequence will display only the information you requested.



convertiGO

5 – JavaScript Scope

How to handle JavaScript in the studio.

6.1 What is the JavaScript Scope ?

6.2 Interactions with JS Scope

6.3 Back-end Objects bound to JS Scope

6.4 Step Sequence JS

6.5 Step Input variables

6.6 Modify a sequence with the JS Scope

5.1 What is the JavaScript Scope ?

By default, **every execution** of a transaction or a sequence has a **JavaScript environment**.
This is called the **JavaScript Scope**.

You can use JS to **manipulate data** in the sequence.
For example, perform calculations and data transformations...

Transaction or sequence input variables

All variables declared as **input vars** (input variables) of the sequence

- are inserted into the global scope of the JS environment.
- are automatically JavaScript variables
- become global variables of the sequence.

5.2 Interactions with JS Scope

In order to **manipulate data in JavaScript**,

Convertigo uses **backend objects** as gateways between the **structured context** and the **JS scope**.

These objects

- manage **interactions** between XML data sources and JavaScript.
- are used as **steps in sequences**.

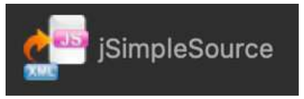
These objects or steps can either

- **transform XML data** from the source defined in the Source property
  into JavaScript variables in the current executed sequence JS scope.
  These JS variables can be manipulated in JS.
- **transform JavaScript scope variables** into **XML data sources**.
- use **JavaScript expressions** as **data sources**.

5.3 Back-end Objects bound to JS Scope

convertigo

Steps transforming XML data sources into JavaScript variables

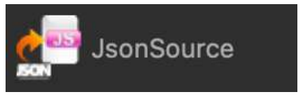


jSimpleSource – JS step

This step **transforms a single node**
from the source defined in the Source property
into a **JS variable** (String)

jSource – JS step

This step **transforms a list of XML nodes**
into a **JS variable** (Java NodeList object)



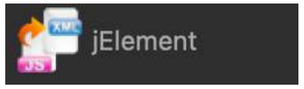
JsonSource – JS step

This step **extracts a JSON typed XML structure**
from the source defined in the Source property,
parses it as JSON,
and sets it as a **JS variable** (JS Object or JS Array).

5.3 Back-end Objects bound to JS Scope

convertigo

Steps transforming JS variables into XML



jElement – XML Step

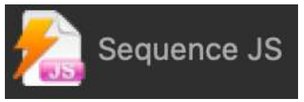
This step **adds an XML element node** based on a **JS expression** to parent XML element in the **sequence XML output**.

JSON to XML – JSON step

This step **adds an XML attribute node** based on a **JS expression** to parent XML element in the **sequence XML output**.

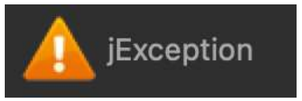
5.3 Back-end Objects bound to JS Scope

Steps used to manipulate JavaScript



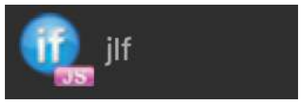
Sequence JS - JS step

This step is used to **write JavaScript code** which is **executed in the sequence scope** (initialize variables, calculations...)



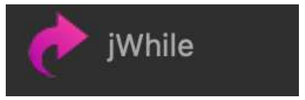
jException - JS step

This step **raises a Convertigo Engine exception**. It **breaks the sequence execution flow**, ending the sequence just after this step.



jIf - Flow control step

This step is **based on a JavaScript condition** and contains other steps executed only **if the condition is fulfilled**.



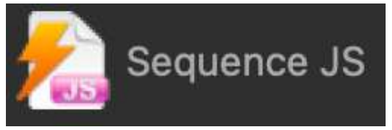
jWhile - Flow control step

This step **executes a group of child steps** as the **condition expression** set in the Condition property remains true.

5.4 Sequence JS Step

The JS Scope is useful to **modify Sequences**.

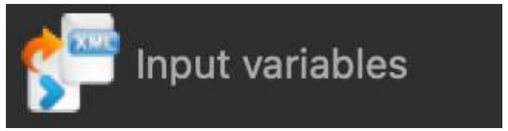
When you need to **write code directly in JavaScript**, the **Sequence JS step** is very helpful.
This JavaScript code will be **executed in the sequence scope**.



With the Sequence JS step, you can :

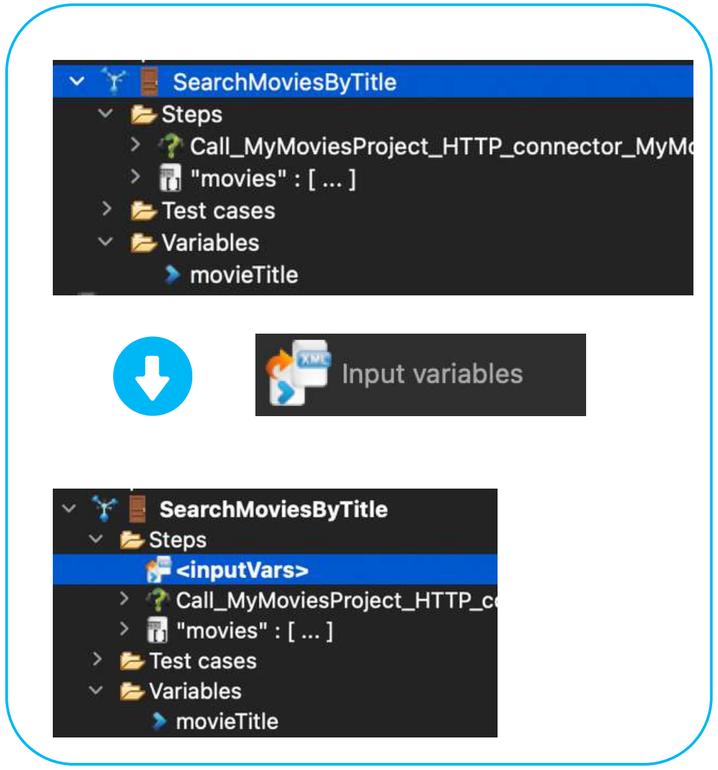
- initialize variables,
- perform complex calculations,
- access the context object to get useful properties (contextID, httpSession, isCacheEnabled, lockPooledContext, etc.)
- use some context methods to manipulate the result XML DOM, encode and decode data, abort sequence...

5.5 Input variables Step

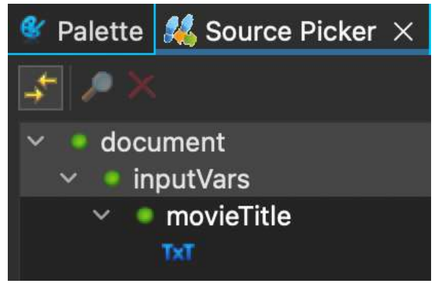


The **step Input variables** is an XML element containing dynamically the **input variables of parent Sequence**.

Placed at the **beginning of a Sequence**, this step allows **steps ordered after** to **use the Sequence input variables as source**.



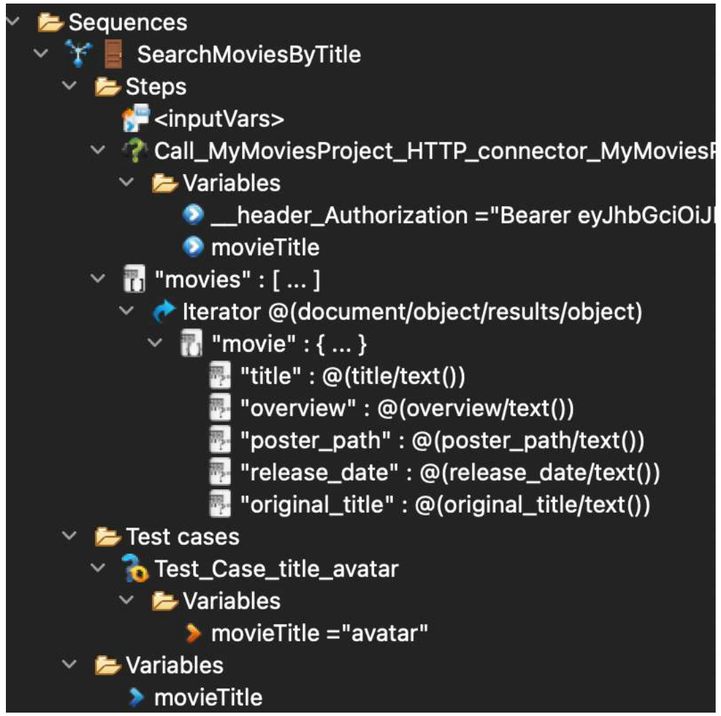
When you add it as the first step of the sequence, it **appears as a source** in the **source picker**.



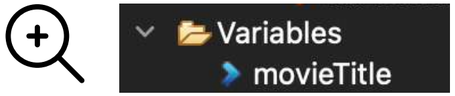
5.6 Modify a sequence with the JS Scope

Exercice 1 : Change a variable name with Sequence JS

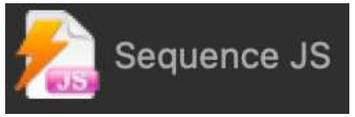
Here is our sequence SearchMoviesByTitle



Let's say we want the name of the **input variable** "**movieTitle**" to appear as **title** in our sequence.



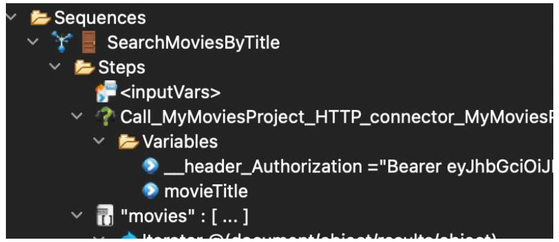
To change the name of the input variable "movieTitle", we are going to use JavaScript in a **Sequence JS**.

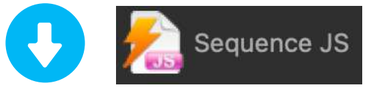


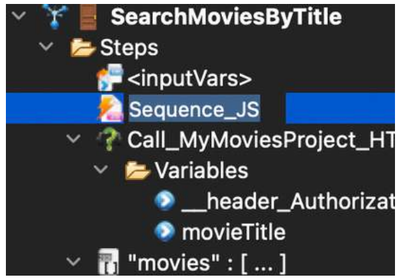
5.6 Modify a sequence with the JS Scope

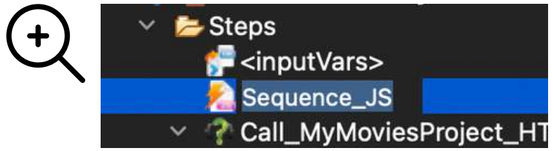
Exercice 1 : Change a variable name with Sequence JS

Drag the **Sequence JS step** from the palette in the steps folder after the step InputVars.

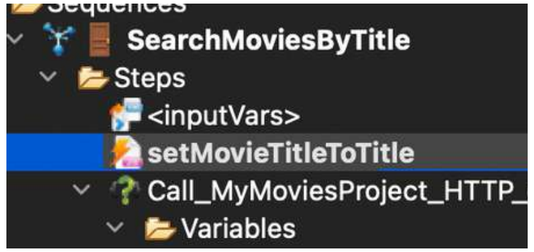








Rename it setMovieTitleToTitle.

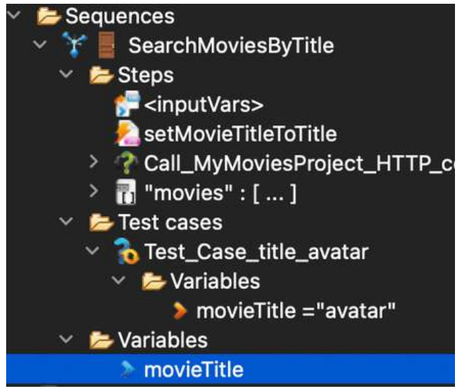


Click twice on **setMovieTitleToTitle** to open the **file setMovieTitleToTitle.js** in the editor panel

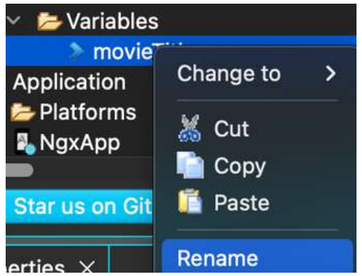
Change the variable name in the file with JavaScript.

5.6 Modify a sequence with the JS Scope

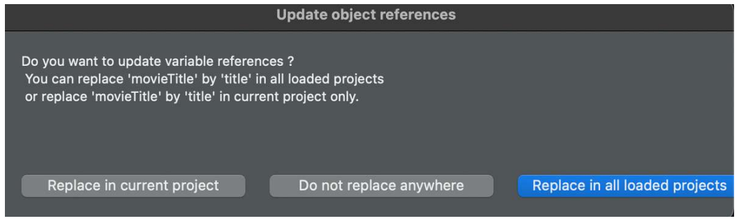
Exercice 1 : Change a variable name with Sequence JS

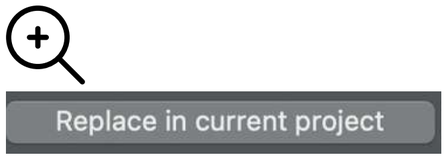


Rename the **movieTitle variable** to **title** in the **Variables folde**r of the sequence.

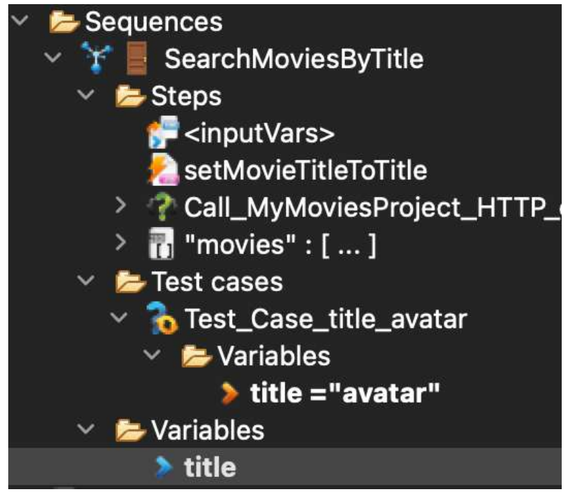


In the **Update object references window**, select **Replace in current project**.





The variable **appears as title** in the sequence (Test cases, Variables folder)



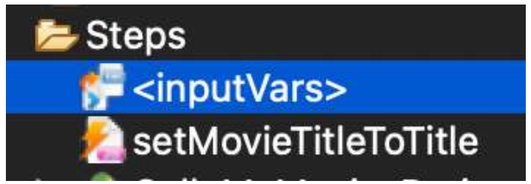
5.6 Modify a sequence with the JS Scope

Exercice 1 : Change a variable name with Sequence JS

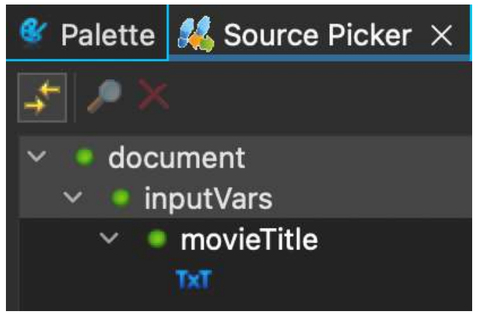
When focused on **inputVars step**,

the source picker shows the entry variable of the sequence as **title**

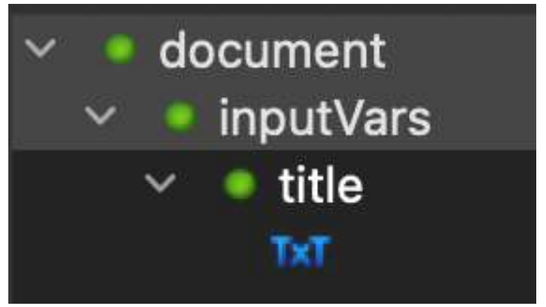
(not as movieTitle anymore).



Before



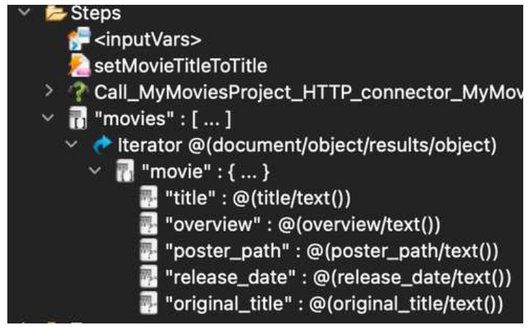
Now

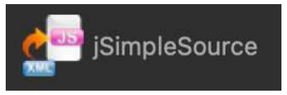


5.6 Modify a sequence with the JS Scope

Exercice 2 : Set the title to uppercase with Sequence JS

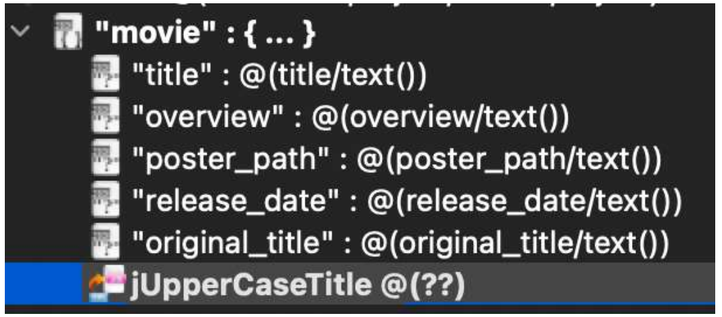
Let’s say we want a field with the title in uppercase in our JSON data.





In the movie object,
after the field steps,
we **add a jSimpleSource step**..

We name it **jUpperCaseTitle**
(for JS variables, good practice is
to add a “j” at the beginning)

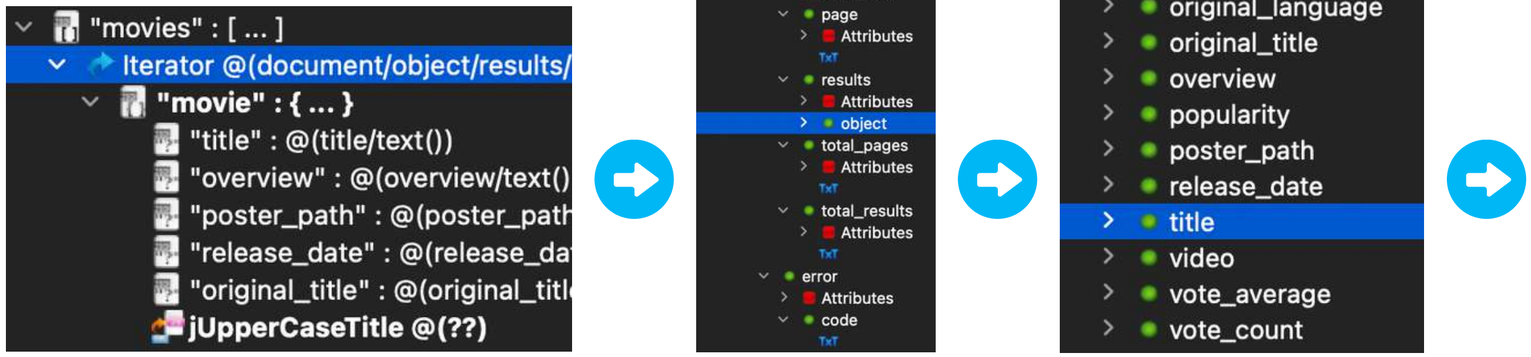


5.6 Modify a sequence with the JS Scope

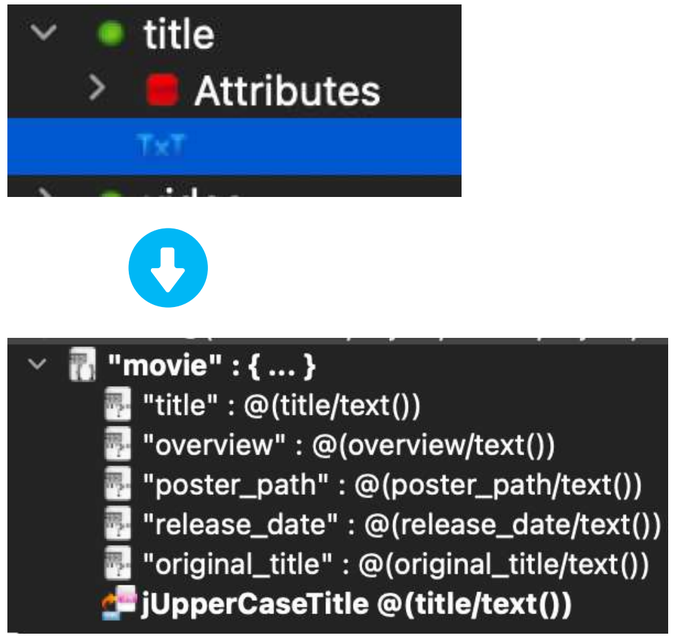
Exercice 2 : Set the title to uppercase with Sequence JS

The jSimpleSource step is used
to **transform a single node** from a source **into a JS variable**.
Now we want to **bind it to the value of the field title in the iterator**.

Double-click on the **Iterator step**
to display its data in the Source Picker
and open the object node.



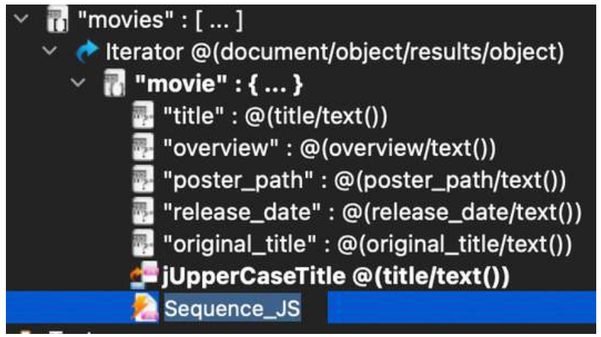
Drag and drop
the **TxT element of the title**
into the jSimpleSource step.



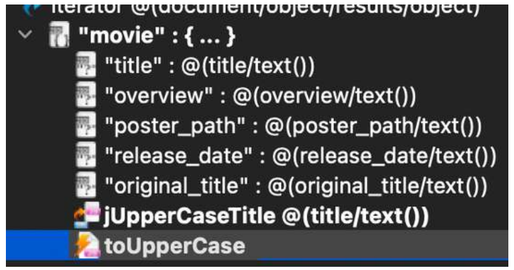
5.6 Modify a sequence with the JS Scope

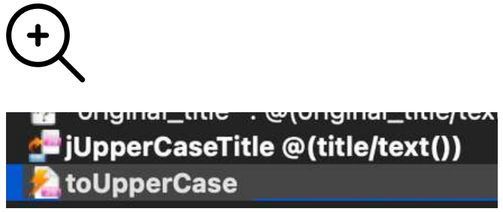
Exercice 2 : Set the title to uppercase with Sequence JS

Add a Sequence JS step
in the sequence,
after the step jUpperCaseTitle.



Rename it toUpperCase.





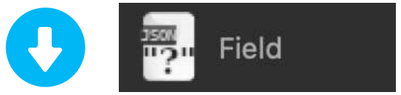
Click twice on the step
to open toUpperCase.js

Edit the toUpperCase.js
with JS code.

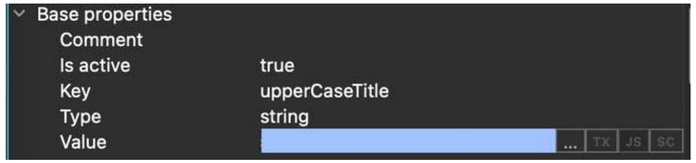
5.6 Modify a sequence with the JS Scope

Exercice 2 : Set the title to uppercase with Sequence JS

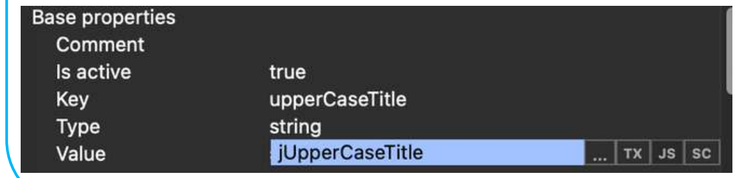
Add a field step after toUpperCase and name it **upperCaseTitle**.



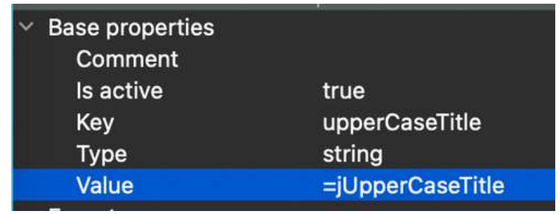
In the **Value property of upperCaseTitle**, select the **JS Scope** by clicking on JS.



Enter **jUpperCaseTitle** to select the JS variable as value.



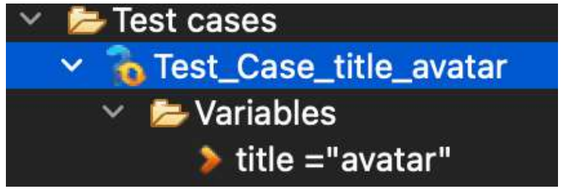
The value of upperCaseTitle is now sourced on the value of the JS variable jUpperCaseTitle.



5.6 Modify a sequence with the JS Scope

Exercice 2 : Set the title to uppercase with Sequence JS

Let’s run the Test Case

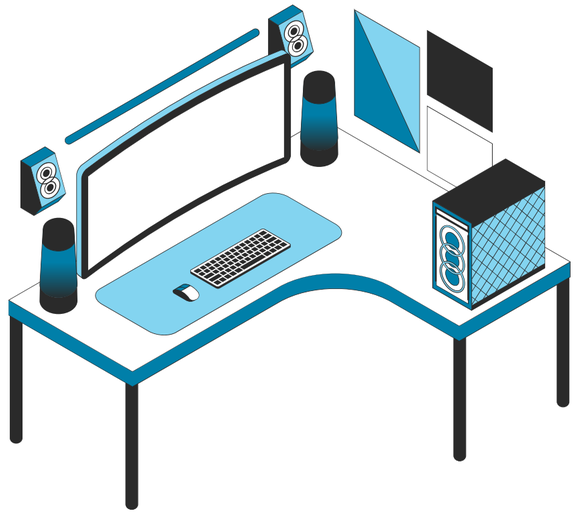


The key upperCaseTitle and the value in upperCase appears in the response data


```
{
  "movies": [
    {
      "title": "Avatar",
      "overview": "Un marine paraplégique, envoyé sur la lune Pandora
      "poster_path": "/3npygfmEhqnmNTmDWhHLz1LPcbA.jpg",
      "release_date": "2009-12-15",
      "original_title": "Avatar",
      "upperCaseTitle": "AVATAR"
    },
    {
      "title": "Avatar : La Voie de l'eau",
      "overview": "Une dizaine d'années se sont écoulés depuis les pr
      "poster_path": "/hYeB9GpFaT7ysabBoGG5rbo9mF4.jpg",
      "release_date": "2022-12-14",
      "original_title": "Avatar: The Way of Water",
```

6 – Error Management

How to handle errors in the studio.



6.1 Basics on Error Management

6.2 Error Management steps

6.3 Error node & error tag

6.4 Using the IfExist step

6.5 Using the Error Structure step

6.6 Using the Return step

6.1 Basics on Error Management

During its execution, a step can fail, and an error happens.

There are two types of errors:

- Functional errors
- System errors

In Convertigo, to handle errors:

- We don't start with "If everything is OK, then...Or else..."

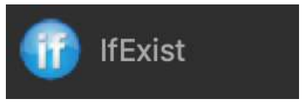
  => otherwise, there would be too much depth in the tree structure.
- We start with "If there is a problem, then...Or else...,"

  => It means we begin error handling before dealing with successful execution.

After each transaction call in a sequence, we test for errors.

6.2 Error Management steps

Convertigo provides steps to handle errors in sequences.

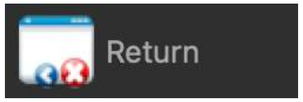


IfExist – Flow control step

This step is used to define an **IF condition looking for node(s) on a source**.
It contains **other steps executed only if** the **source** defined through the Source property **exists**.

Error structure – XML step

This step is used
to **generate an output XML structure** corresponding to an **applicative error**.
It **doesn't break** the **sequence execution flow**.

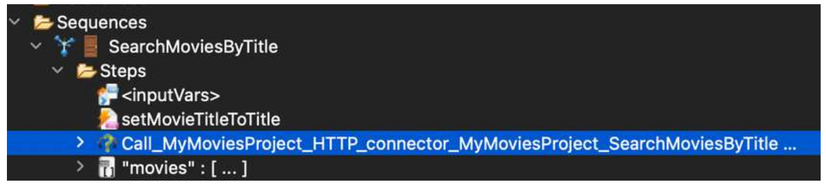


Return – Flow control step

This step is used to **exit the current sequence** in which it is positioned.

6.3 Error node & error tag

Let's have a look on the XML structure
of the transaction call
in the sequence SearchMoviesByTitle.



Double click on the transaction
to display its structure
in the source picker

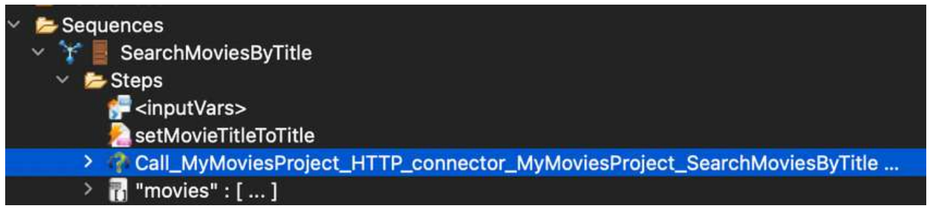
In the **XML structure of a source**,
there is always an **error node**,
so that **errors can be picked or sourced**
if they are present.

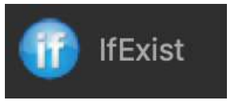
When a system error
or a functional error happens,
it **generates an error tag**
which "fills" the error node.

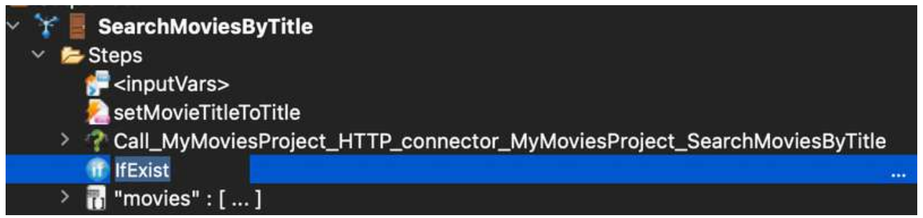
The **error tag structure** is **standardized**.
The error will always be in the same place
and have the same format in the sequence.

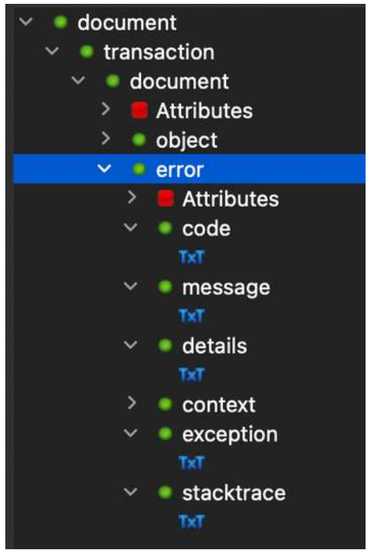
6.4 Using the IfExist step

In the **sequence SearchMoviesByTitle**,
just **after the transaction call**
and **before the array movies**,
let's **add an IfExist step**.

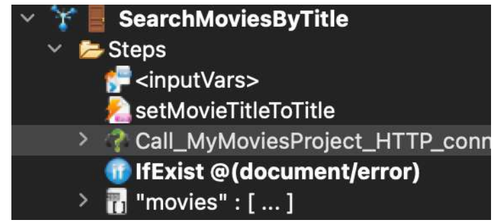








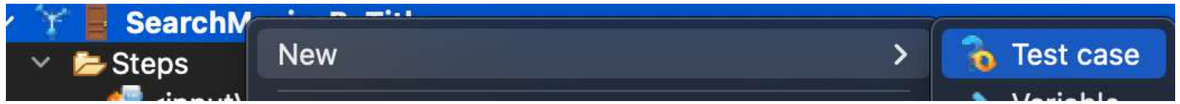
Then **drag the node error**
of the XML structure
of the transaction call
in the step IfExist.



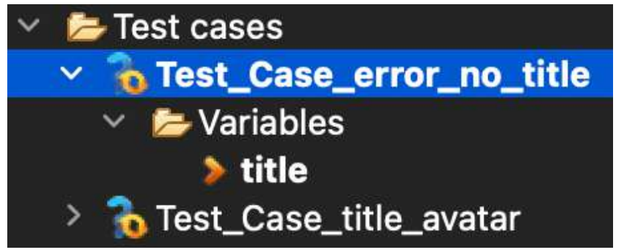
When there is any kind of error,
an **error tag is generated within the error node**.
The IfExists step **checks the XPath of the transaction call**
to see if there is an **error tag in the XPath**.

6.4 Using the IfExist step

Let’s test it by adding a new test case with an error.



Our test case is created.



In the properties,
the Default value of variable title
is null.

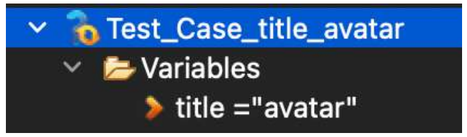
The transaction needs a title to search a movie.

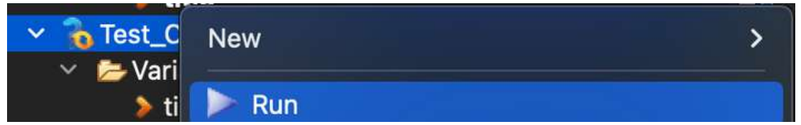
If we forget to add a value to the variable title, an error will be generated.

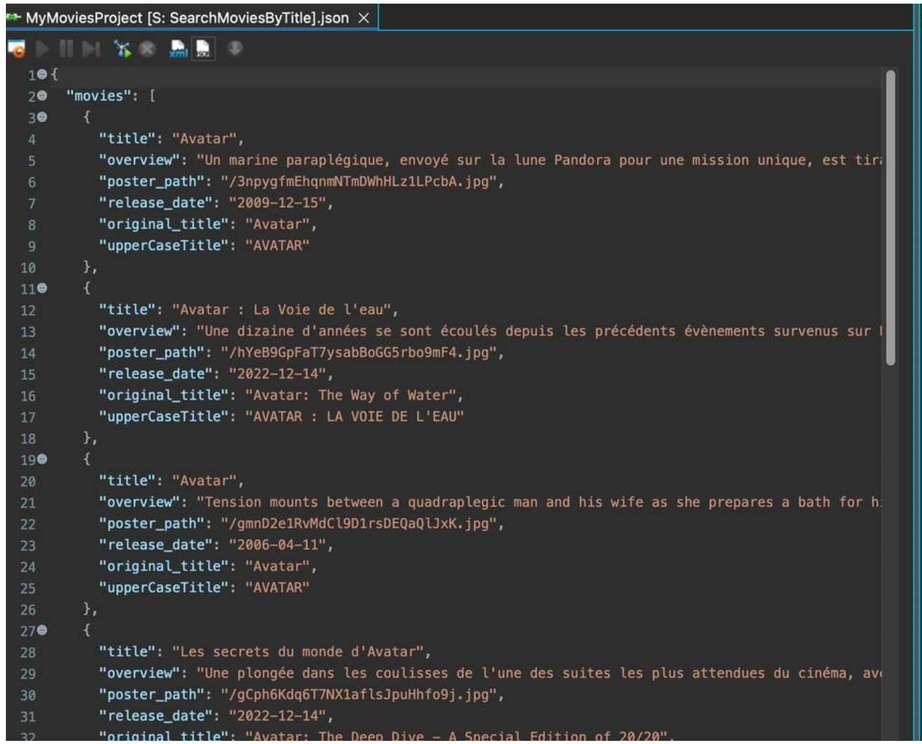
6.4 Using the IfExist step

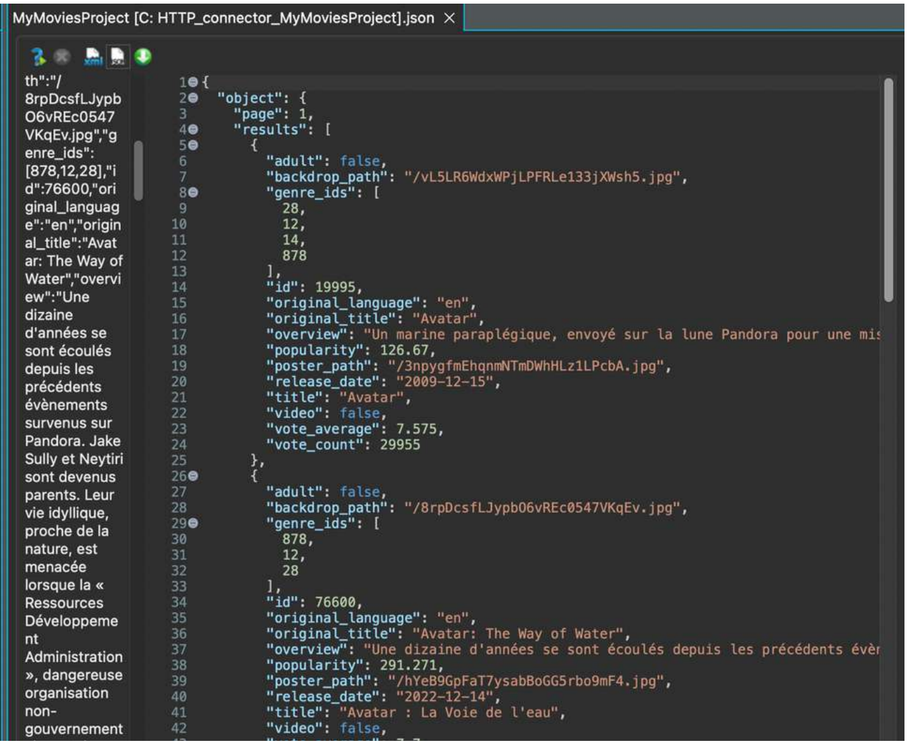
A a reminder, this is the result of the sequence execution

with a test case where the title has a value and everything is OK.



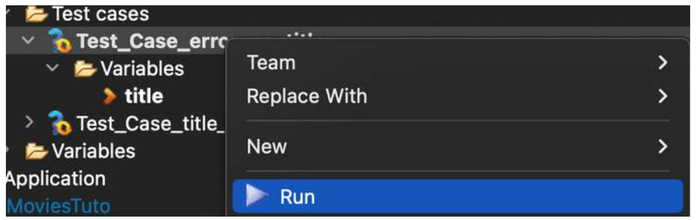






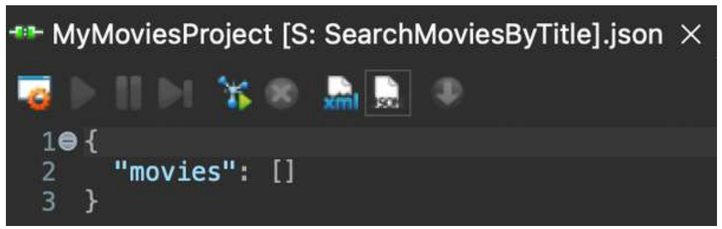
6.4 Using the IfExist step

Let’s run the error test case



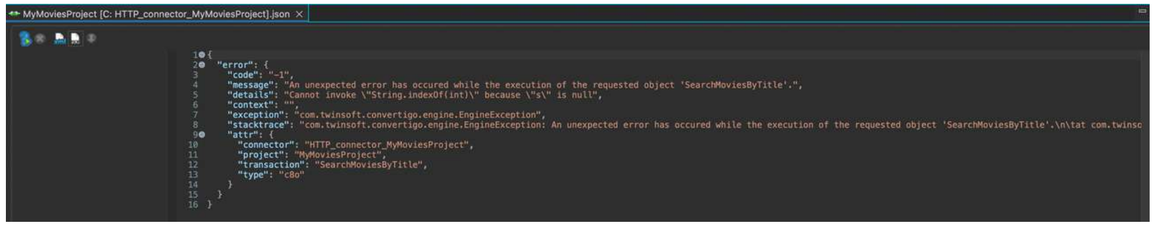
Without a title,

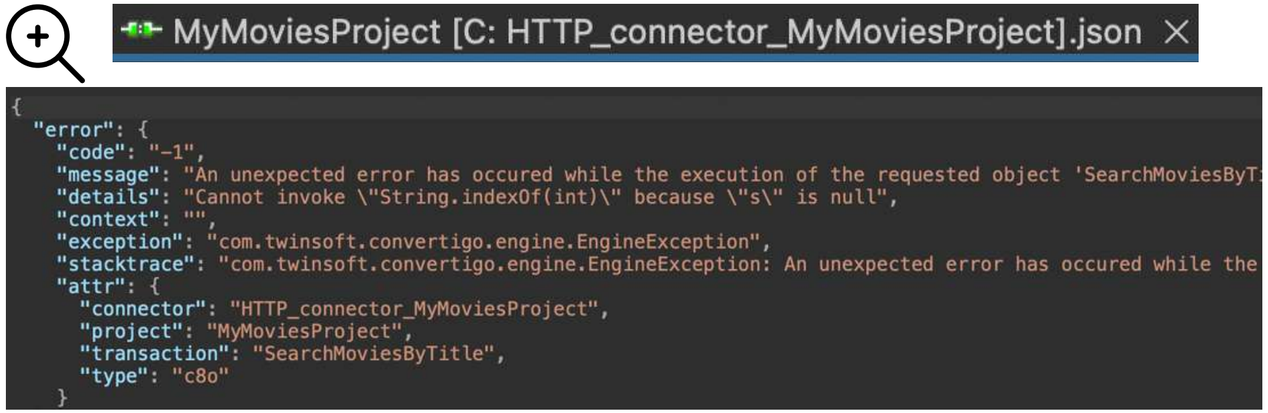
the sequence returns an empty array



The **transaction returns an error**,

and an **error tag is generated.**





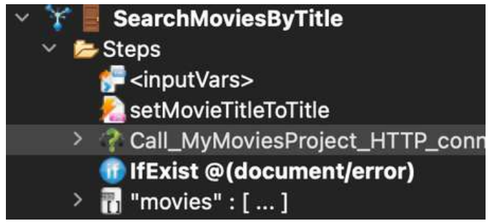
The IfExists step detects the error.

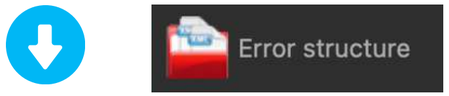
We are going to use it in combination to other steps to handle this error.

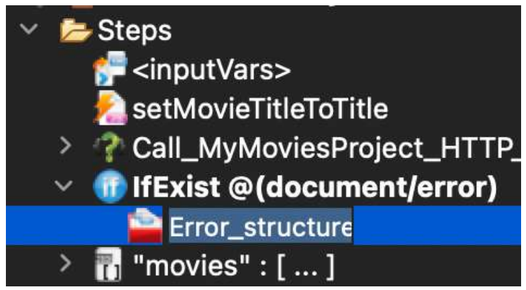
6.5 Using the Error Structure step

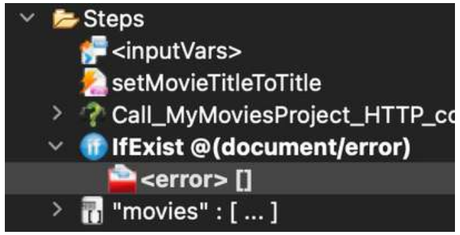
To report the error to the client, we use the **Error Structure step**,

Let's add an **Error structure step** in the **IfExist step**.







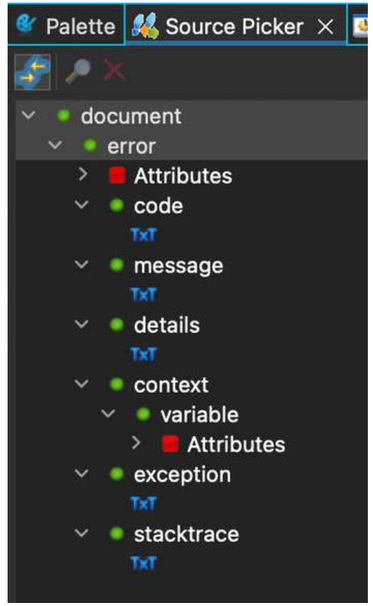


This step has the properties

- **Code** (error status code)
- **Message**
- **Details**

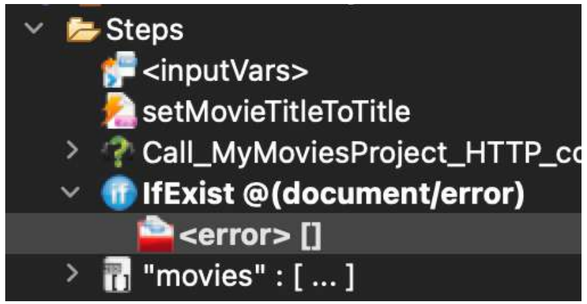
These properties can be **sourced from the original error message** returned by the API in the transaction.

In the source picker, the **XML Structure** of the **Error structure step** has the same properties.

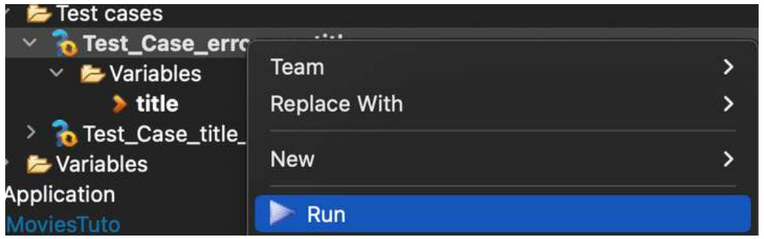


6.5 Using the Error Structure step

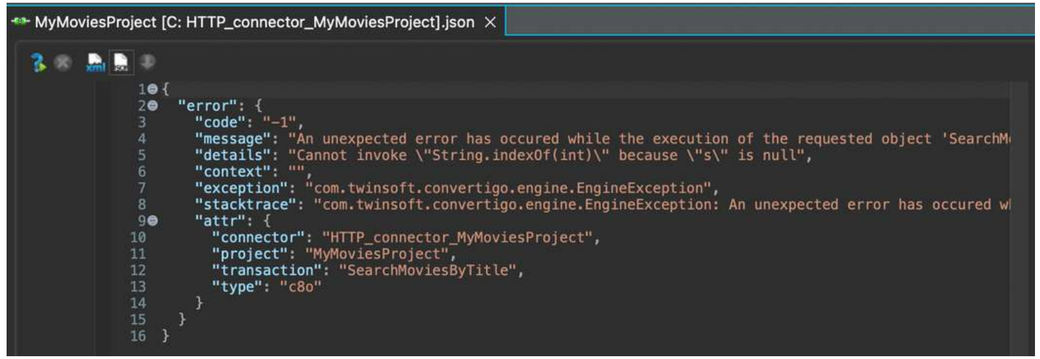
Let’s see what difference it makes to have this Error structure step in the sequence.



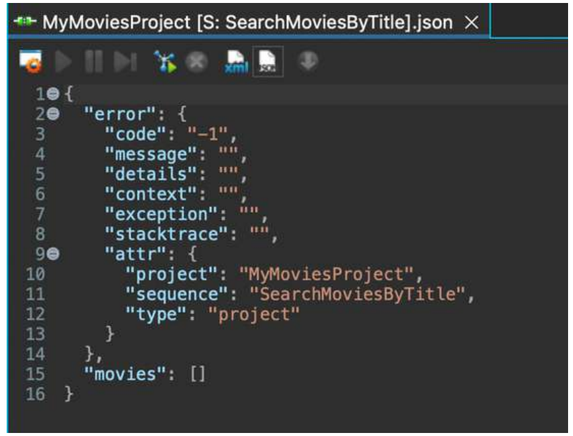
Let’s run the error test case again.



The original error message returned by the transaction is the same

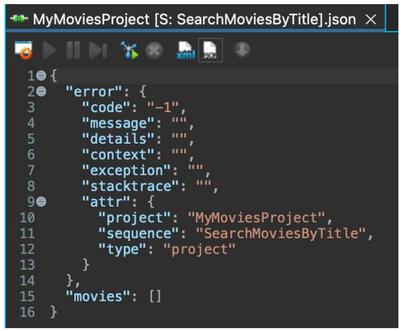


The sequence returns **an empty array and an error**.

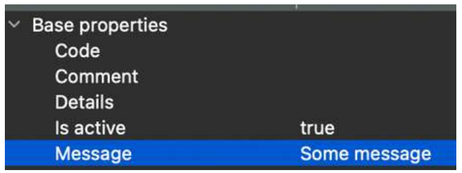


6.5 Using the Error Structure step

By changing the properties of the Error structure step,
we can customize the error returned by the sequence.

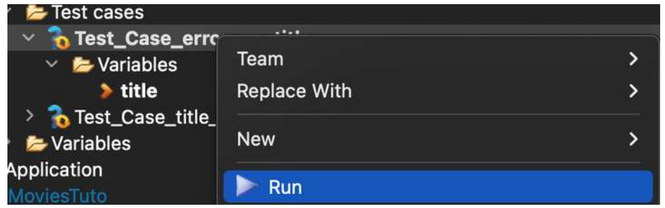


Let's customize our error message.

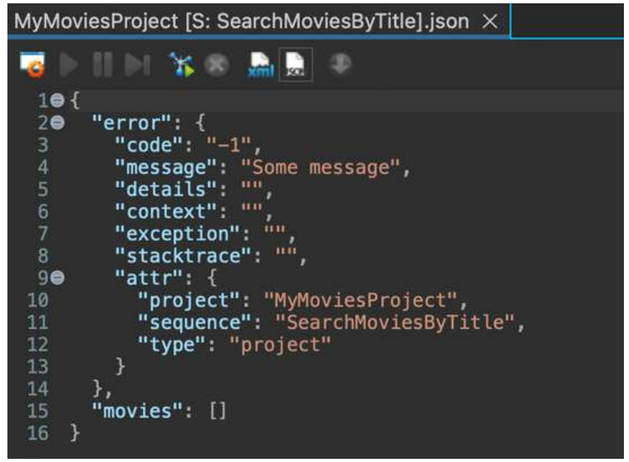


We can enter a error message
directly in the message properties.

The error message appears
in the error structure step of the sequence.



When running the test case,
the message will appear in the return
of the execution of the sequence

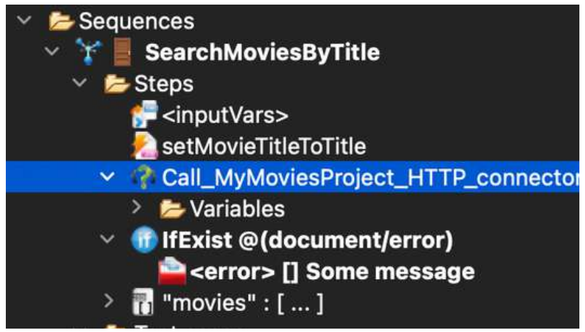


6.5 Using the Error Structure step

Now, let's **customize the error** returned by the sequence **dynamically**.

by using the original error message returned by the API,

and the properties of the Error Structure step.



Click twice on the transaction call

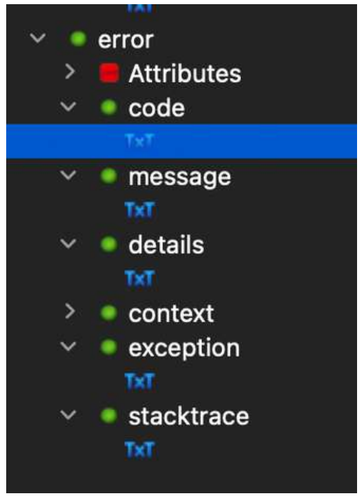
to display its XML structure in the source picker.

In the source picker,

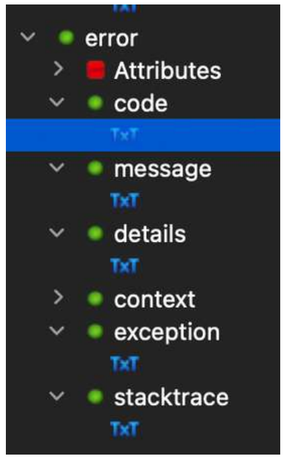
we can see the **nodes code, message and details**.

We are going to use these nodes

to **source the properties** of the Error Structure step.



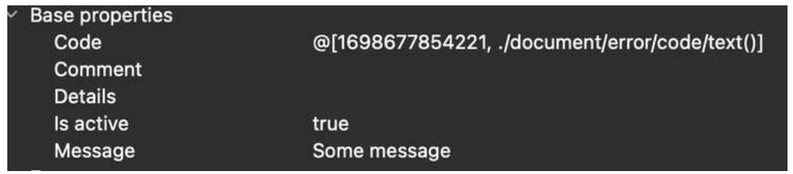
6.5 Using the Error Structure step



Drag the TxT from the code node

into the error structure step in the sequence.

The source appears in the properties



Do the same thing with the TxT from the message and the detail nodes

6.5 Using the Error Structure step

When we run the test case, the error returned by the sequence
has the same code, message and details as the original error message returned by the API.

Error returned by the sequence

MyMoviesProject [S: SearchMoviesByTitle].json

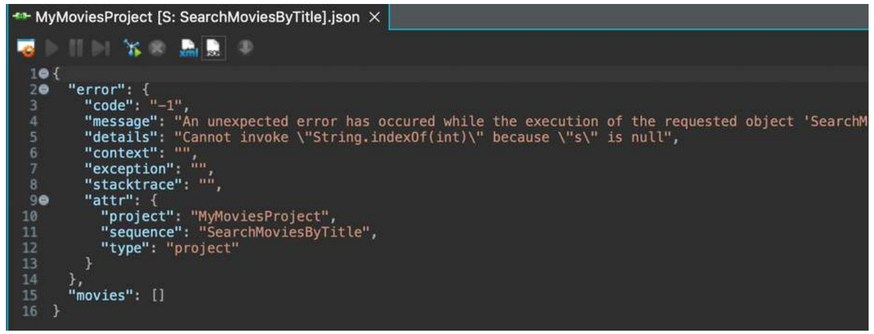

```
{
  "error": {
    "code": "-1",
    "message": "An unexpected error has occured while the execution of the requested object 'SearchM
    "details": "Cannot invoke \"String.indexOf(int)\" because \"s\" is null",
    "context": "",
    "exception": "",
    "stacktrace": "",
    "attr": {
      "project": "MyMoviesProject",
      "sequence": "SearchMoviesByTitle",
      "type": "project"
    }
  },
  "movies": []
}
```


Original error message returned by the API

MyMoviesProject [C: HTTP_connector_MyMoviesProject].json


```
{
  "error": {
    "code": "-1",
    "message": "An unexpected error has occured while the execution of the requested object 'SearchM
    "details": "Cannot invoke \"String.indexOf(int)\" because \"s\" is null",
    "context": "",
    "exception": "com.twinsoft.convertigo.engine.EngineException",
    "stacktrace": "com.twinsoft.convertigo.engine.EngineException: An unexpected error has occured wi
    "attr": {
      "connector": "HTTP_connector_MyMoviesProject",
      "project": "MyMoviesProject",
      "transaction": "SearchMoviesByTitle",
      "type": "c8o"
    }
  }
}
```

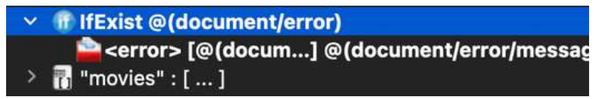
6.6 Using the Return step

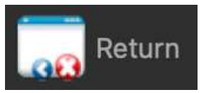


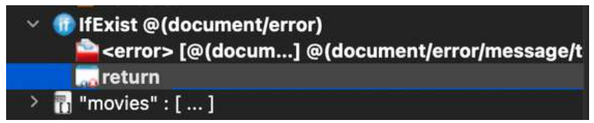
After the error structure step, the sequence didn't stop and the following steps were executed.

That's why we see an empty movies array after the error.

To stop the sequence after an error,

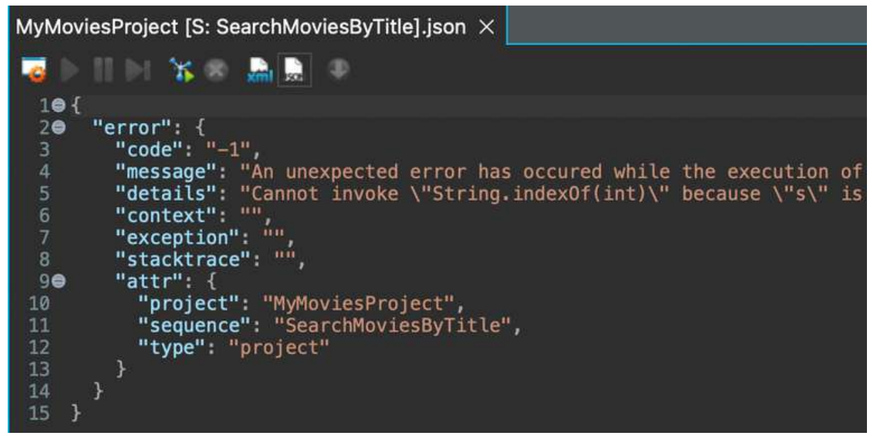






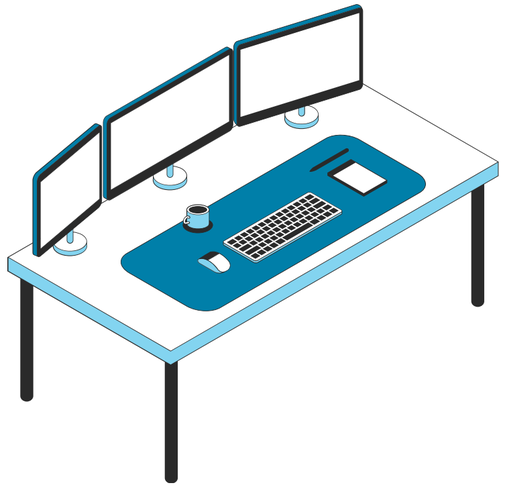
we add a **Return step after the Error Structure step in the IfExist step.**

The sequence execution is stopped and the empty movies array disappears from the result of the sequence.



7 – Collaboration with Git

How to share your projects with Git Versioning.



7.1 Git basics with Convertigo

7.2 Git Repositories View

7.3 Git Staging View

7.4 Compare mode

7.5 Commit your changes

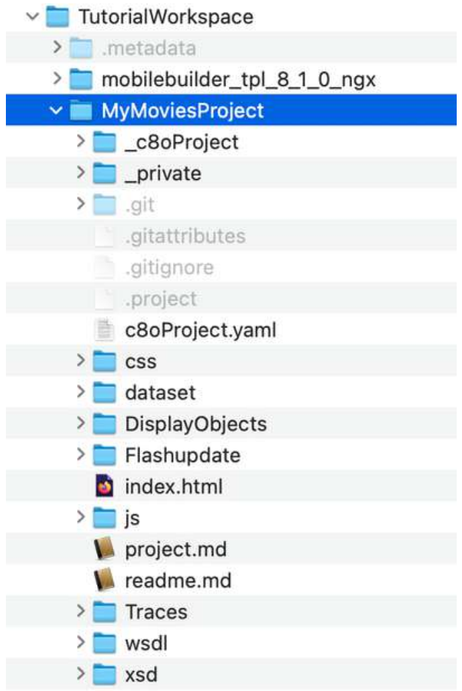
7.6 Clone a project

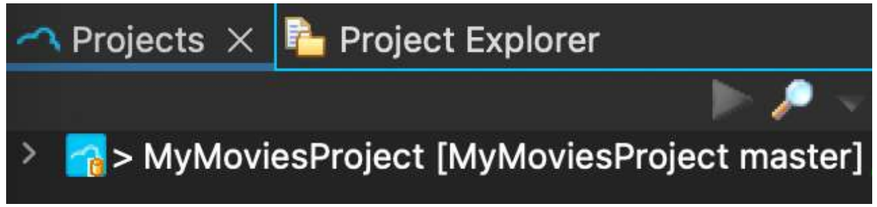
7.1 Git basics with Convertigo

When you create a new project, a **Git Repository is automatically created**.

In the Projects folder, the name of your project

is followed by the **name of the branch** you're currently working on.





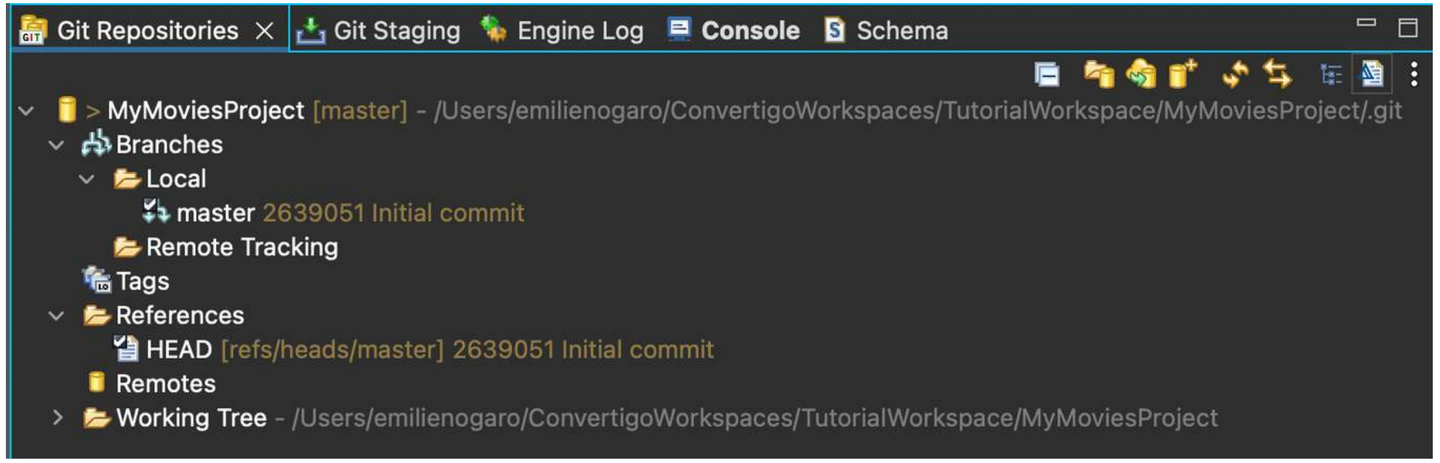
In the studio interface, two views are used to manage Git in your projects.

- Git Repositories 

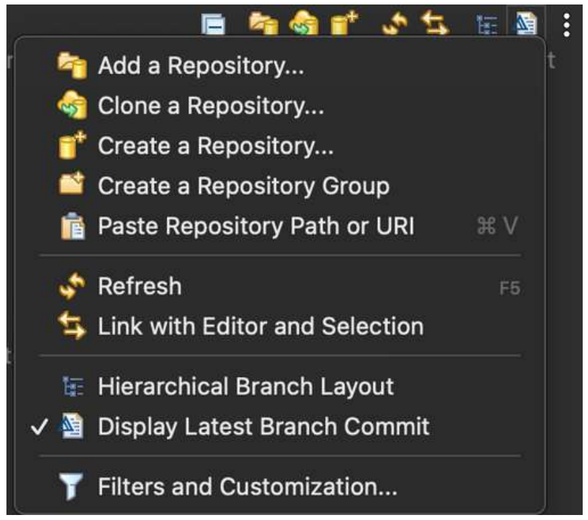
- Git Staging 

7.2 Git Repositories View

In the **Git Repositories View**,
you can see the Git Repositories of **all the projects in your workspace**.



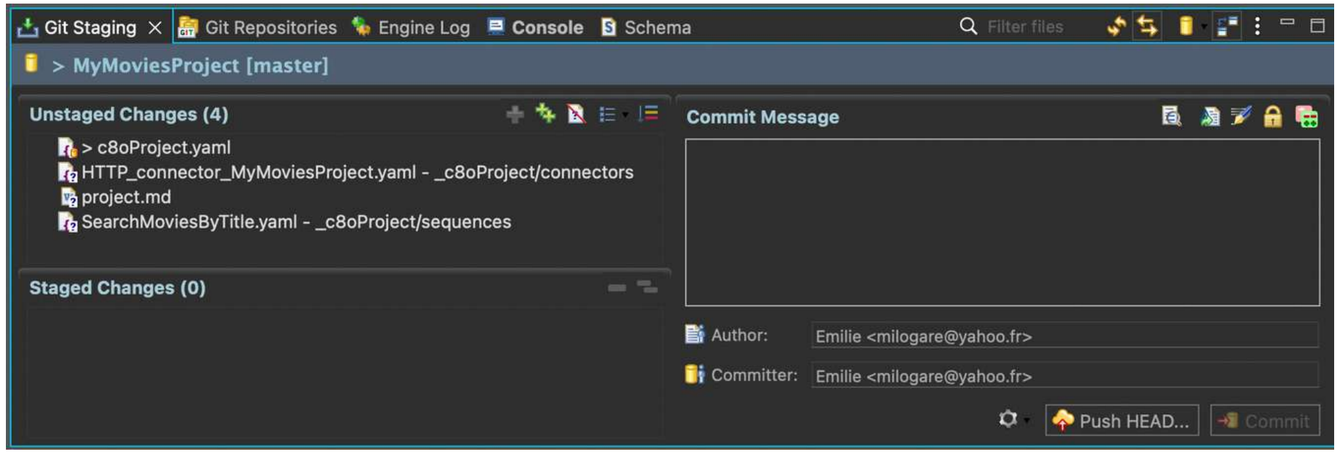
It provides Eclipse features
to manage your Git repositories.



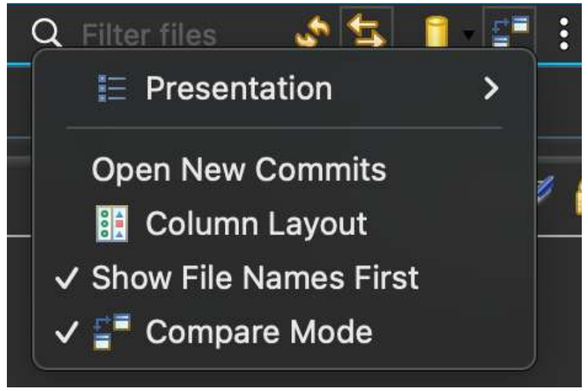
7.3 Git Staging View

In the **Git Staging view**, you can manage your **git workflow**,
and **commit your changes** to your **local and remote directories**.

The files that have been modified since the last commit are shown in **Unstaged changes**.

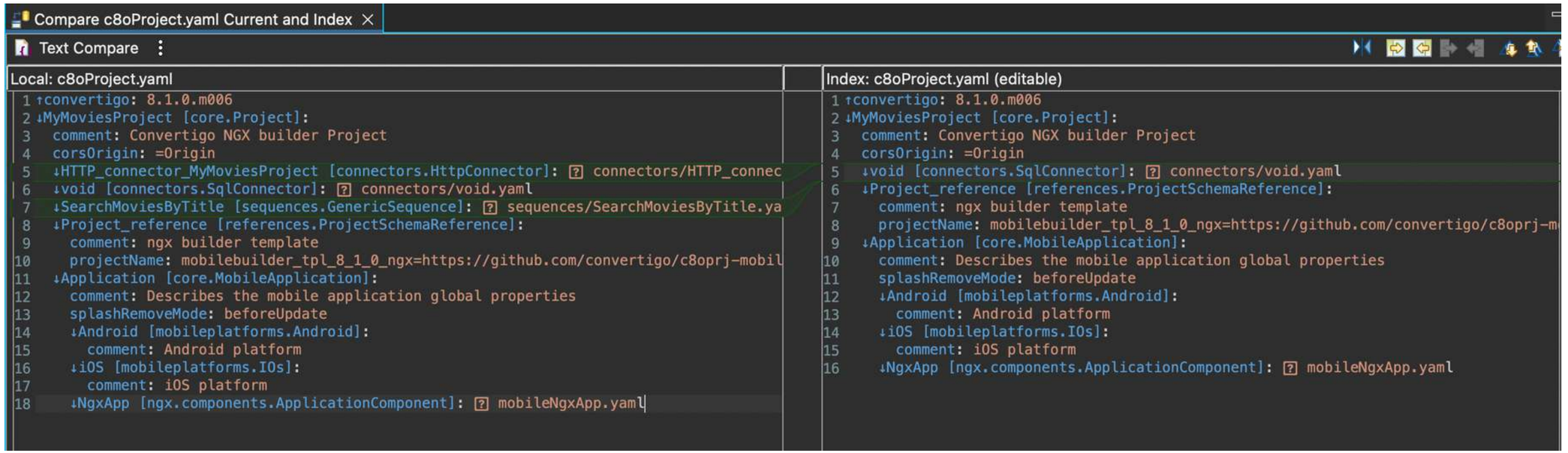


It provides Eclipse features
to manage your Git workflow.



7.4 Compare Mode

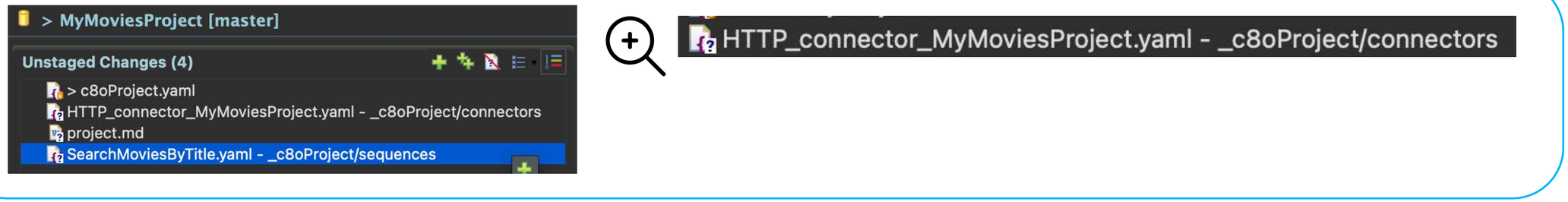
With the **Compare Mode**, you can display the **differences with the previous commit**, and **resolve conflicts** when necessary.

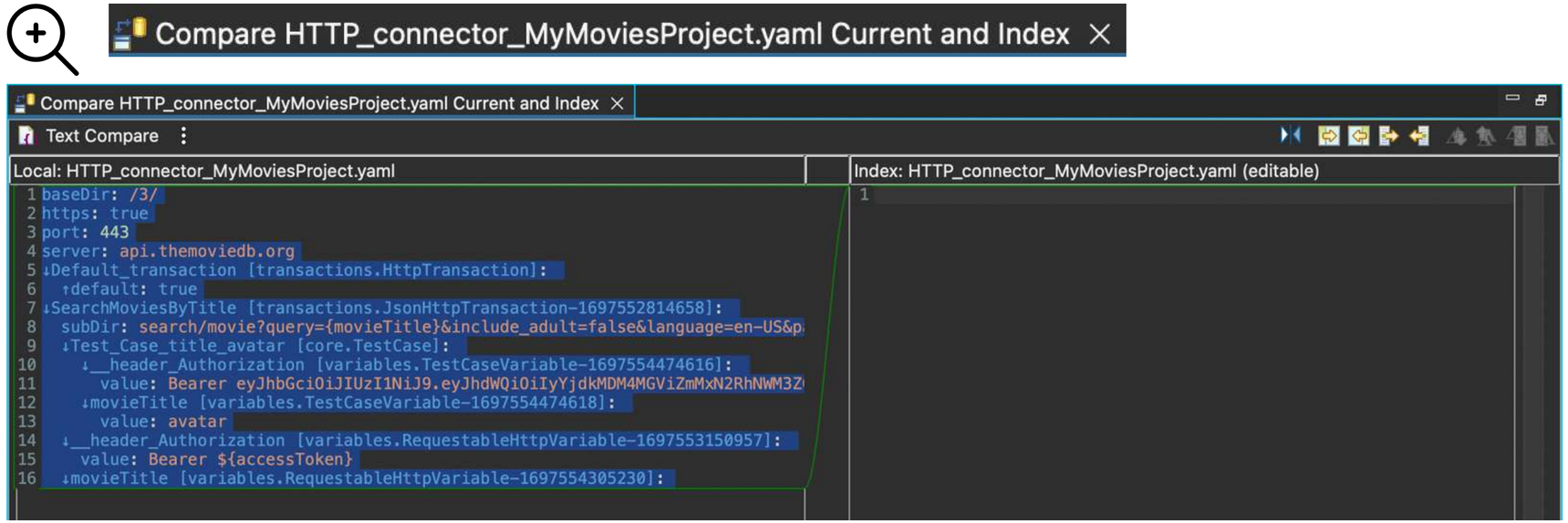


These icons represent **features** allowing you to **manipulate the files** : navigate to the next or previous changes, swap the views, copy changes from one view to the other...

7.4 Compare Mode

Right-click twice on a file in the Staging view to open the compare mode.





You can see the changes since the last commit :

Here the **index view** is empty because the connector was created after the last commit

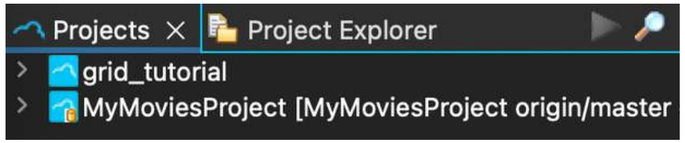
7.5 Create a repository

When you **create a new project** in your workspace, a **Git Repository is automatically created**.

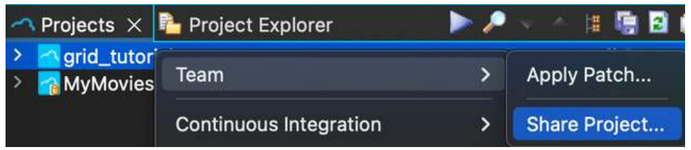
But if you **import a project from a .car file**, you have to **create it manually**.

Let's say we want to create a Git Repository for the the project grid_tutorial

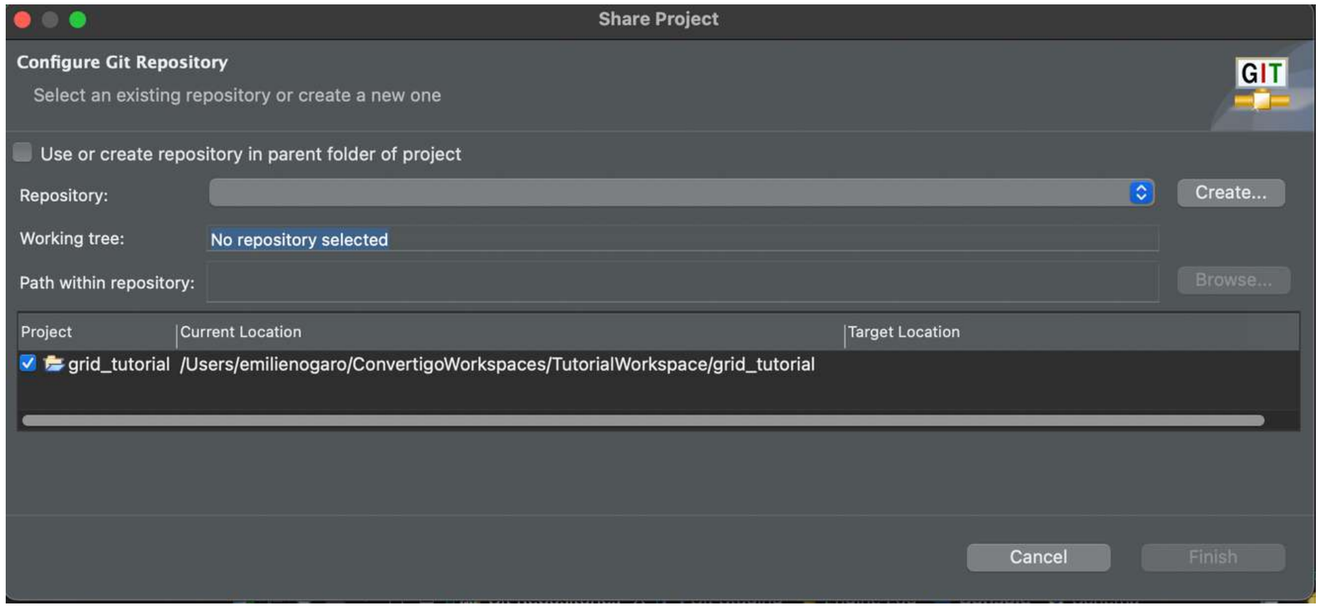
Right-click on the project grid_tutorial in the Projects view.



Select Team > then Share Project

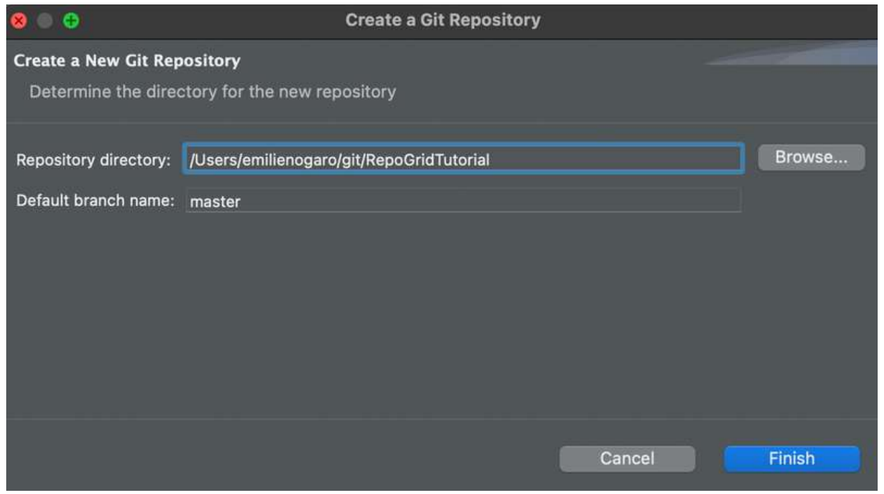


In the Configure Git Repository window of the Share Project Window, click on Create to create the repository

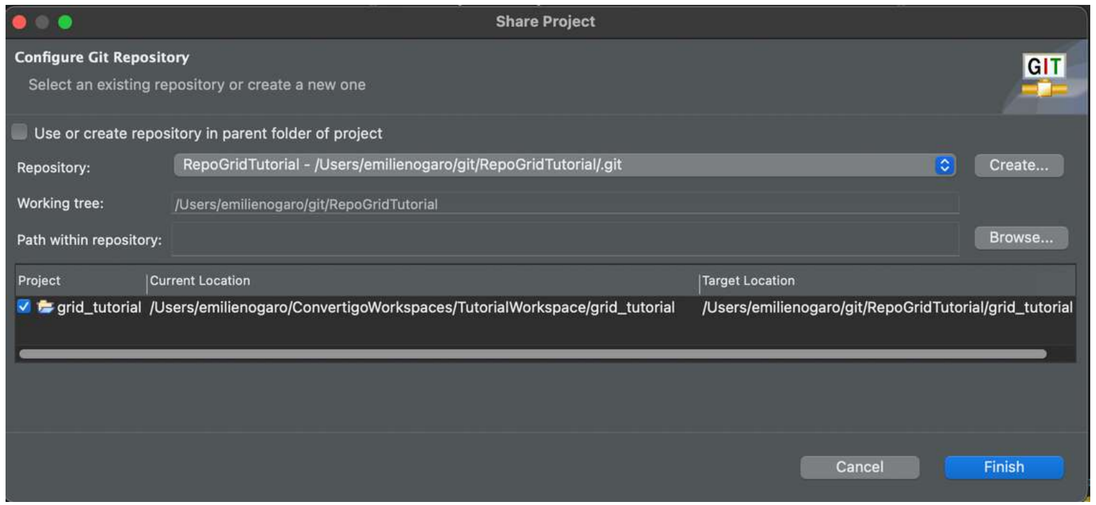


7.5 Create a repository

In the **Create a new Git Repository window**,
change the repository name
(repository by default)
to RepoGridTutorial.
Then click on **Finish**.

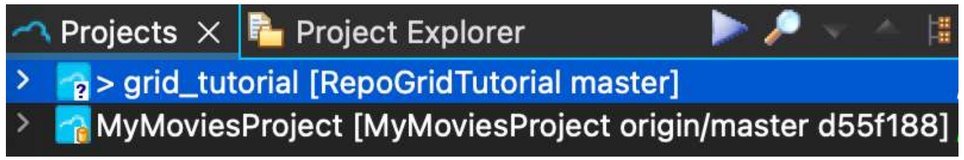


In the **Configure Git Repository window**,
you can see the **repository name and its path**.
Click on **Finish**.

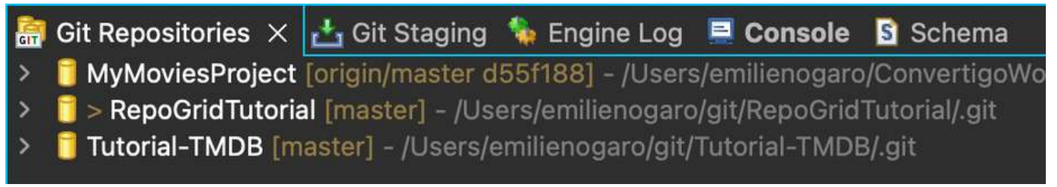


7.5 Create a repository

In the **Projects view**,
the **repository name** and the **branch name**
appears after the project name.



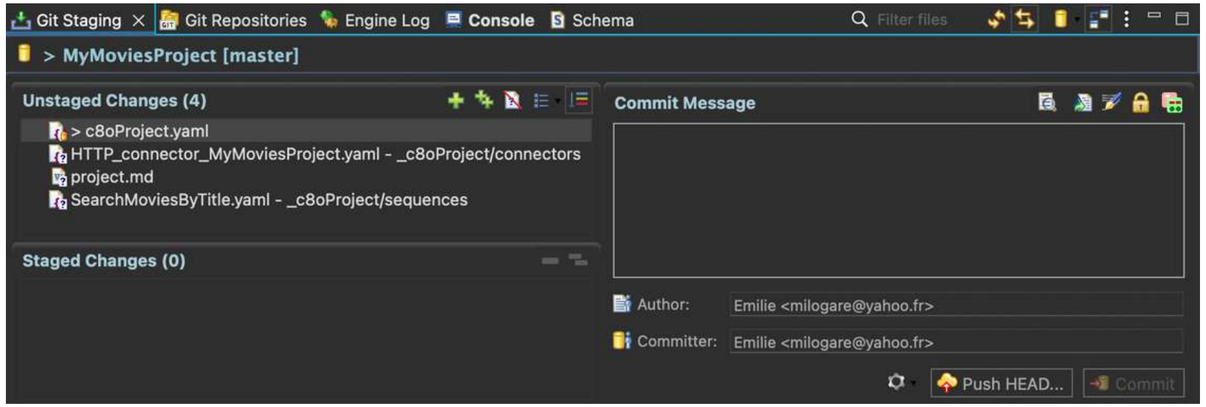
In the **Git Repositories view**,
the **repository name**, the **branch name**
and the **path to the Git repository** appears.

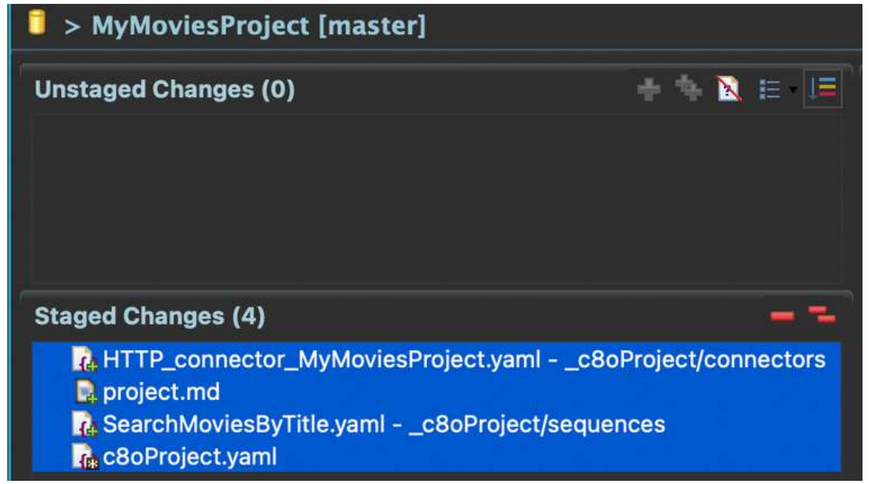


7.6 Commit your changes

Let's say you have **made a few changes** in your project and you want to **commit them on a Git repository**.

Go the Git Staging view and stage your files.





Stage your files with the green cross

 One by One

 Or All at Once

You can also Unstage them with the red line

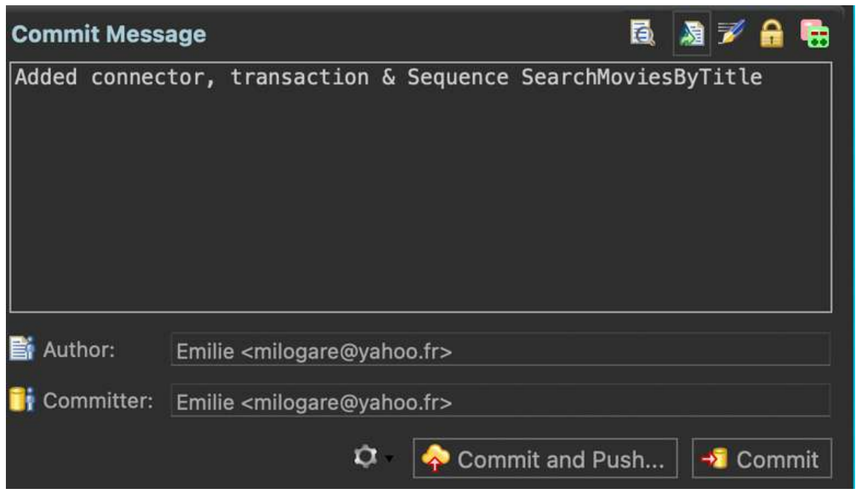
 One by One

 Or All at Once

7.6 Commit your changes

Add a commit message

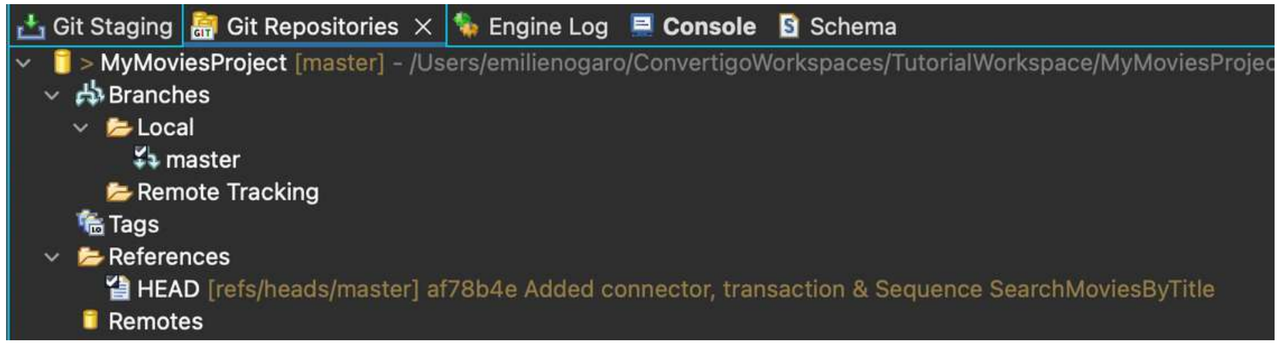
and click on **Commit**.



Your changes have been **committed to your local Git repository**

In the **Git Repositories view**,

you can see the **latest commit** in the **References folder**.



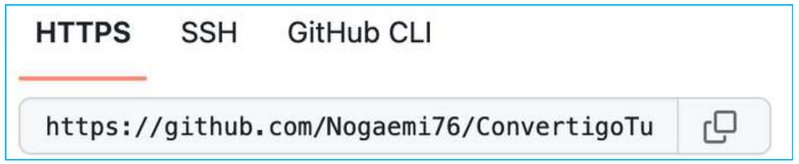
7.6 Commit your changes

At this stage, only your **local Git repository** has been initialized.

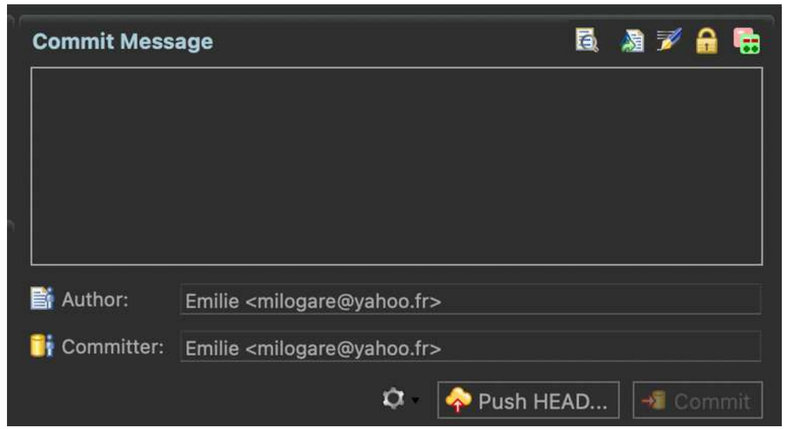
Let's add a **remote repository** to your project.

Create an **empty remote repository in GitHub or GitLab.**

Copy your repo URI to the clipboard.



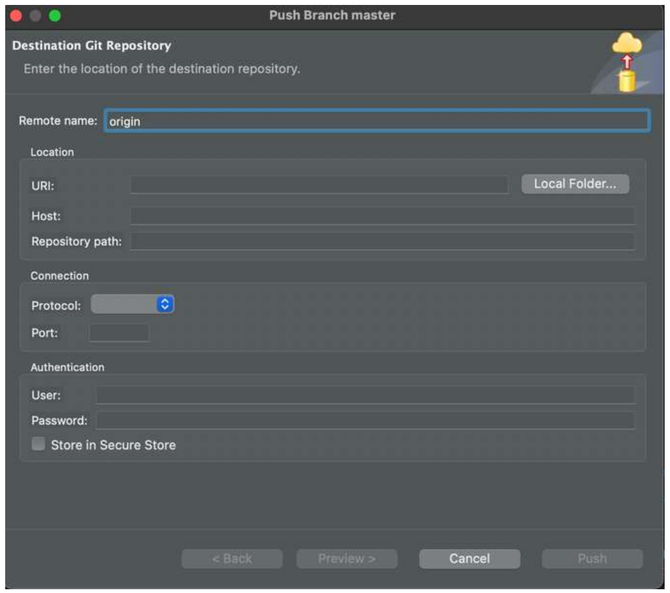
In the **Git Staging view**



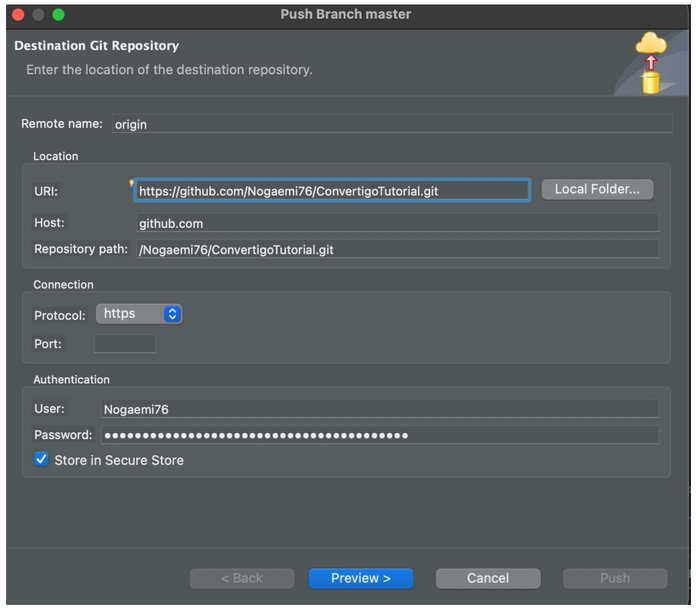
Click on **Push HEAD**.

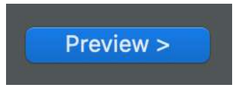
7.6 Commit your changes

In the **Destination Git Repository window** of the **Push Branch Master window**, paste the **URI in the URI field**.



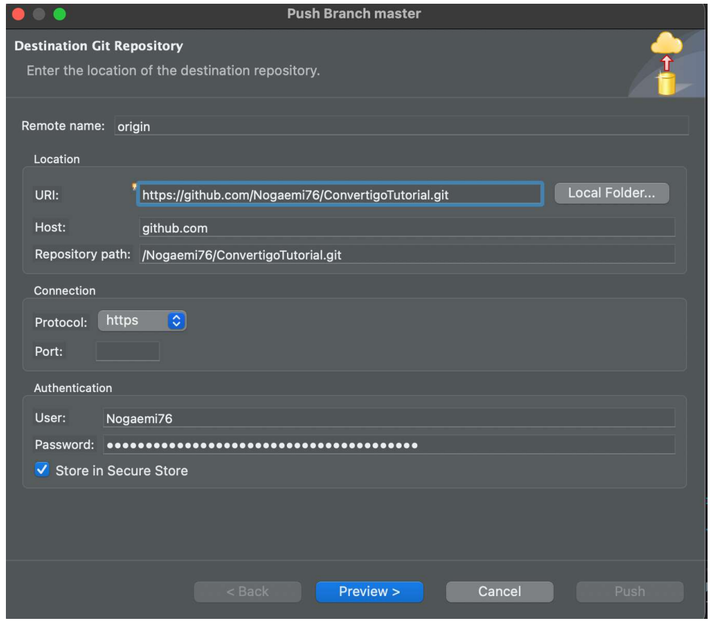
The other fields will update automatically





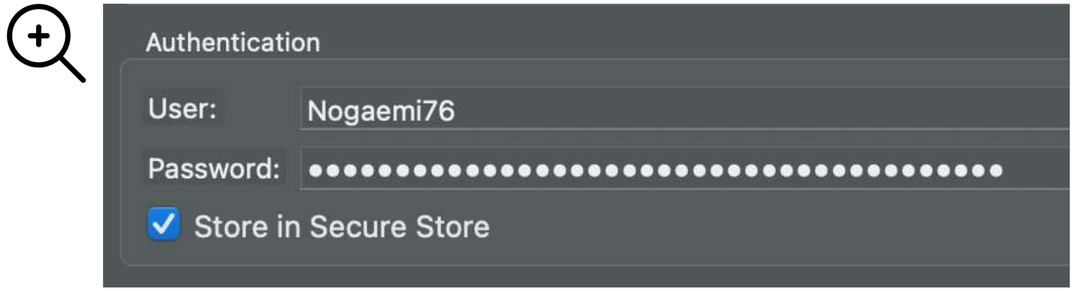
Click on **Preview>**.

7.6 Commit your changes



Reminder : In the Authentication part of the Push Branch Master window.

- User is your GitHub Username
- Password is a Personal access token from GitHub

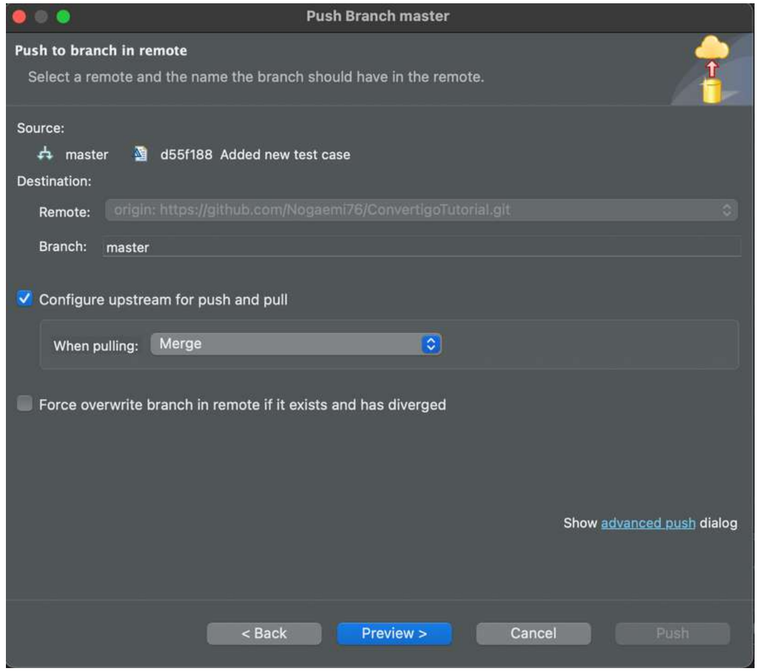


7.6 Commit your changes

The **Push to branch in remote** window appears.

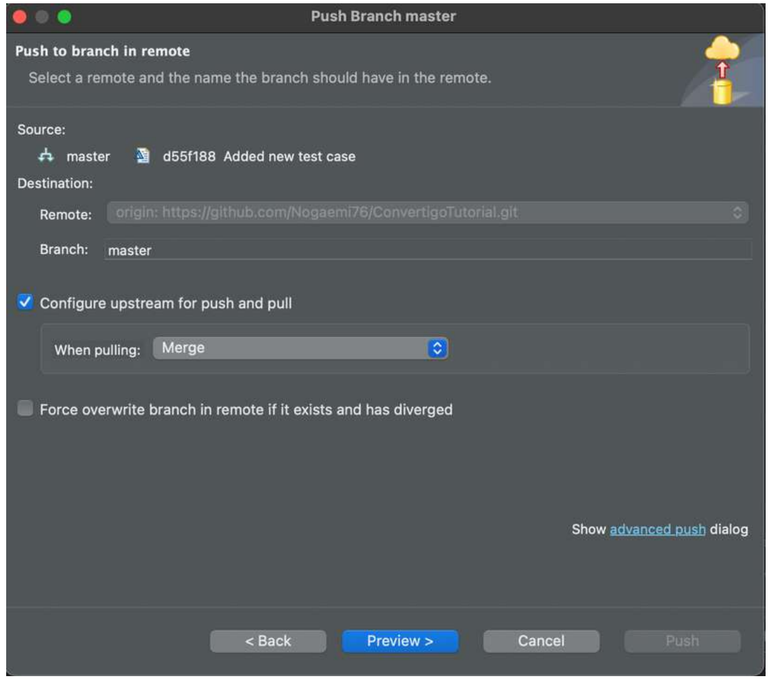
You can change the remote branch if necessary.

Click on **Preview >**.



The **Push to branch in remote window** appears.

Click on **Preview >**

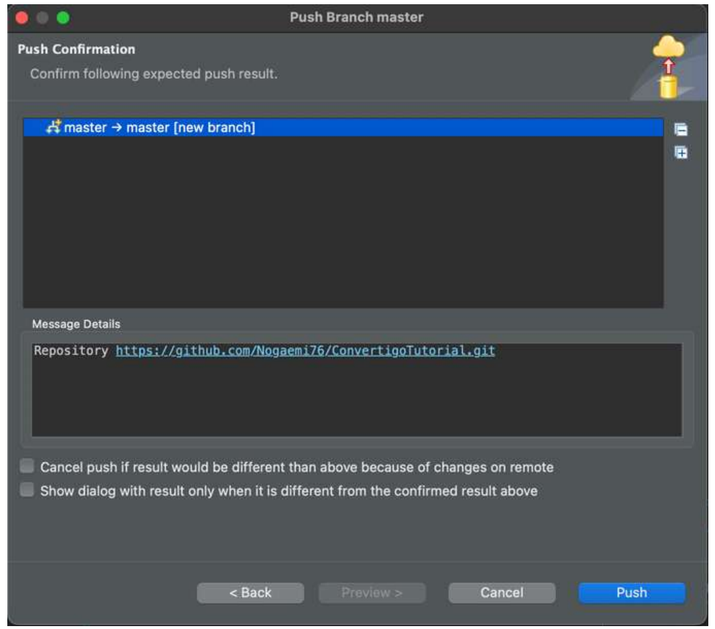


7.6 Commit your changes

The **Push Confirmation window** appears.

Click on **Push**

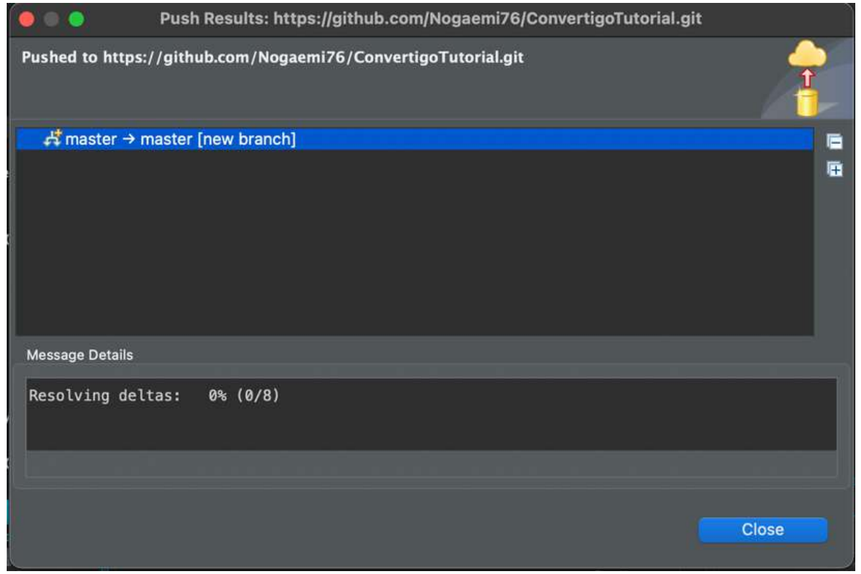
to **push your project on your remote repository**.



A **Push Results window** appears

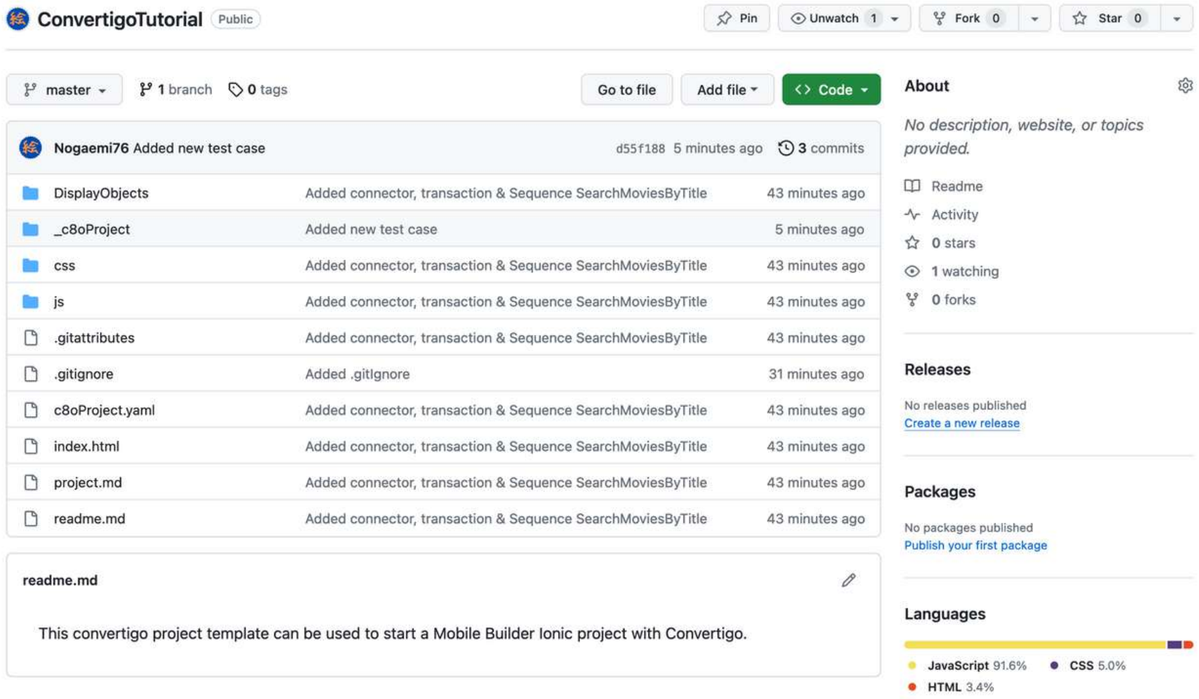
to confirm that your project has been pushed

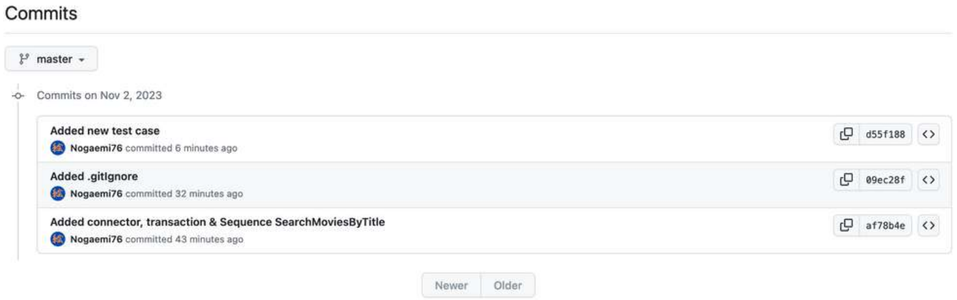
on your remote repository.



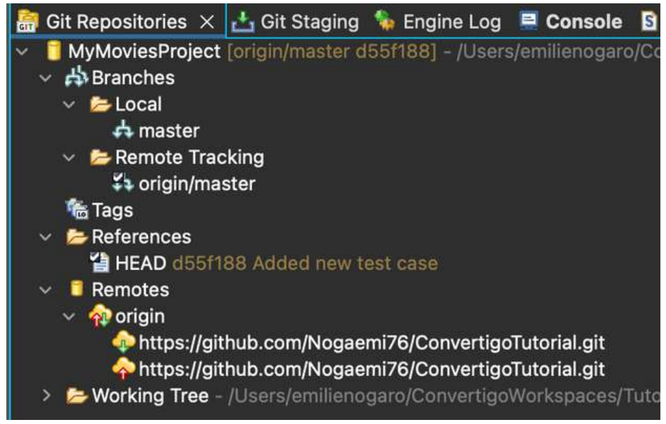
7.6 Commit your changes

Your project appears in your **remote repository**.





The **Remote branch** appears in the **Git Repository view**.



7.7 Clone a project

Let’s say you want to clone a project in your studio.

For example, you want to use the **library lib_UserManager developed by Convertigo**.

It is used to **include user management and authentication** in a **Convertigo project**.

lib_UserManager

User management and Authentication for your projects

The lib_UserManager enables your projects to include user management and authentication in your apps. This library will handle :

- user login with user/password using a salted password security
- user login using OpenID (Google, Azure & linkedin)

When using user/password, the library will use the **lib_usermanager_fullsync** database to store userids and salted/hashed password

You can find the repository in GitHub :

https://github.com/convertigo/c8oprj-lib-user-manager

7.7 Clone a project

As explained in the ReadMe of lib_UserManager in GitHub,

the **simplest way** to clone a project is

- **NOT** by using the Git Repositories view (more complex eclipse-based process).
- by **using the Convertigo project import Wizard** in the **Project view** (customized process developed by Convertigo).

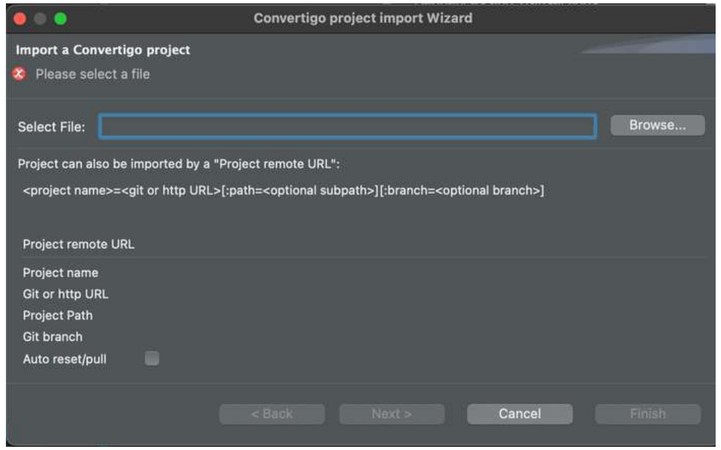
Copy the project url from the ReadMe of the repo in GitHub :

lib_UserManager=https://github.com/convertigo/c8oprj-lib-user-manager/archive/8.0.X.zip

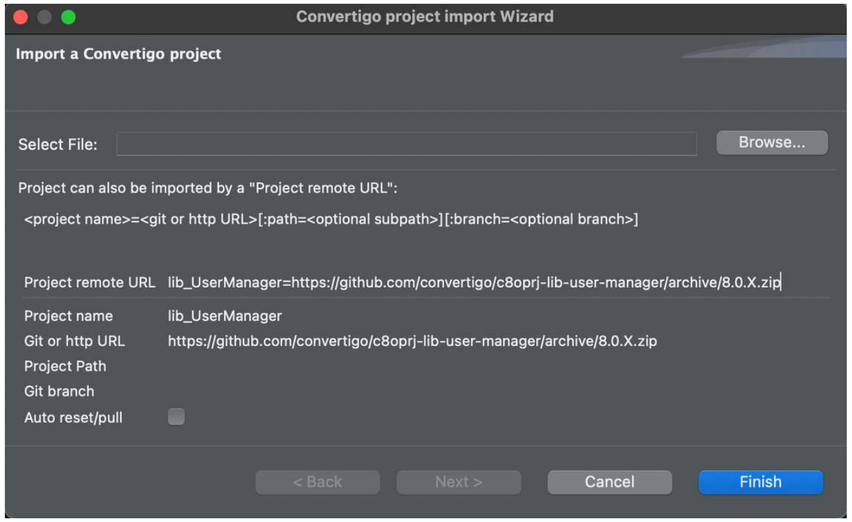
Click on the **Import a project in treeview** button to open the **Convertigo project import Wizard**.

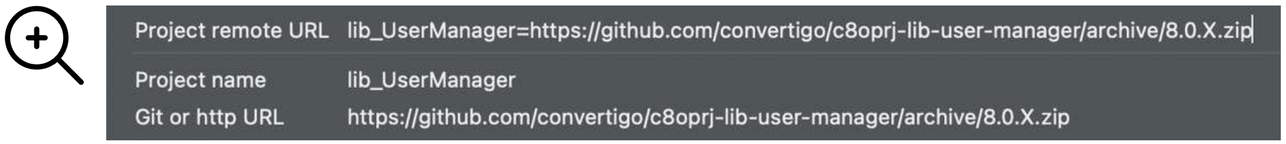
7.7 Clone a project

The **Convertigo project import Wizard** opens.

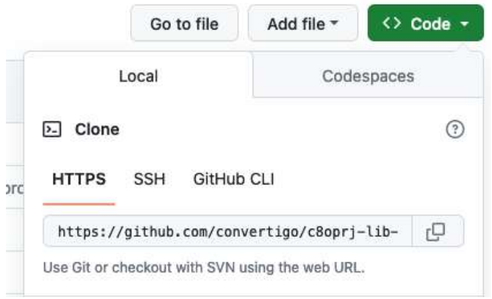


Paste the **project url in the Project remote URL field** and click on **Finish**.





7.7 Clone a project



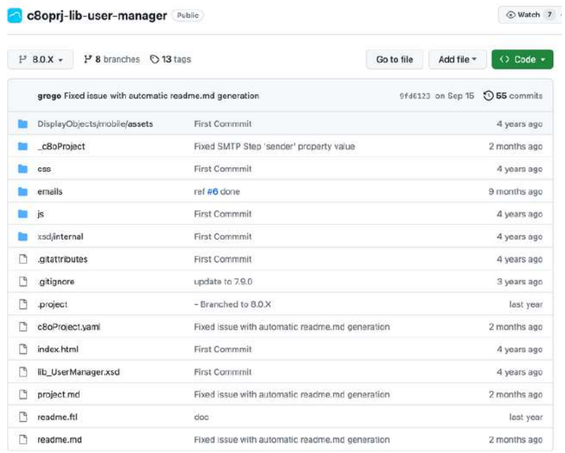
Important :

Usually, when **cloning a GitHub repo**, you copy it from the usual repo url and the project name is not already present in the url.

In that case, you have to **include it manually in the project name field**.

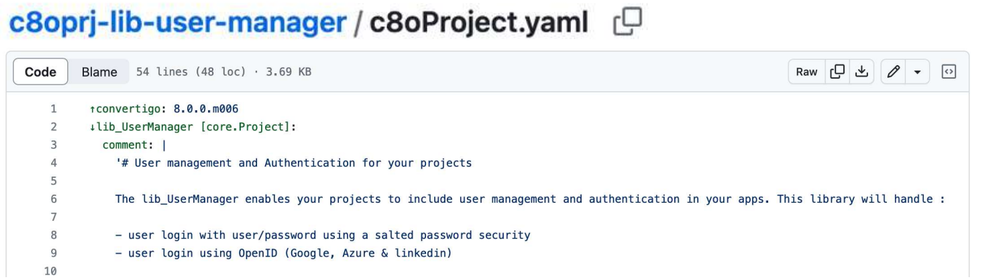
To find the **project name of a Convertigo library**,

go to the **c8oProject.yaml file** in GitHub.



The project name (lib_UserManager)

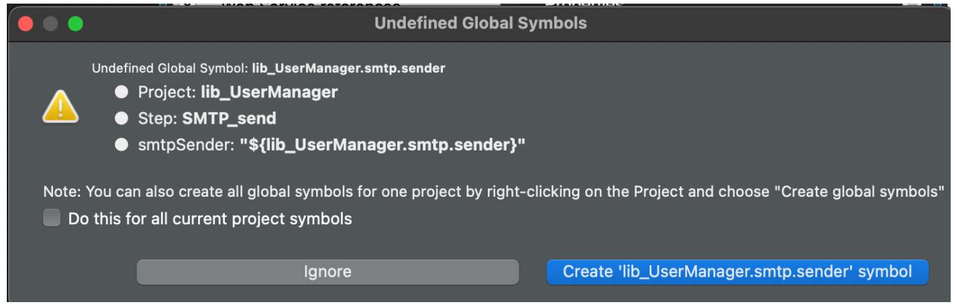
is indicated at the very beginning of the file.



7.7 Clone a project

If the cloned project has symbols,

the **Undefined Global Symbol window** appears.

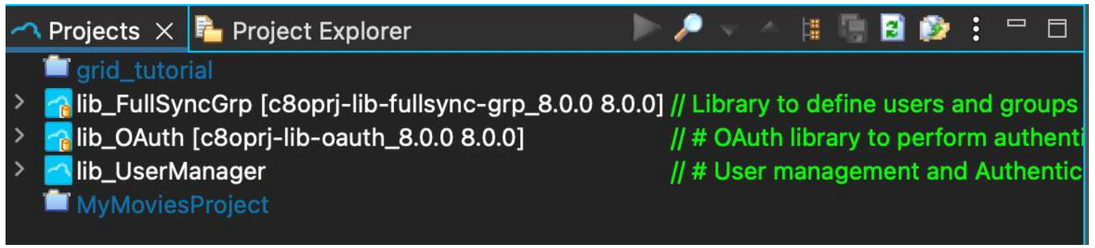


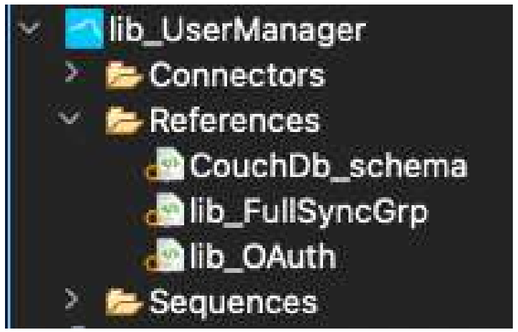
Select **Do this for all current symbols**.

Click on Create ‘XXX’ symbol

(‘XXX’ depends on the symbol name).

The **library is imported** and appears in the **Projects view**.





Important :

The library lib_UserManager

uses other libraries

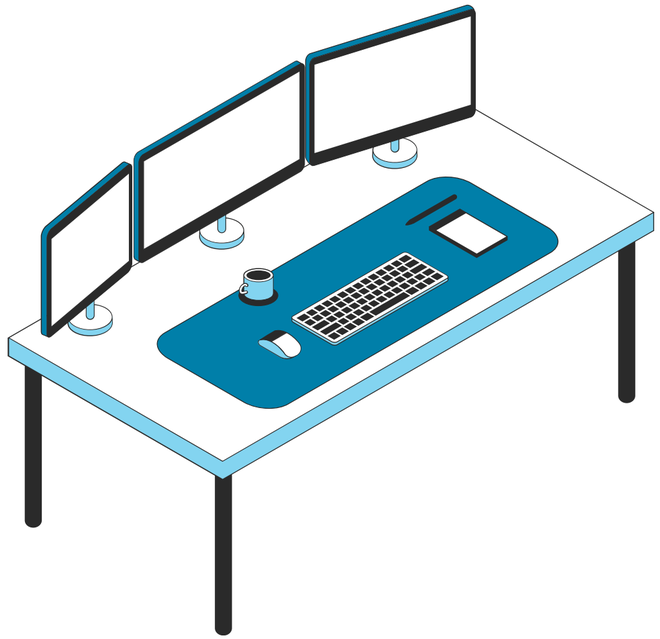
(as shown in References folder)

and **they were imported** as well.

convertiGO

8 – Test platform

How to test your backend.



8.1 Access the Test platform

8.2 Test a transaction

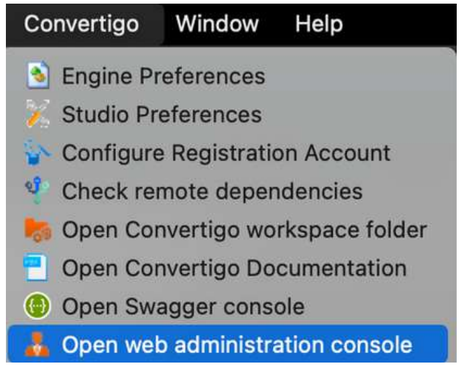
8.3 Test a sequence

8.1 Access the Test platform

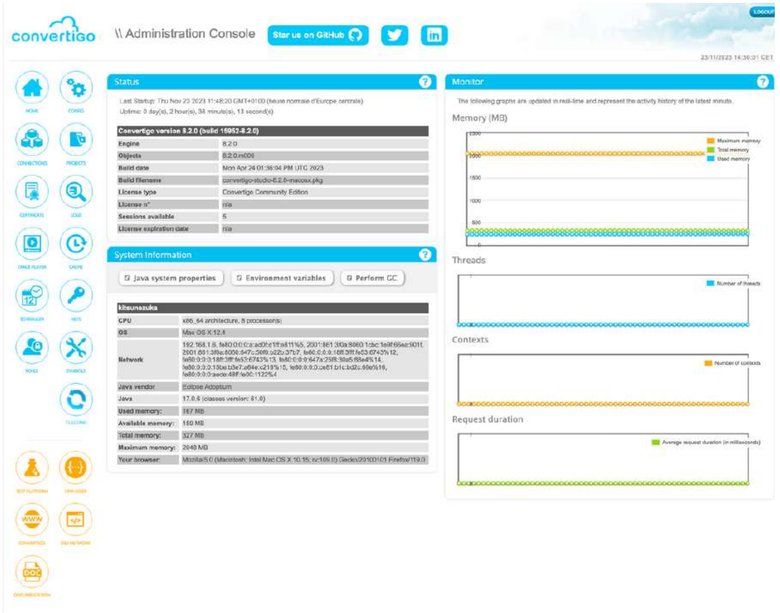
Convertigo provides a Test platform to test your backend and your frontend.

To access the Test platform

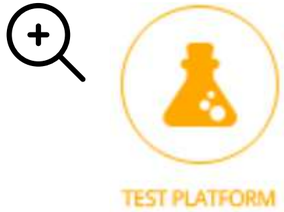
Open the **web administration console**.



In the web administration console



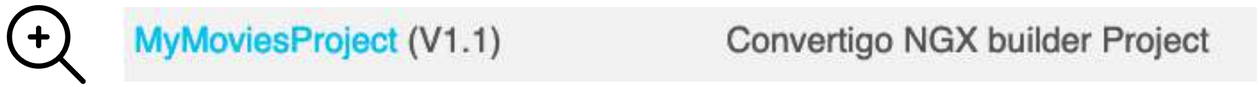
Click on the **Test platform icon**.



8.1 Access the Test platform

In the Test platform are displayed **all the projects of your workspace**.

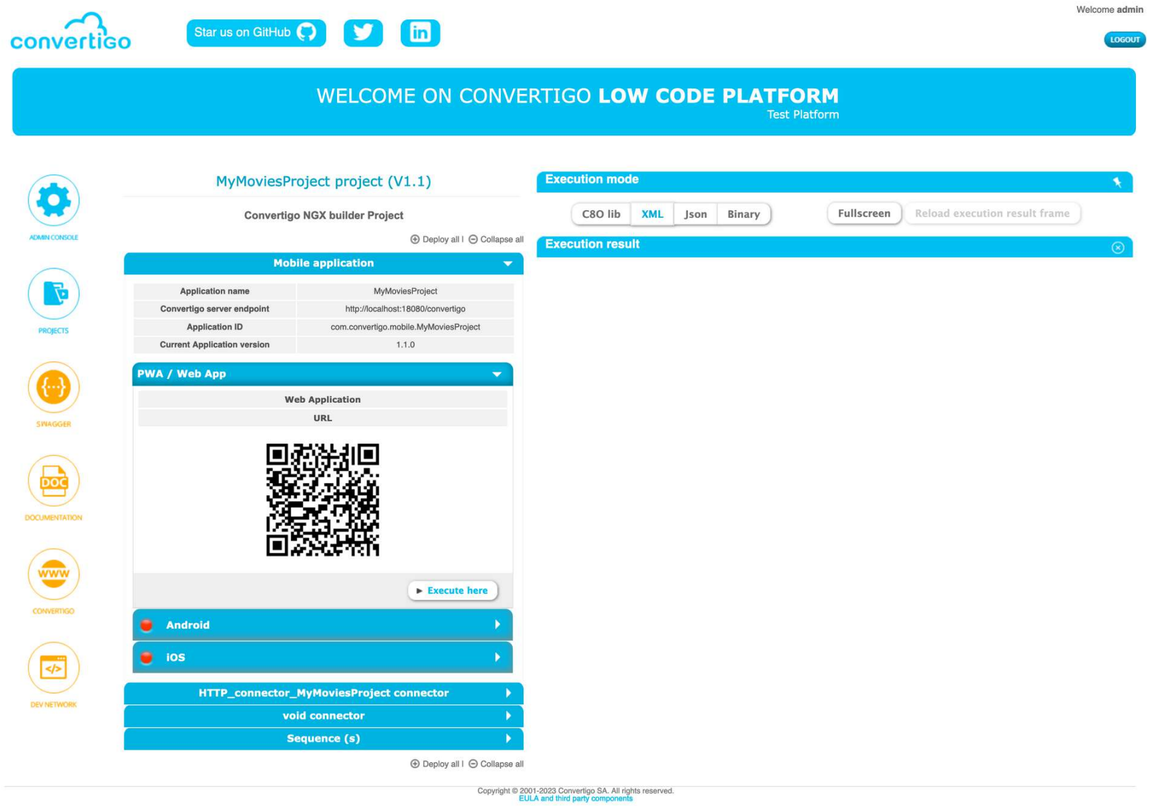
Click on MyMoviesProject to select it.

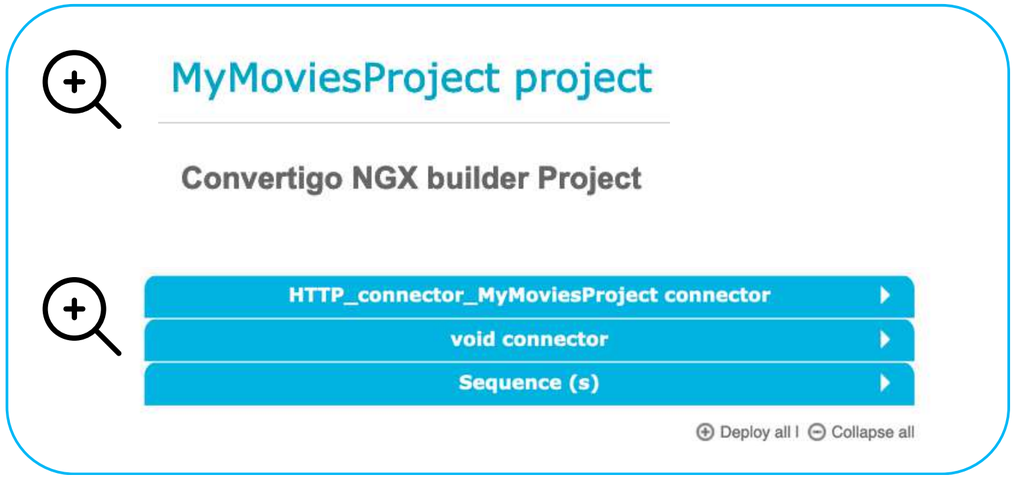


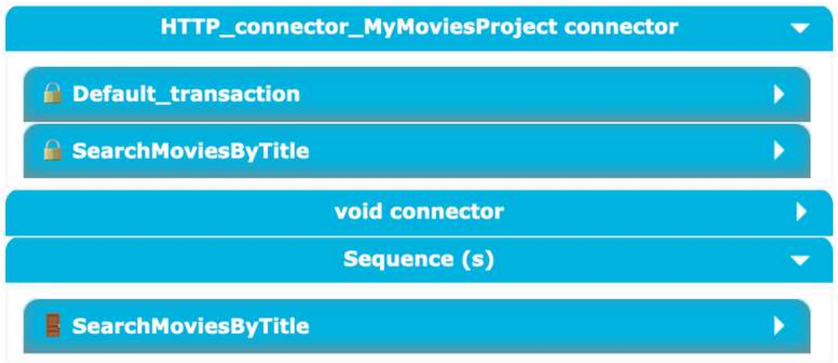
8.1 Access the Test platform

In the MyMoviesProject page of the Test platform,

we can see **all the transactions and sequences** of the project.



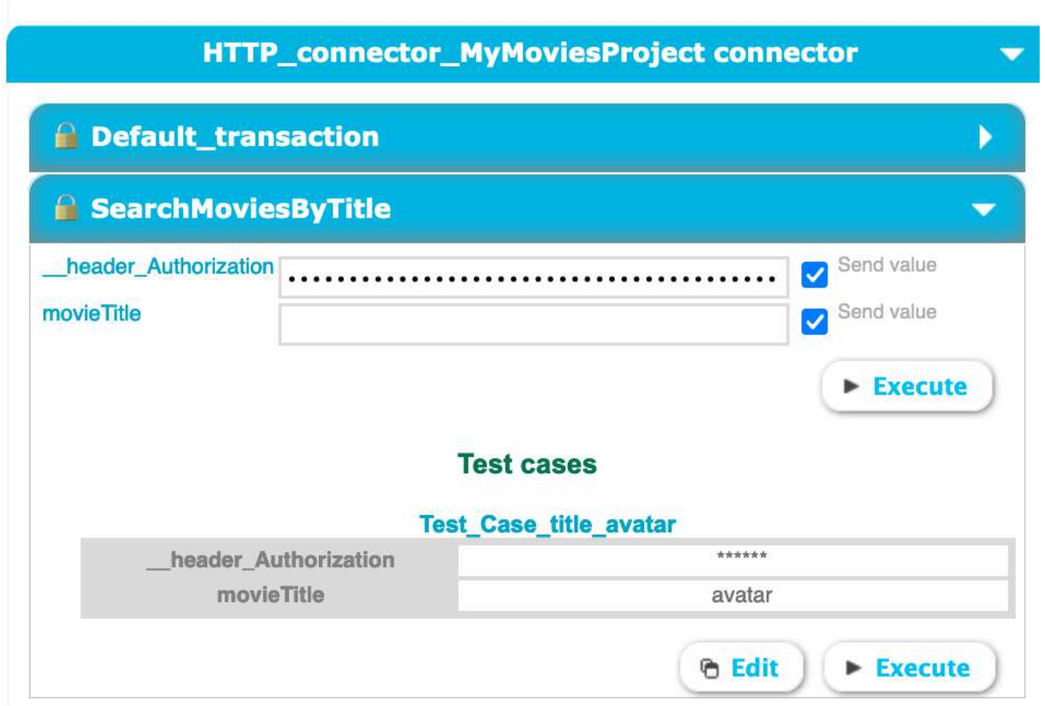




8.2 Test a transaction

Let's test our SearchMoviesByTitle transaction.

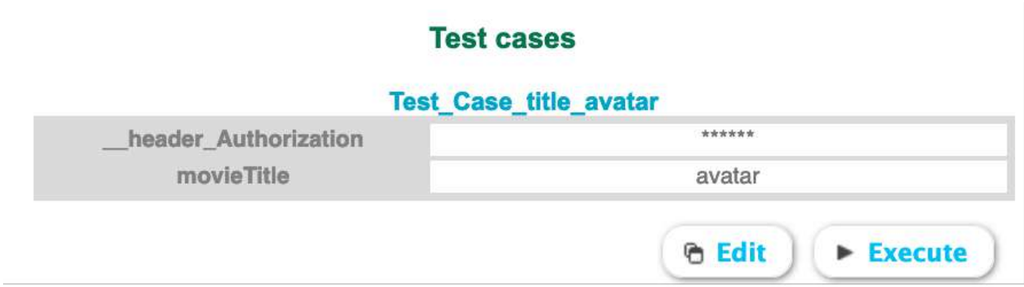
When we deploy the transaction tab, we can see 2 parts:



- an **editor with our transaction variables**

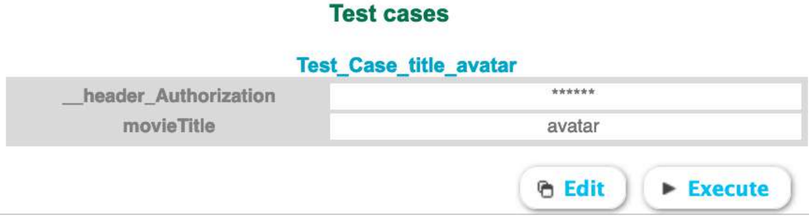
  where we can enter a movieTitle variable.

- the **test case we created in our transaction**.



8.2 Test a transaction

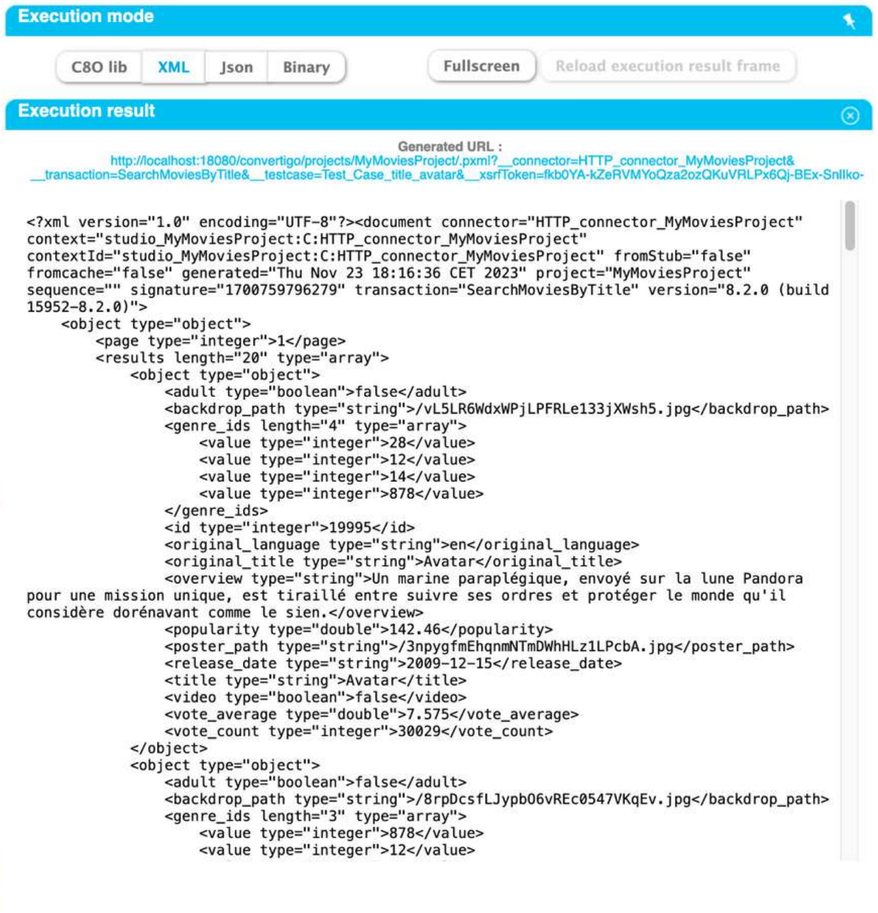
Let's try the **test case** we created in our project with "avatar" as value for the movieTitle variable.



The result will be **displayed in XML** by default.

Click on **Execute** to run the test case.

The result is **displayed in XML**.



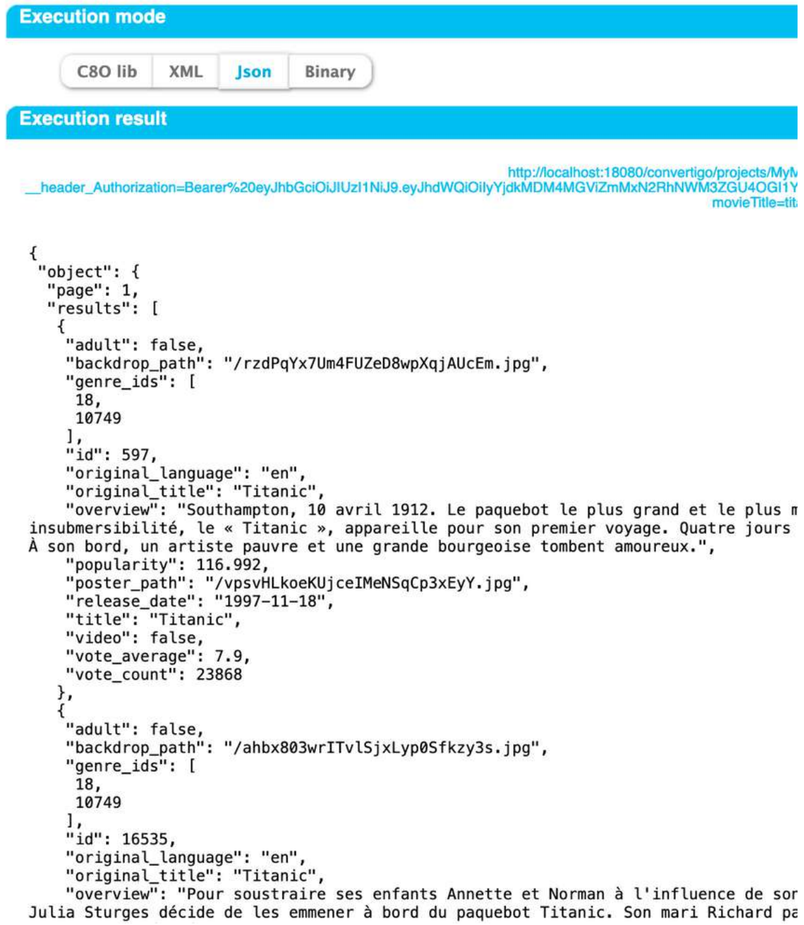
8.2 Test a transaction

Let’s try the editor with our transaction variables with “titanic” as value for the movieTitle variable.

Let’s change the **Execution mode to Json.**

Click on **Execute** to run the test case.

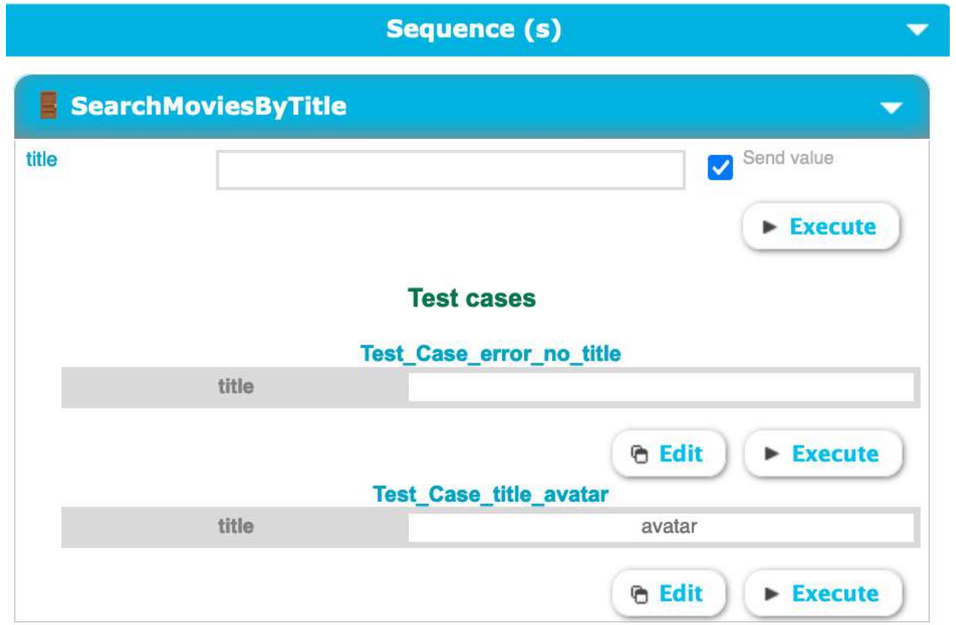
The result is **displayed in JSON**.



8.3 Test a sequence

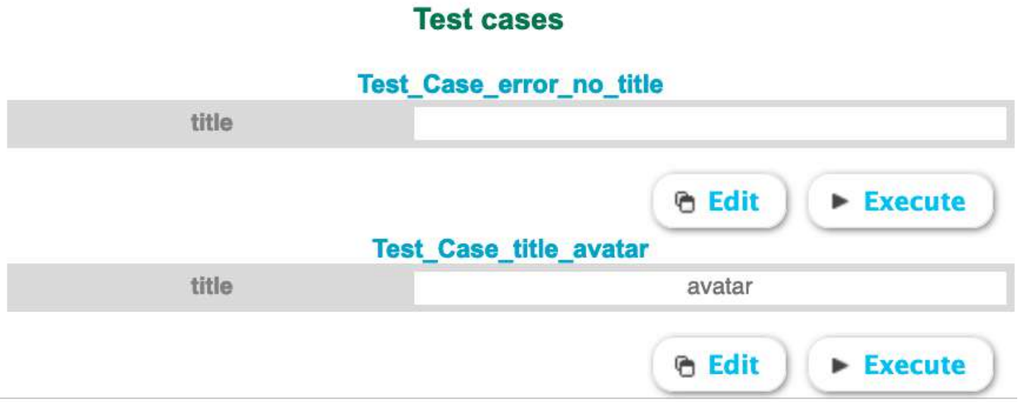
Now, let's test our SearchMoviesByTitle sequence.

Here again, we can see 2 parts:



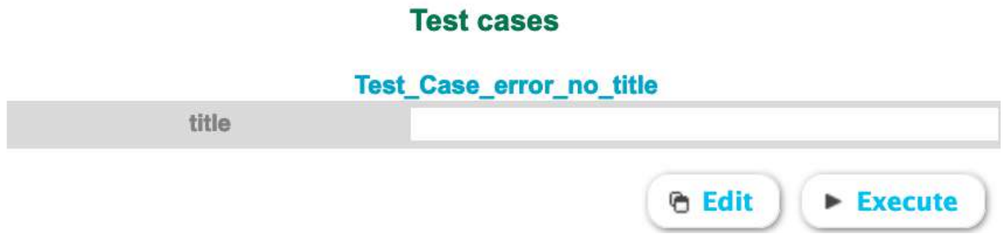
- an editor with our sequence variable where we can enter a title variable.

- the test cases we created in our transaction.



8.3 Test a sequence

Let’s try the error test case we created in our project with no value for the title variable.



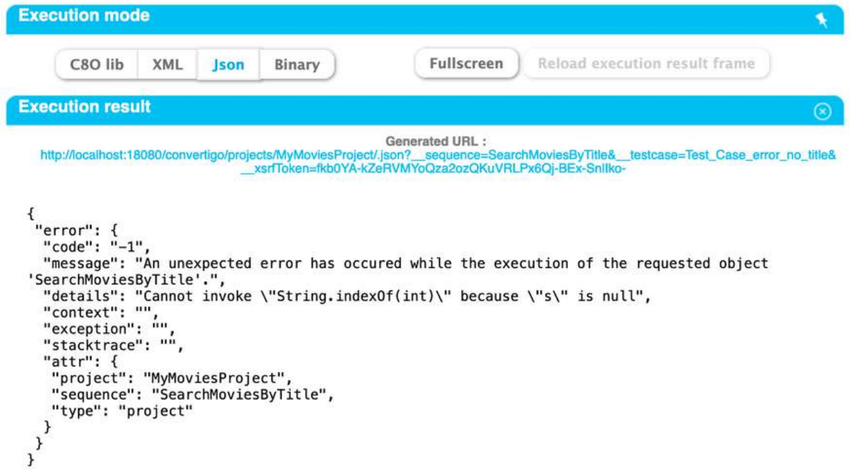
The **Execution mode** is in **XML**.

Click on **Execute** to run the test case.

The result is **displayed in XML**.

Let’s change the **Execution mode** to **Json**, and execute the test again.

The result is **displayed in JSON**.



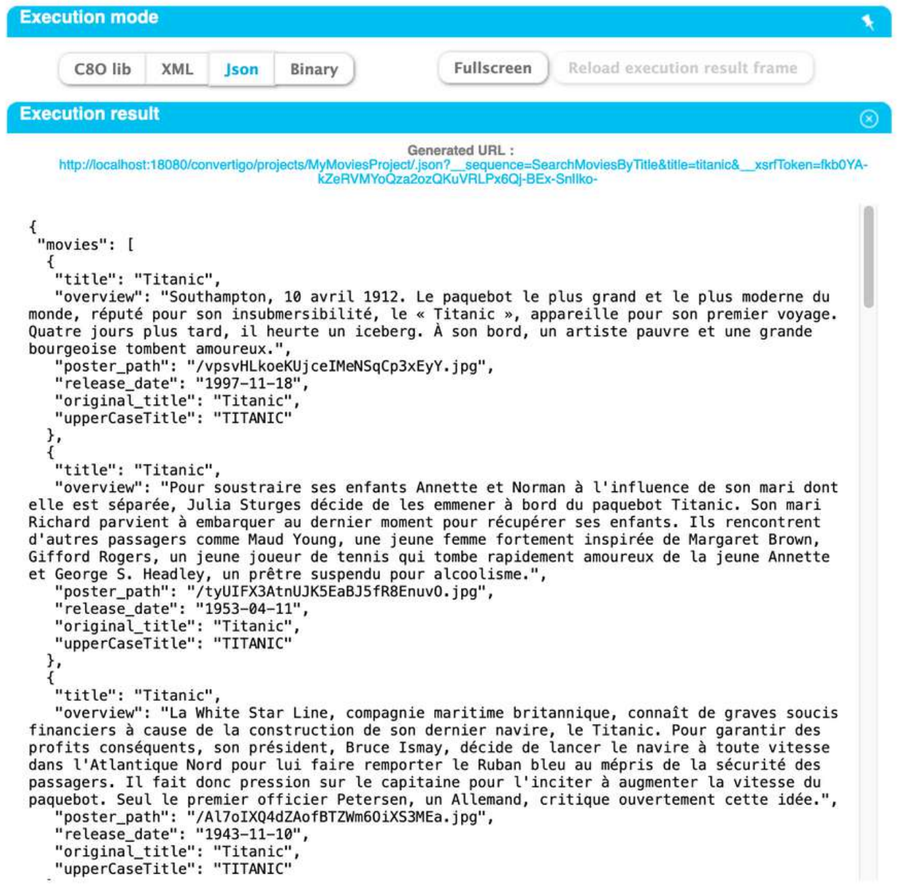
8.3 Test a sequence

Let’s try the editor with our sequence variable with “titanic” as value for the title variable.

The **Execution mode is in Json**.

Click on **Execute** to run the test case.

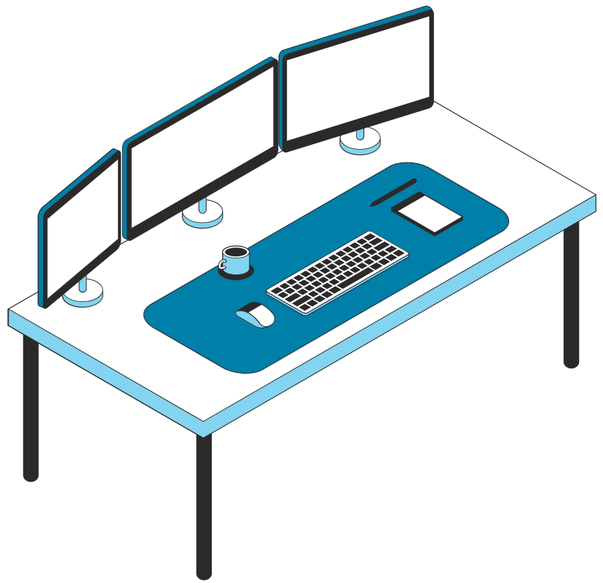
The result is **displayed in JSON**.



convertigo

9 – URL mapper

How to expose an API REST.



9.1 What is the URL mapper ?

9.2 URL mapper steps

9.3 Create an URL mapper for a transaction

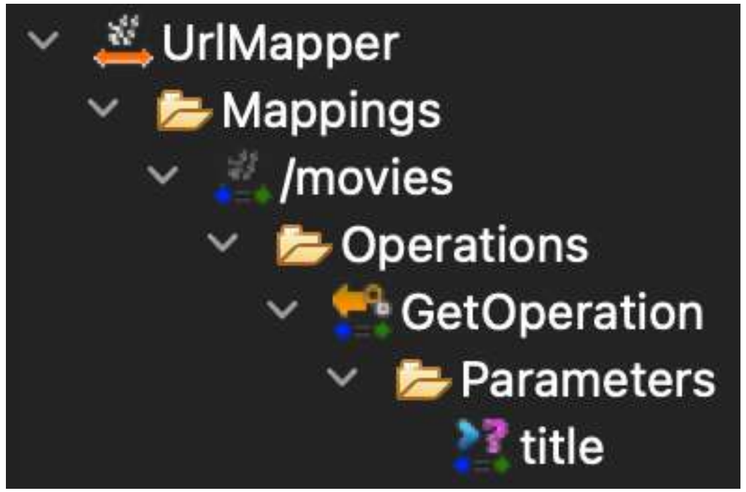
9.4 Test the URL mapper on Swagger

9.1 What is the URL mapper ?

The **URL mapper** is able to map **RESTful urls** to **Convertigo requestables** such as **Sequences and Transactions**.
This way Convertigo **can expose RESTful APIs** to the outside world.

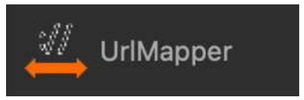
You can have **only one URLMapper per project**,
but an URLmapper **can map URLs to any otherproject deployed on the server**.

Example of URL mapper structure in a Convertigo project



9.2 URL mapper steps

Convertigo provides steps to create the URL mapper.



UrlMapper

This step defines the **URL mapper** to use in the project.

PathMapping – Mapping step

This step defines a **mapping path associated with the mapper**, the **base URL structure** an API user will have to use **to access this API Service**.
For example: /accounts/{accountid}.

9.2 URL mapper Objects

Operations Steps

These steps define the **HTTP operations associated with the mapping**.

For a **given operation on a given mapping**,
you define here **what should be the Requestable (Sequence or Transaction) to be executed**,
and **how will the variables for this requestable will be mapped**.

=> HTTP **GET** operation

=> HTTP **POST**operation

=> HTTP **PUT** operation

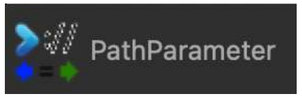
=> HTTP **DELETE** operation

=> HTTP **HEAD** operation

9.2 URL mapper Objects

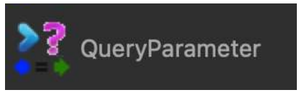
Parameters Steps

Convertigo provides steps to define parameters associated with the operation.



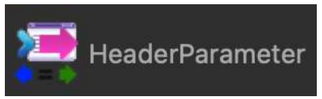
PathParameter – Parameters step

This step defines a path parameter by extracting the variable value from a segment of the URL path between {}. ex: /accounts/{accountid}



QueryParameter – Parameters step

This step defines a query parameter by extracting the variable value from the query string.
ex: /accounts?verbose=1



HeaderParameter – Parameters step

This step defines a header parameter by extracting the variable value from the HTTP Header of this parameter name.

9.2 URL mapper Objects

Responses Step

OperationResponse

OperationResponse – Responses step

This step defines an **HTTP response associated with the operation**.

When a service is invoked, it **responds with a HTTP status code**.

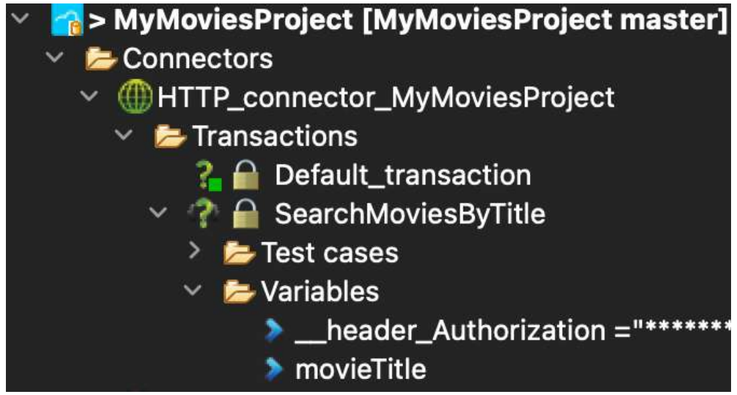
This mapping object will help you **define status codes** such as 200, 401 or any other **according to XPaths resolution** done on a **Convertigo Sequence response**.

The Sequence response will be **scanned by all the UrlMappingResponse objects** defined for a given operation. The **first one having its XPath matching** will **generate the corresponding status code**.

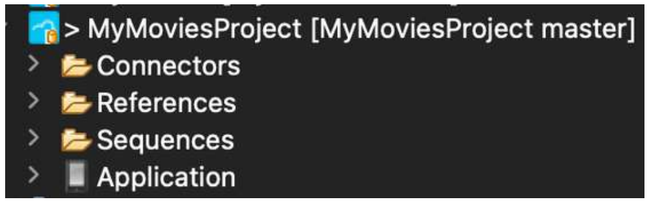
9.3 Create an URL mapper for a transaction

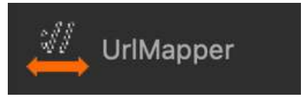
In our project, we have a **SearchMoviesByTitle transaction**, with a **variable named movieTitle**.

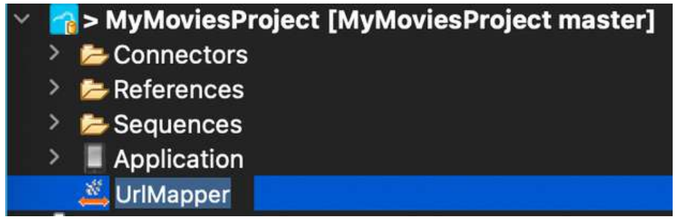
Let's create an URL mapper for this transaction.



Drag and drop the **UrlMapper step** from the palette in the project.

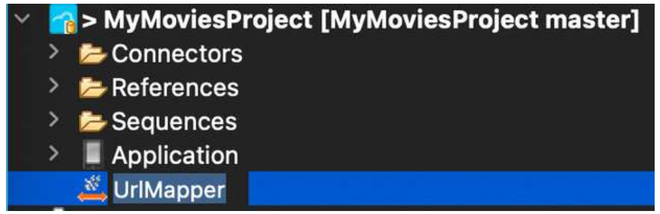






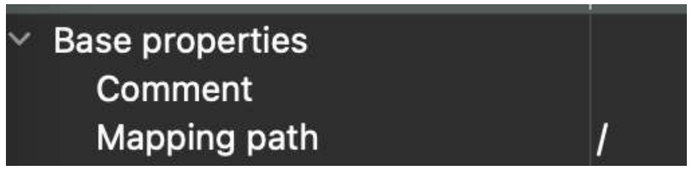
9.3 Create an URL mapper for a transaction.

Drag and drop a **PathMapping step** from the palette in the **UrlMapper step**.



In the properties,

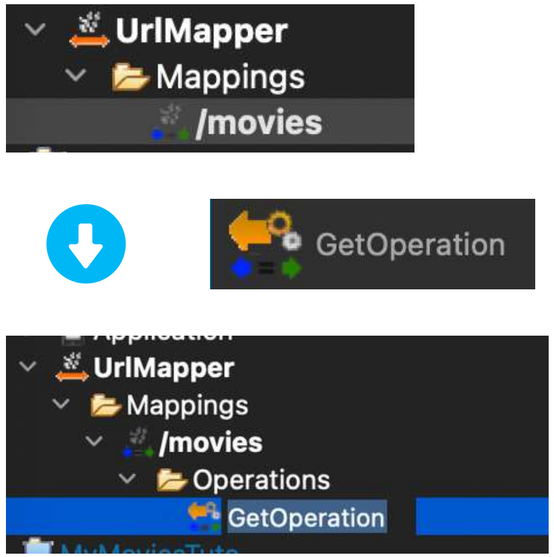
rename the Mapping path as /movies.



The path will appear as **/movies in the url.**

9.3 Create an URL mapper for a transaction.

Drag and drop a **GetOperation step** from the palette in the **PathMapping /movies step**.

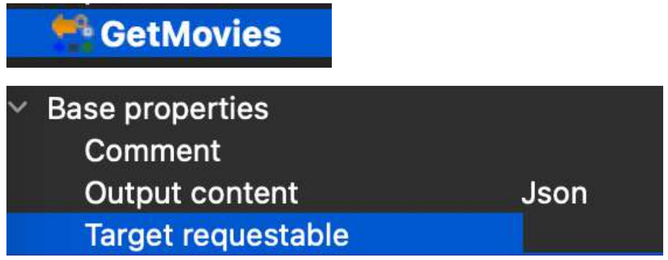


Rename the GetOperation step as **GetMovies**.

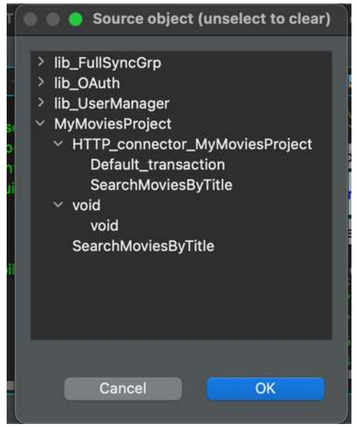
9.3 Create an URL mapper for a transaction.

Now, let's select which transaction or sequence we are going to map.

In the properties of GetMovies

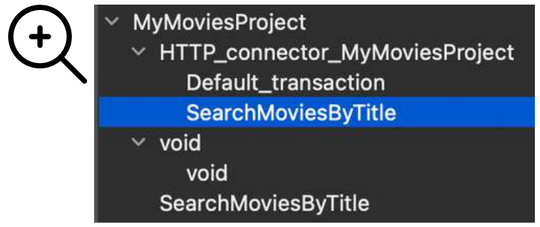


At the end of the line
of the **Target requestable property**,
click on this icon.

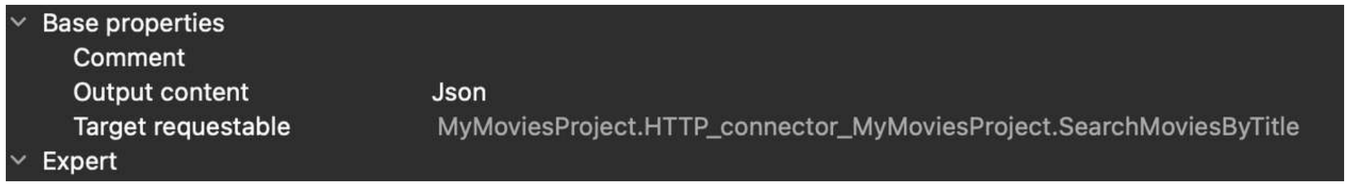


The **Source object window** appears.

Select the **SearchMoviesByTitle transaction**.

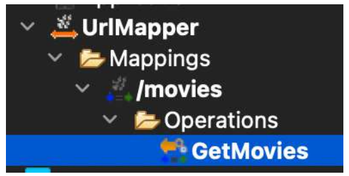


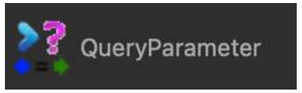
The SearchMoviesByTitle transaction appears as value
in the Target requestable property of GetMovies.

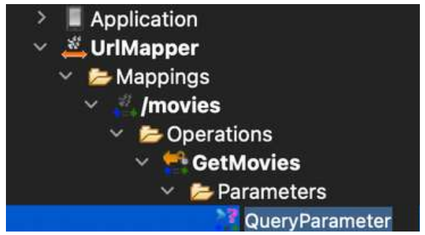


9.3 Create an URL mapper for a transaction.

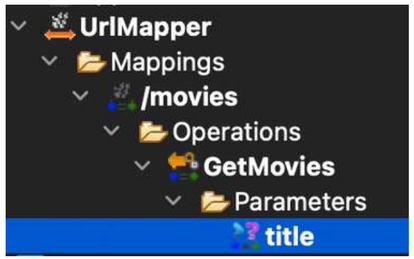
Drag and drop a **QueryParameter step** from the palette in the **GetMovies step**.



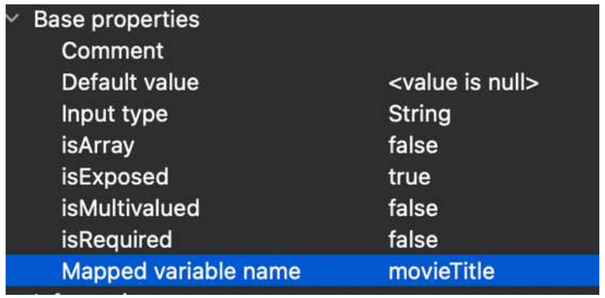




Rename the QueryParameter step as **title**.



In the **properties of the QueryParameter**, enter **movieTitle** (transaction variable name) as value of **Mapped variable name**.



9.4 Test the URL mapper on Swagger

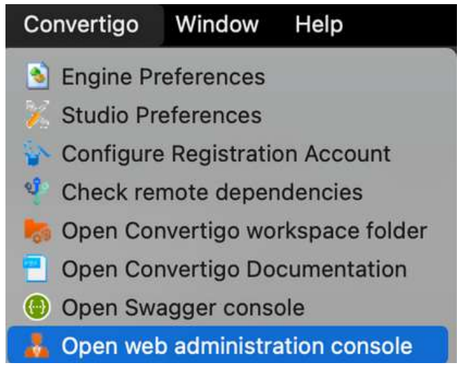
Now, let’s test our URL mapper on Swagger.

To open the Swagger console in your browser.

Click on

Open Swagger console.

Or open the **web administration console**.



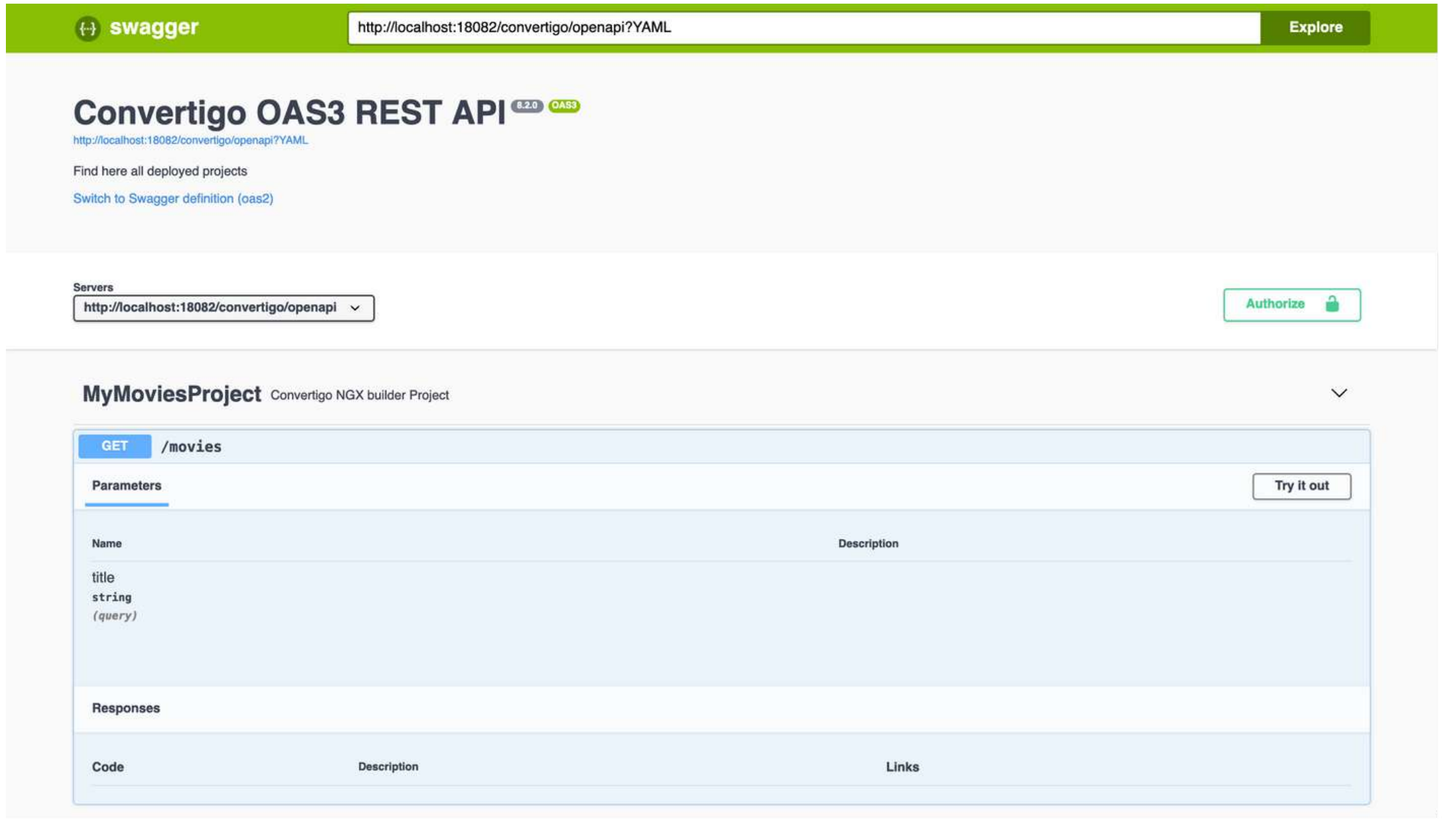
In the web administration console

Click on the **Swagger icon**.

9.4 Test the URL mapper on Swagger

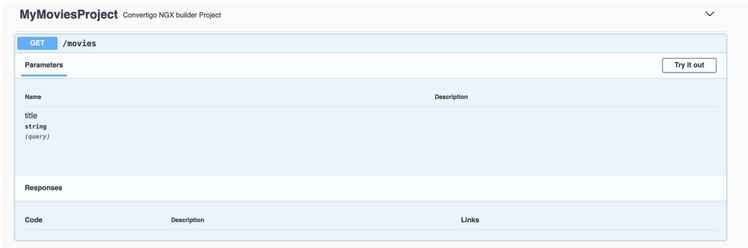
In the Swagger console of your browser,

we can see a GET /movies request with a title parameter.

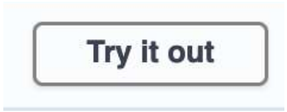


9.4 Test the URL mapper on Swagger

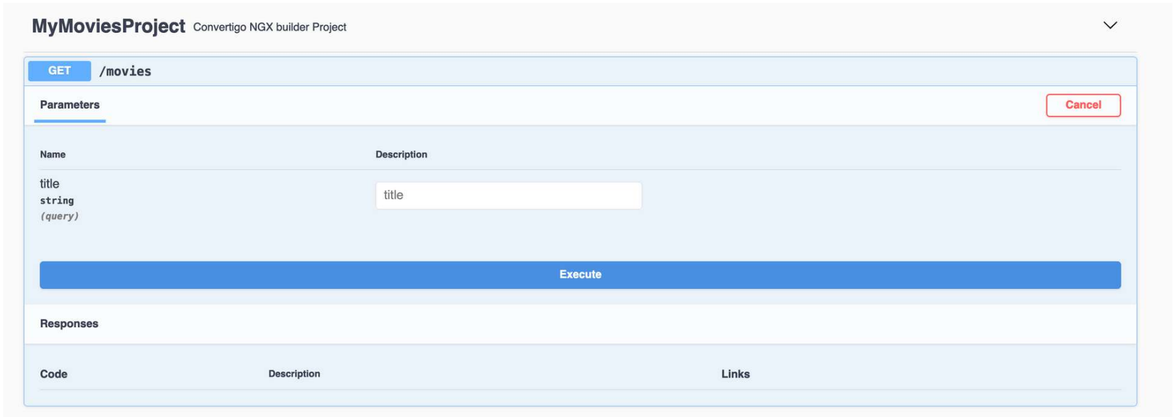
Let’s test the GET /movies request with **Try it out**.

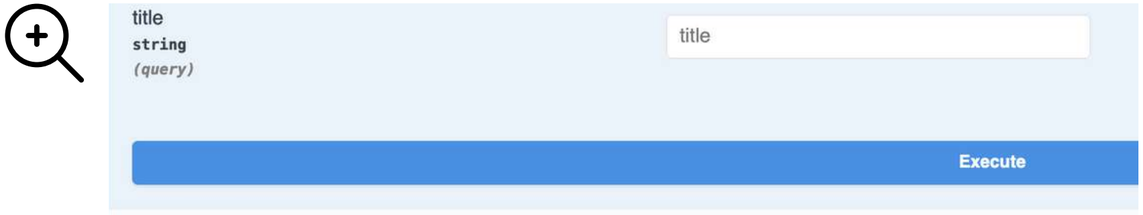


Click on the Try it out button.

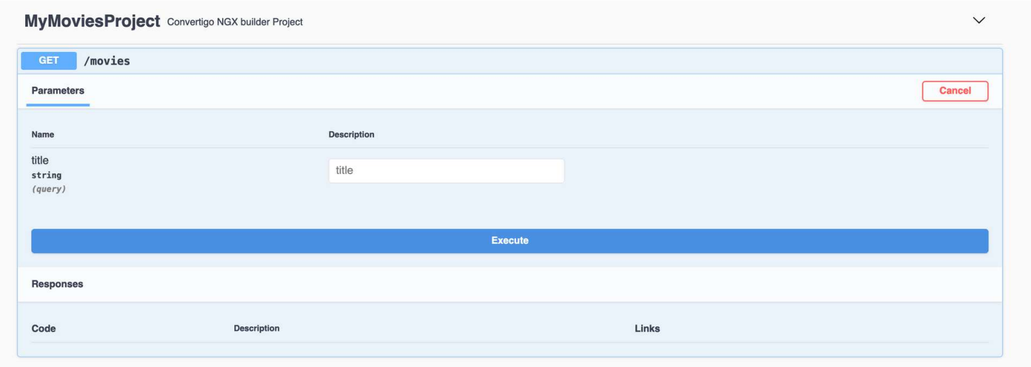


A **title field** and an **Execute button** appear.

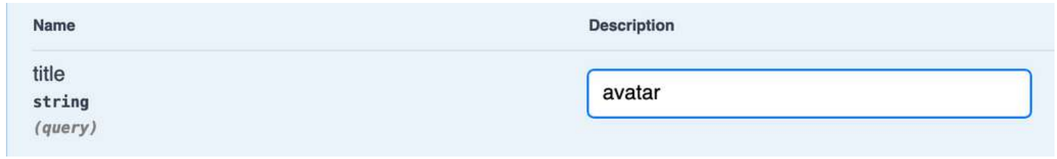




9.4 Test the URL mapper on Swagger



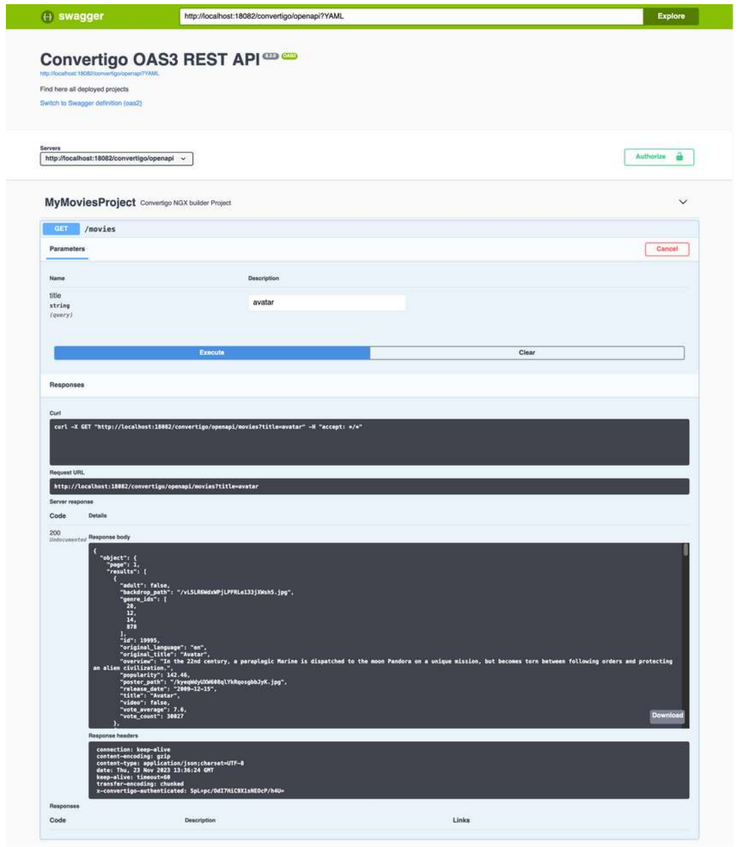
Enter a value in the title field (here “avatar”).

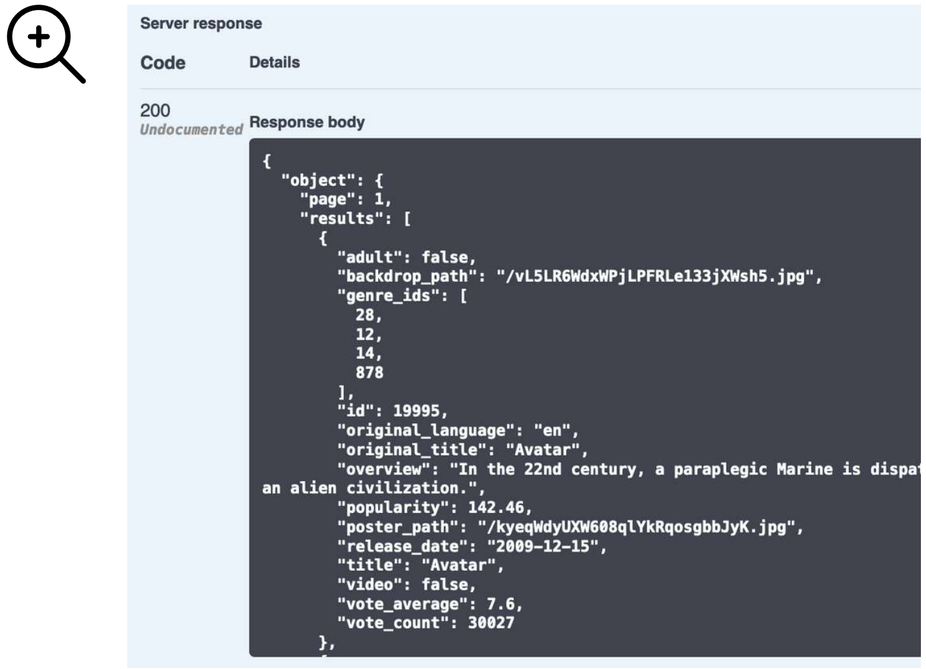


Click on Execute

9.4 Test the URL mapper on Swagger

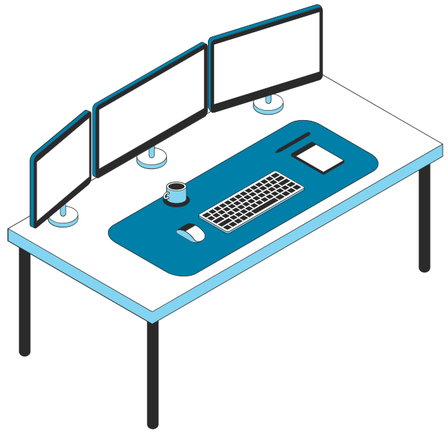
A response result of the GET /movies request appears in the Swagger





10 – Nocode database

How to use the NoCode Database.



10.1 Presenting Baserow

10.2 Set up your Baserow account

10.3 Create a database

10.4 Import CRUD sequences into a project

10.5 Add filters in a table

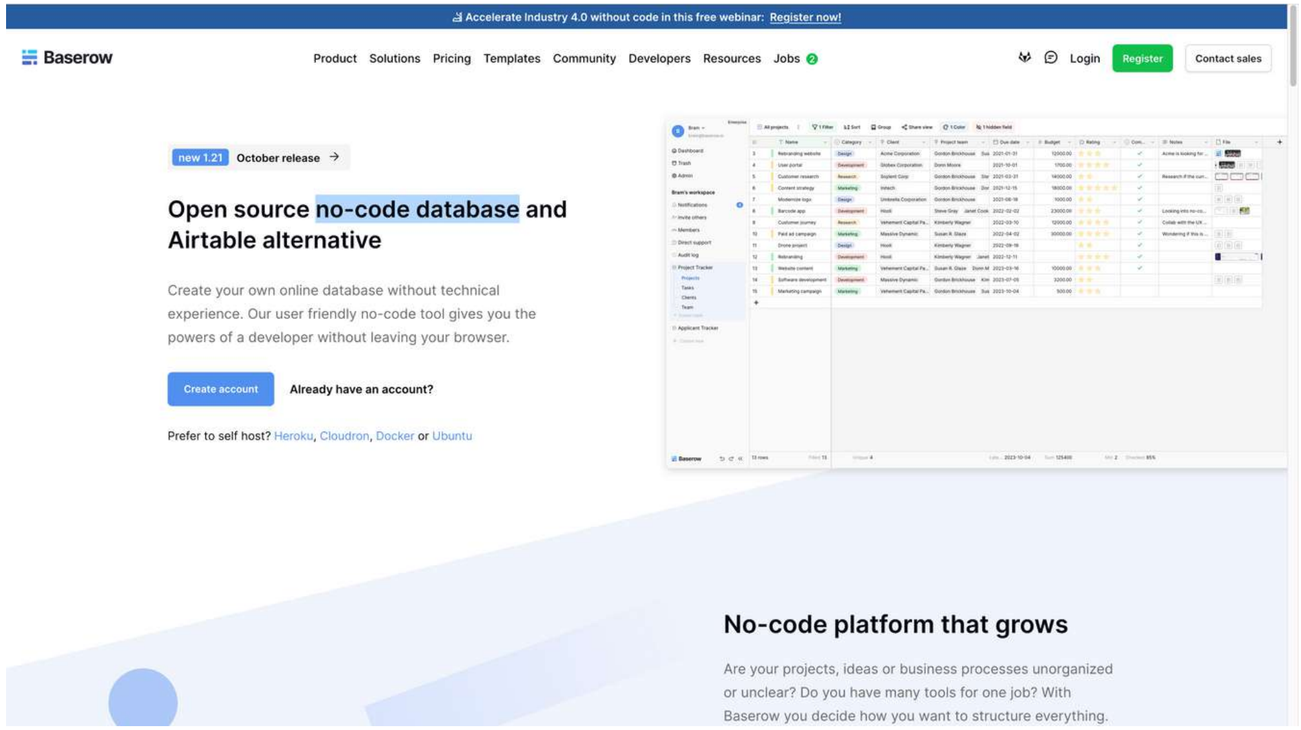
10.6 Test the CRUD sequences

10.1 Presenting Baserow

Convertigo Low code studio **integrates Baserow as no-code database**.

Baserow is an **open-source no-code database**
that **allows users**
to create databases and web applications
without the need for **coding**.

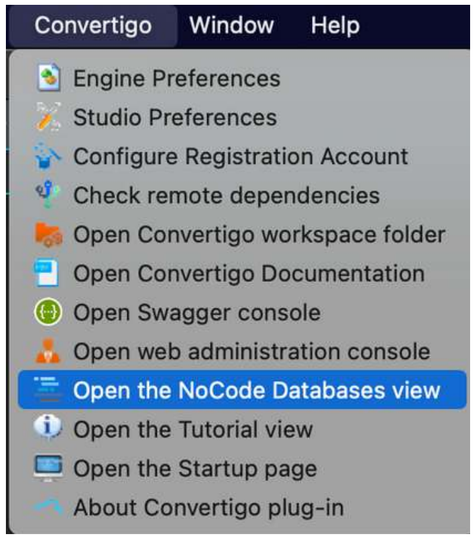
It provides a **user-friendly interface**
for designing databases,
setting up tables,
and views for data entry and visualization.



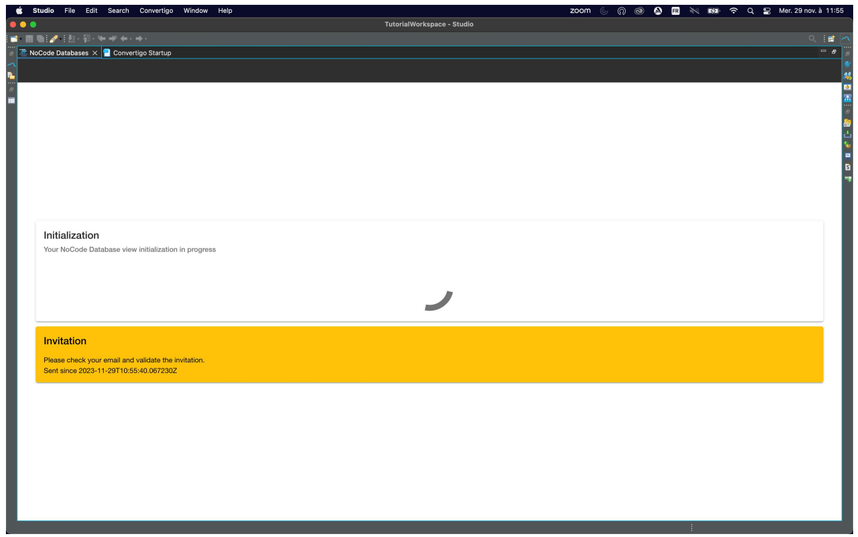
10.2 Set up your Baserow account

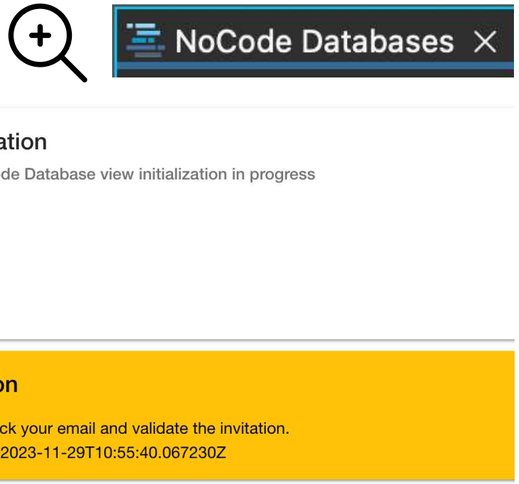
Let’s start by opening the **No Code Database view** in the studio
to set up your account.

Click on **Convertigo**,
then click on
Open the NoCode Databases view.



In the **NoCode Databases view**,
a message tells you to check your emails to create your profile
(The email you used to create your Studio account).



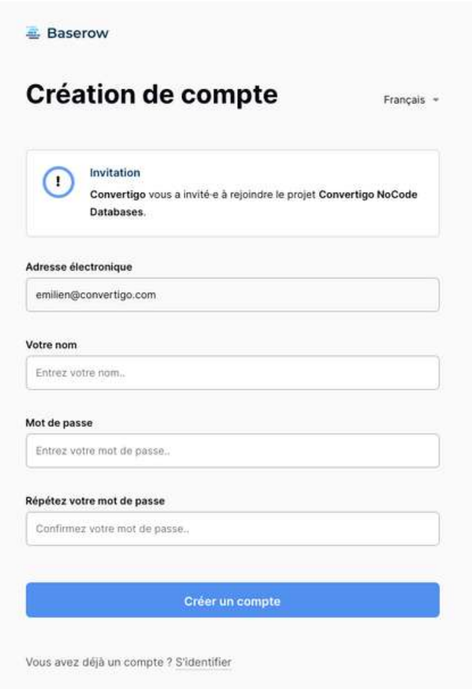


10.2 Set up your Baserow account

In this email, click on **accept invitation** to open the account creation page.

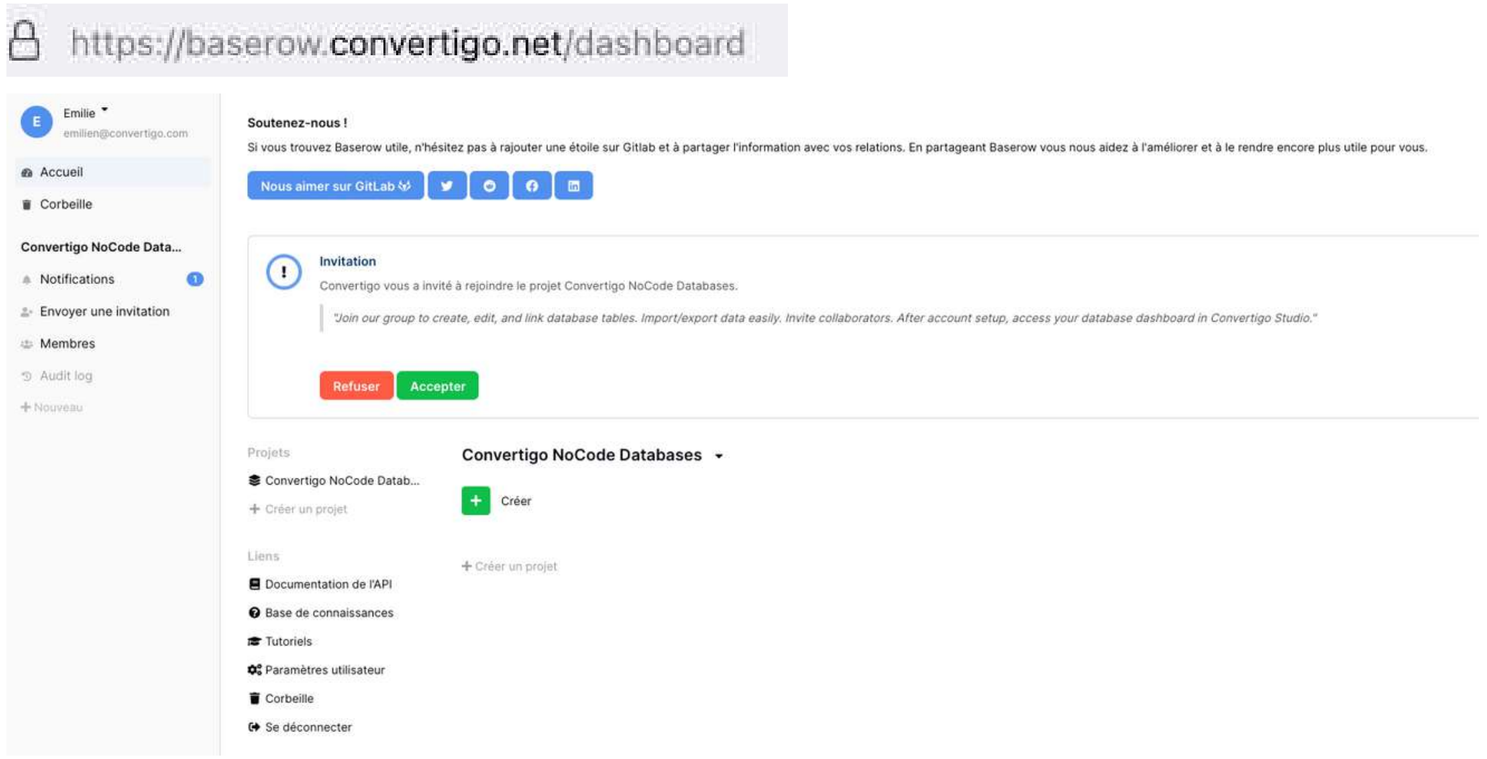
Fill in the information

and Sign up.



Once your account is created,

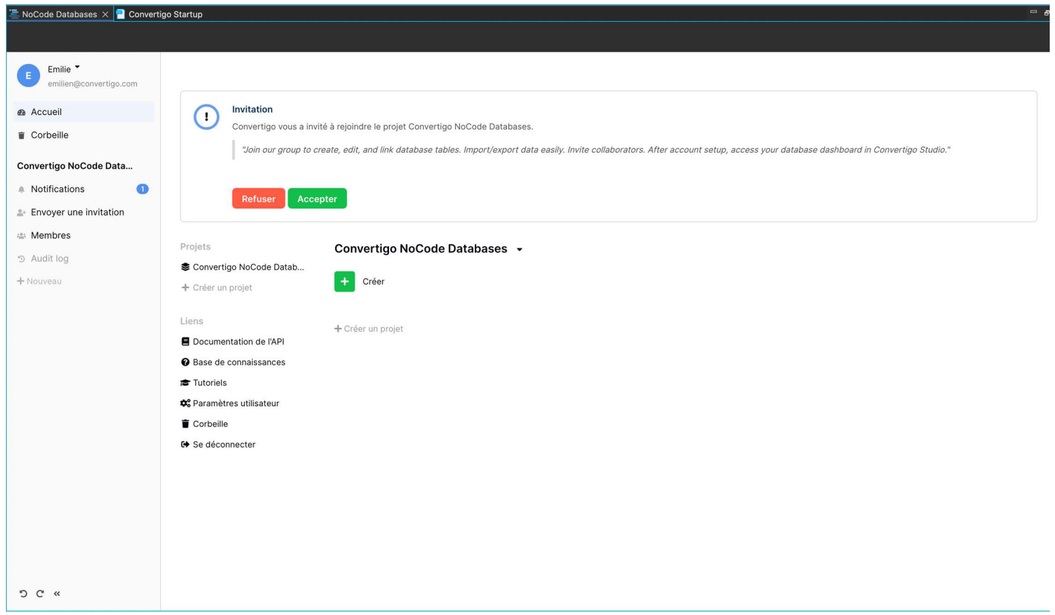
a tab with the **Convertigo baserow dashboard** opens in your browser.

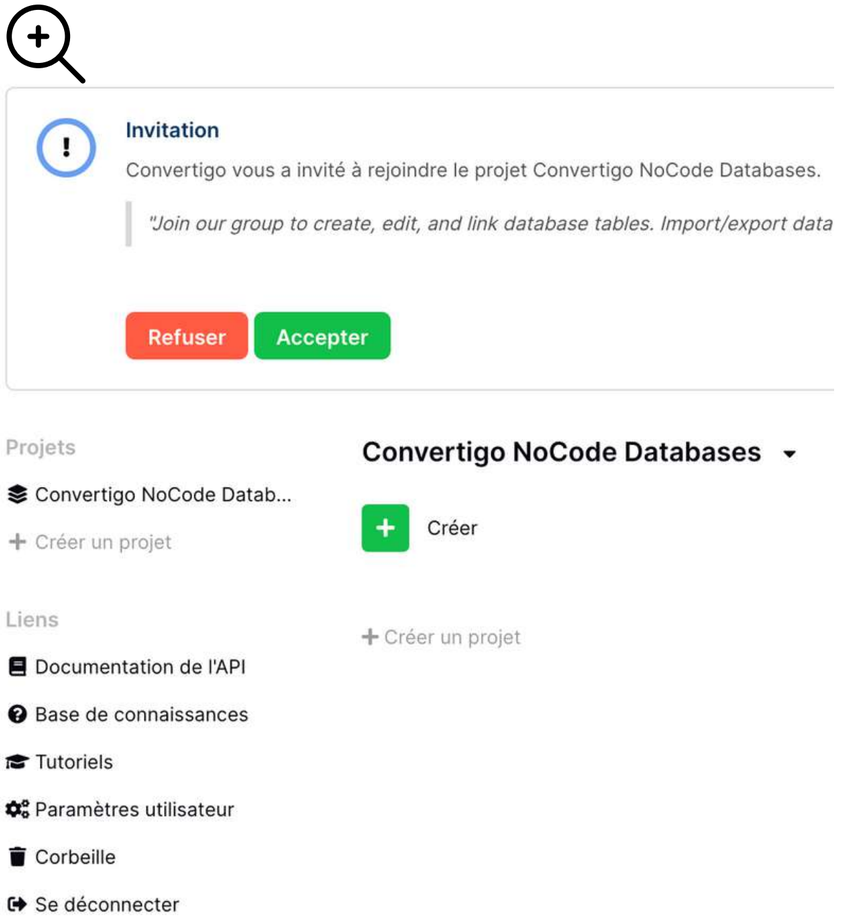


10.2 Set up your Baserow account

The same dashboard opens in the **NoCode Databases view** of the studio.

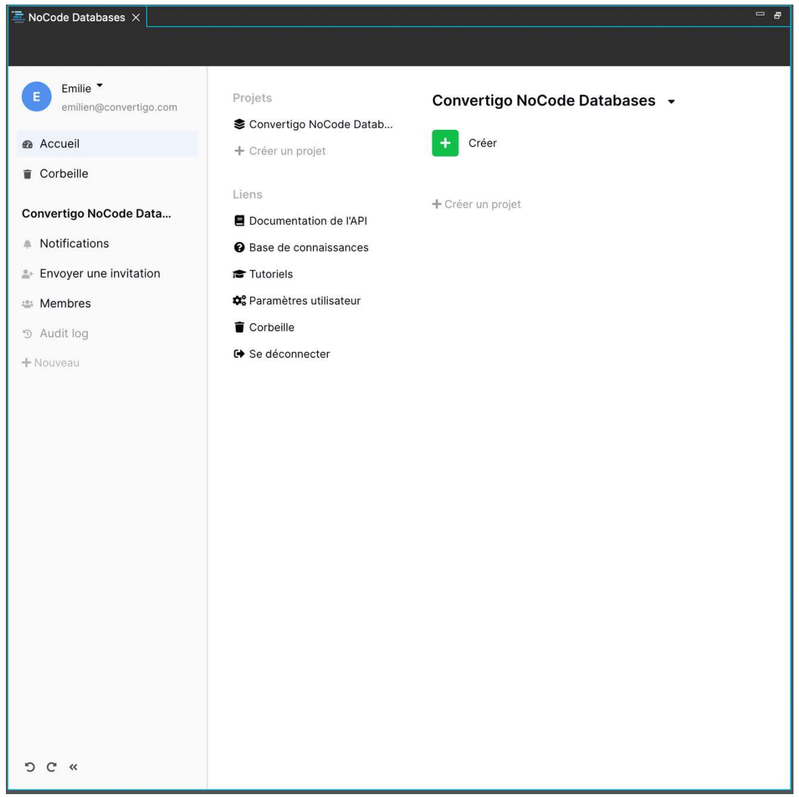
Click on **Accept** to join the Convertigo NoCode Databases project (either in the studio or in the browser).

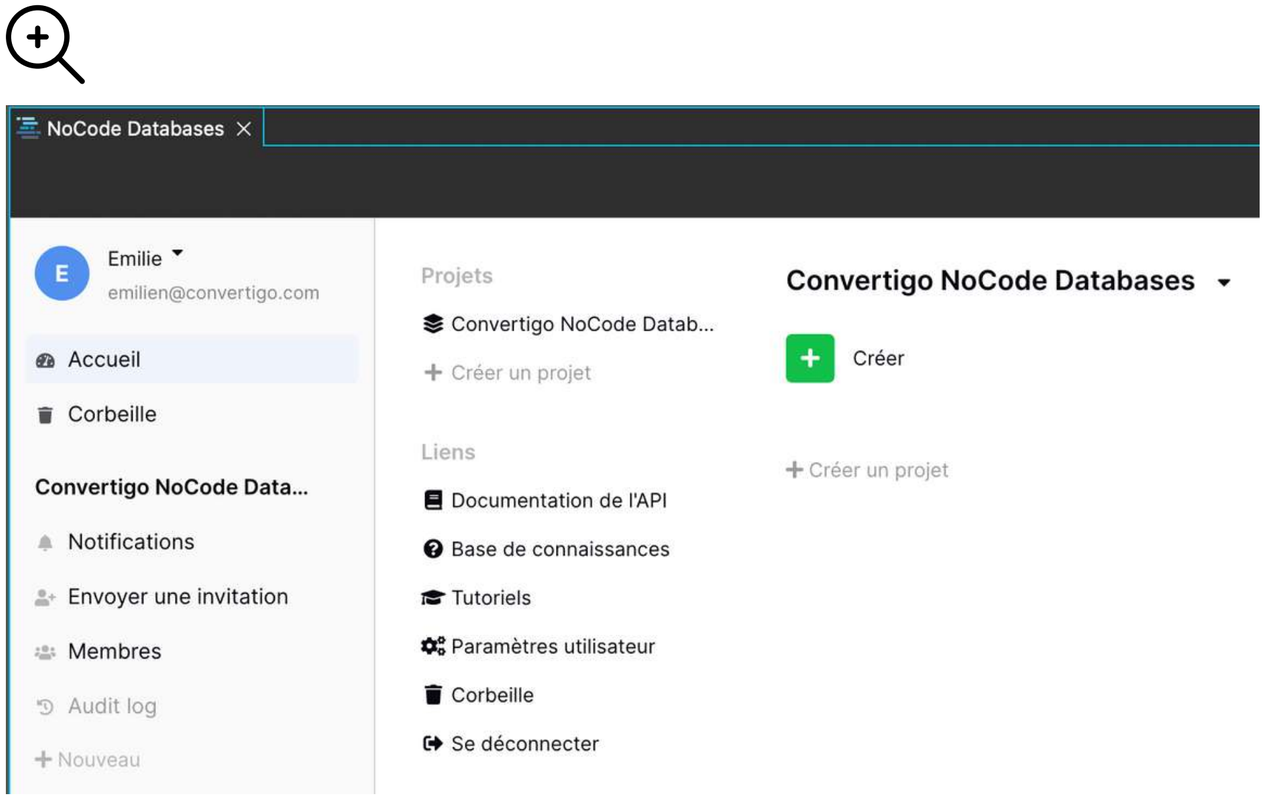




10.2 Set up your Baserow account

You are now ready to use your nocode database.

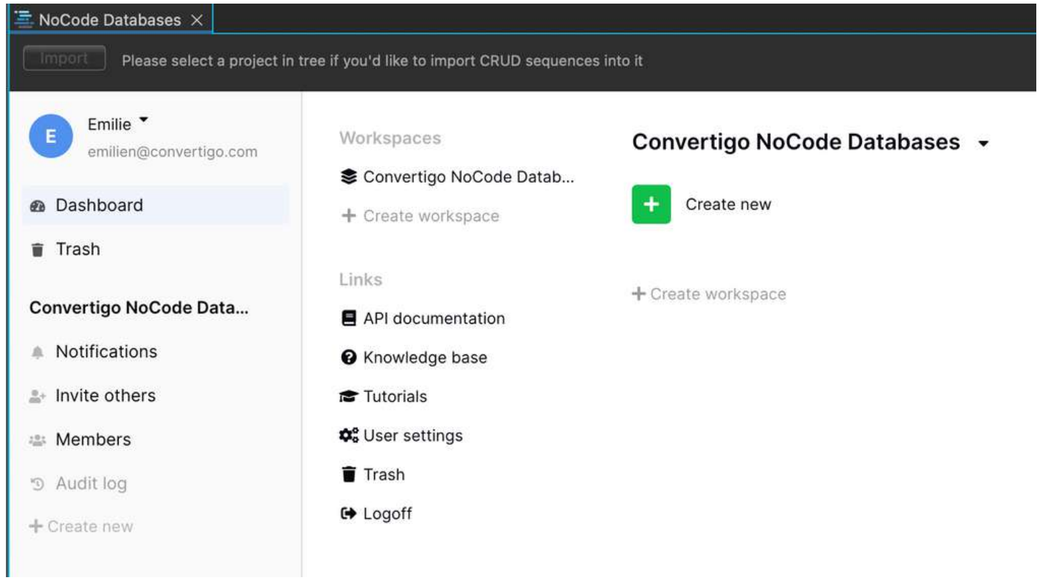




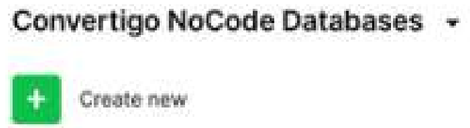
10.3 Create a database

Let’s create a new database

from the dashboard in the NoCode Databases view.

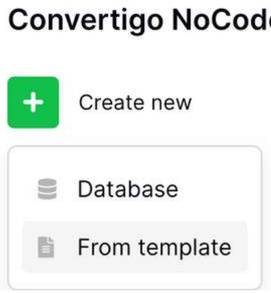


Click on **Create new**.



You can either

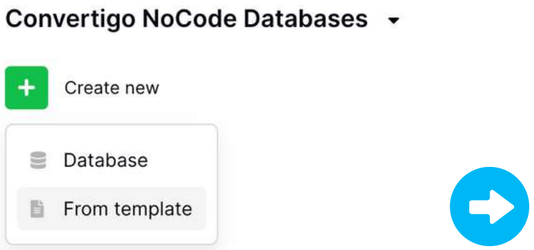
- start a new database from scratch
- or select a template



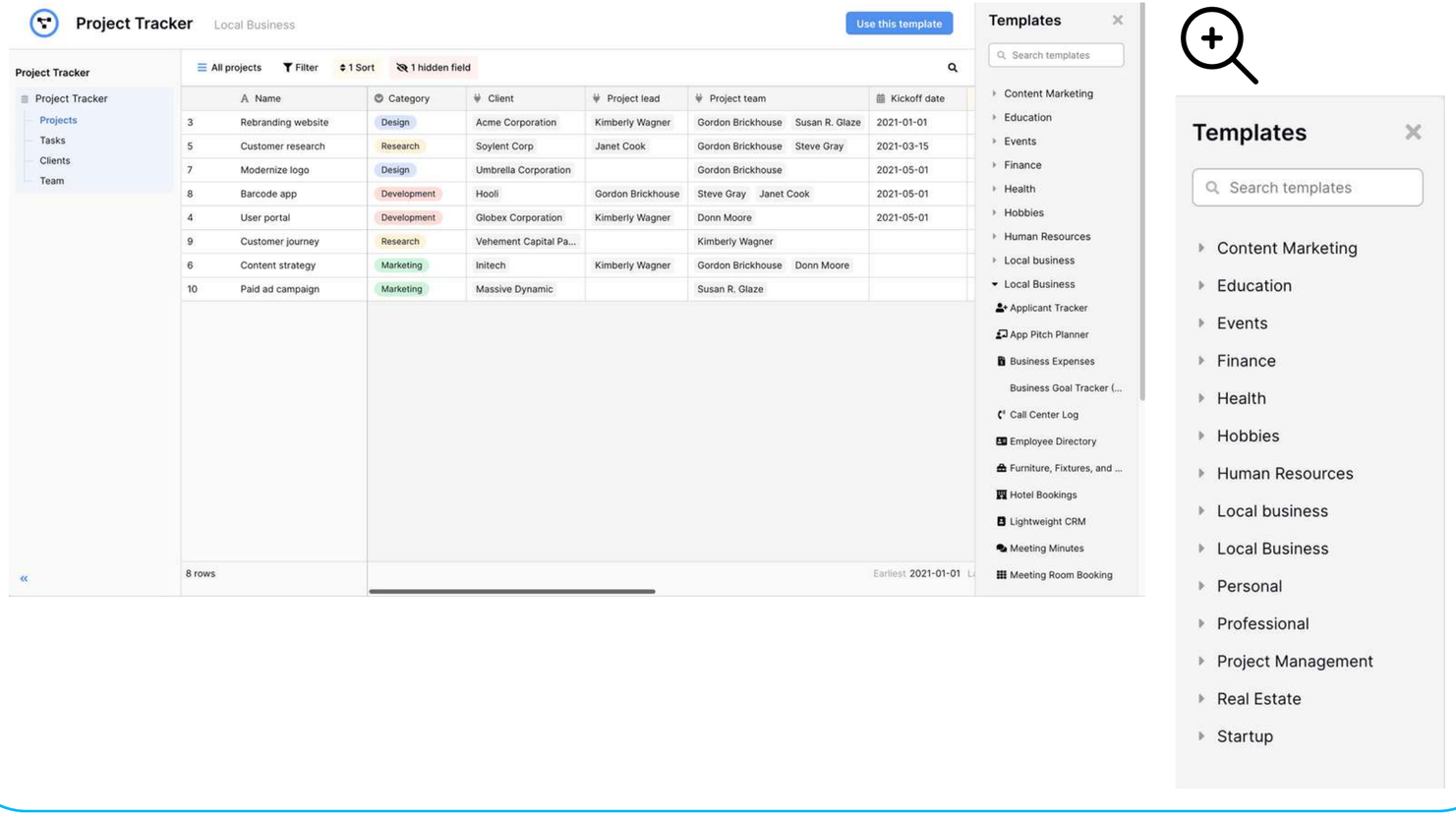
10.3 Create a database

First option :

you click on **From template**,



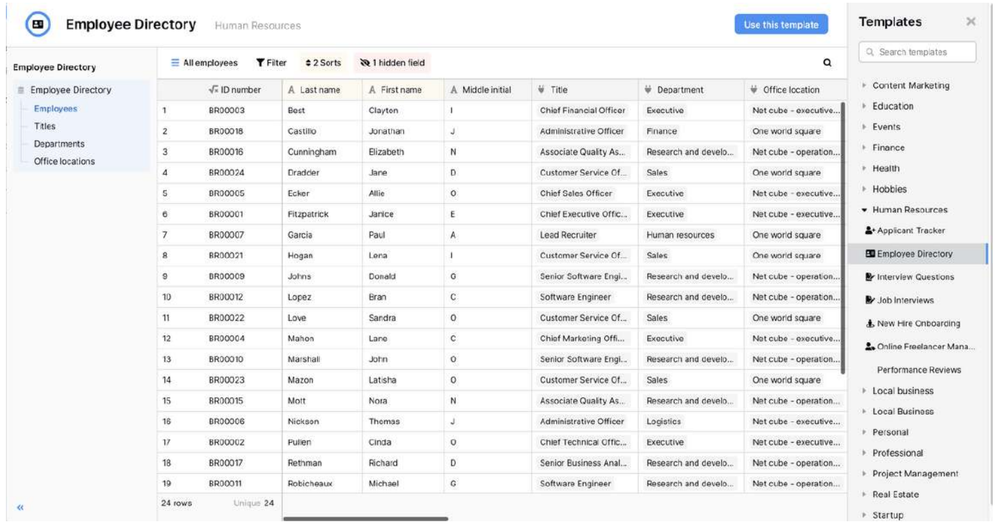
You will see a **selection of templates organized in categories**.

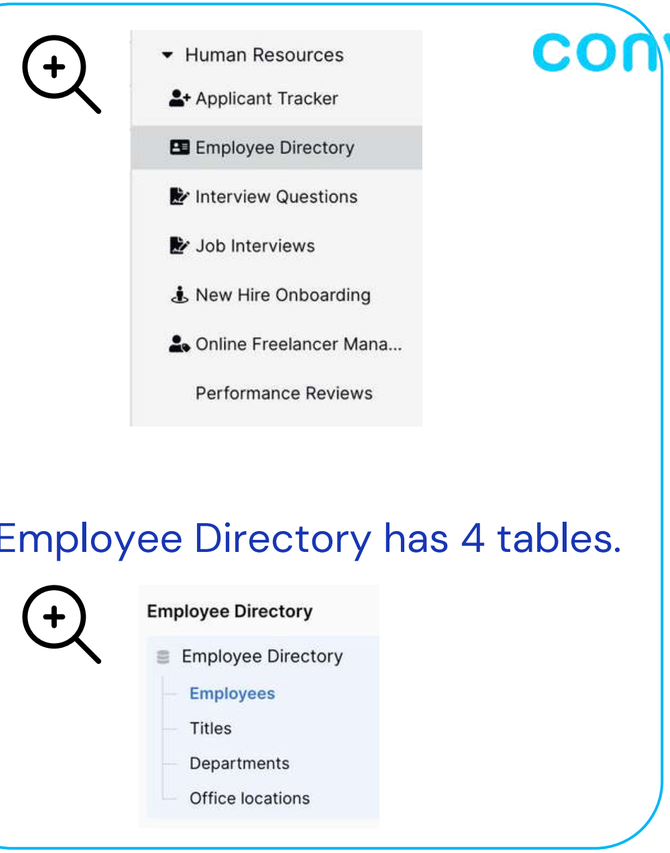


10.3 Create a database

Let’s select a template to manage an employee list.

Click on **Employee Directory** in the category **Human Resources** to display the **Employee Directory** database.





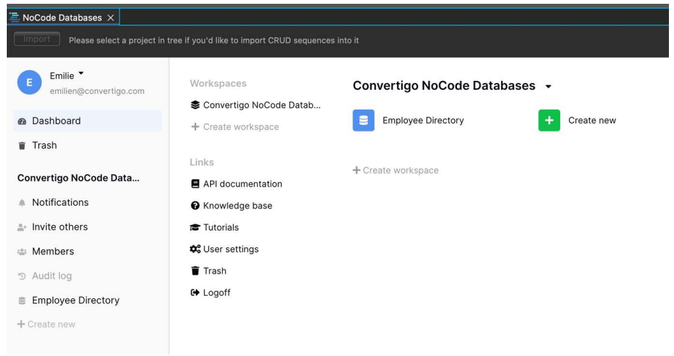
Click on **Use this template**.

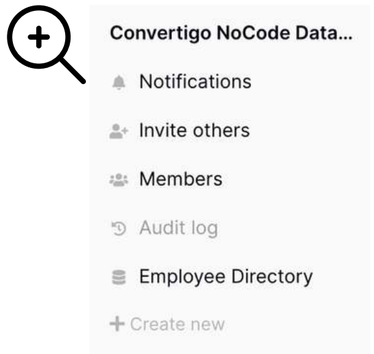
10.3 Create a database

Once created,

the **Employee Directory** database appears

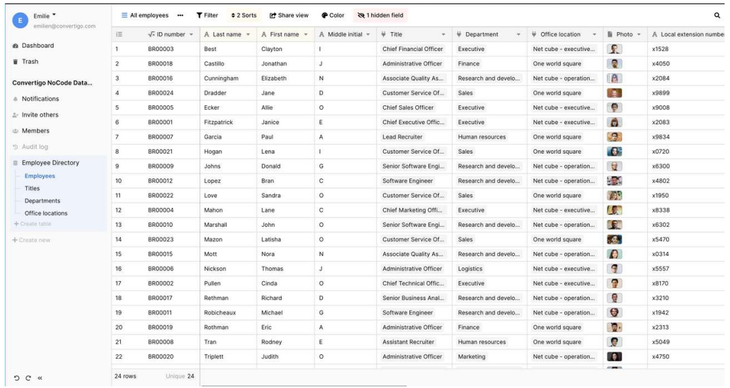
in the dashboard of the **NoCode Databases view**.

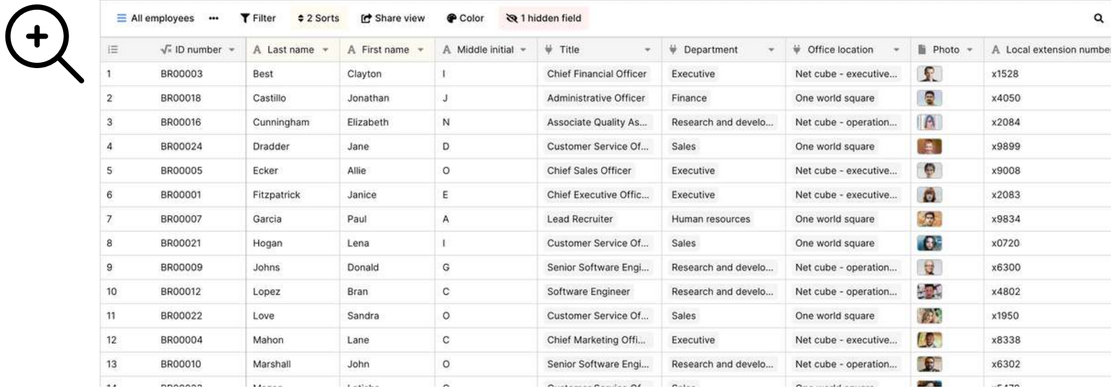




When you click on **Employee Directory**,

you can **see the tables and the data of the selected table**.



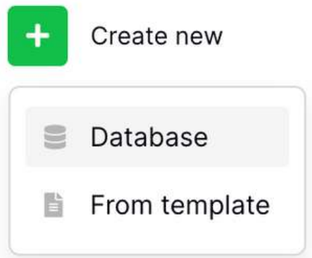


	ID number	Last name	First name	Middle initial	Title	Department	Office location	Photo	Local extension numbe
1	BR00003	Best	Clayton	I	Chief Financial Officer	Executive	Net cube - executive...		x1528
2	BR00018	Castillo	Jonathan	J	Administrative Officer	Finance	One world square		x4050
3	BR00016	Cunningham	Elizabeth	N	Associate Quality As...	Research and develo...	Net cube - operation...		x2084
4	BR00024	Dradder	Jane	D	Customer Service Of...	Sales	One world square		x9899
5	BR00005	Ecker	Allie	O	Chief Sales Officer	Executive	Net cube - executive...		x9008
6	BR00001	Fitzpatrick	Janice	E	Chief Executive Offic...	Executive	Net cube - executive...		x2083
7	BR00007	Garcia	Paul	A	Lead Recruiter	Human resources	One world square		x9834
8	BR00021	Hogan	Lena	I	Customer Service Of...	Sales	One world square		x0720
9	BR00009	Johns	Donald	G	Senior Software Engi...	Research and develo...	Net cube - operation...		x6300
10	BR00012	Lopez	Bran	C	Software Engineer	Research and develo...	Net cube - operation...		x4802
11	BR00022	Love	Sandra	O	Customer Service Of...	Sales	One world square		x1950
12	BR00004	Mahon	Lane	C	Chief Marketing Offi...	Executive	Net cube - executive...		x8338
13	BR00010	Marshall	John	O	Senior Software Engi...	Research and develo...	Net cube - operation...		x6302

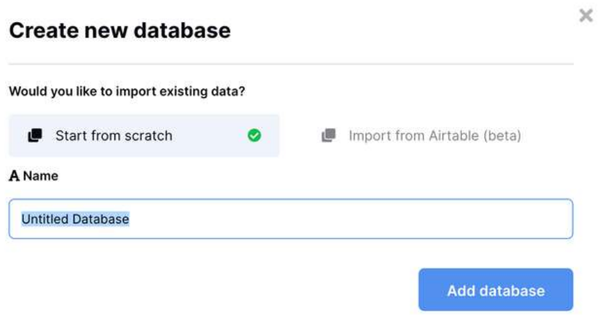
10.3 Create a database

Second option:

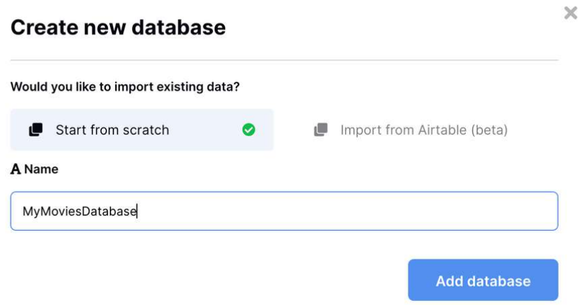
you click on **Database**



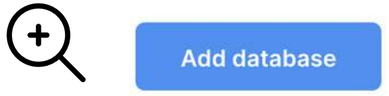
The **Create new database window** appears.



Choose a name for the **new database**.



Click on Add database.

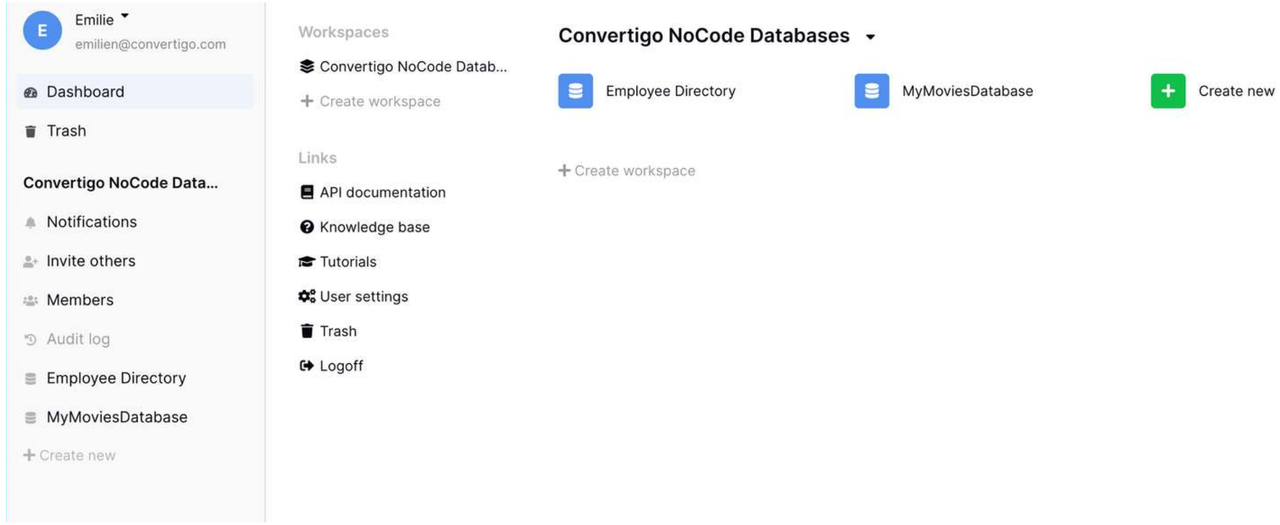


10.3 Create a database

Once created,

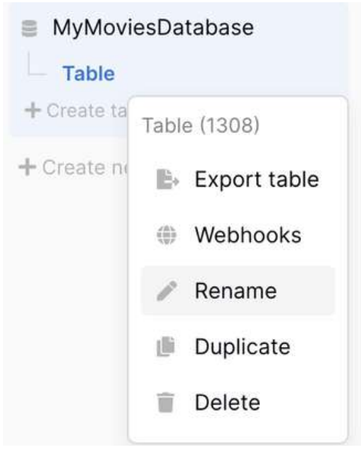
the **MyMoviesDatabase** appears

in the dashboard of the **NoCode Databases view**.

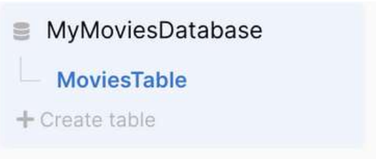


MyMoviesDatabase

has a table by default

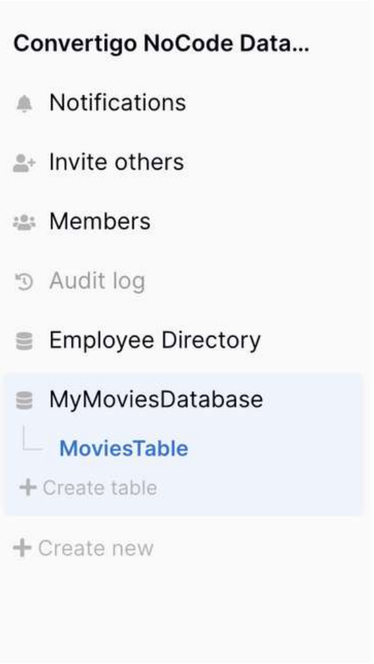


Rename it as **MoviesTable**

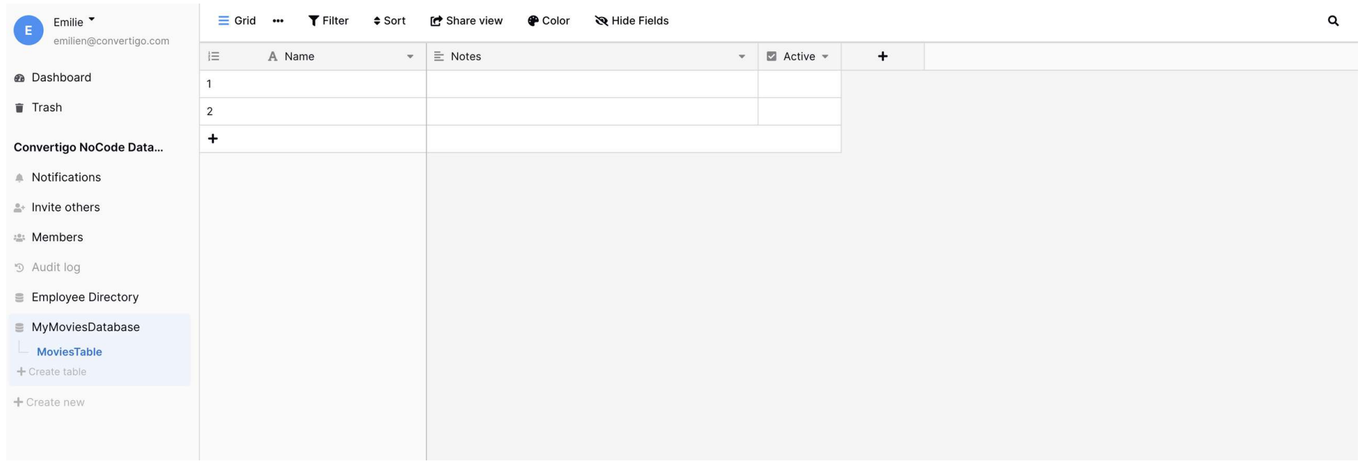


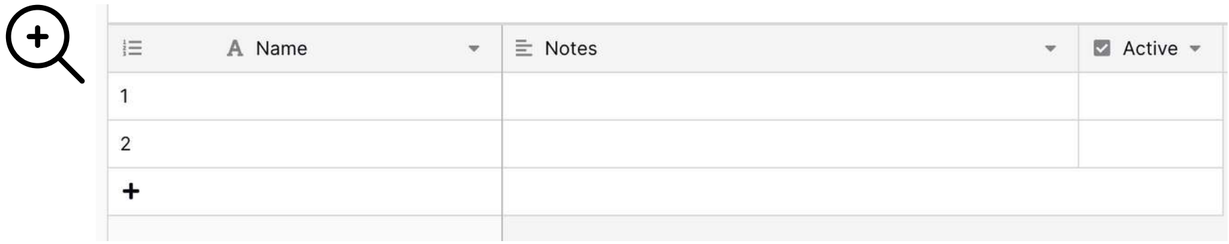
10.3 Create a database

Click on MoviesTable to display it.



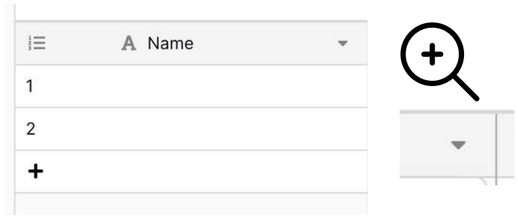
MoviesTable has a few fields by default.





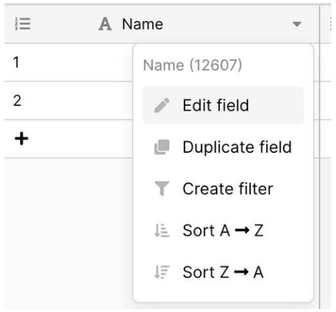
10.3 Create a database

Let's create a **"Title" field** in MoviesTable by editing the **Name field**.



Click on the down arrow

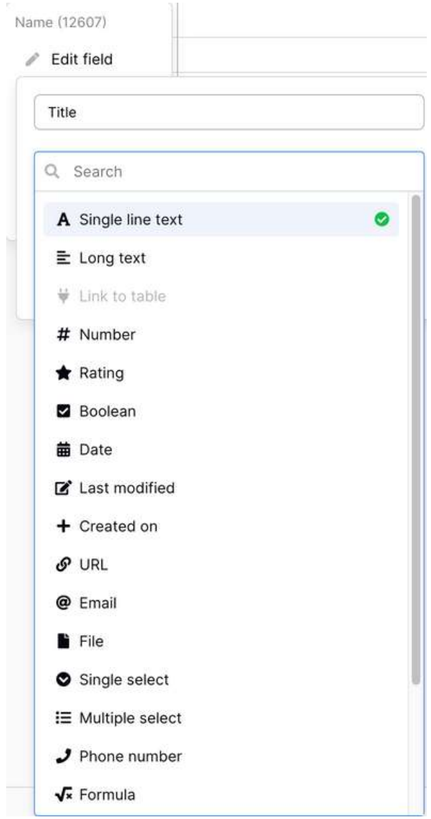
to display the menu



Change Name to Title

Select **Single line text**

as type of data for **Title field**.



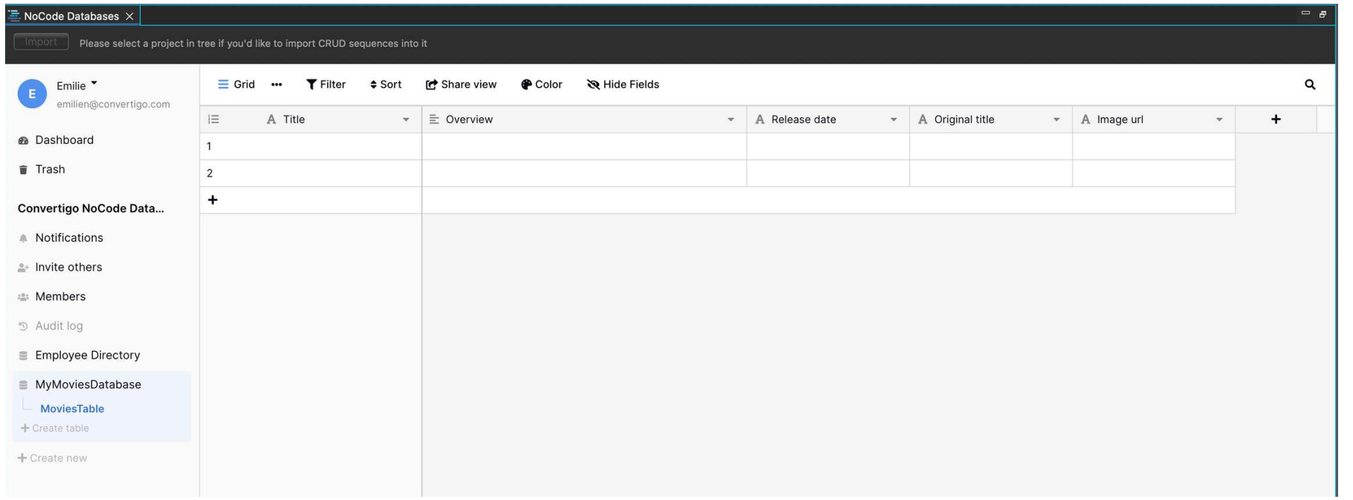
10.3 Create a database

Now we have a **Title field** in **MoviesTable**

Let’s do the same thing and add a few more new fields in MoviesTable.

Now we have 5 fields in MoviesTable:

Title, Overview, Release Date, Original title and Image url.



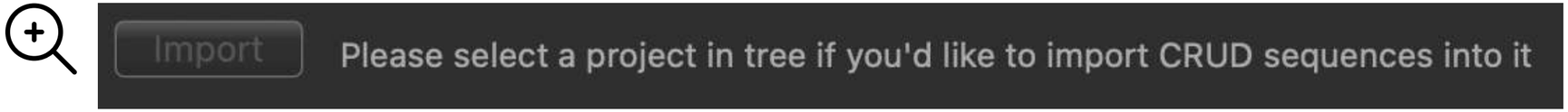
10.4 Import CRUD sequences into a project

You are now ready to use your nocode database.

When you select a project and click on **Import**,

CRUD sequences will automatically be **created from baserow definitions of the table you selected**,

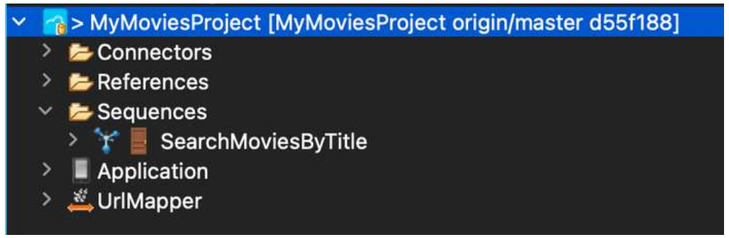
then **imported in your project**.



10.4 Import CRUD sequences into a project

In the Project view,

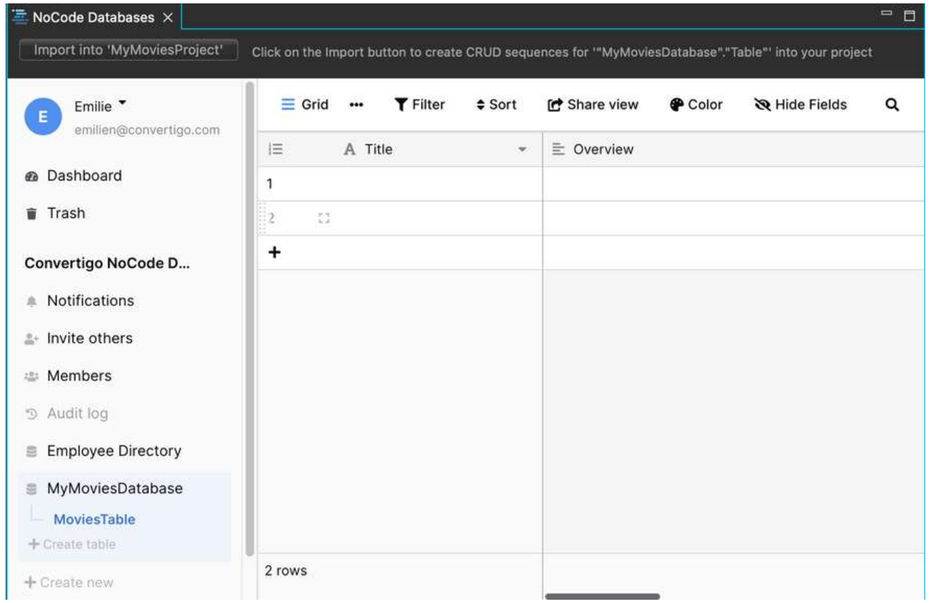
click on MyMoviesProject to select it.

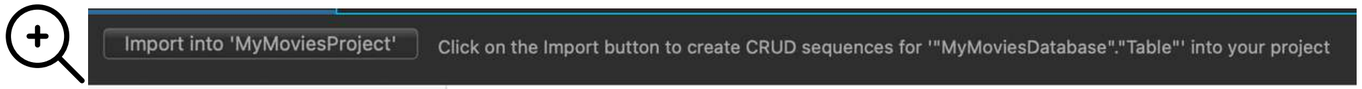


In the NoCode Databases view,

the Import button text has changed

from **Import** to **Import into “MyMoviesProject”**.





10.4 Import CRUD sequences into a project

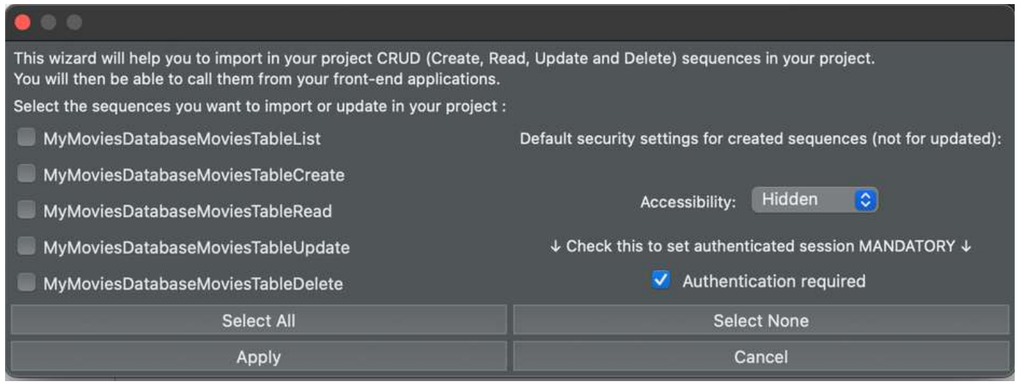
In the NoCode Databases view,

click on **Import into “MyMoviesProject”**.

A window allowing you

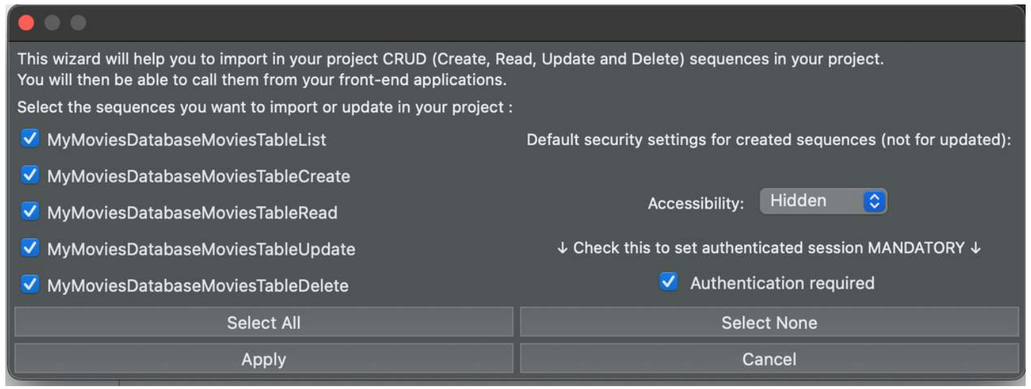
to automatically create and import CRUD sequences

into your project appears.



Click on Select All

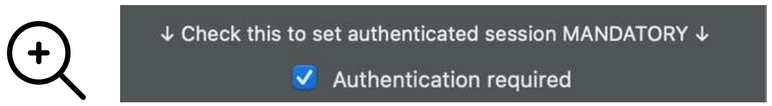
to select all CRUD sequences .

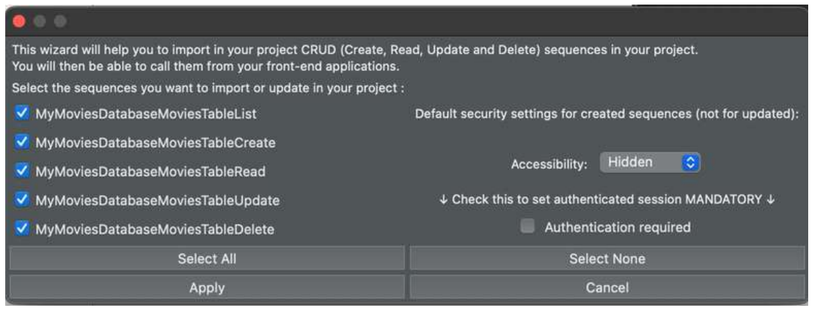


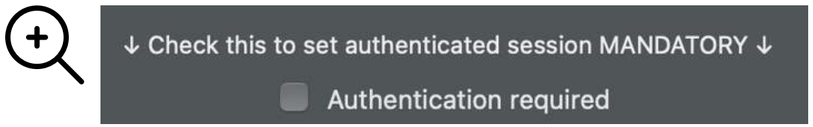
10.4 Import CRUD sequences into a project

We don’t use authentication

so **uncheck Authentication required**.



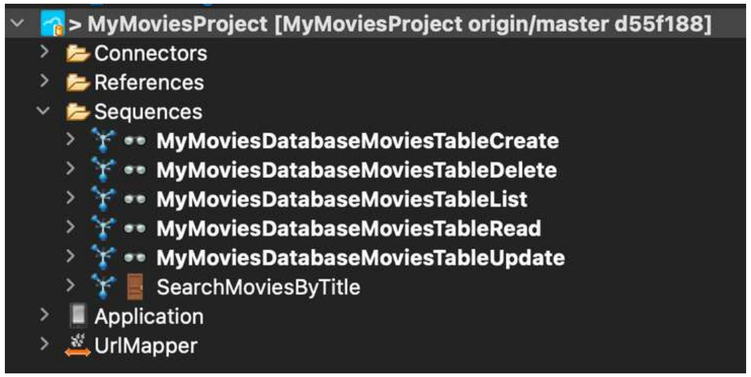




Click on Apply

The CRUD sequences were imported

and appear in **MyMoviesProject** in **Project view**.

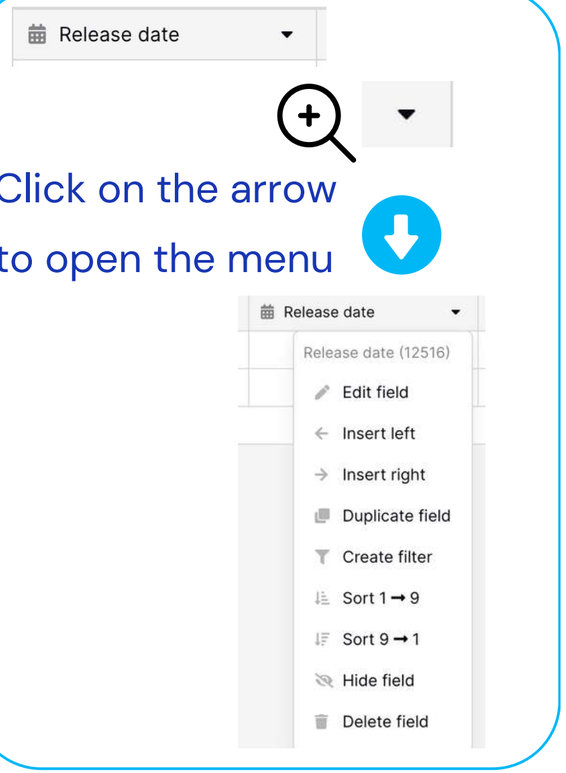


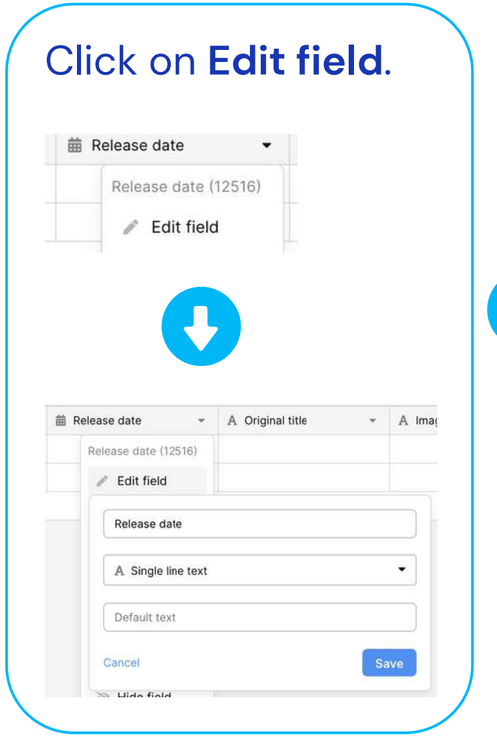
10.4 Import CRUD sequences into a project

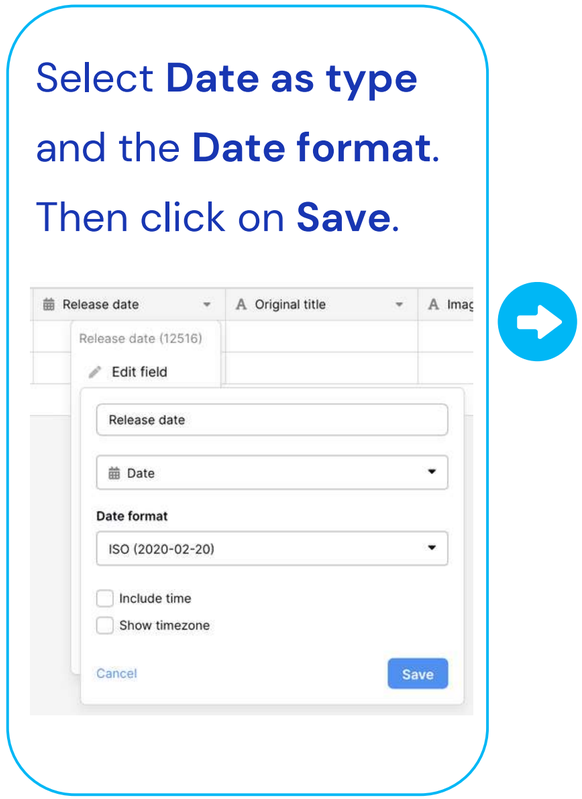
If you **change anything in the table**,

the **table definition in Baserow changes** and you have to **reimport the table**.

For example, let's **edit the Release date field** to **change the type** from Single line text to **Date**.







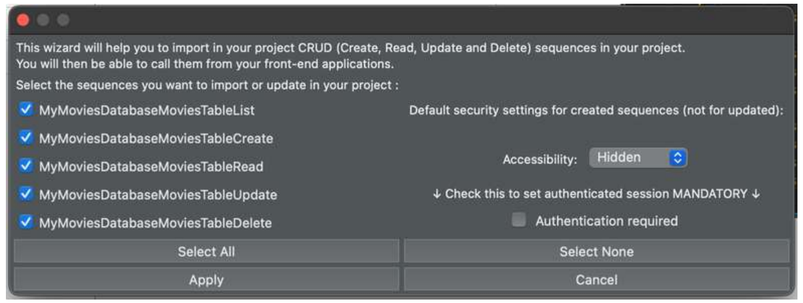
10.4 Import CRUD sequences into a project

In the **NoCode Databases view**,

click on **Import into "MyMoviesProject"**.

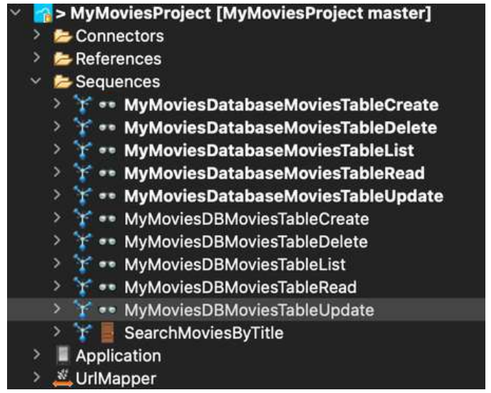
Click on **Select All**

then click on **Apply**.

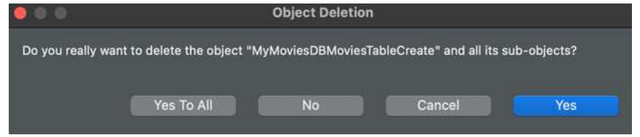


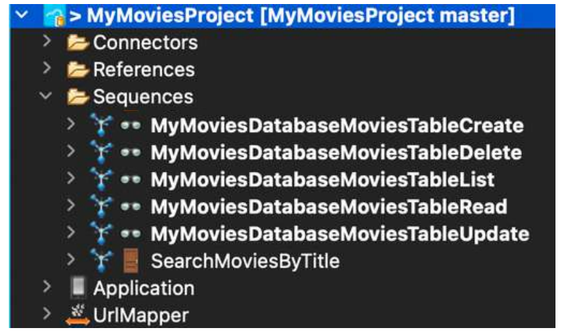
The new CRUD sequences were imported

and appear in **MyMoviesProject** in **Project view**.



Delete the old CRUD sequences.

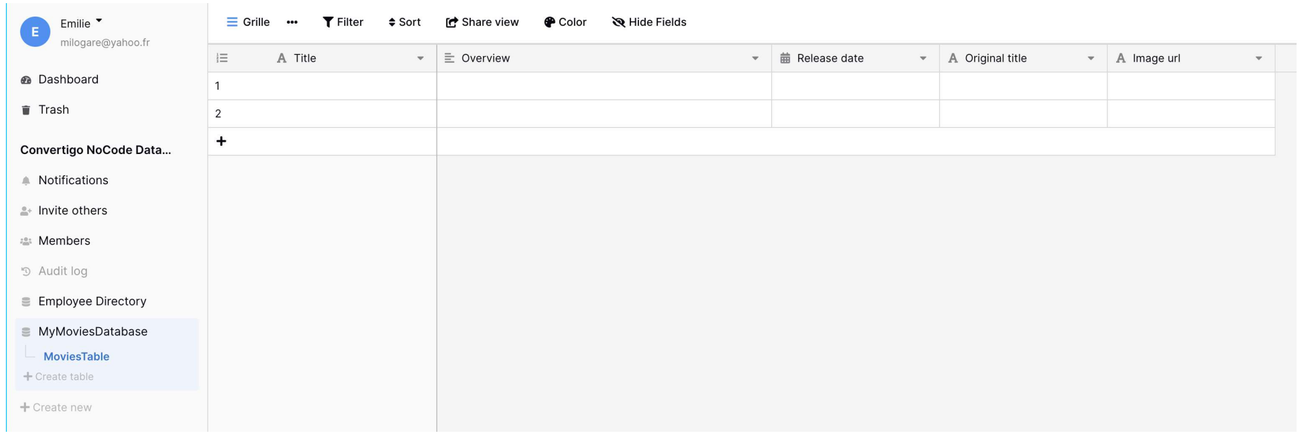


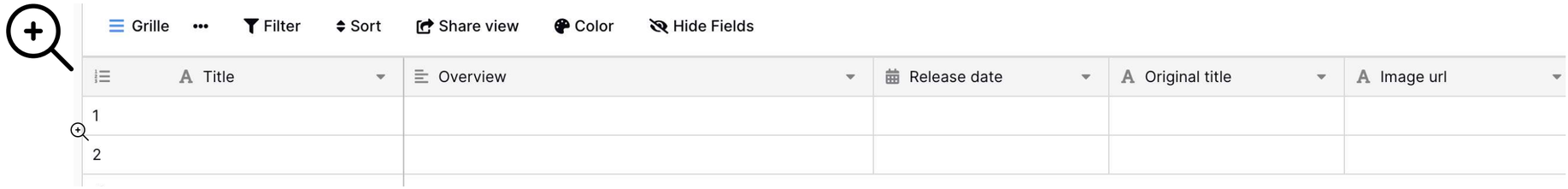


10.5 Add filters in a table

Let's say you want to **add a filter** on the **Release Date** field

to **select only the movies released after a specific date**.

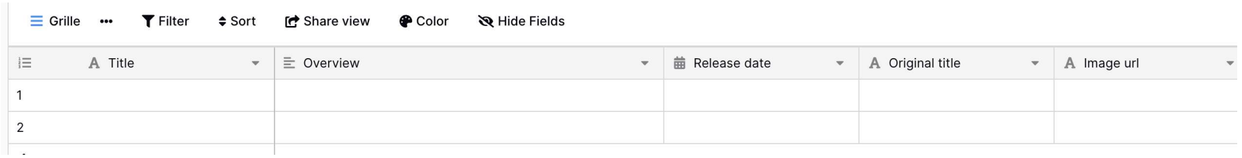




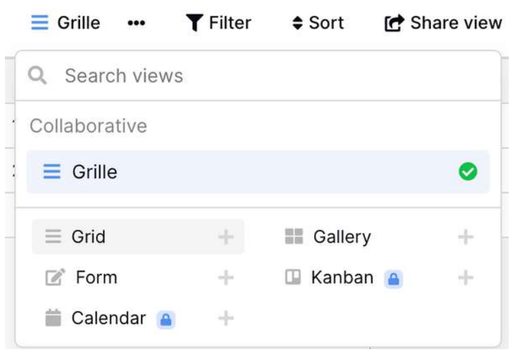
We are in the **Grille view**.

To **add filters**, we need to **add a new view**.

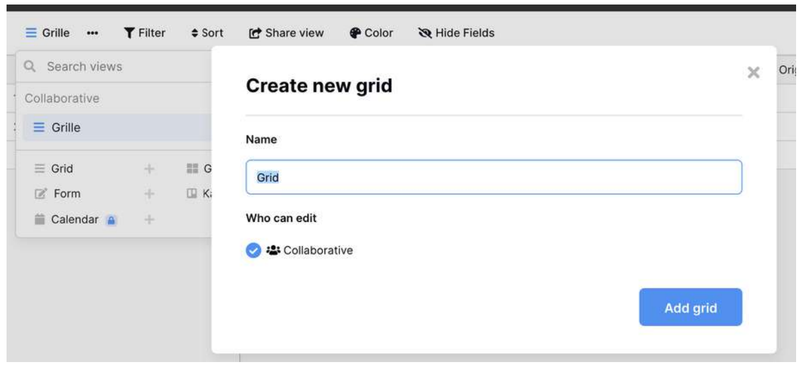
10.5 Add filters in a table



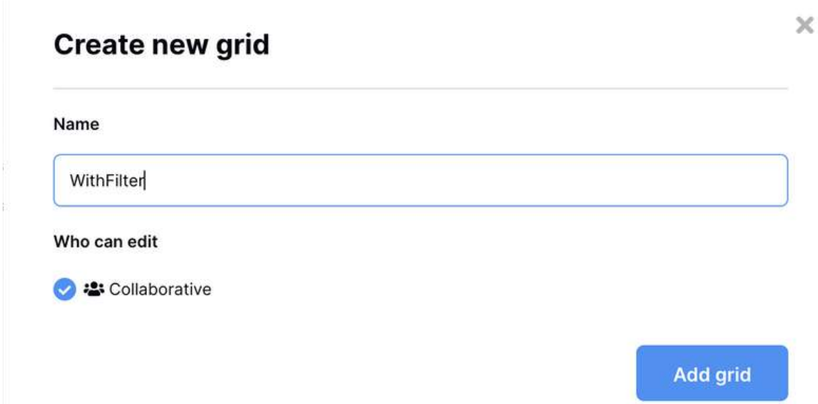
Click on the **view name** (Grille, in our case) to **open the views menu**.



Click on **Grid** to open the **Create new grid window**.



Rename the Grid as **WithFilter** and click on **Add grid**.

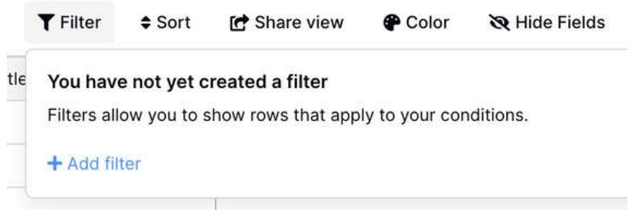


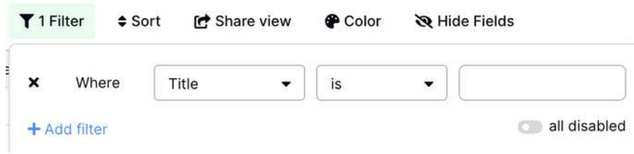
10.5 Add filters in a table

We are now in the new **view WithFilter**.

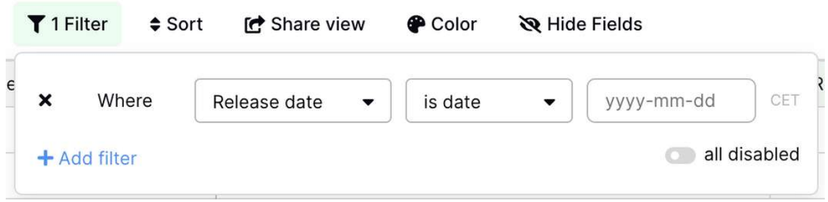
Filter Click on **Filter**.

Click on **Add filter**.

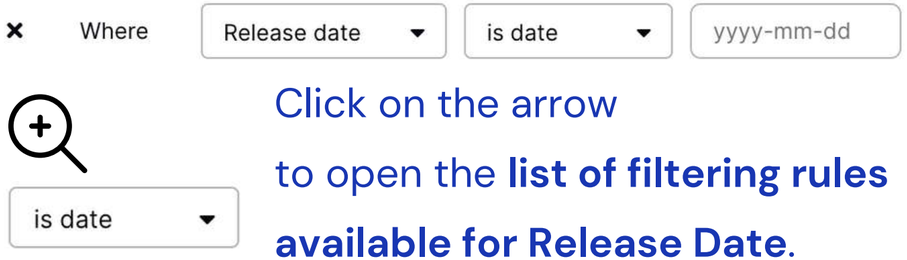




Select the field where you apply the filter by **changing Title into Release date**.



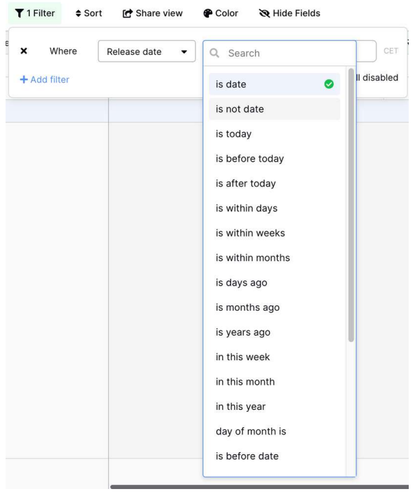
10.5 Add filters in a table



Click on the arrow

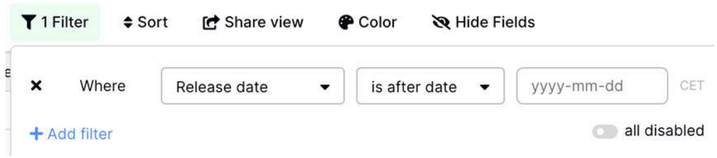
to open the **list of filtering rules available for Release Date**.

Select the filtering rule

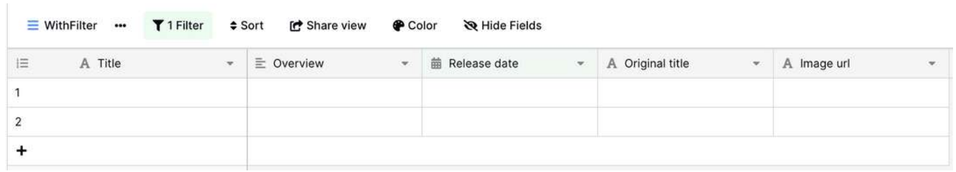


Set the date to **is after date**.

The last input field (where you set the date) stays empty.

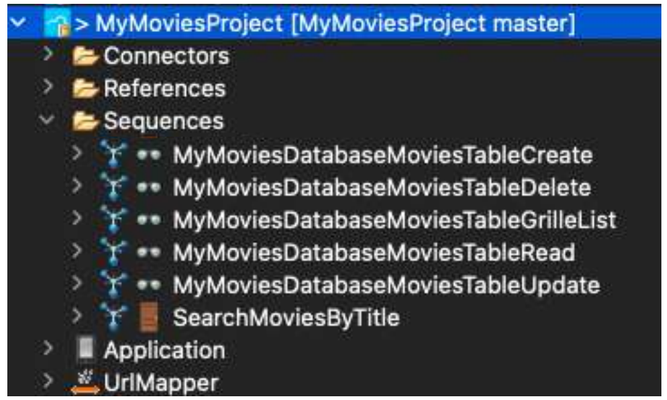


Your filter is created in the view.

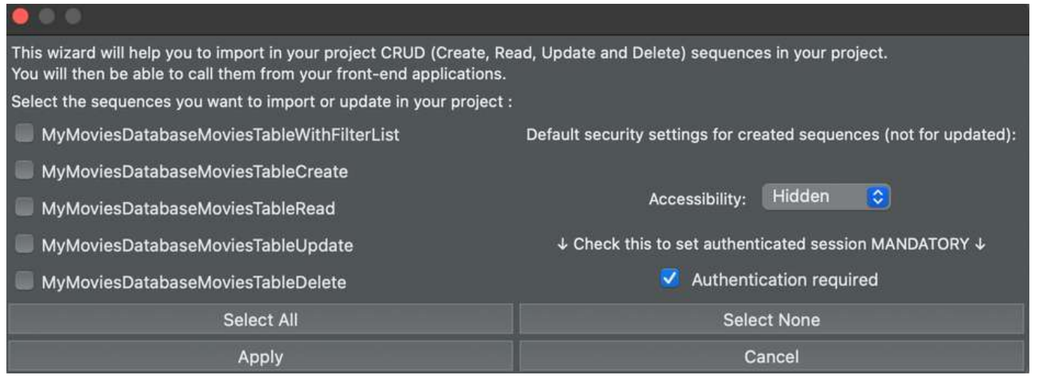


10.5 Add filters in a table

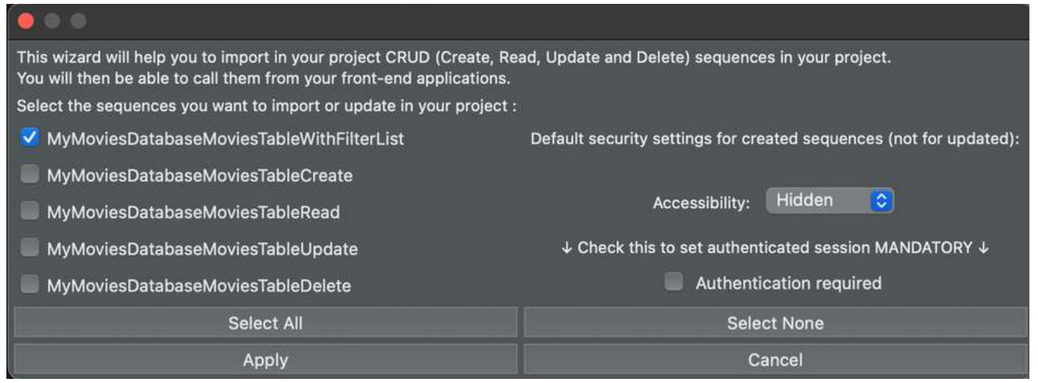
Let’s create and import the sequence corresponding to the filter in **MyMoviesProject**.



In the **NoCode Databases view**, click on **Import into “MyMoviesProject”**.

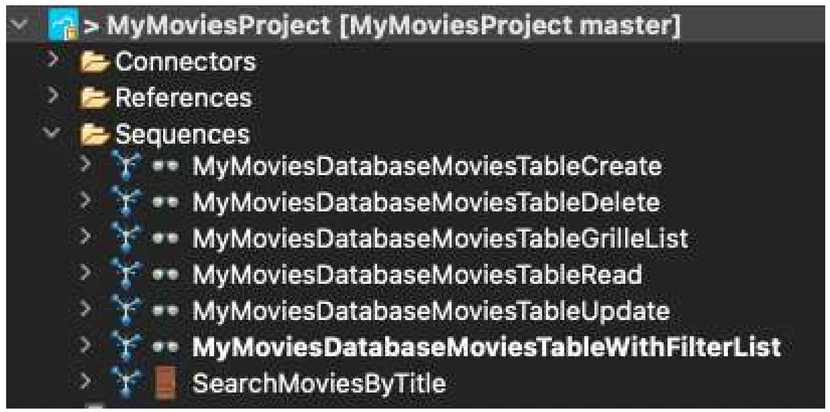


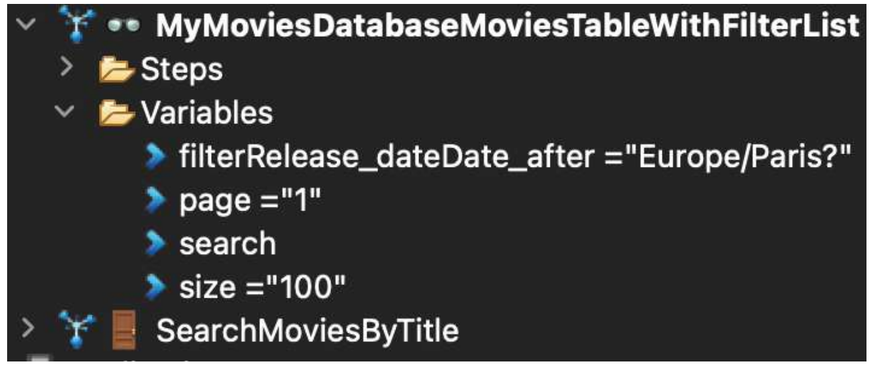
Select the sequence, uncheck **Authentication required** and click on **Apply**.



10.5 Add filters in a table

The sequence corresponding to the filter has been imported in **MyMoviesProject**.





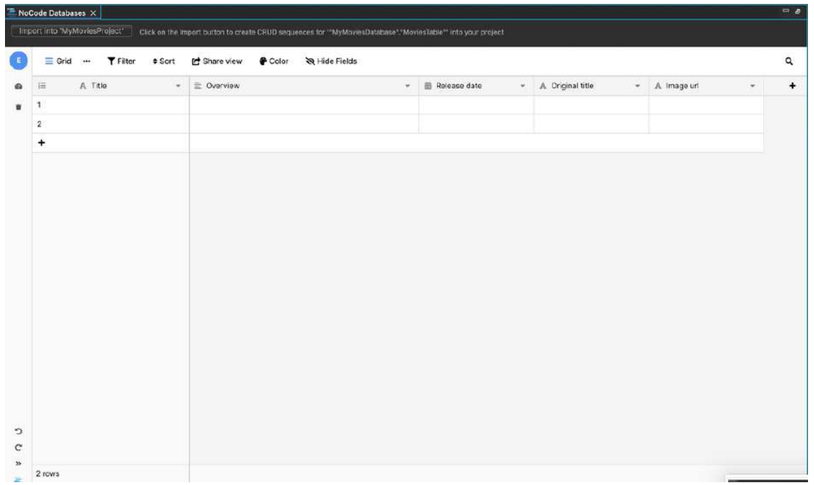
In the variables folder,

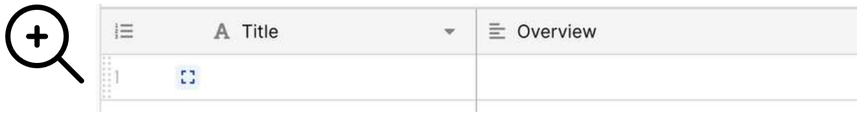
we can see a variable corresponding to the filter.

10.6 Test the CRUD sequences

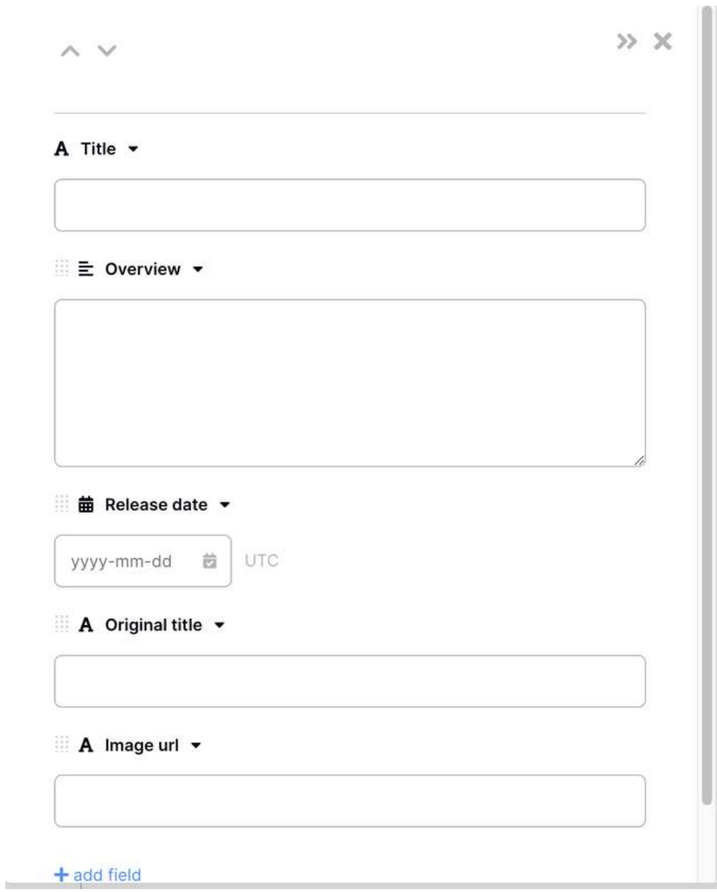
Let’s say we want to **read data from the first row** in MoviesTable.

First, let’s create an entry directly in the table.





Click on this icon in the field Title to open the field editor

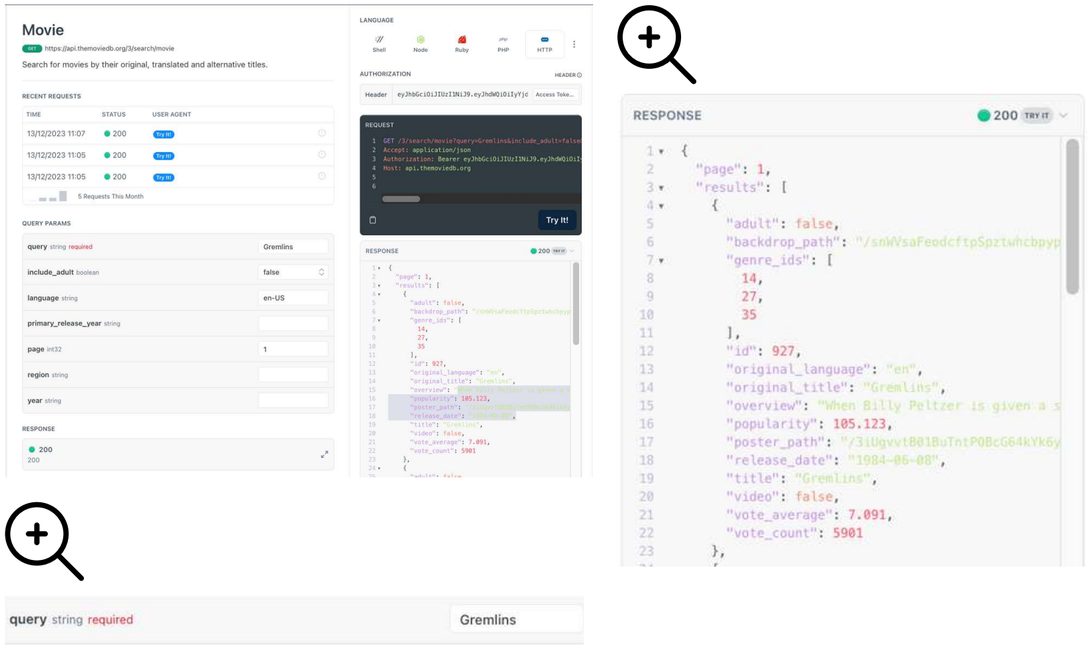


10.6 Test the CRUD sequences

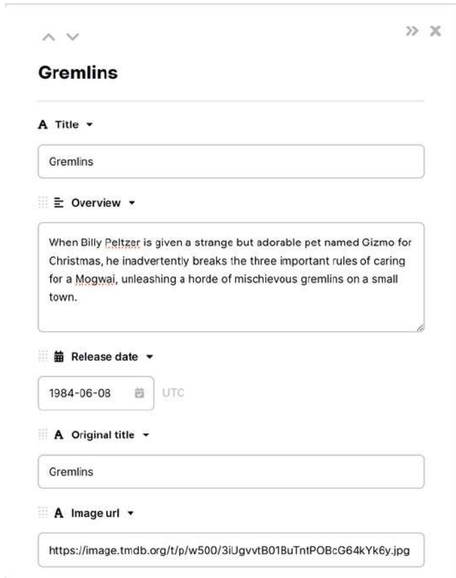
You can search data in the editor available in the TMDB API Search Movie page.

https://developer.themoviedb.org/reference/search-movie

For example, you can try it with the movie ‘Gremlins’



Fill the fields in the editor.

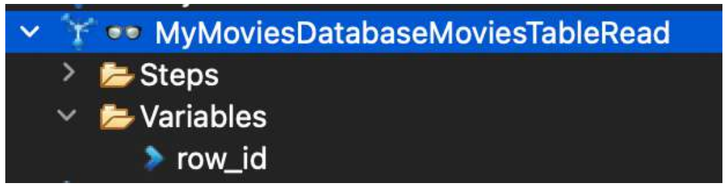


Now, we have data in the first row in MoviesTable

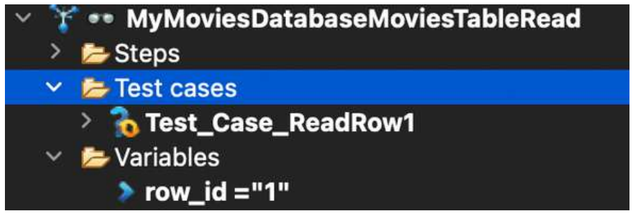
10.6 Test the CRUD sequences

To **read data from a row** in MoviesTable, we use the **sequence MyMoviesDatabaseMoviesTableRead**.

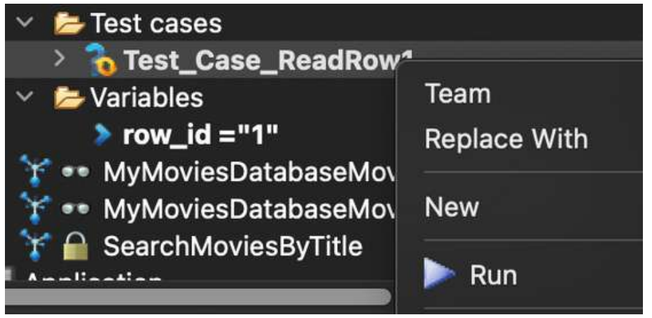
The sequence has one variable : row_id



Let's create a test case with 1 as row_id value

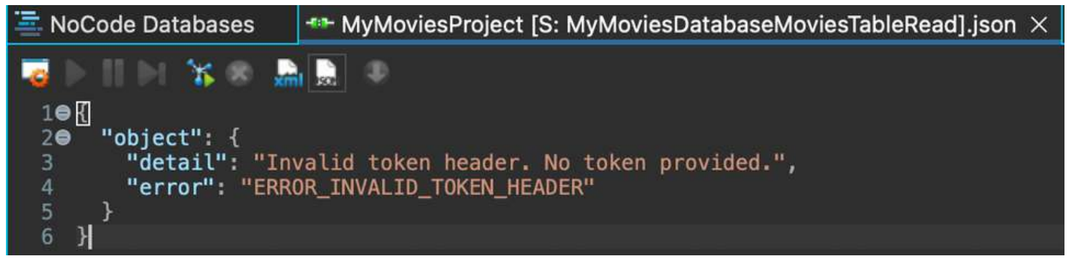


Run the test case



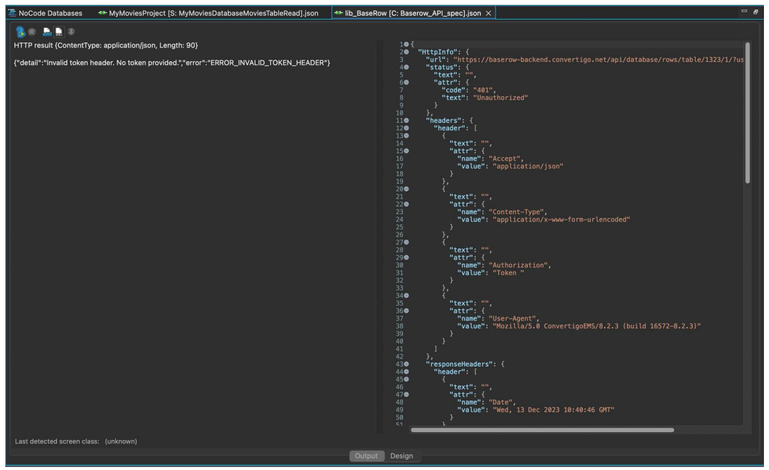
10.6 Test the CRUD sequences

The sequence returns an error: invalid token.



If we look at the **Baserow_API response**,

we can see that the database returns an **error 401 Unauthorized**.

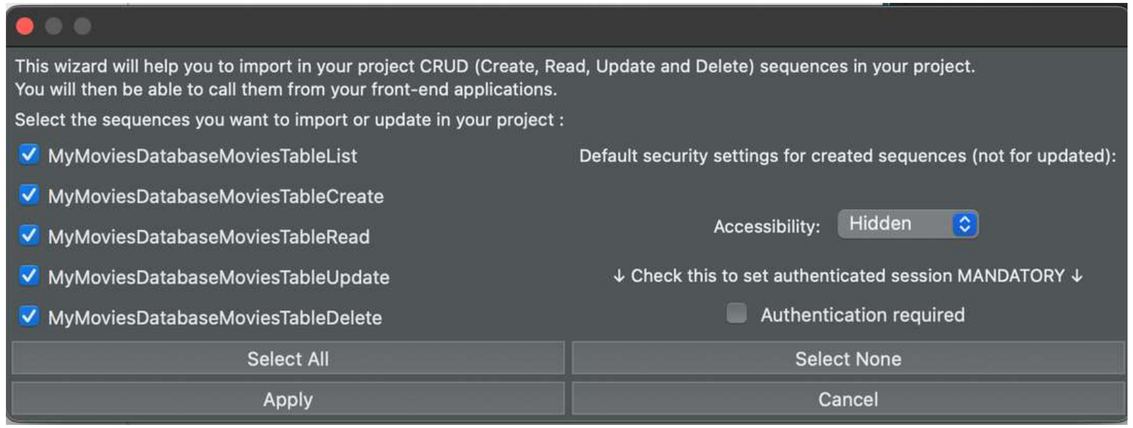


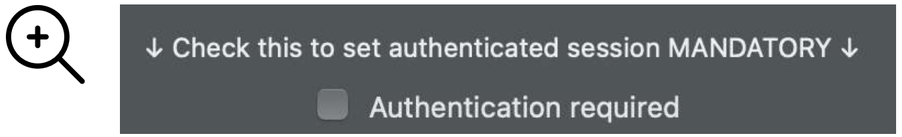
10.6 Test the CRUD sequences

Remember,

when we created and imported the CRUD sequences,

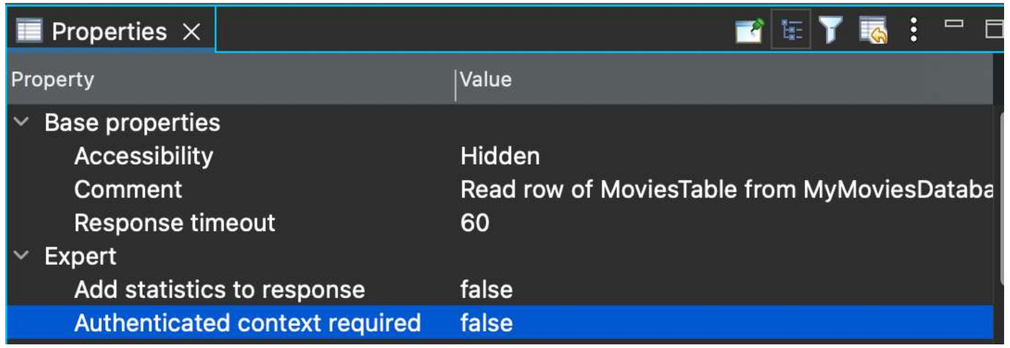
we unchecked the field **uncheck Authentication required**.

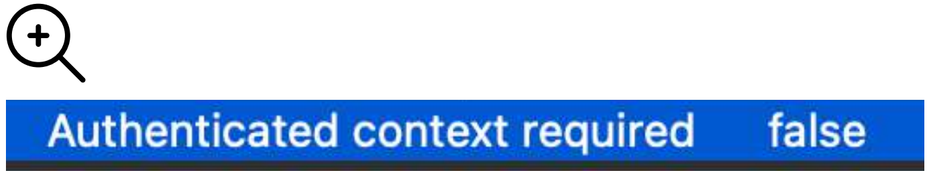




In the sequence properties,

the **Authentication context required property value** is **false, as expected.**

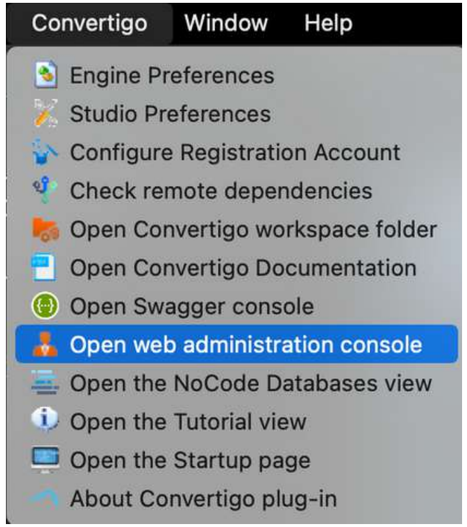




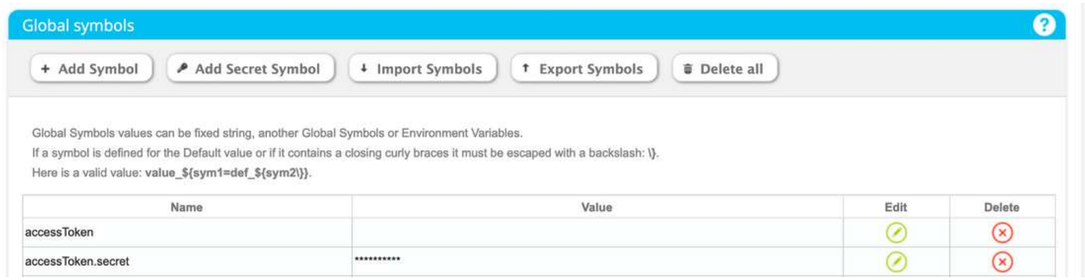
If we don’t need an authentication, why does the Baserow_API ask for a token ?

10.6 Test the CRUD sequences

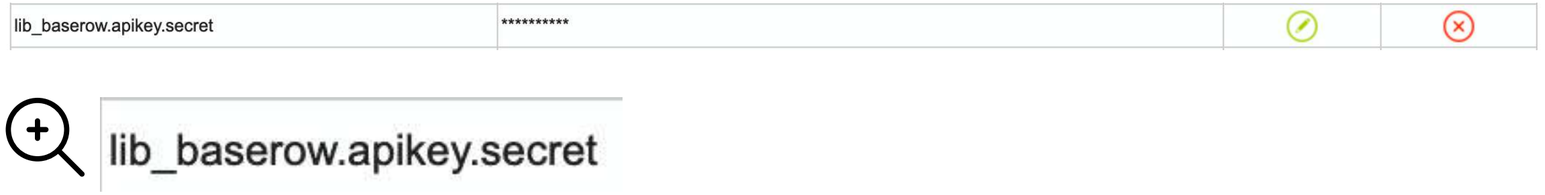
The answer is in the Web administration console.



Let's look at the **Global symbols**.



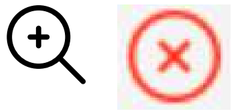
Baserow **adds automatically an apikey** in the symbols.



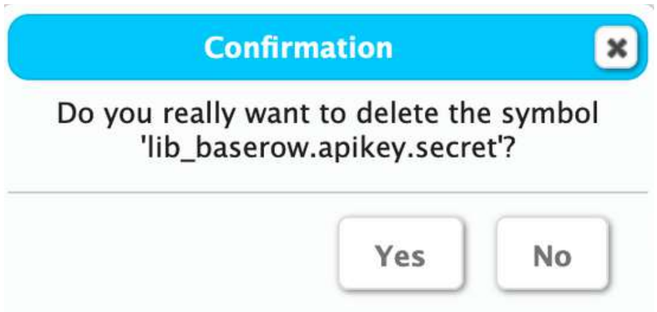
10.6 Test the CRUD sequences

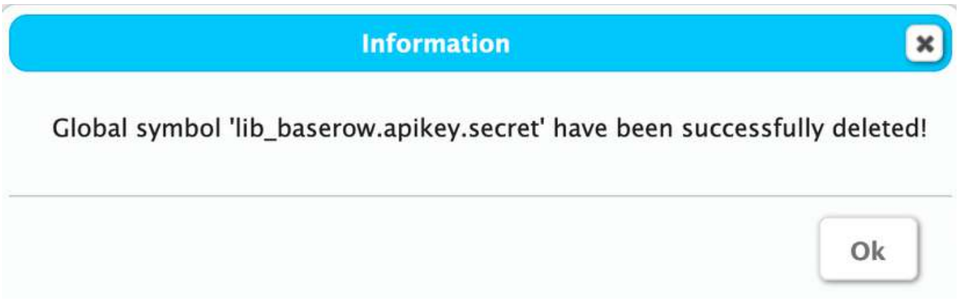
We need to delete this apikey.

lib_baserow.apikey.secret	**********		



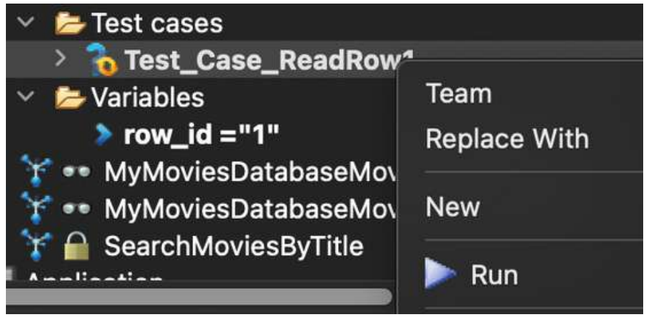
Click on the Delete icon.





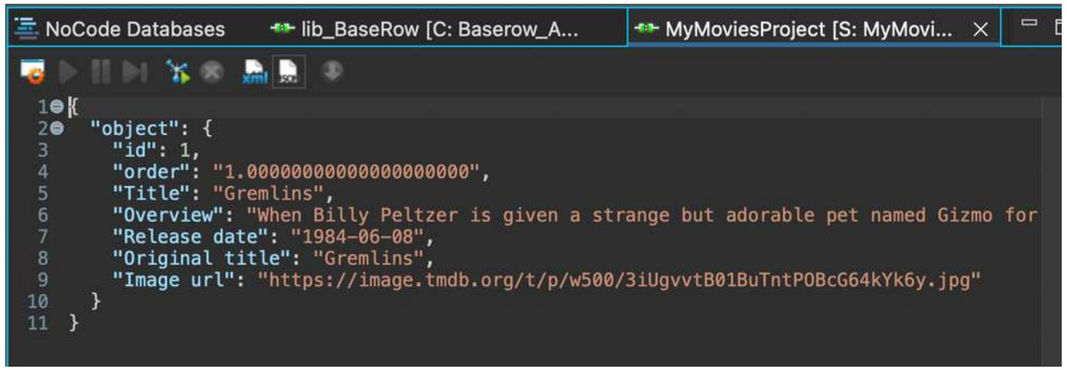
10.6 Test the CRUD sequences

Close the No Code database view and run the test case again.



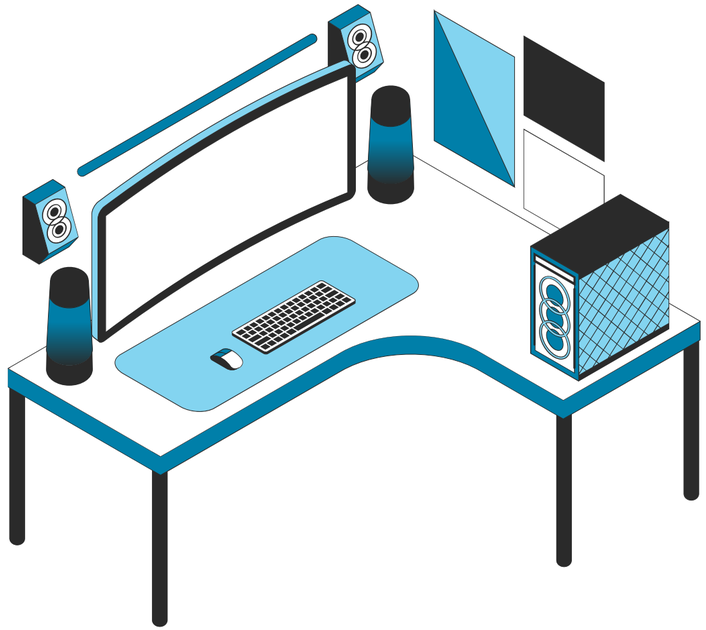
The request is authorized by Baserow

The sequence returns the data from the table.



11 – Logs

How to manage logs in the studio.



11.1 Engine Log view

11.2 Basics in Log level Configuration

11.3 Configure Log level

11.4 Logs in the web administration console

11.5 Log step

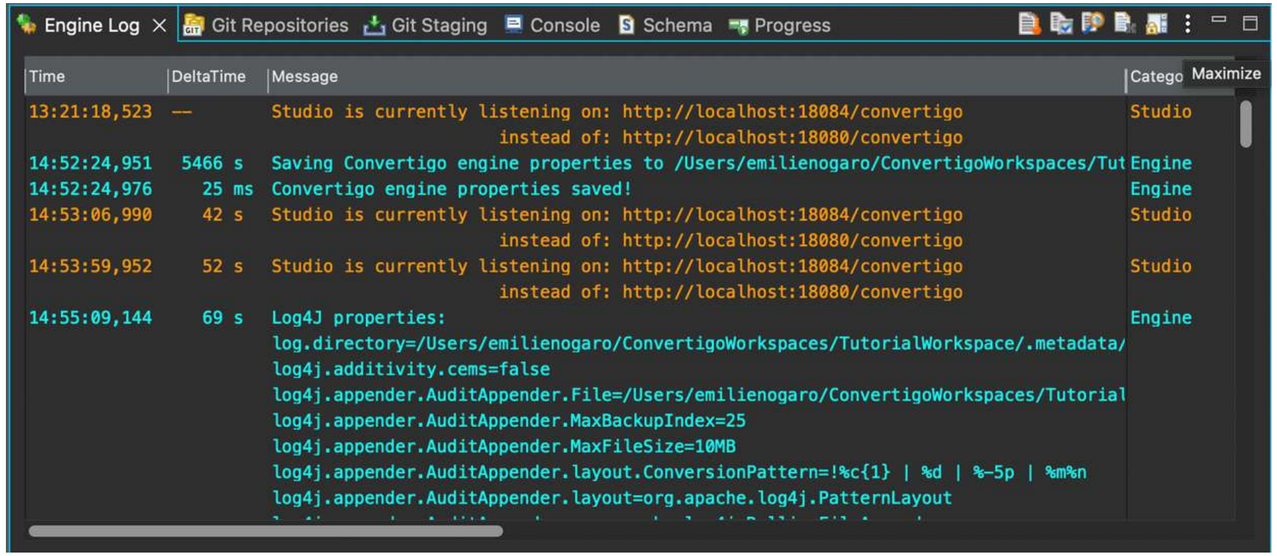
11.6 Using Log step in a sequence

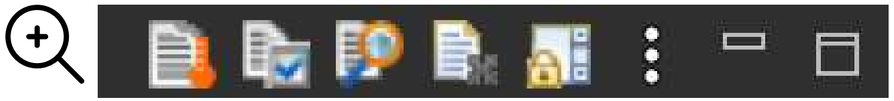
11.1 Engine Log view

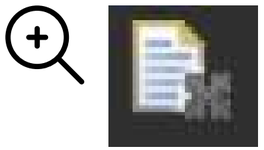
In the studio,

the logs are displayed in the **Engine Log view**

in the **Logs & Git panel**.

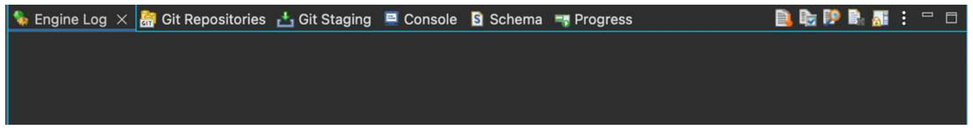




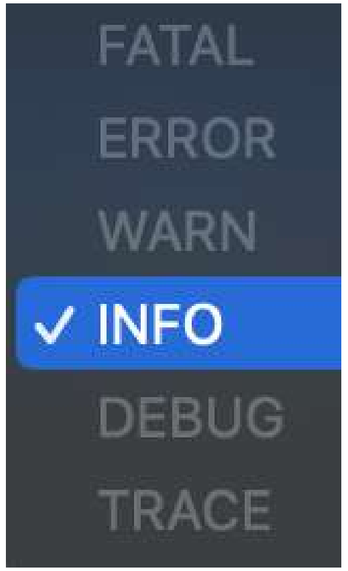


Click on the **Clear log viewer** icon

to clear the logs.



11.2 Basics in Log level Configuration



By default, the **root logger** is set to **INFO**.
The others loggers are set to **WARN** or **INFO**.

The most useful and important logs (beans, context, engine and user context)
are set to **“Inherited from root logger“**.

In **development mode**, to analyze issues in a project,
the root logger is set to **DEBUG**.

In **Production mode**, we usually set **root logger to WARN**.

Sometimes, other loggers (specific and not commonly used)
are lowered to WARN to gain space and speed.

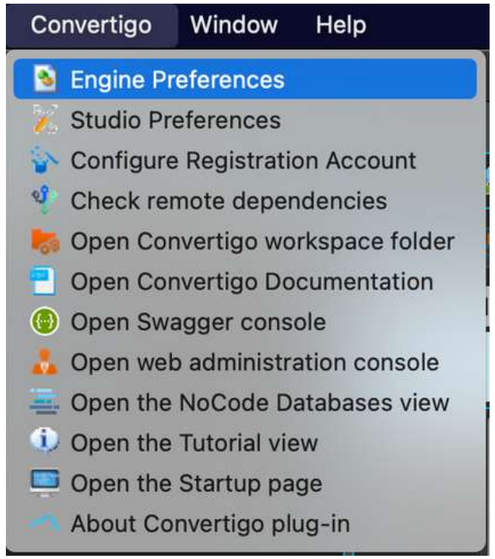
11.3 Configure Log level

In the studio, you can configure the log level in differents places.

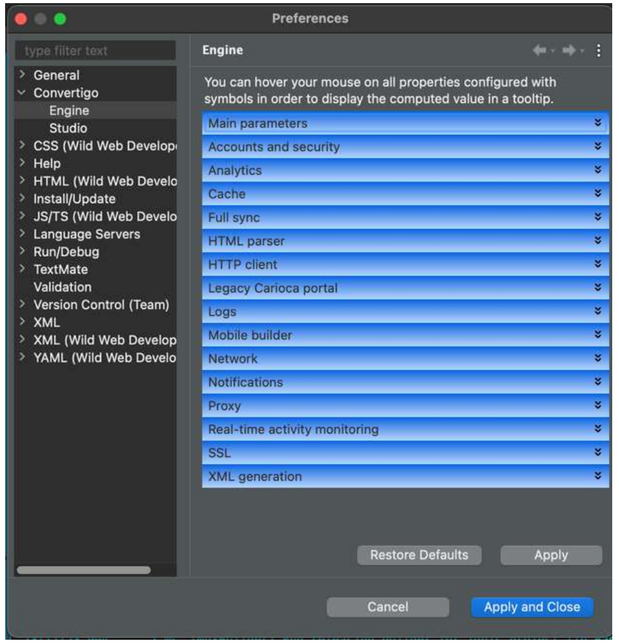
First option:

Click on **Convertigo**,

then select **Engine Preferences**

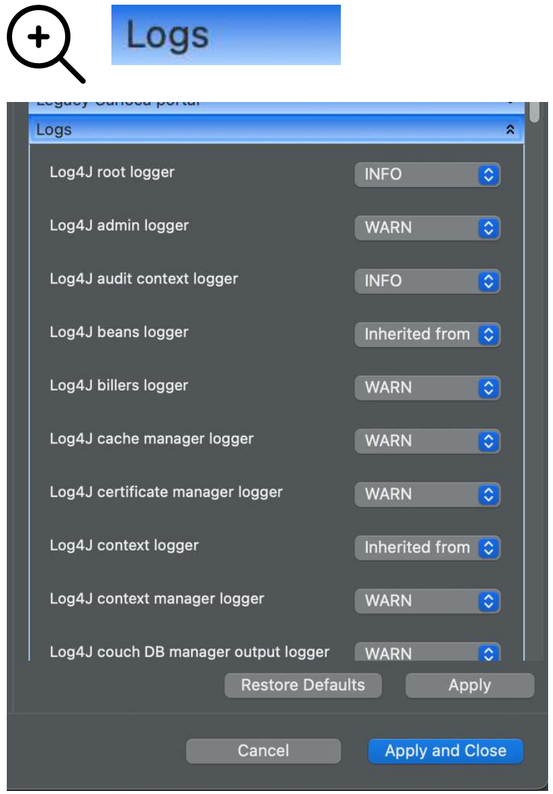


The **Preferences windows** appears.



Click on the **Logs** tab,

to open the logs settings



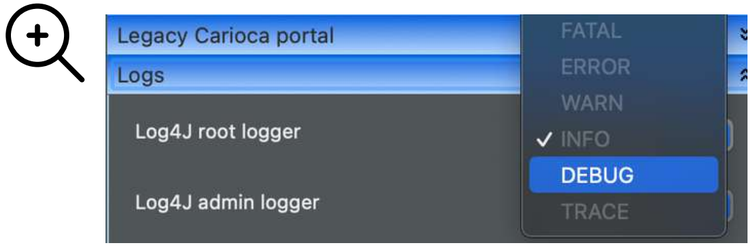
11.3 Configure Log level

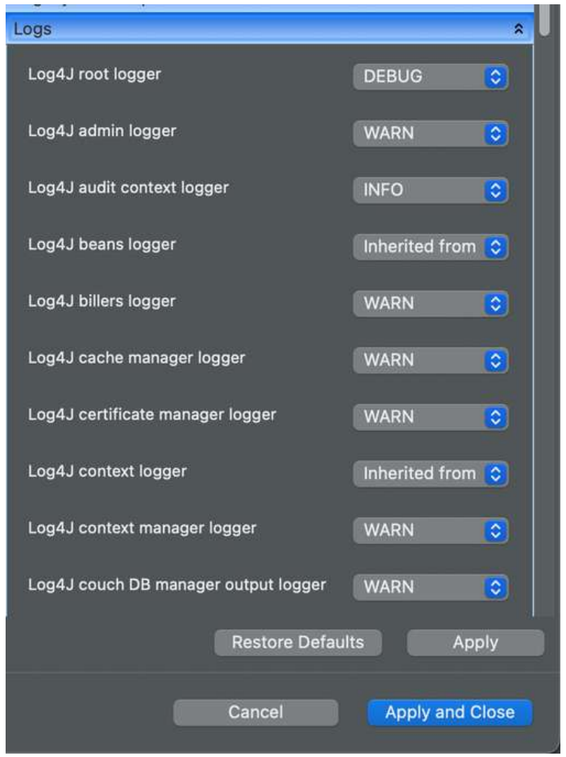
By default, the root logger is set on INFO.

Click on the select button

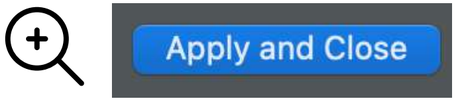
to display the different **log settings**

and select **DEBUG**.





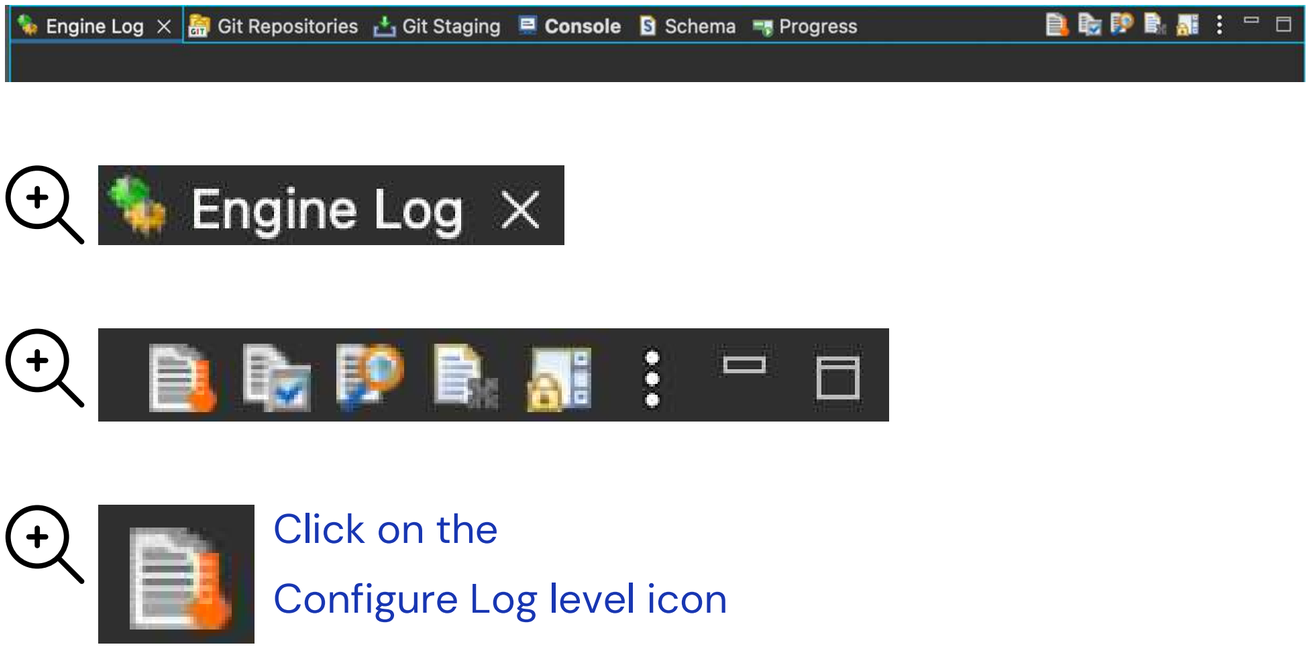
Click on Apply and Close.



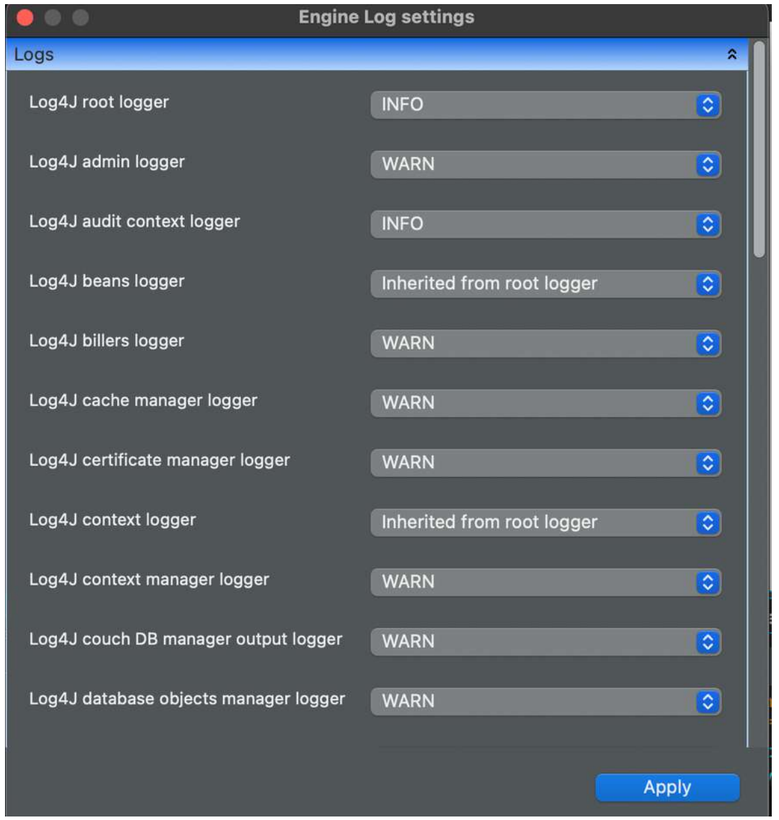
11.3 Configure Log level

Second option:

In the **Engine Log view** in the **Logs & Git panel**

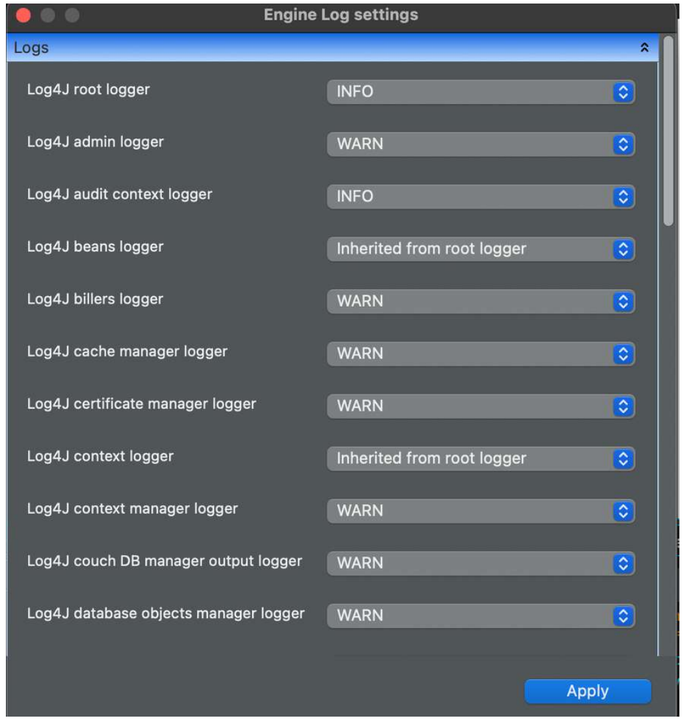


The **Engine Log settings** window appears.



11.3 Configure Log level

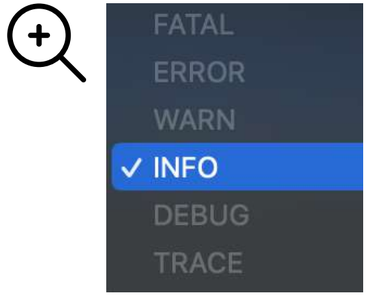
In the **Engine Log settings** window,



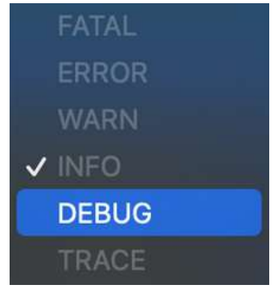
Let's look at the root logger.

Click on the select button

to display the different **log settings.**



Select **DEBUG.**

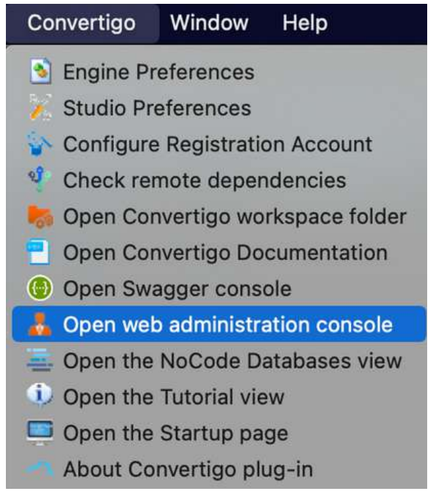


Click on Apply.

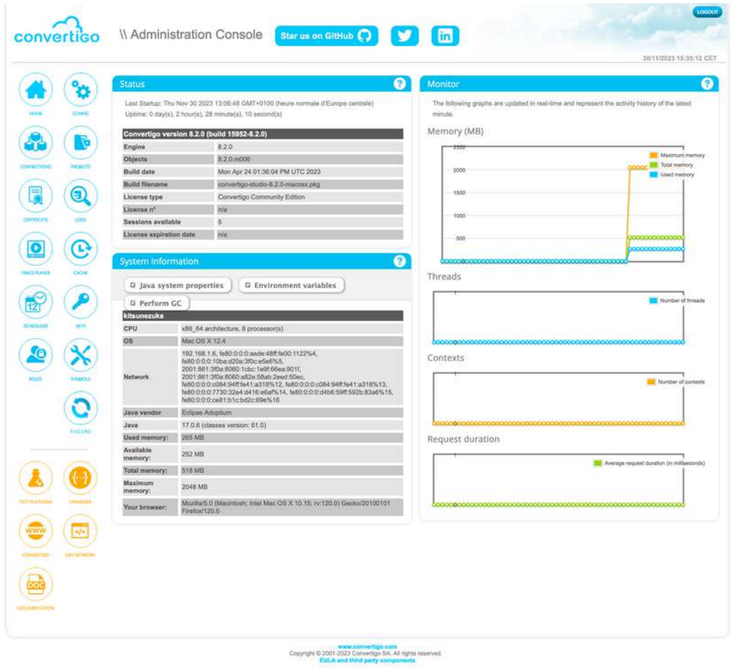
11.4 Logs in the web administration console

You can also configure and view the logs in the web administration console.

Open the **web administration console**.



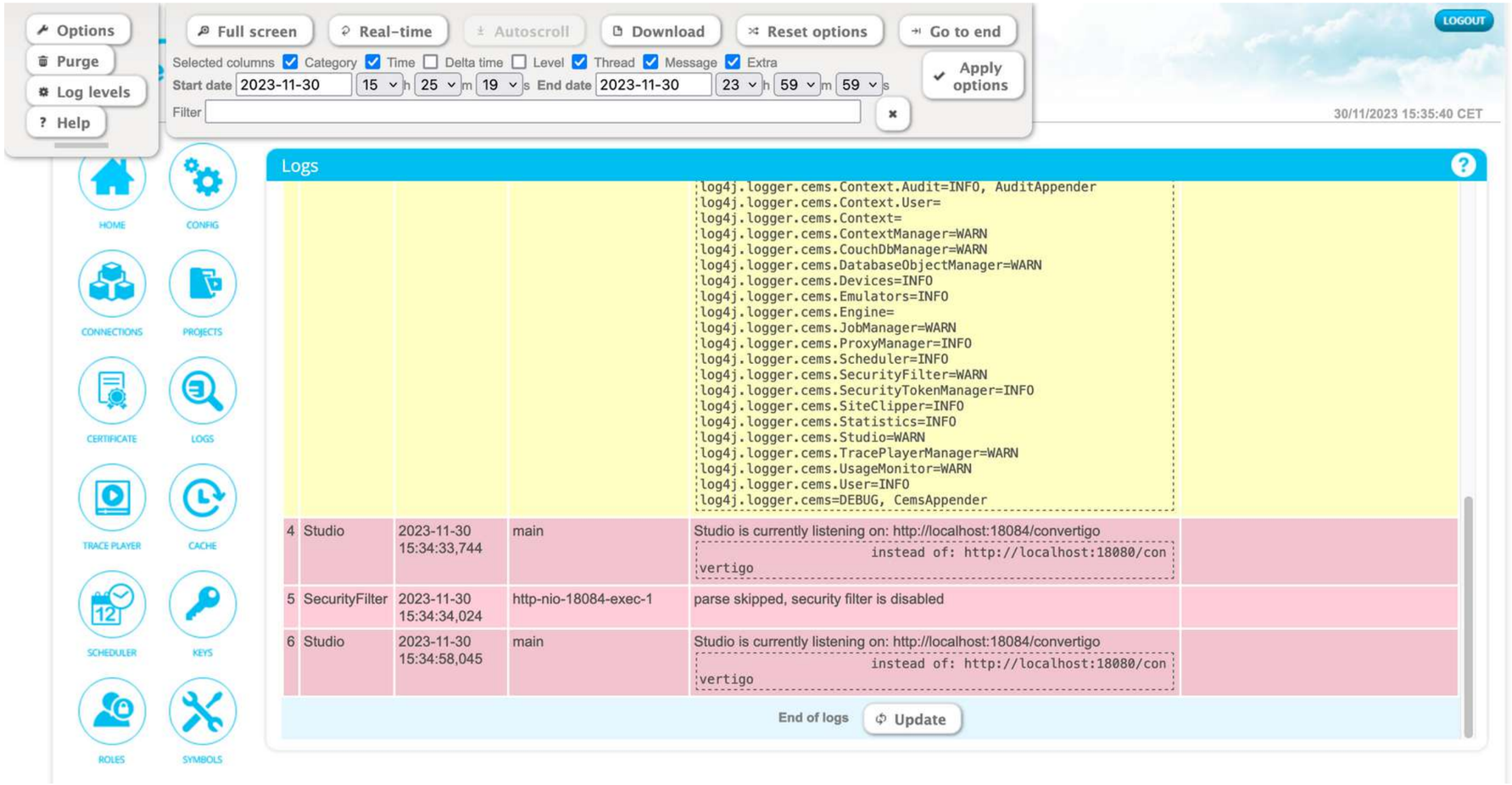
In the web administration console



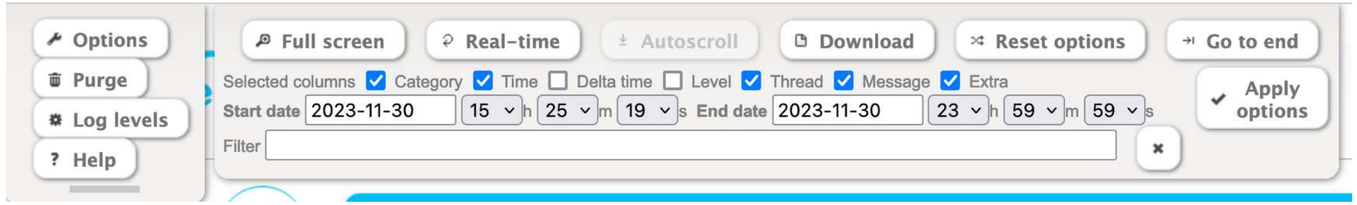
Click on the **Logs icon**.

11.4 Logs in the web administration console

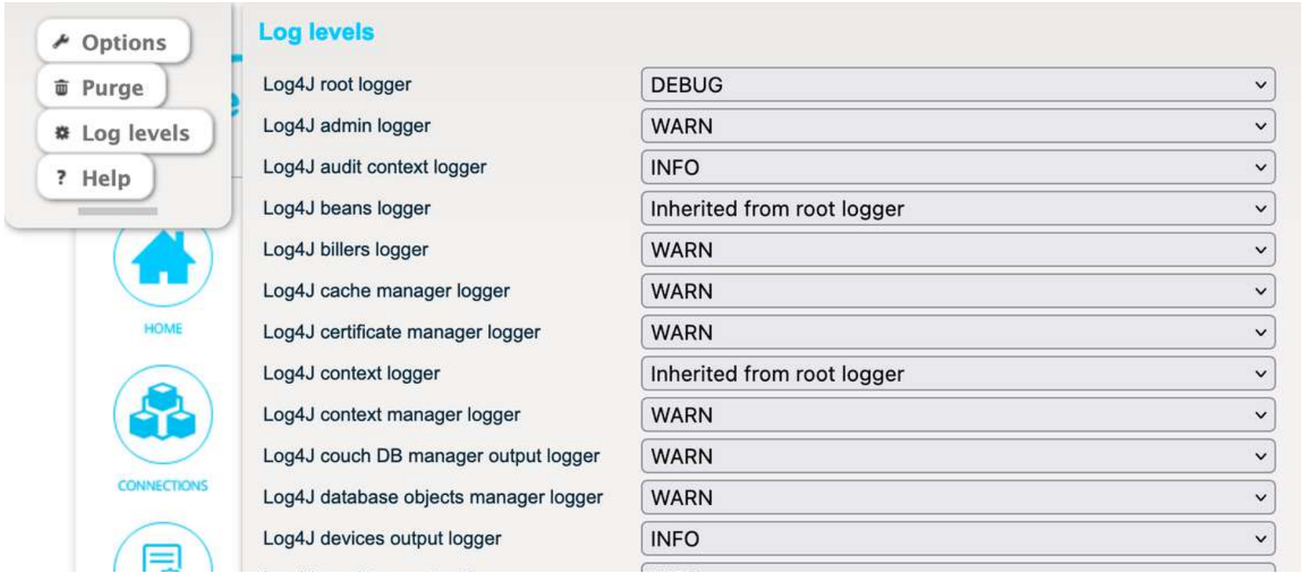
In the **Logs page of the web administration console**, you can see and filter the logs.



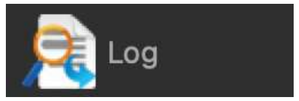
11.4 Logs in the web administration console



Click on **Log levels** to display a window where you can **configure the Log levels**.



11.5 Log step



Log – Others step

Property	Value
Base properties	
Comment	
Expression	//todo
Is active	true
Level	INFO
Logger	cems.Context

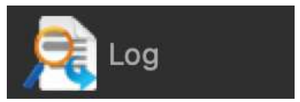
This step **outputs data in a log file**.

More accurately, it outputs

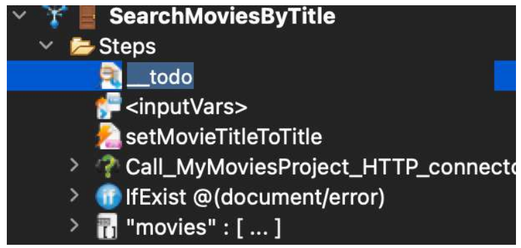
- a **message** (text string) generated from the **JavaScript expression** defined in **Expression property**
- in the **Convertigo logger** defined in the **Logger property**,
- for the **log level** defined in the **Level property**.

11.6 Using Log step in a sequence

Let's say we want to **log the inputVars** of the **sequence SearchMoviesByTitle in the Engine Log**.

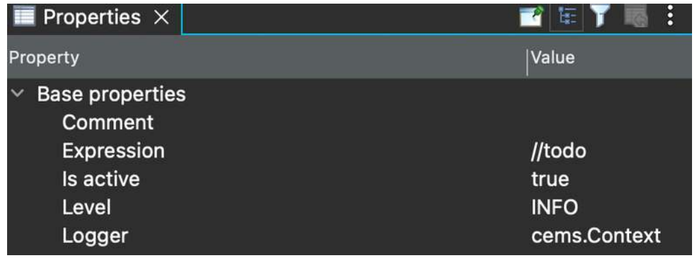


Drag and drop a **Log step** in the sequence

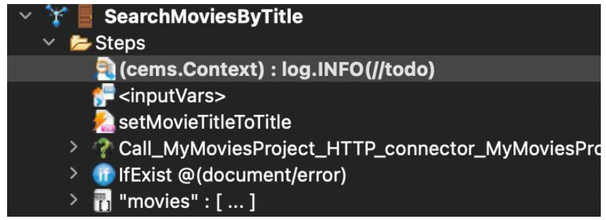


In the **Log step** properties,

- the **Level** is set to **INFO** (default)
- the **Logger** is set to **Context** (default)
- the **Expression** is yet to be defined



The log step appears like this in the treeview.



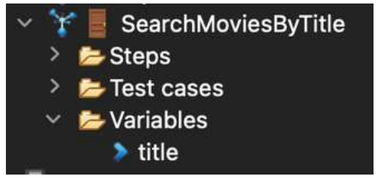
11.6 Using Log step in a sequence

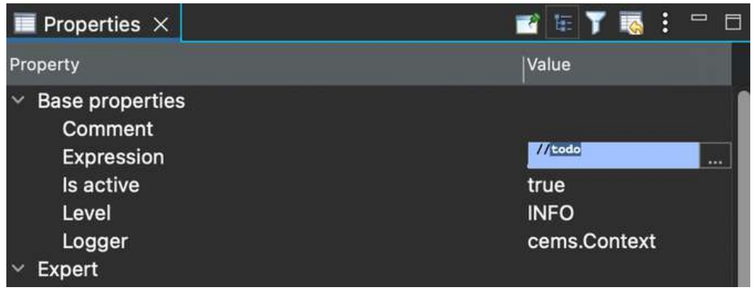
Let's **define the JavaScript expression**.

title is

the **input variable**

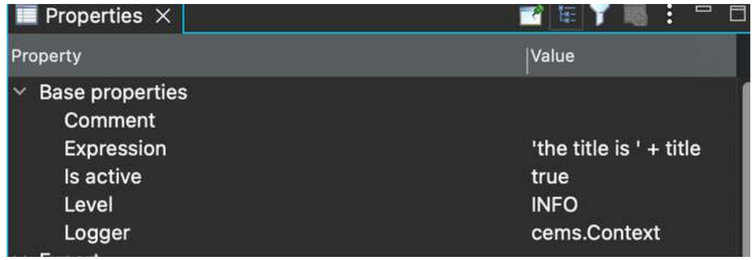
of the sequence.





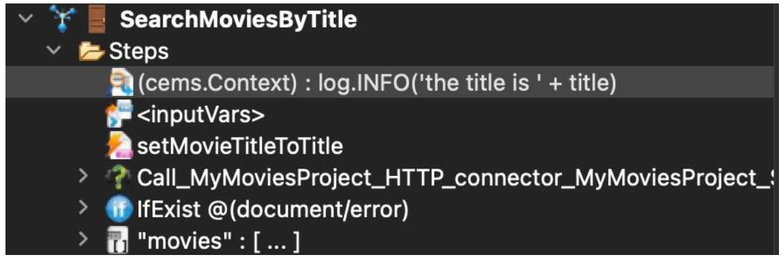
In the **Expression property**,

replace **//todo** by **'the title is ' + title**.



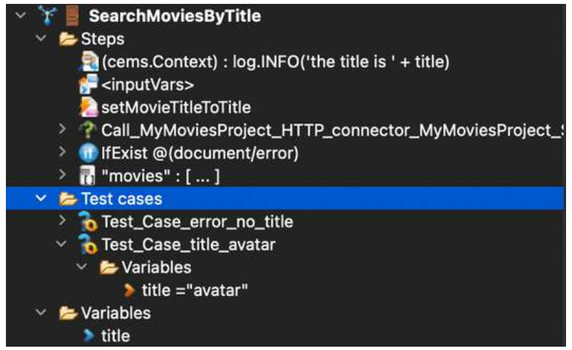
The log step appears like this

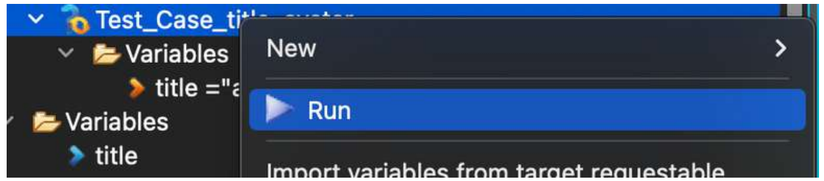
in the treeview.

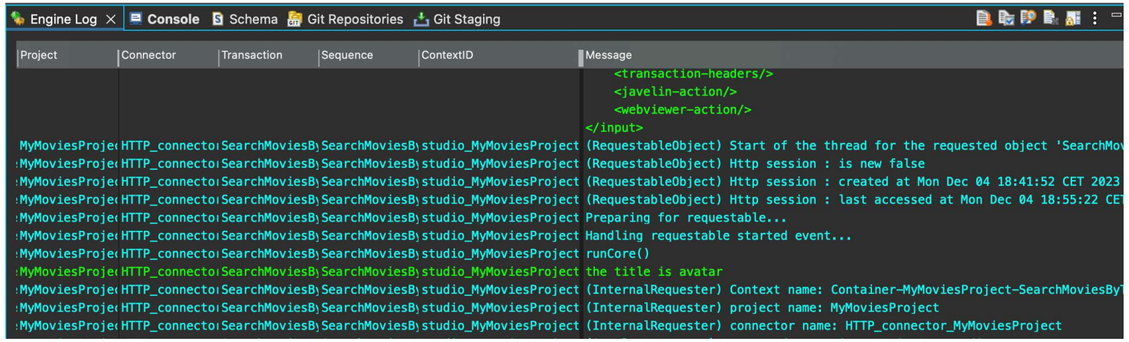


11.6 Using Log step in a sequence

Let’s run the test case with avatar as title.





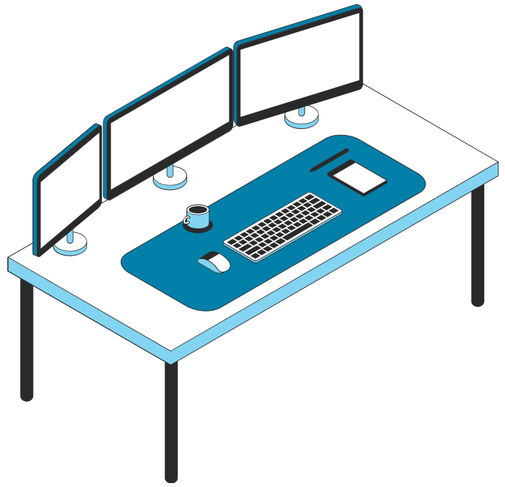


In the **Engine Log**,

the log **the title is avatar** appears in green (INFO level color).

12 - Authentication

How to manage authentication in the studio.



12.1 Accessibility property

12.2 Http session management steps

12.3 Create a Login sequence

12.4 Test the Login sequence

12.5 Add authentication to a sequence

12.1 Accessibility property

Sequences and transactions have a **property** called **Accessibility**.

It can take the following values:

- **Public**: The transaction/sequence is **runnable from everyone and everywhere**, and **visible in the Test Platform.**
- **Hidden**: It is **runnable** but **only** if you **know the execution URL**, and **not visible in the Test Platform**.
- **Private**: It is only **runnable from within the Convertigo engine** (Call Transaction/Call Sequence steps), and is **not visible in the Test Platform**.

  This value is used for tests, unfinished transactions/sequences or functionalities not to be exposed.

  Private transactions/sequences remain runnable in the Studio, for the developer to be able to test its developments.

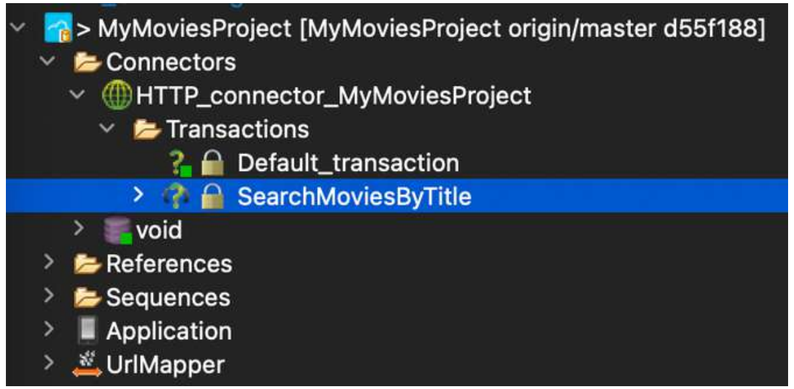
Note: In the Test Platform:

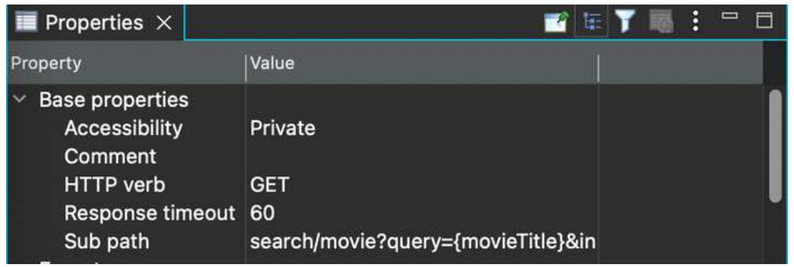
- The **administrator user** (authenticated in Administration Console or Test Platform) can **see and run all transactions / sequences, no matter what their accessibility is**.
- The **test user** (authenticated in the Test Platform or in case of anonymous access) can **see and run public transactions/sequences** and **run hidden ones if he knows their execution URL**.

12.1 Accessibility property

For a **transaction**,

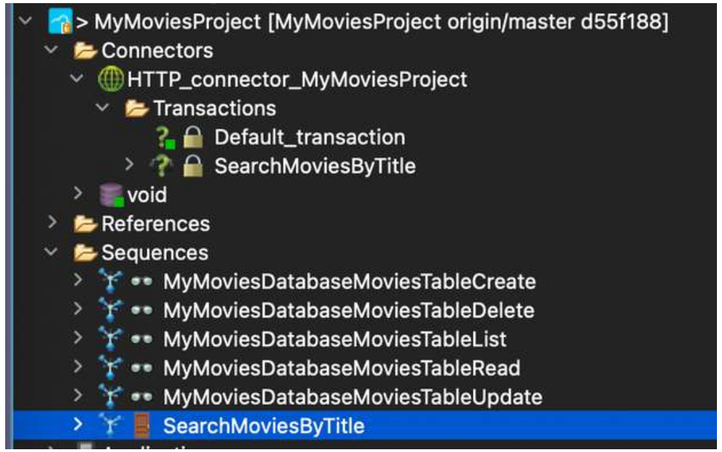
the **Accessibility property** is **private by default**.

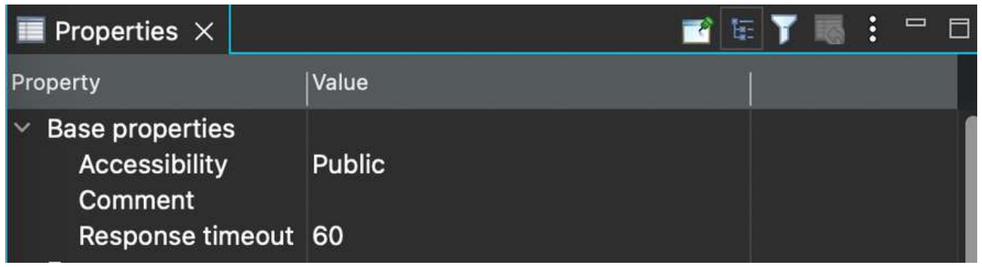




For a **sequence**,

the **Accessibility property** is **public by default**.





12.2 Http session management steps

Set authenticated user

This step sets a **user ID** as the **authenticated user ID** in the **current context/session** and sets the **current context/session** as **authenticated**.

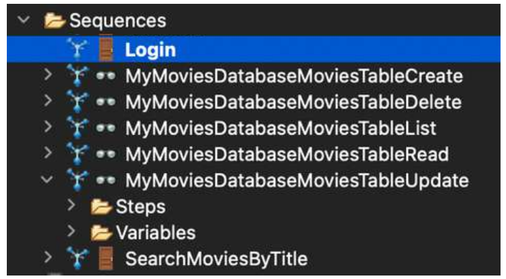
Remove authenticated user

This step allows to easily **remove a value stored in the session** using its **key**, i.e. the **variable name**.

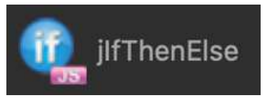
12.3 Create a Login sequence

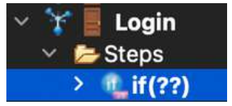
Let’s create a basic authentication in the project. We dont’ have a database with user accounts to log in so we will **simulate a user with a fake email and password** with hardcoded values.

Add a new sequence and rename it **Login**.



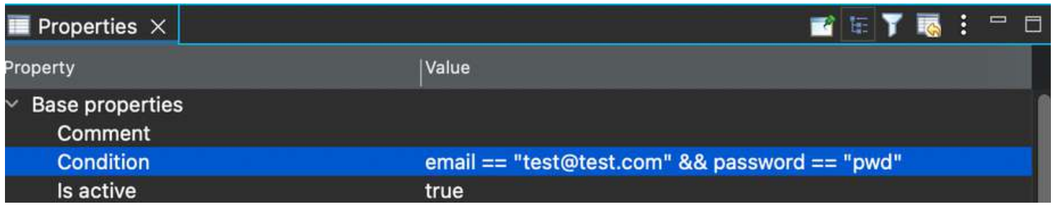
Add a **jIfThenElse step** in the Login sequence.



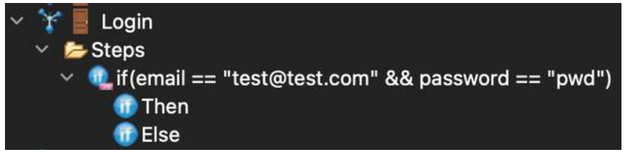


In the **Condition property** of the **jIfThenElse step**, let’s add a condition with an email and a password.

```
email == "test@test.com" && password == "pwd"
```

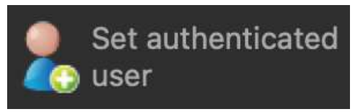


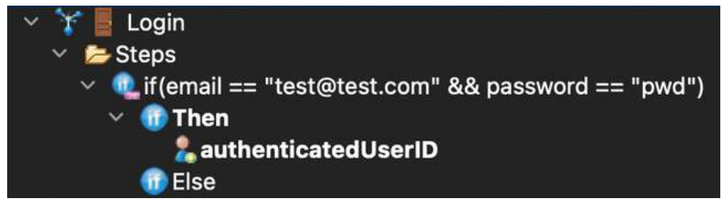
It appears like this in the treeview.



12.3 Create a Login sequence

In the Then of the jIfThenElse step, add a **Set authenticated user step**.





In the **User ID property** of the **Set authenticated user step**, enter **email** as value in **JS**.

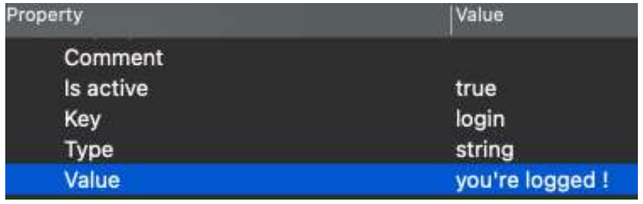
12.3 Create a Login sequence

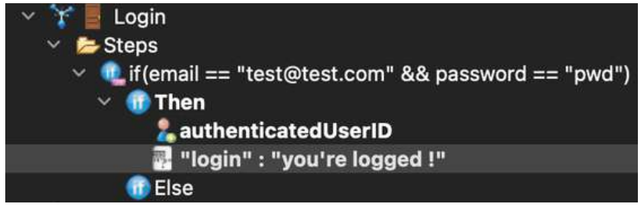
Add a **Field step** after the **Set authenticated user step**

Rename it login.

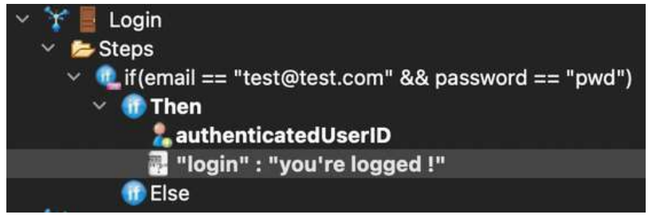
In the **Value property** of the **login field step**

Enter “You’re logged !” (or any other message) as value in text (TX)



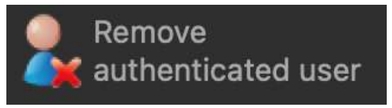


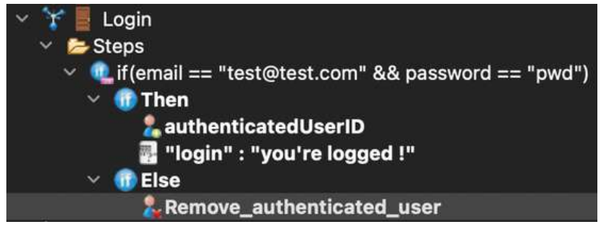
12.3 Create a Login sequence



In the **Else** of the **jIfThenElse step**,

add a **Remove authenticated user step**.

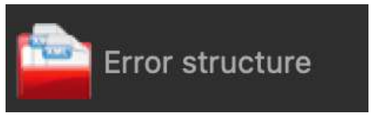




In the **Else** of the **jIfThenElse step**,

after the **Remove authenticated user step**,

add an **Error structure step**.



12.3 Create a Login sequence

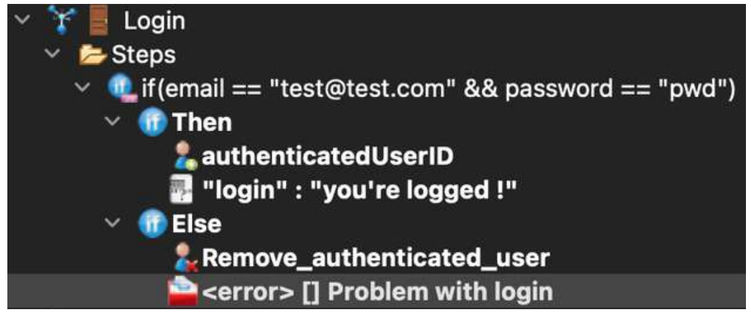
In the **Error structure step properties**

Property	Value
Code	
Comment	
Details	
Is active	true
Message	

In the **Message property**,

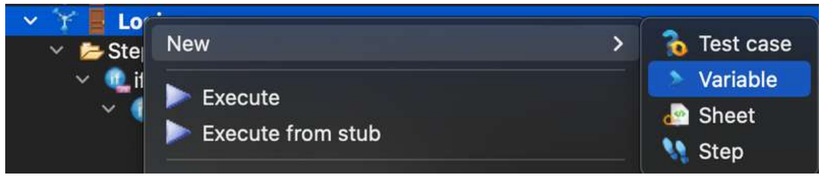
add **Problem with login** as **value**.

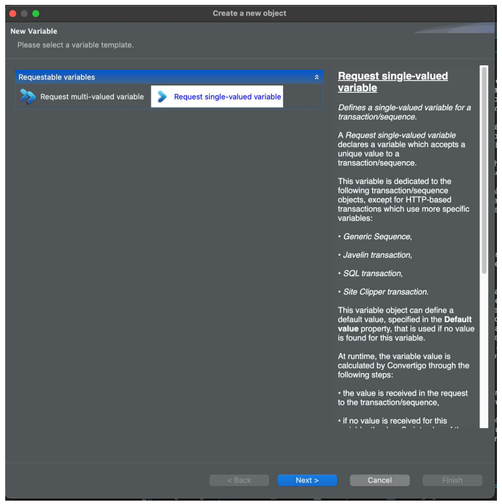
Property	Value
Code	
Comment	
Details	
Is active	true
Message	Problem with login



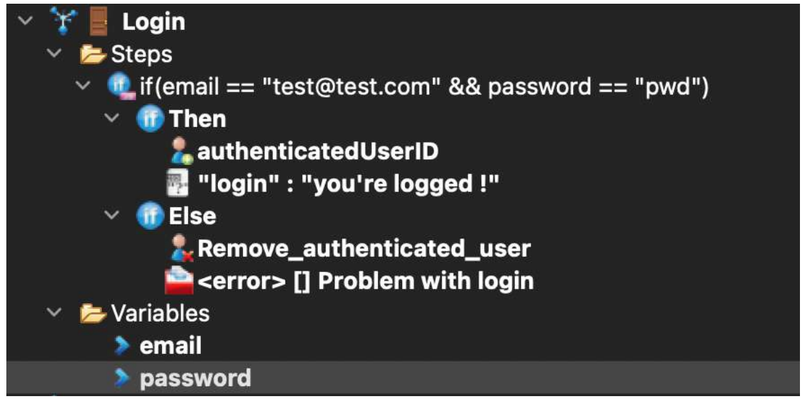
12.3 Create a Login sequence

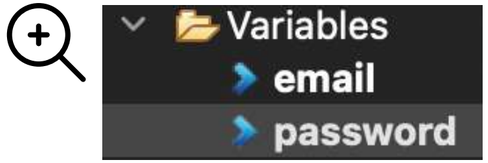
Let's **add email and password variables** to the **Login sequence**.





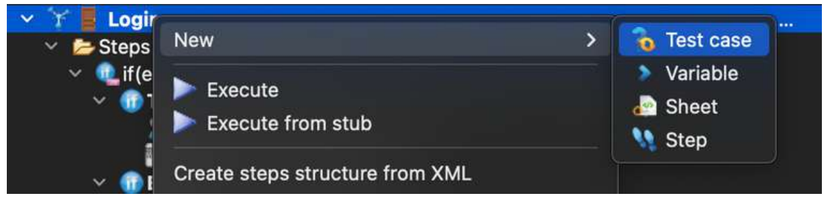
Our Login sequence with error management is completed

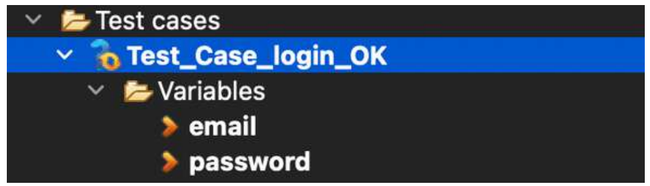




12.4 Test the Login sequence

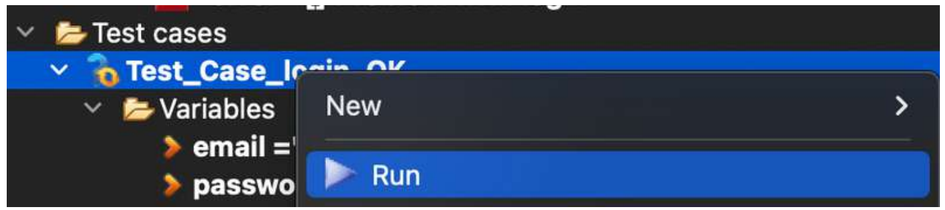
Now, let's create a test case with the right email and password values.



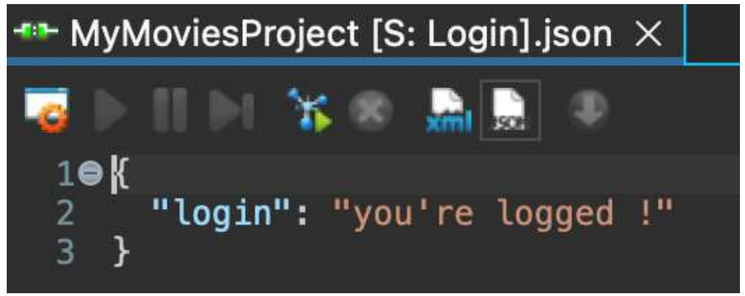


Enter the right email and password values in the test case corresponding variables

Run the test case.

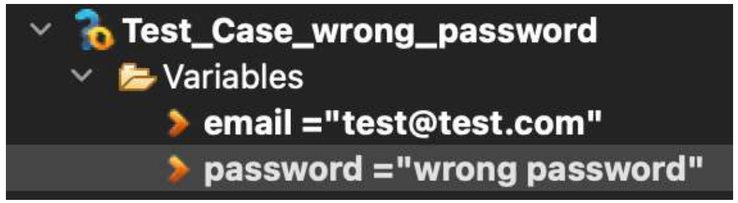


The sequence returns the login message.

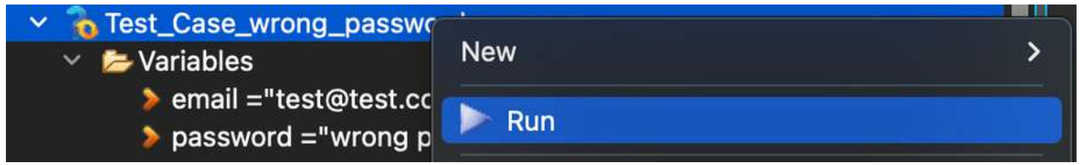


12.4 Test the Login sequence

Now, let’s create a test case with the right email and a wrong password

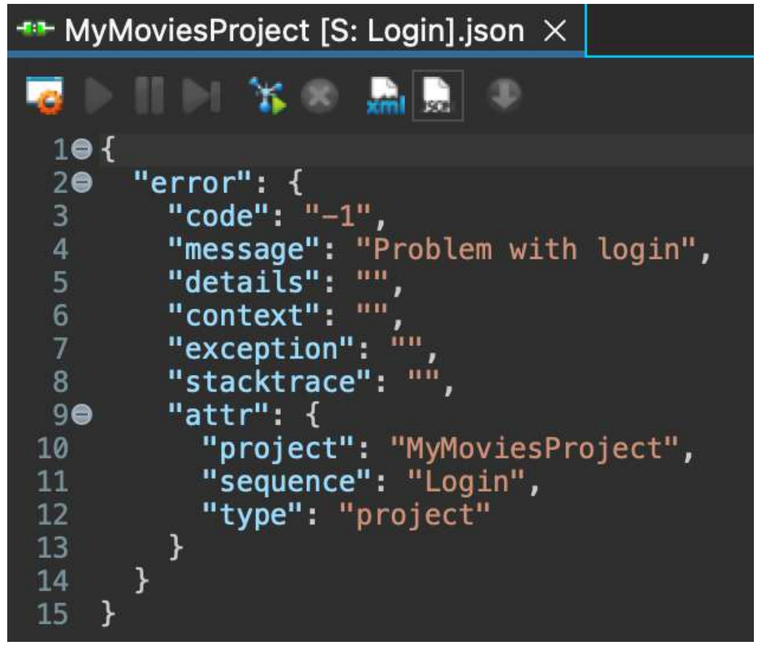


Run the test case.



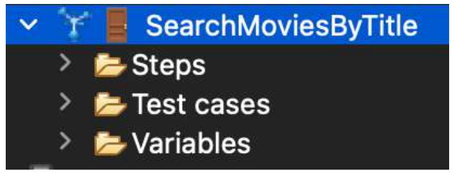
The sequence returns an error

with the error message we indicated.



12.5 Add authentication to a sequence

Let’s say we want to add authentication to the SearchMoviesByTitle sequence.

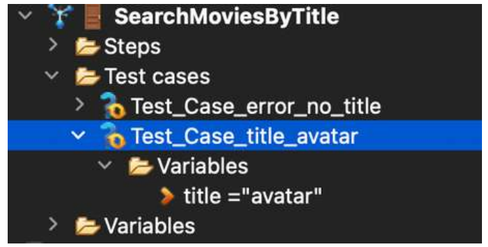


In the **sequence properties**

Base properties	
Accessibility	Public
Comment	Search a movie
Response timeout	60
Expert	
Add statistics to response	false
Authenticated context required	false

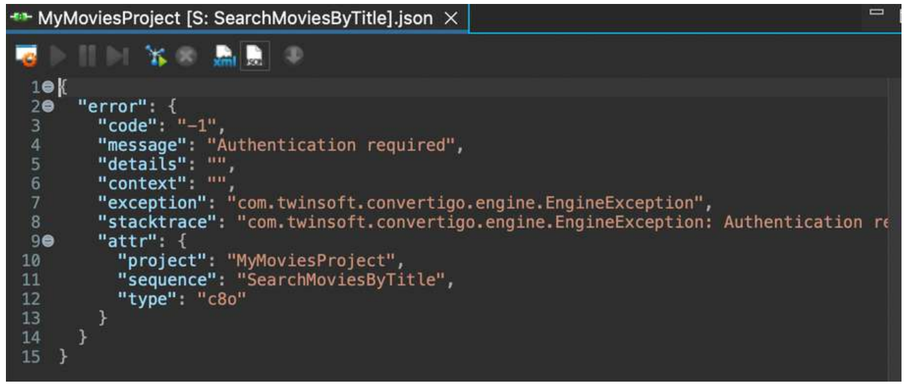
Let’s **change the value** of the **Authenticated context required property** from **false to true**.

Base properties	
Accessibility	Public
Comment	Search a movie
Response timeout	60
Expert	
Add statistics to response	false
Authenticated context required	true



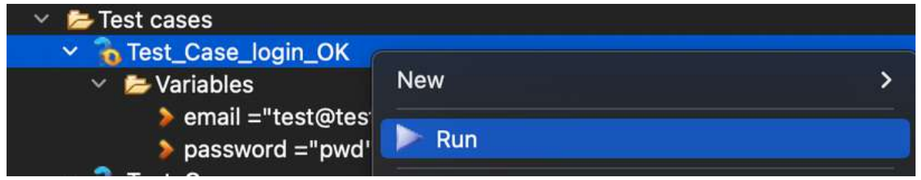
Run the test case.

The sequence returns an error with an error message : **Authentication required.**

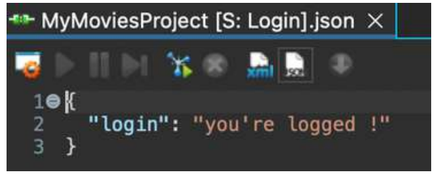


12.5 Add authentication to a sequence

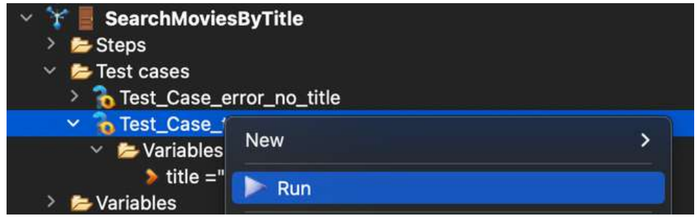
Now, let's **run the Login sequence test case before the SearchMoviesByTitle sequence test case**.



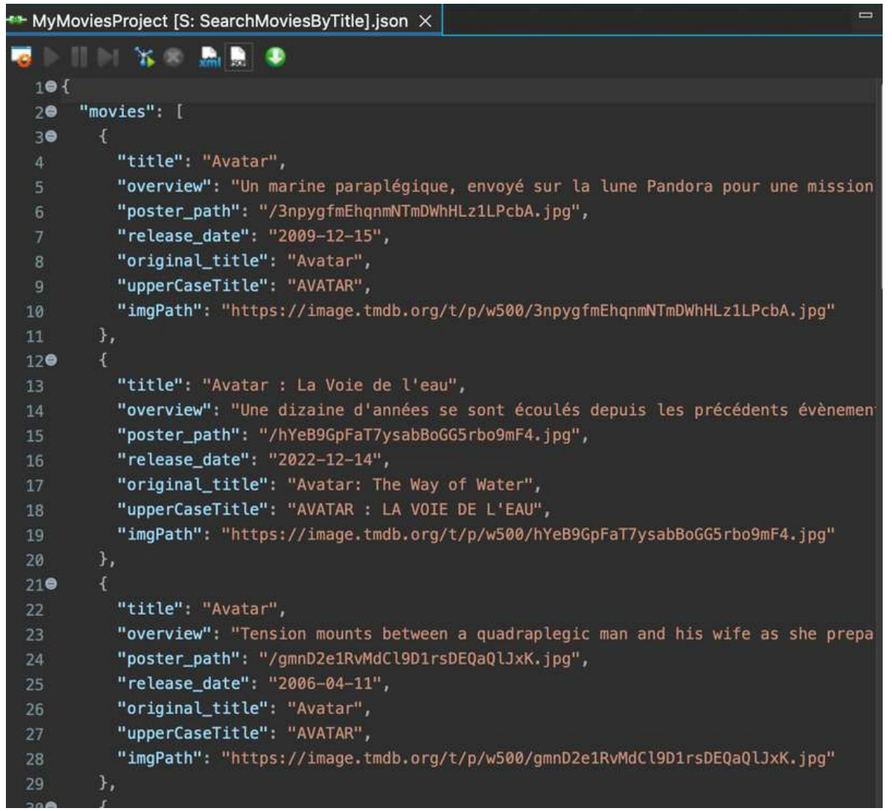
The sequence returns the login message and we are logged in.



Run the SearchMoviesByTitle sequence test case again.

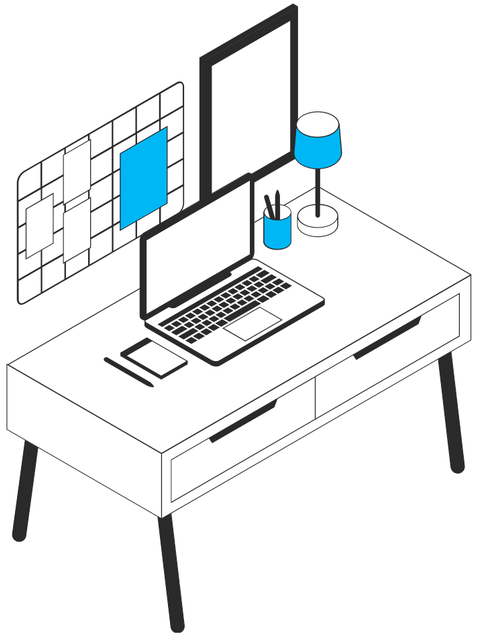


The sequence returns data from the TMDB API.



convertiGo

Appendix



A.1 Use Java classes in JS SCOPE

A.2 Generate documentation in ReadMe file

A.1 Use Java classes in JS SCOPE

In the studio, we can use Java classes in the JS Scope (thanks to the Rhino js engine).

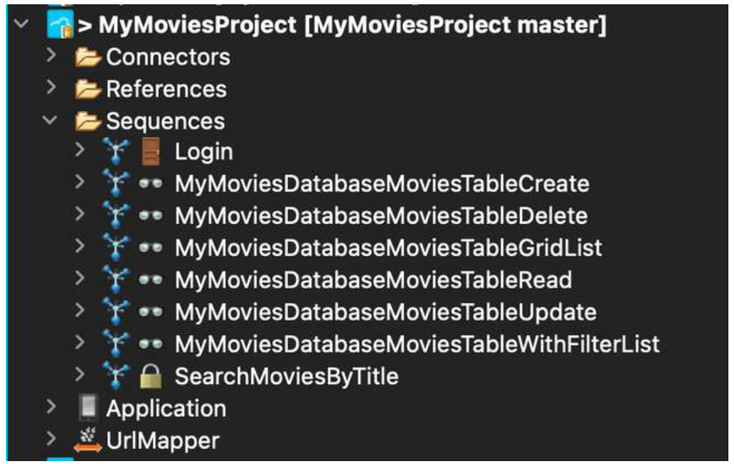
We can integrate **calls to Java code from the studio or external libraries**

into the sequence in a **Sequence JS step**.

Let's say we want to **manage dates with Java classes** in our application.

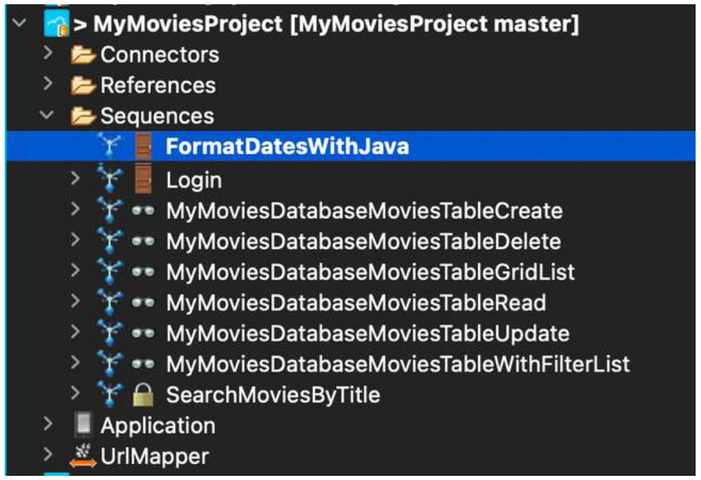
We can use

- the **Calendar class** for **handling date and time** in Java.
- the **SimpleDateFormat class** to **format Calendar instances** in a **human-readable** way.



Add a **new sequence**

and name it **FormatDatesWithJava**



A.1 Use Java classes in JS SCOPE

Add a Sequence JS step in the sequence,

and **name it FormatDatesWithJava**.

Click twice on the **Sequence JS step**

to open the **JS file** in the editor panel

Edit the JS file as following.

FormatDatesWithJava.js

```
// Import the Java Classes with JavaImporter
var JI = JavaImporter(
    Packages.java.util.Calendar,
    Packages.java.text.SimpleDateFormat
);

//Declare a variable
var formattedDate;

with (JI) {

    // Create a Calendar instance
    let calendar = Calendar.getInstance();

    // Create a SimpleDateFormat instance with desired format
    let dateFormat = new SimpleDateFormat("EEEE d MMMM yyyy");

    // Format the calendar's time
    formattedDate = dateFormat.format(calendar.getTime());
}
```

A.1 Use Java classes in JS SCOPE

convertigo

FormatDatesWithJava.js

```
// Import the Java Classes with JavaImporter
var JI = JavaImporter(
    Packages.java.util.Calendar,
    Packages.java.text.SimpleDateFormat
);

//Declare a variable
var formattedDate;

with (JI) {

    // Create a Calendar instance
    let calendar = Calendar.getInstance();

    // Create a SimpleDateFormat instance with desired format
    let dateFormat = new SimpleDateFormat("EEEE d MMMM yyyy");

    // Format the calendar's time
    formattedDate = dateFormat.format(calendar.getTime());
}

```

Here, a Calendar instance is created
and The SimpleDateFormat is used to format this date into a readable format.
In the pattern "EEEE d MMMM yyyy", we have

- the day of the week (EEEE)
- the day of the month (d)
- the month in full (MMMM)
- and the year (yyyy)

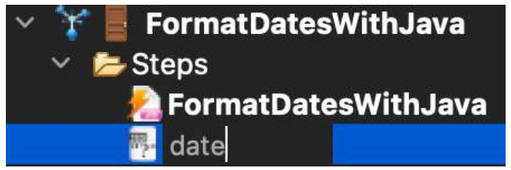
This will produce output like "Mercredi 15 février 2023".
You can adjust the pattern in SimpleDateFormat to suit your preferred date format.

A.1 Use Java classes in JS SCOPE

Add a Field step

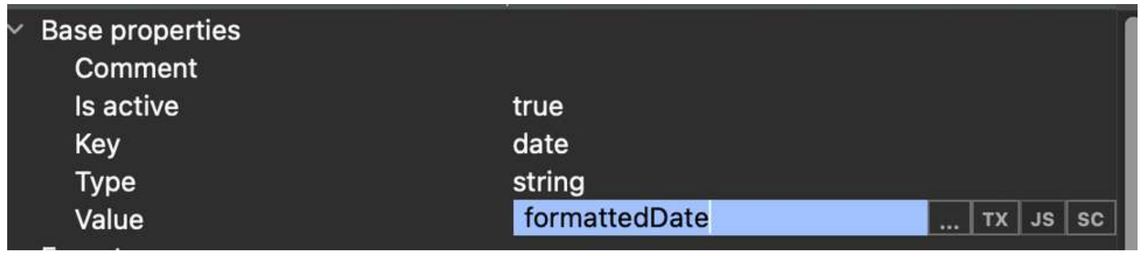
after the Sequence JS step.

Name it **date**.



in the **Value property** of the Field step properties,

enter **formattedDate as value in JS**.



A.1 Use Java classes in JS SCOPE

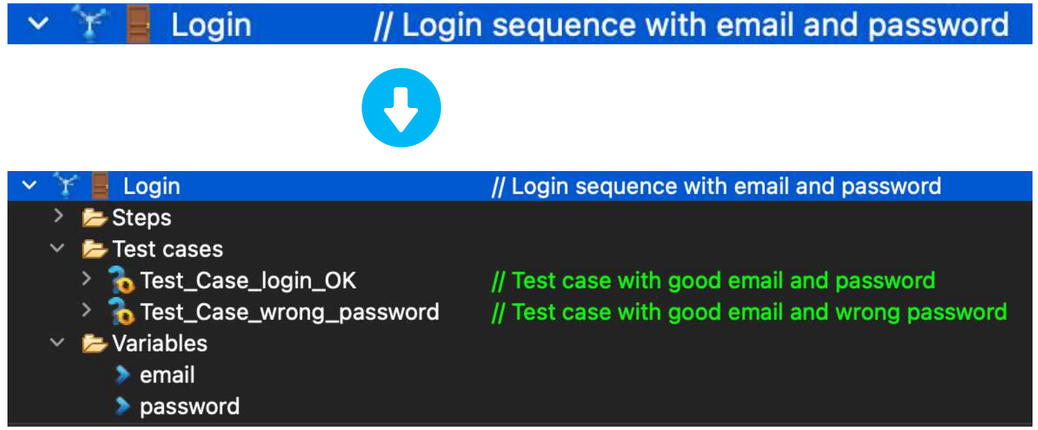
Execute the Sequence.

The sequence returns the formatted date.

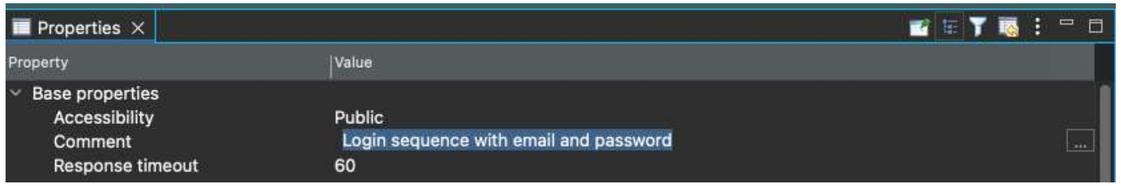
A.2 Generate documentation in ReadMe file

In the studio, we can automatically generate a documentation in the ReadMe.md file.

You can edit the comment in the treeview.



Or in the comment property.

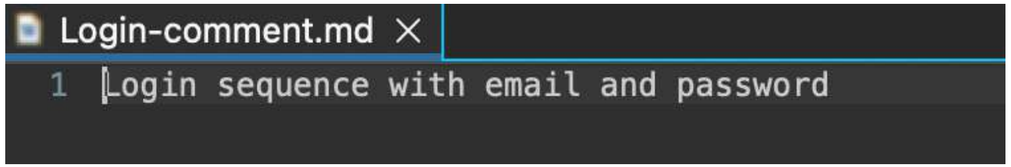


You can also click on this icon

to open the markdown file of the comment

and write a more detailled comment,

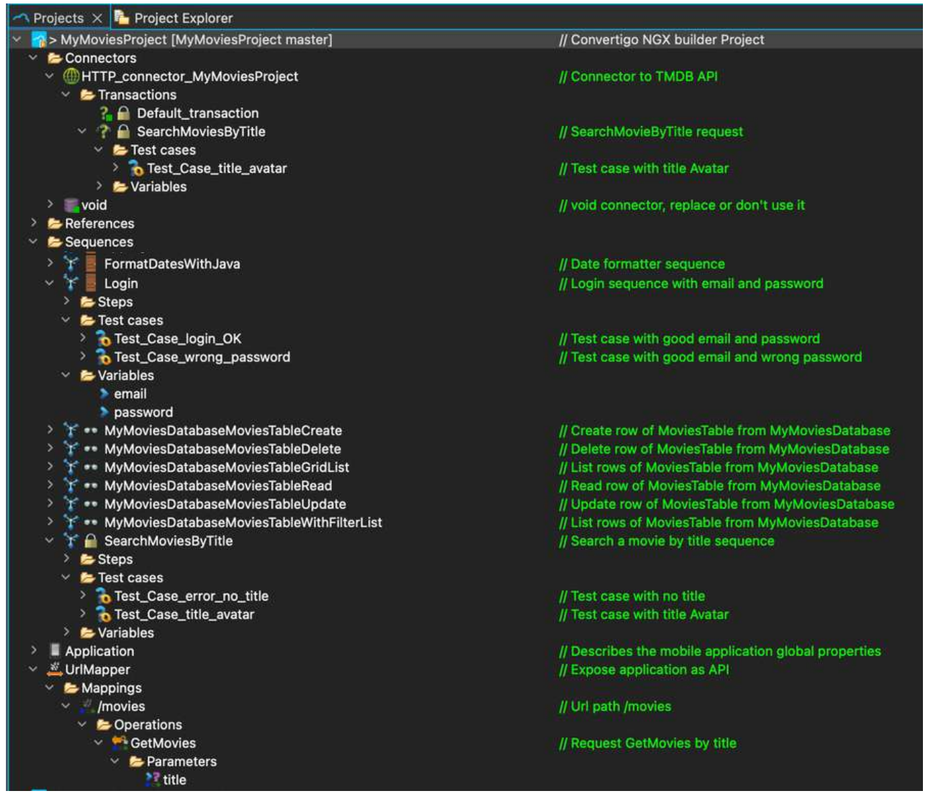
if needed.

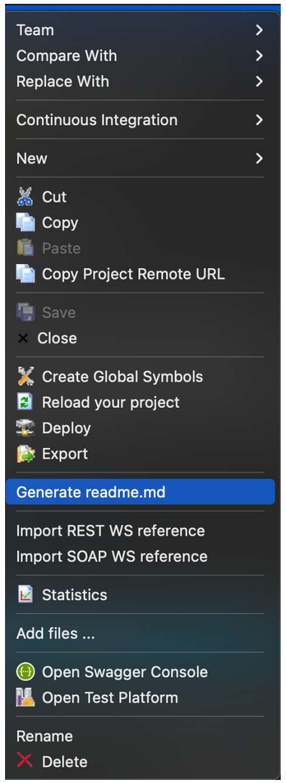


A.2 Generate documentation in ReadMe

You can write comments on every step

in the treeview.





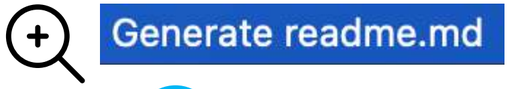
To **enable the automatic generation**

of the **documentation** in the **ReadMe.md file**,

right-click on the **project name** in the **Projects view**.

Select **Generate readme.md**

to open the **Readme generation window**.

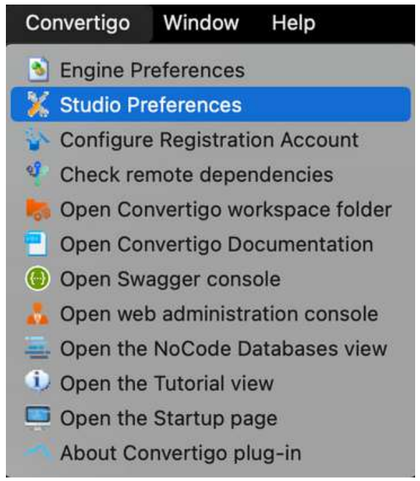


Click on **Always** to confirm.

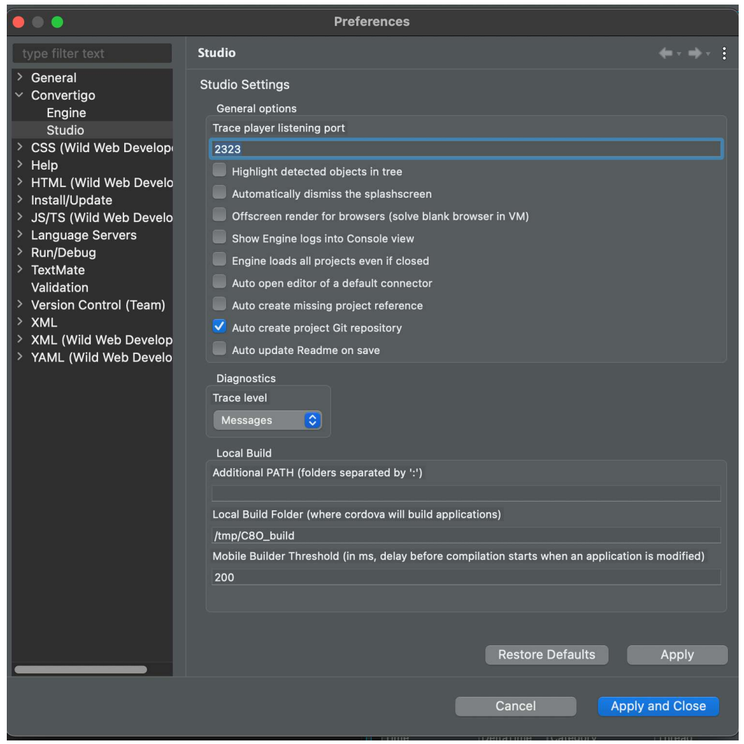
A.2 Generate documentation in ReadMe

You can also use **Studio Preferences** to enable the automatic generation of the documentation in the ReadMe.md file.

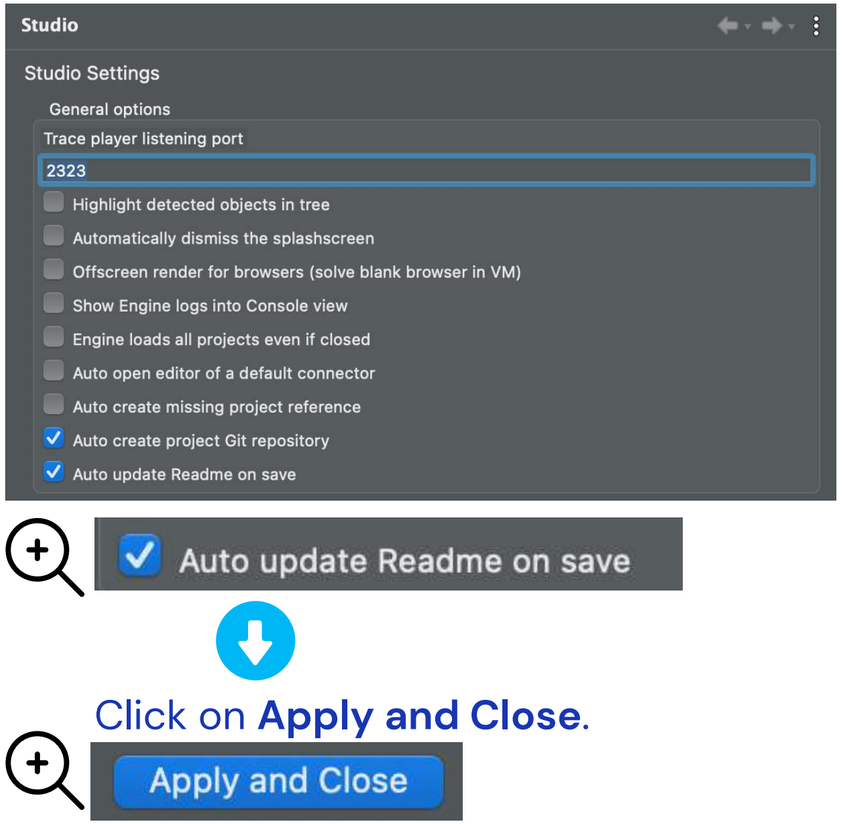
Each time the project is saved, the file is updated.



In the Preferences window



Click on **Auto update Readme on save**.

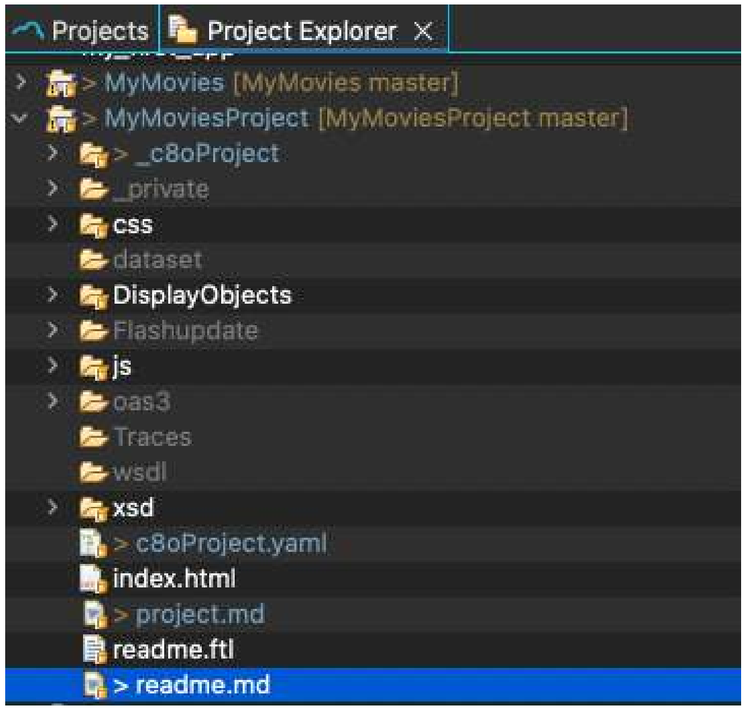


Click on **Apply and Close**.

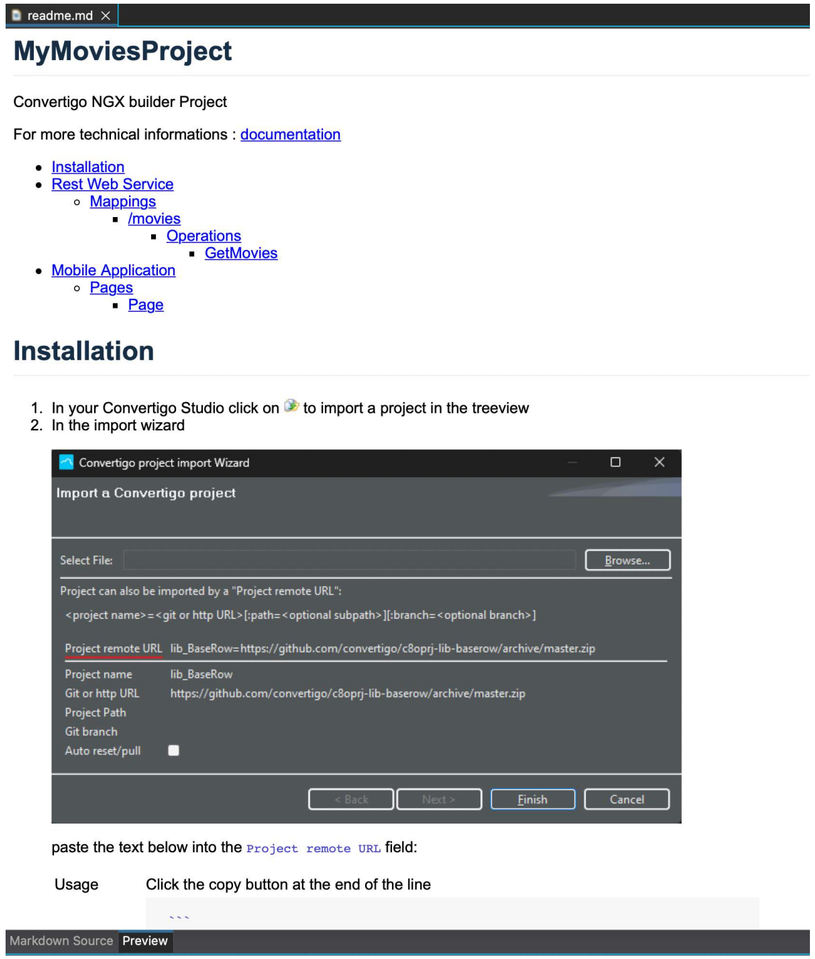
A.2 Generate documentation in ReadMe

In the **Project Explorer view**,

click twice on the **readme.md file** to open it.



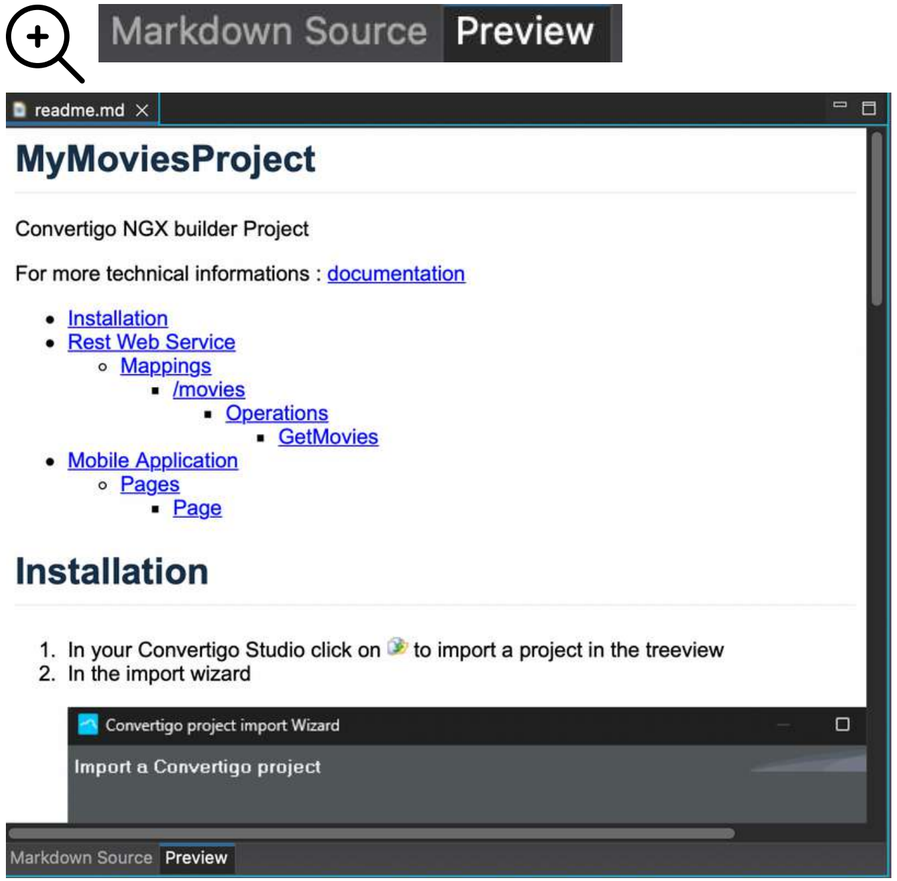
The content of the file is automatically generated



A.2 Generate documentation in ReadMe

The ReadMe.md file is **displayed in Preview by default**. You can switch it to **Markdown** to **edit it in markdown**.

Preview view



Markdown Source view