DEI
DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
TÉCNICO LISBOA

Tutorial BPMN 2.0
modelling, developing, deploying, installing, testing and operating using Camunda

Instituto Superior Técnico
Universidade de Lisboa
Integração Empresarial / Enterprise Integration

The goal of this document is to present the details related with: Installation and execution of the Camunda platform, Designing the BPMN 2.0 models using the main constructors of the language, Deploy those previous modelled BPMN 2.0 models, Testing the models, and Operating the platform.

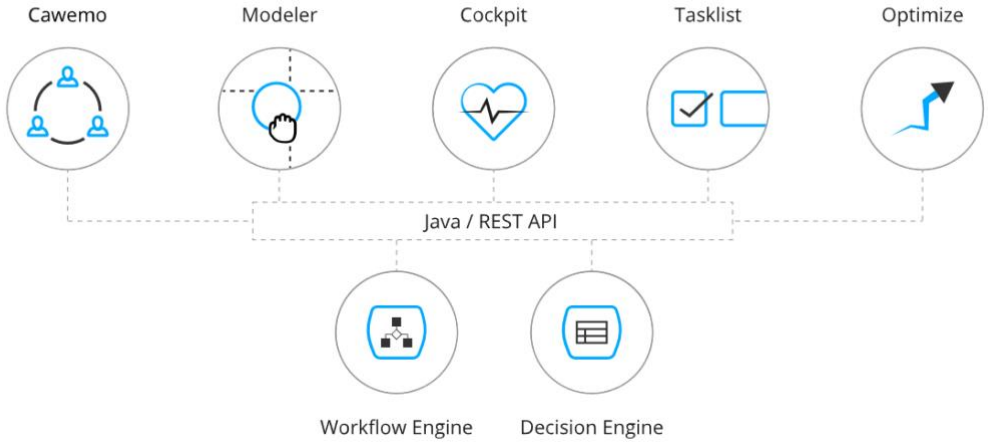
The examples address the following aspects: how to integrate a BPMN process model with an external worker written in JAVA, how to add a user task to the BPMN process model, how to add gateways to the BPMN process model, how to implement DMN rules in the BPMN process model, and how to invoke a cloud REST API.

The following contents is presented in this document.

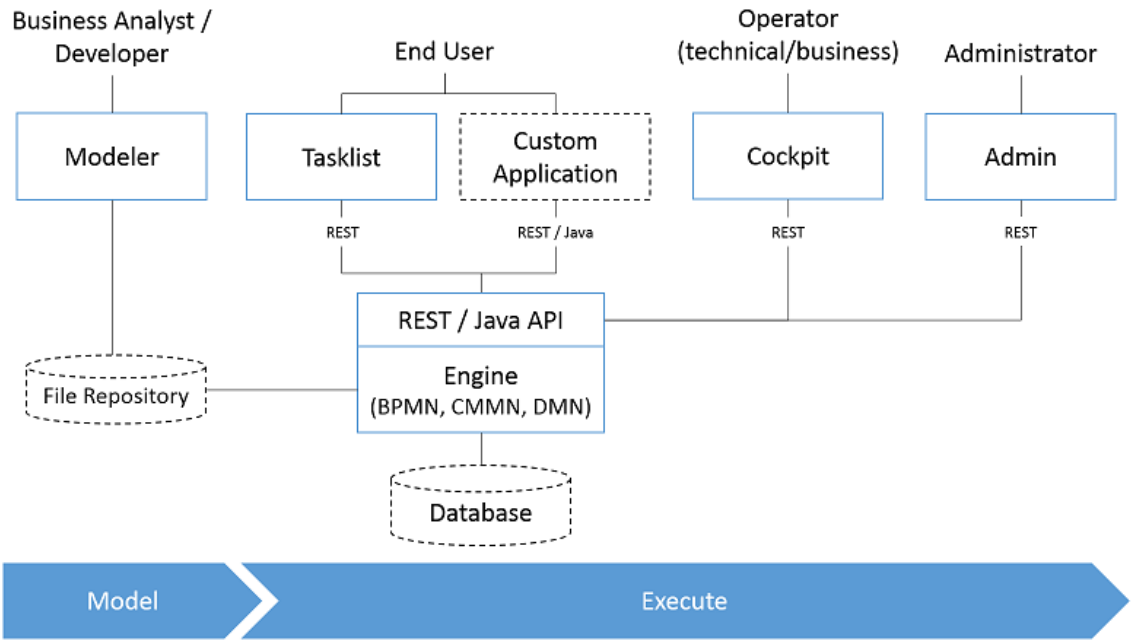
Contents

A. Camunda brief overview

Camunda encompasses the following software modules.



The Community edition that we are going to use have a basic cockpit and optimize component is also not available. The main stakeholders involved in Camunda are the following:



B. Installation and operation of Camunda Engine in AWS EC2

C.1. Create an EC2 new instance of the type: Amazon Linux 2 AMI (HVM), SSD Volume Type. The exact versions may change with time. And define the inbound rules to allow the 8080 and 22 accessed from anywhere.

C.2. Install the most recent version of camunda-modeler (**version 7 recommended**) using the download public available at: https://camunda.com/download/modeler/

C.3. Access the new EC2 instance and execute the following commands:

```
sudo yum update -y
sudo yum install -y docker
sudo service docker start
sudo usermod -aG docker ec2-user
docker ps
exit
```

C.4. Then, access again your EC2 instance and install and execute the Camunda engine with the following command:

```
docker pull camunda/camunda-bpm-platform:latest
docker run -d --name camunda -p 8080:8080 camunda/camunda-bpm-platform:latest
```

Test the installed Camunda engine opening the following URL in your browser (the default user/password is **demo**/demo):

```
# open browser with url: http://YOUR_AWS_EC2_INSTANCE:8080/camunda-welcome/index.html
```

Hint 1: Later if you would like to list all the available container use the following command:

```
PS C:\Users\Sérgio Guerreiro> docker container ls --all

CONTAINER ID        IMAGE                                  COMMAND                  CREATED             STATUS
PORTS               NAMES
44509250aa0c        camunda/camunda-bpm-platform:latest    "/sbin/tini -- ./cam…"   11 days ago         Exited
(143) 1 second ago                        camunda
```

Hint 2: Later if you would like to stop the container use the following command:

```
docker container stop <containerID_hash>
```

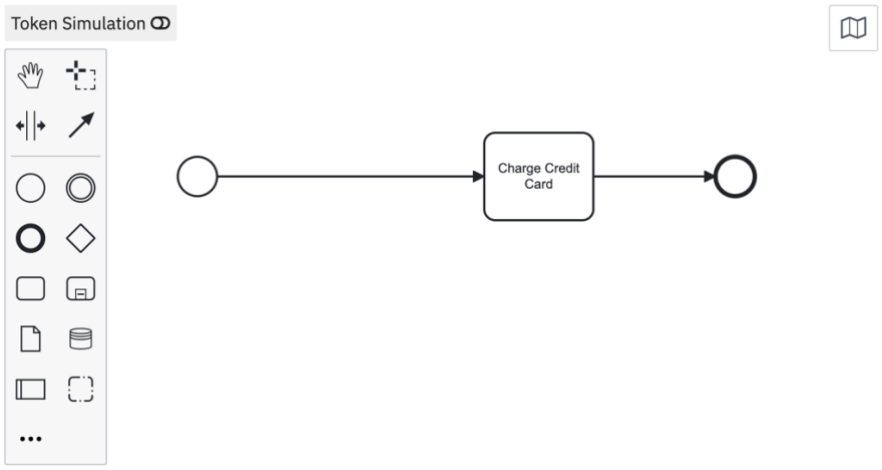
Hint 3: Later if you would like to restart the container use the following command:

```
docker container start <containerID_hash>
```

C. Add a User Task to the Process

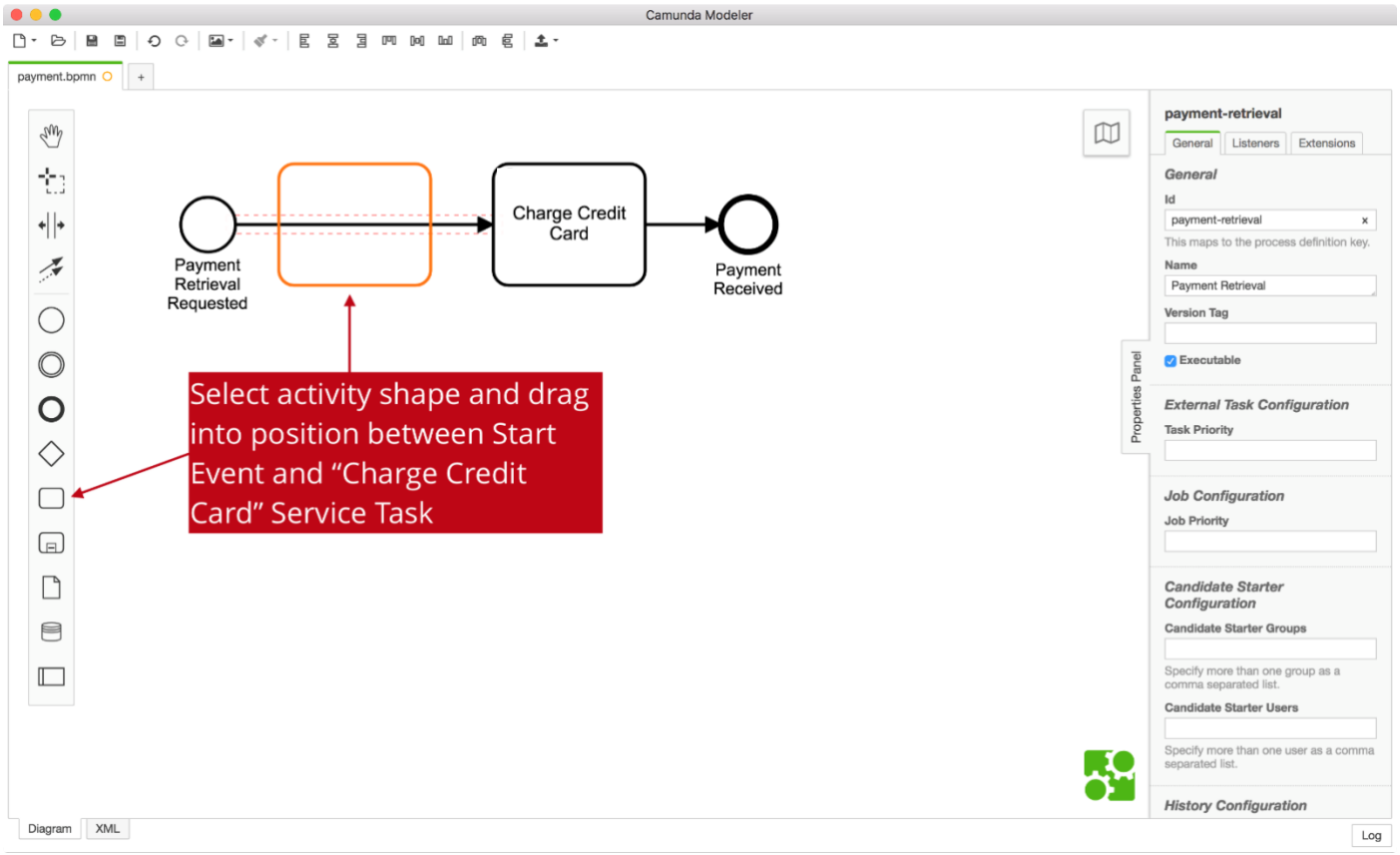
This example is based in the Camunda documentation available at https://docs.camunda.org/get-started/quick-start/user-task/ . The goal of the example is to show how to add a human task in the middle of a process execution. After deploying, go to the tasklist (all tasks) identify the "Approve payment", select the claim, and check approved and then complete. Then, check that instances and tasklist are not available anymore.

D.1. Add an abstract task with the name "Charge Credit Card" between a start and an end event.

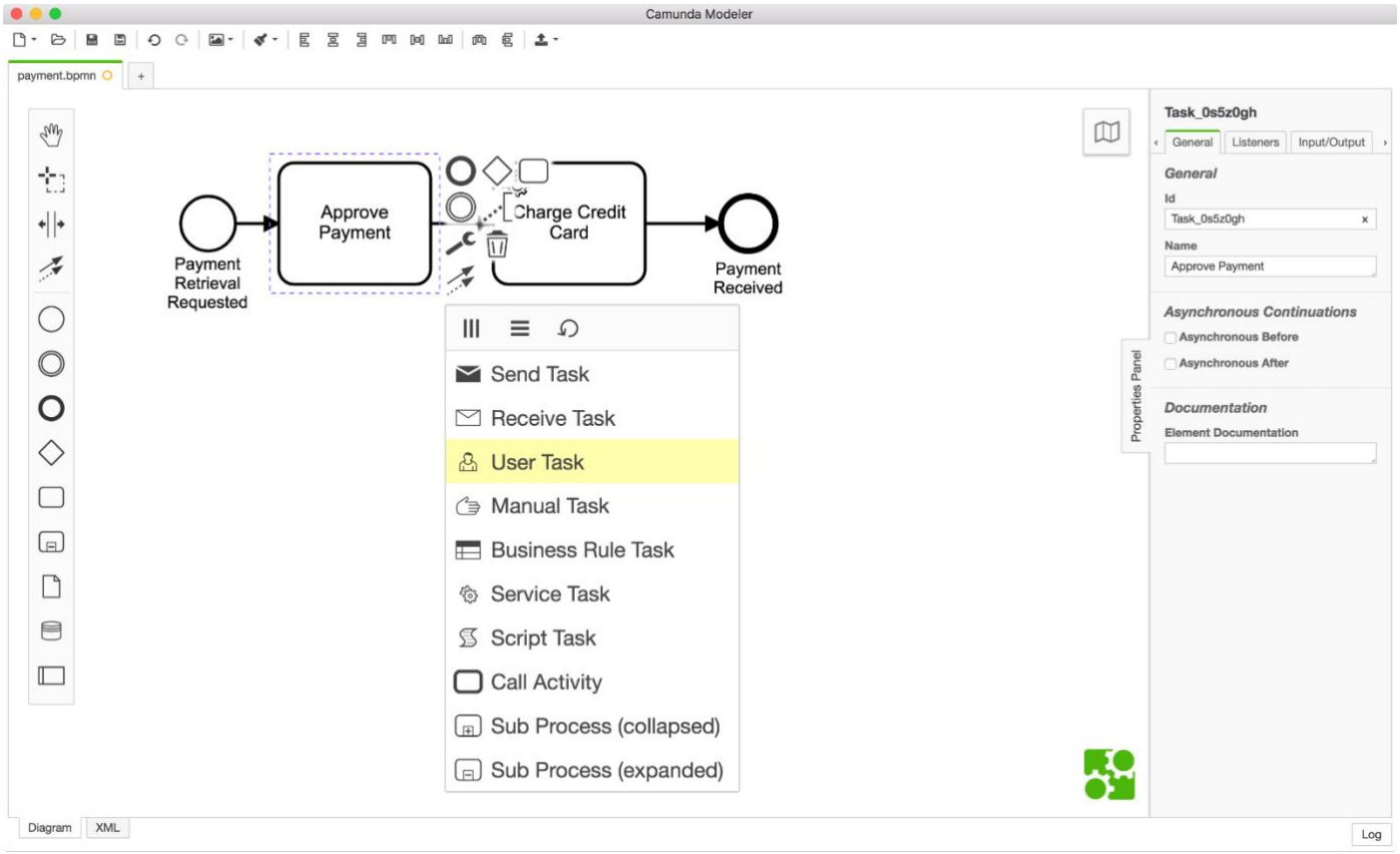


D.2. Add a User Task

Now, we want to modify our process so that we can involve humans. To do so, open the process in the Camunda Modeler. Next, from the Modeler's left-hand menu, select the activity shape (rectangle) and drag it into position between the Start Event and the "Charge Credit Card" Service Task. Name it *Approve Payment*.



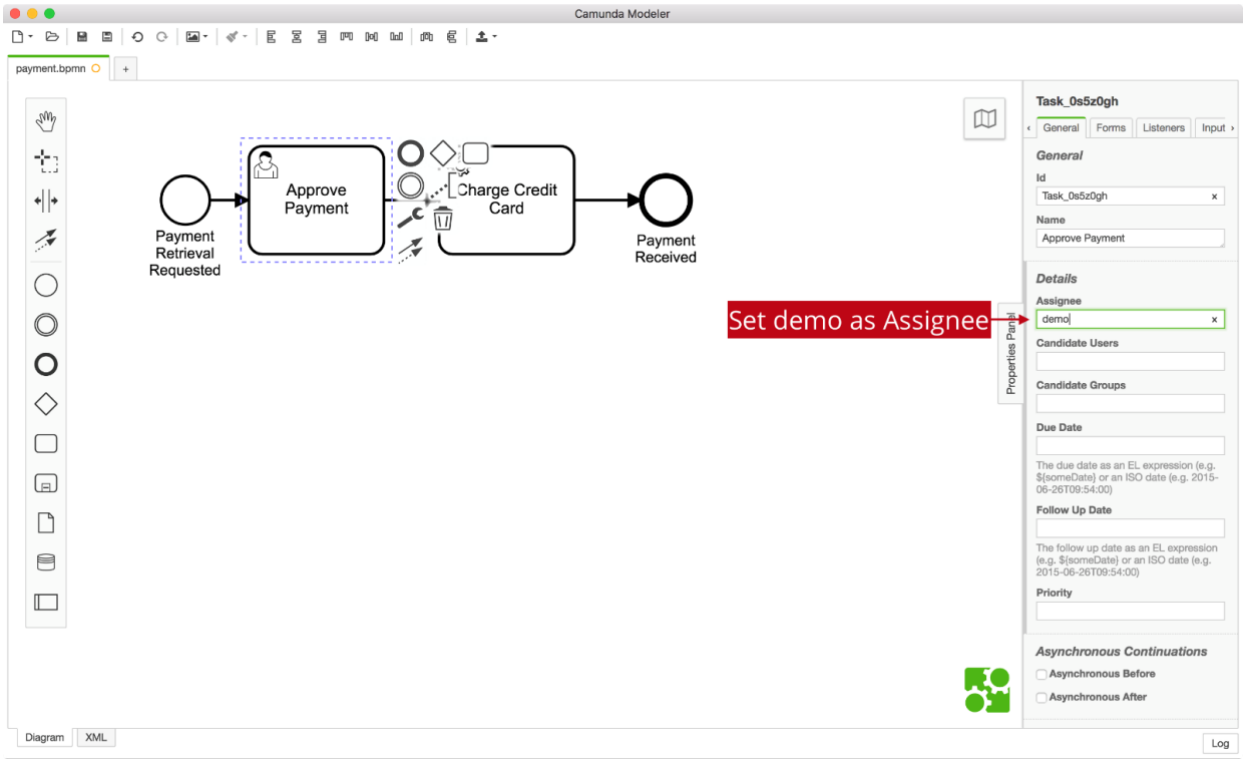
Change the activity type to *User Task* by clicking on it and using the wrench button menu.



D.3. Configure a User Task

Next, open the properties view. If the properties view is not already visible, click on the "Properties Panel" label on the right-hand side of the Modeler canvas.

Select the User Task on the canvas. This will update the selection in the properties view. Scroll to the property named `Assignee`. Type *demo*.



D.4. Configure a basic form in the User Task

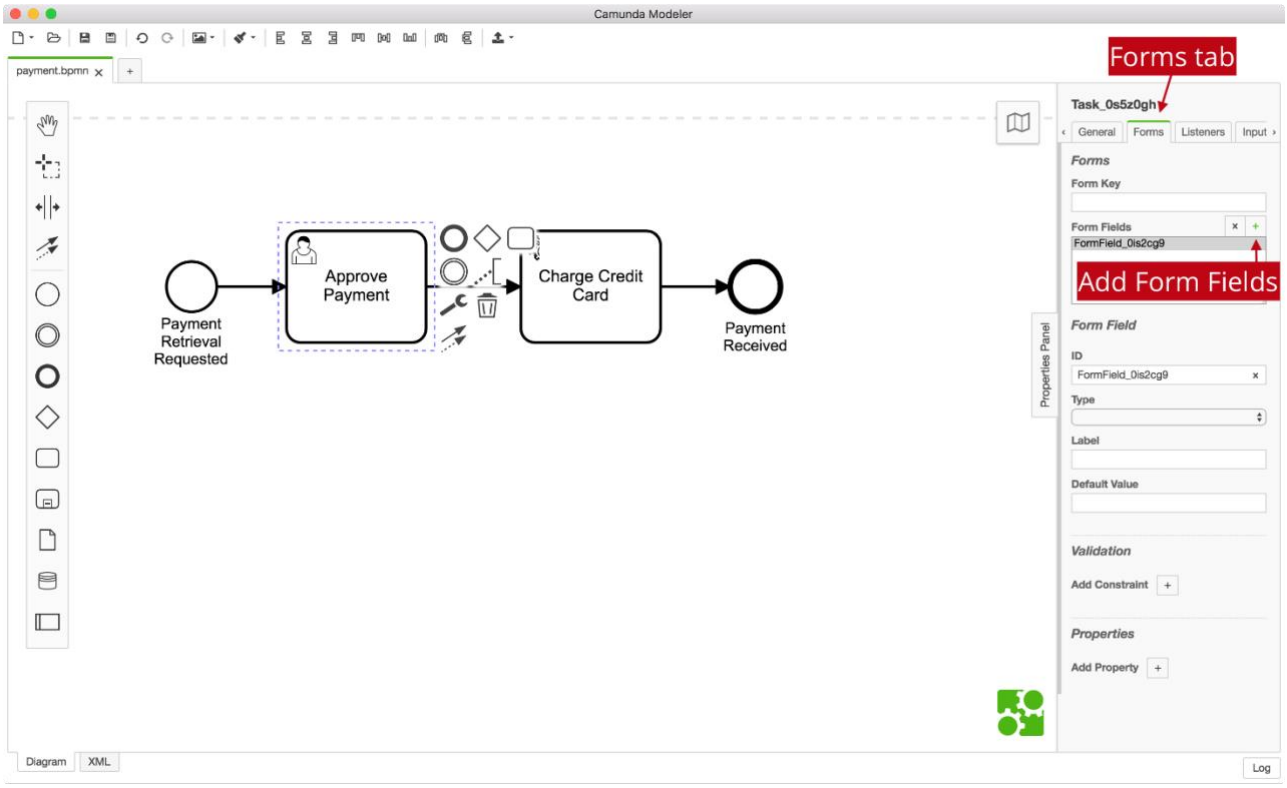
This step will also take place in the properties panel. If the panel is not already visible, click on the “Properties Panel” label on the right-hand side of the Modeler canvas.

Select the User Task on the canvas. This will update the selection in the properties view.

Click on the Tab `Forms` in the properties panel.

(version 5.8.0) -> In the type of Forms choose “Generated Task Forms”

Add three form fields by clicking on the `plus` button:

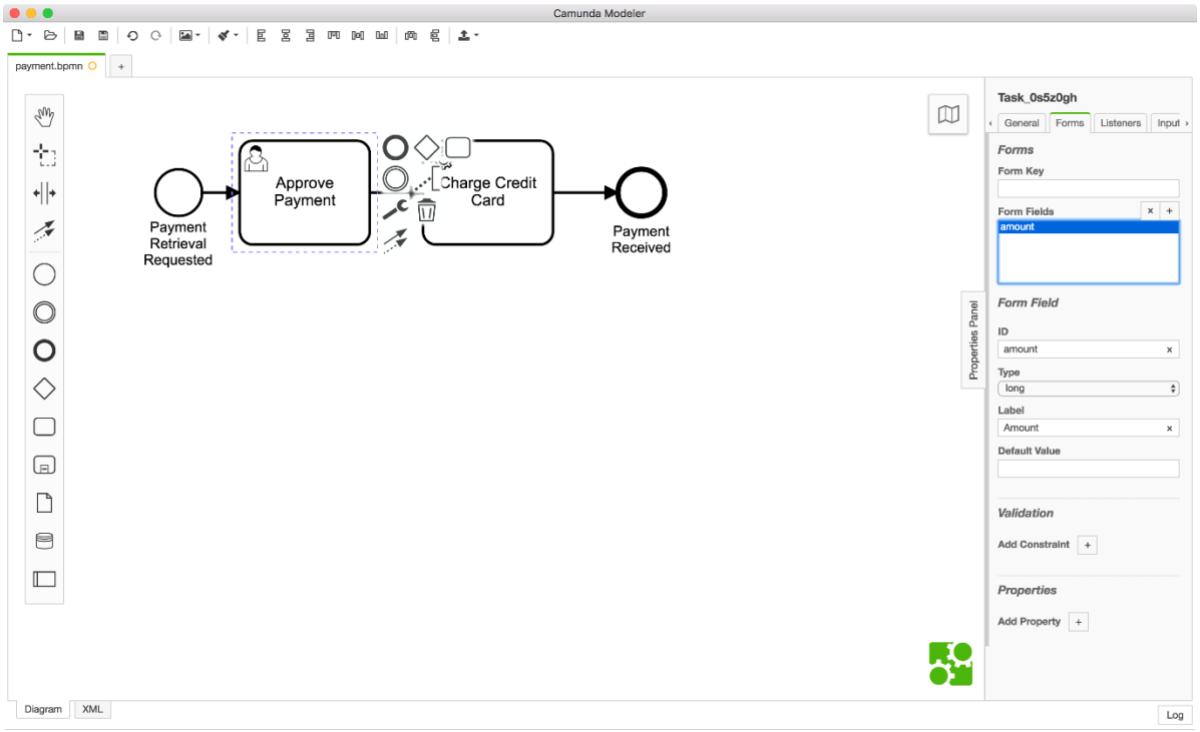


Field 1:

ID: amount

Type: long

Label: Amount

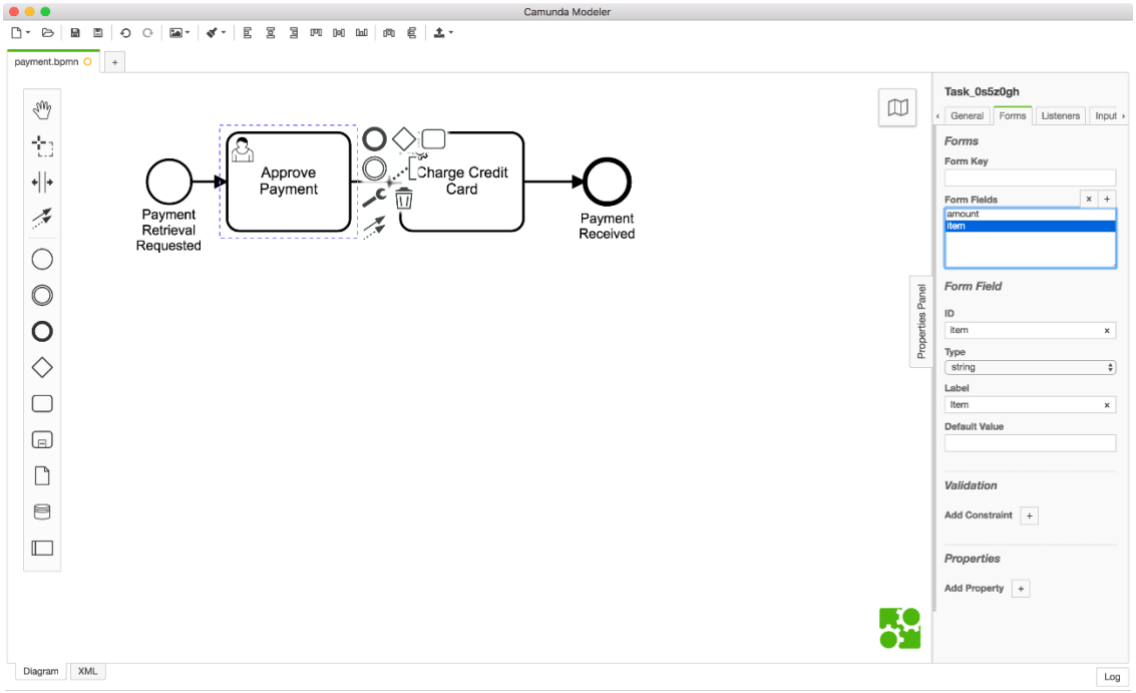


Field 2:

ID: item

Type: string

Label: Item

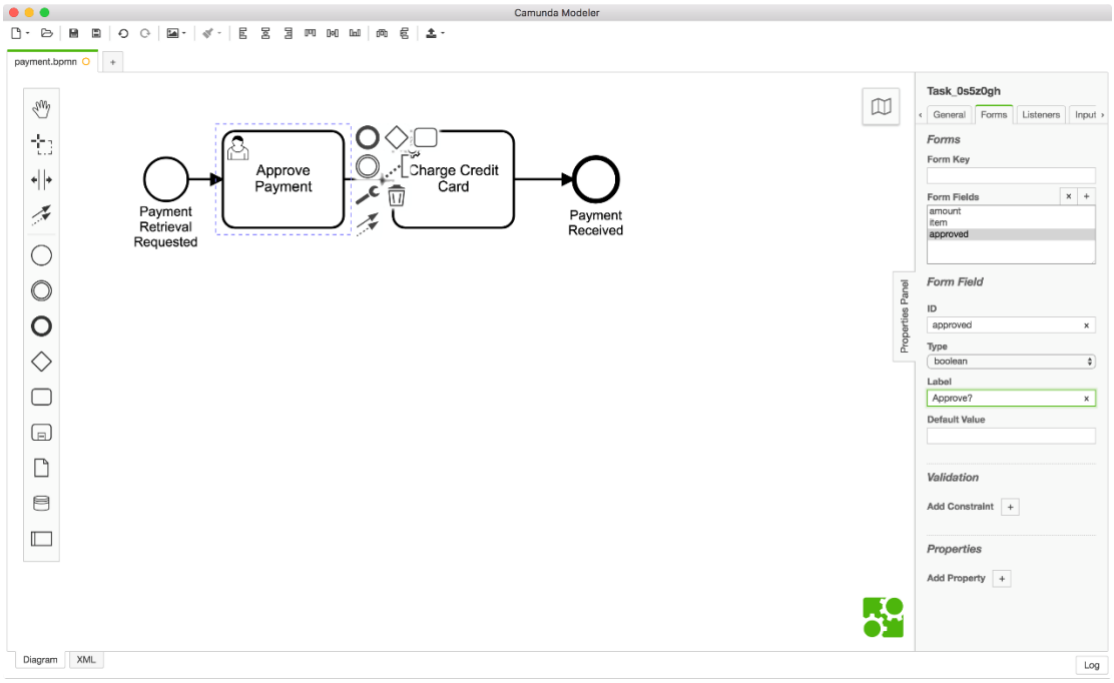


Field 3:

ID: approved

Type: boolean

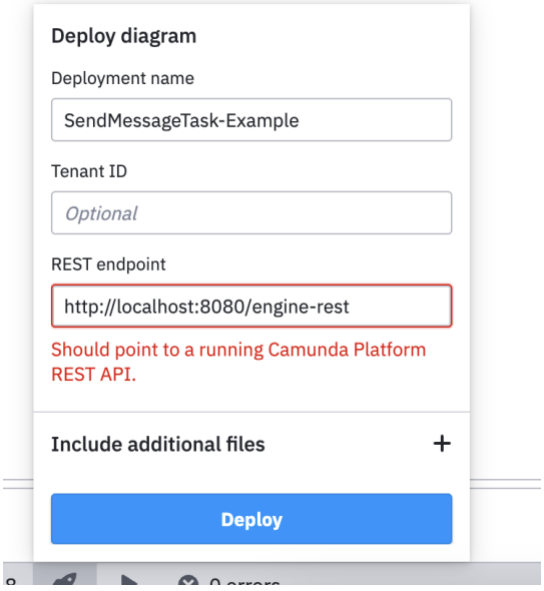
Label: Approved?



D.5. Deploy the Process

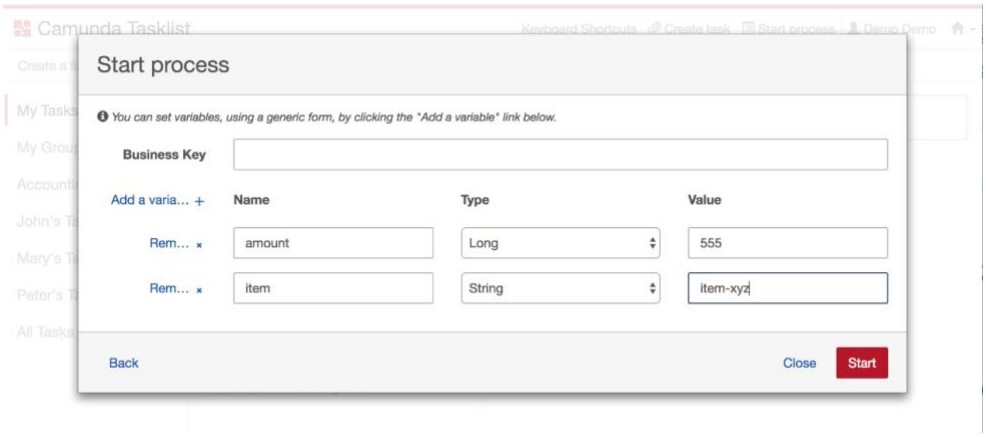
Use the `Deploy` Button in the Camunda Modeler to deploy the updated process to Camunda.

Replacing the localhost for the AWS EC2 DNS_name where you deployed the camunda-engine.



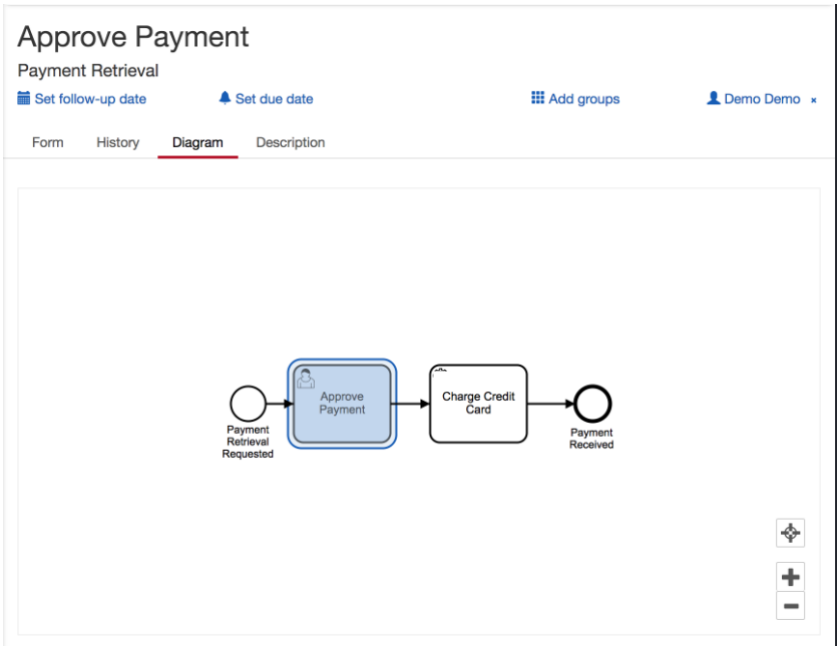
D.6. Work on the Task

Go to Camunda Tasklist and log in with the credentials “demo / demo”. Click on the button to start a process instance. This opens a dialog where you can select *Payment Retrieval* from the list. Now you can set variables for the process instance using a generic form.

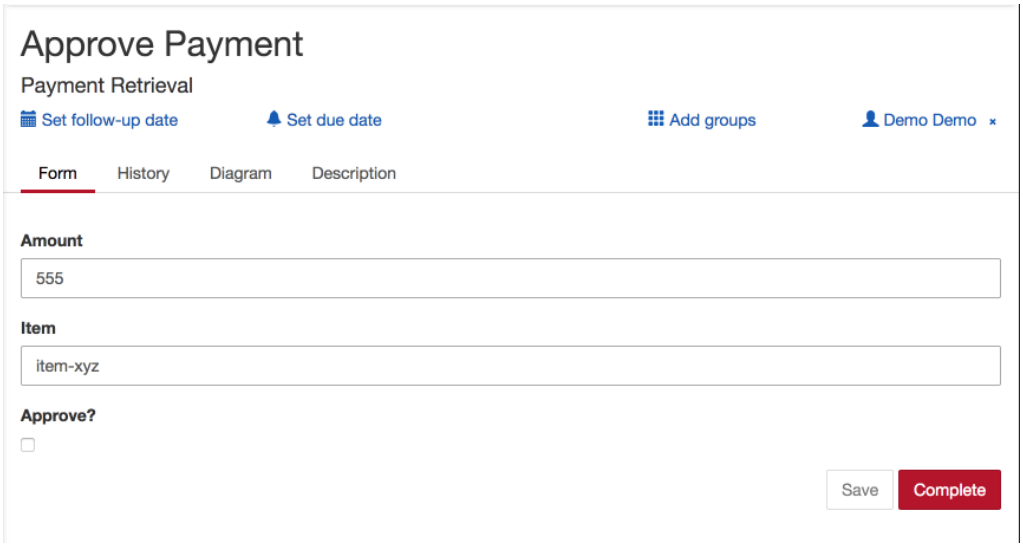


The generic form can be used whenever you have not added a dedicated form for a User Task or a Start Event. Click on the *Add a variable* button to create a new row. Fill in the form as shown in the screenshot. When you're done, click *Start*.

You should now see the *Approve Payment* task in your Tasklist. Select the task and click on the *Diagram* tab. This displays the process diagram highlighting the User Task that's waiting to be worked on.



To work on the task, select the *Form* tab. Because we defined the variables in the Form Tab in the Camunda Modeler, the Tasklist has automatically generated form fields for us.



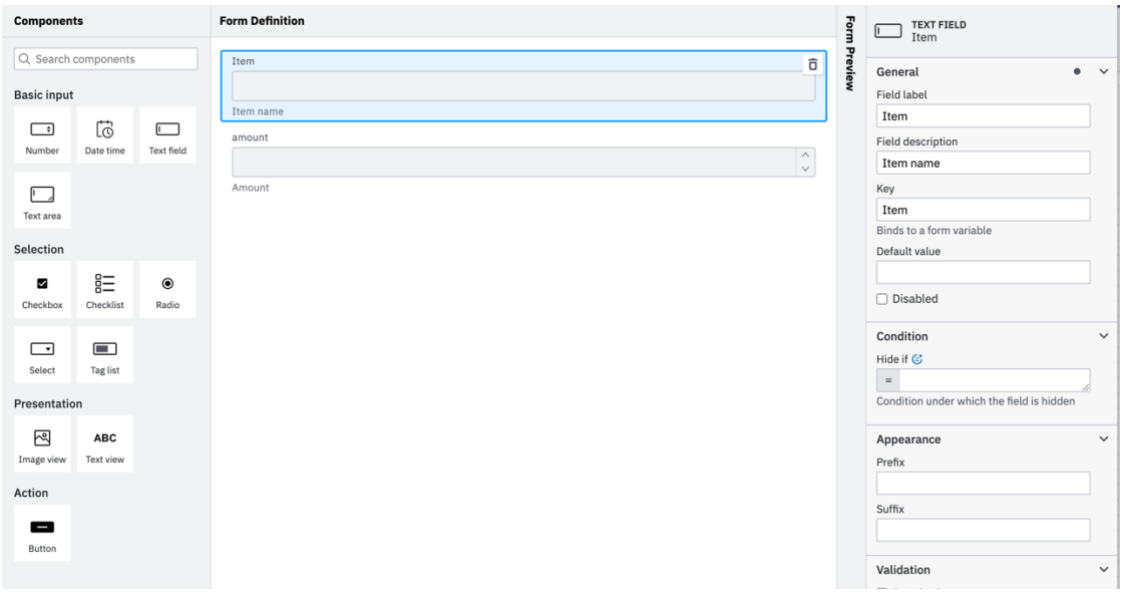
D. Yet, another option using Camunda forms

Instead of hard coding the form directly in the user task you can create the form separately, deploy it, and then refer to it in your process. However, remind that Camunda is a process engine, not an optimized platform for user interaction. When intensive user interactions are required consider the usage of solutions such as https://github.com/formio/formio.

E.1. Create a new form, using the option: File > New File > Form (Camunda Platform 7)

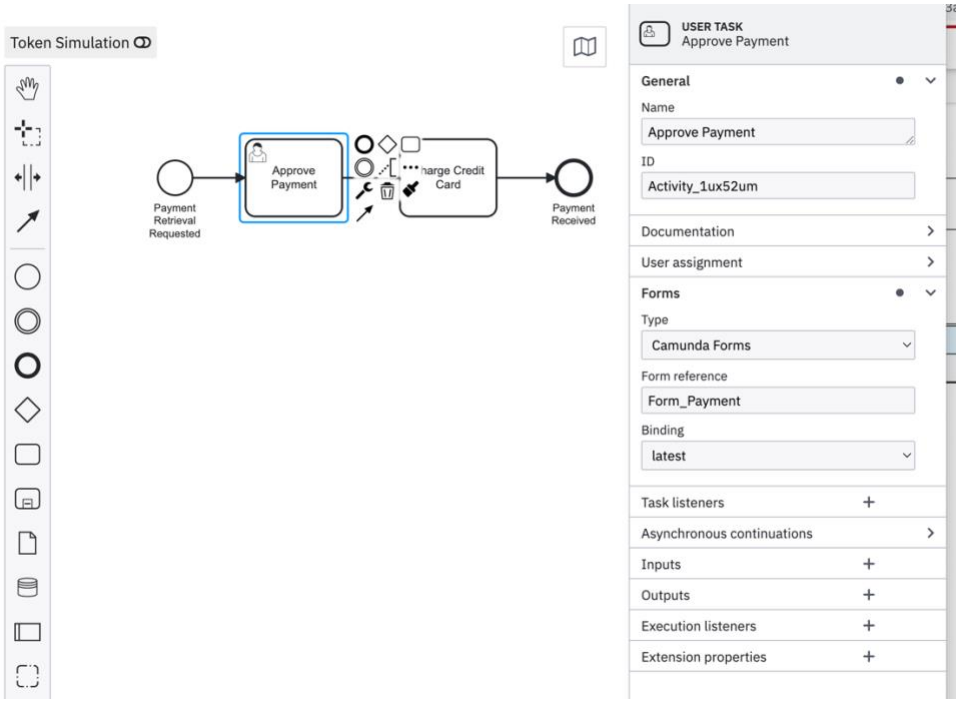
E.2. Define an ID for the entire form, i.e., Form_Payment

E.3. Add 1 text field and 1 number, similar with the following figure:



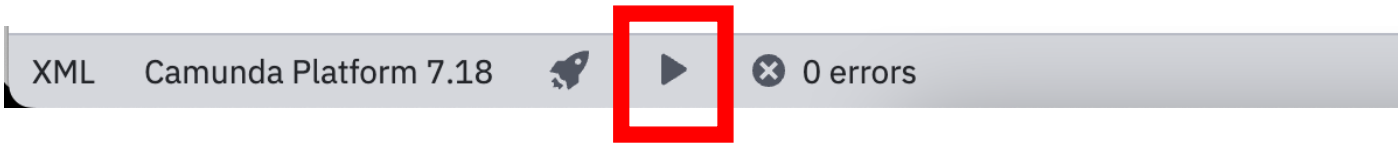
E.4. Deploy the form.

E.5. Change the configuration of the previous user task accordingly with the following figure:

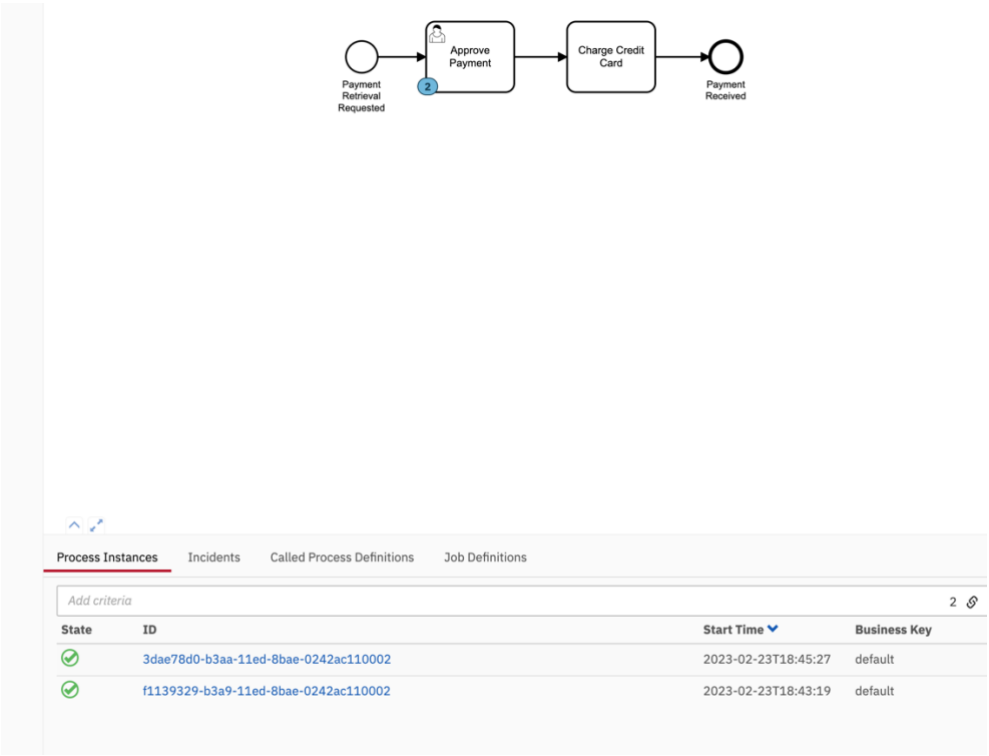


E.6. Deploy the process

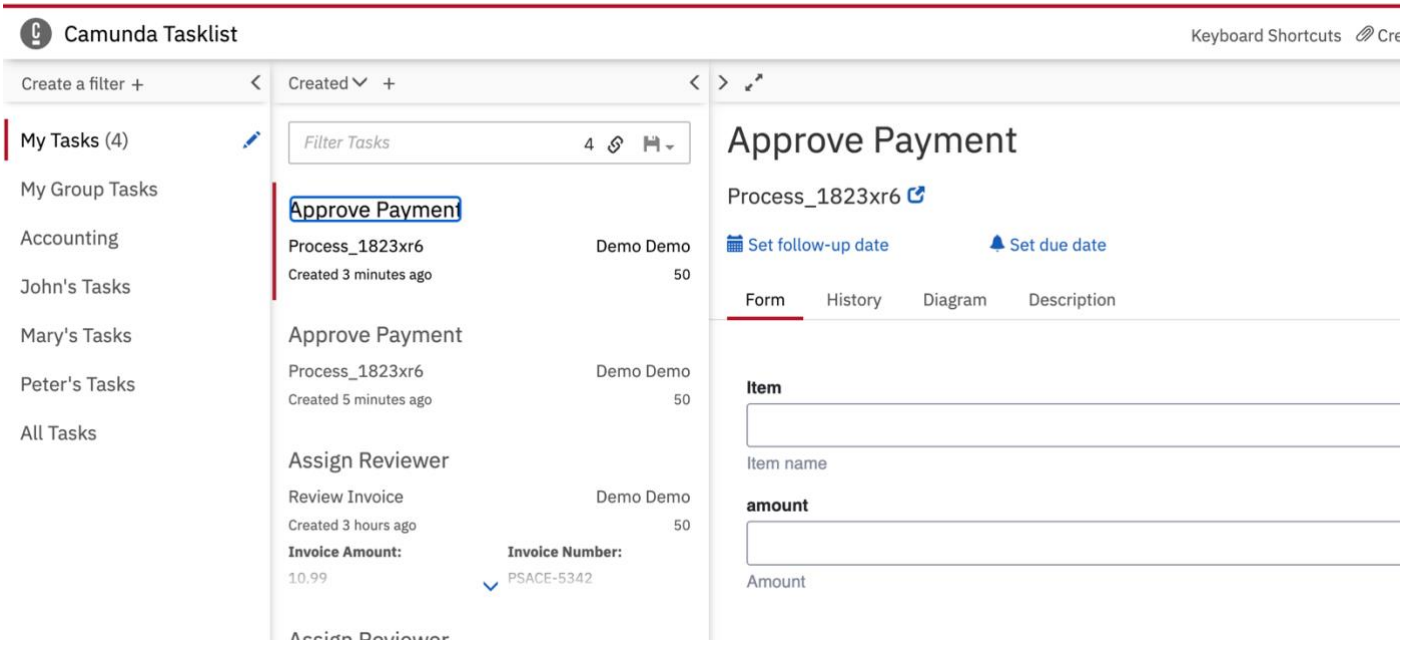
E.7. Start a process instance, using the **`Start instance`** Button in the Camunda Modeler to deploy the updated process to Camunda.



E.8. Check in the cockpit that a pending instance is waiting for a user task:



E.9. Choose your process instance, and then change to user tasks tab, and select one pending action in the task list. You can now see your form!



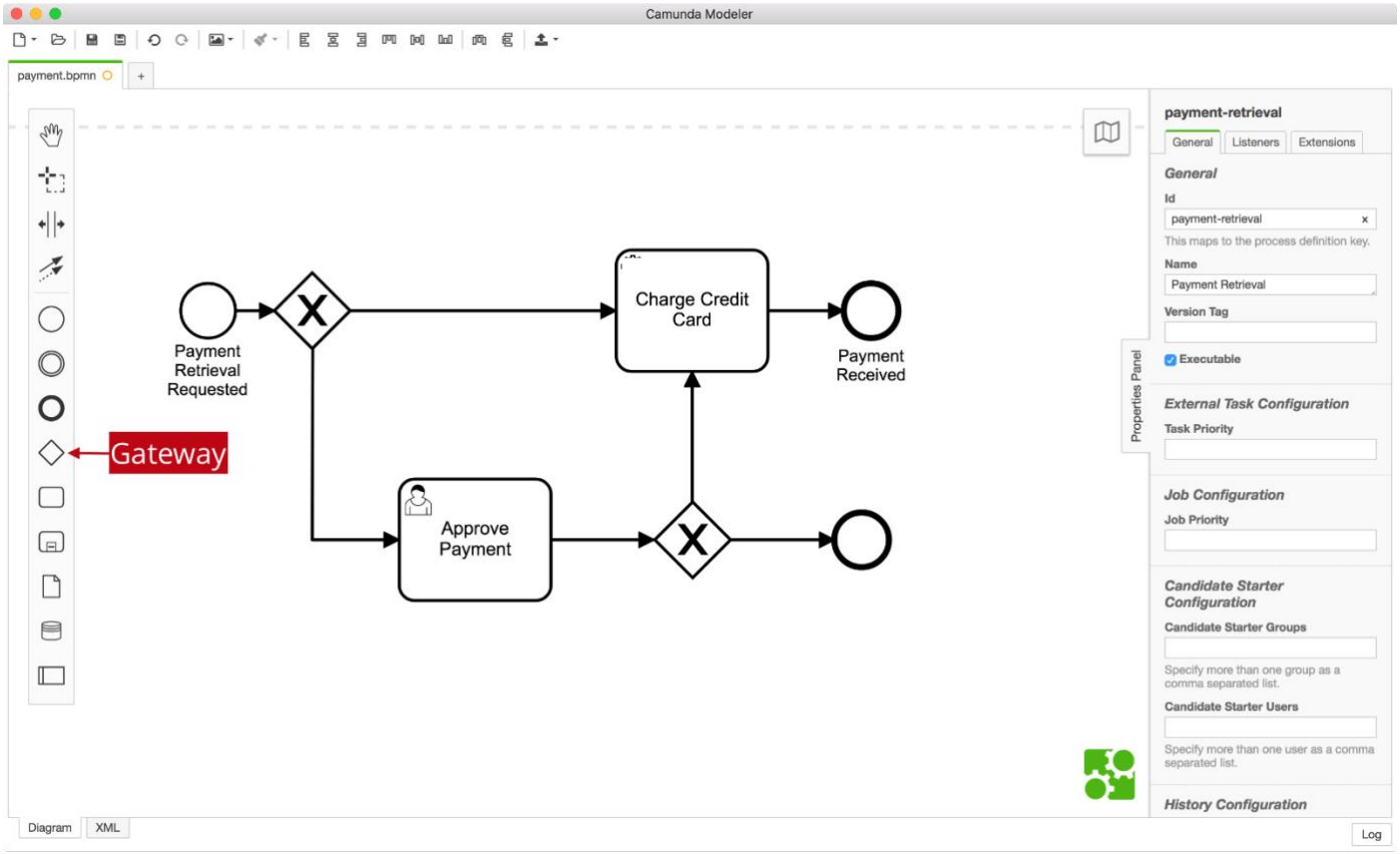
E. Add Gateways to the Process

This example is based in the Camunda documentation available at https://docs.camunda.org/get-started/quick-start/gateway/ . The goal of the example is to implement decisions in the flow of the business process, using the BPMN gateway element.

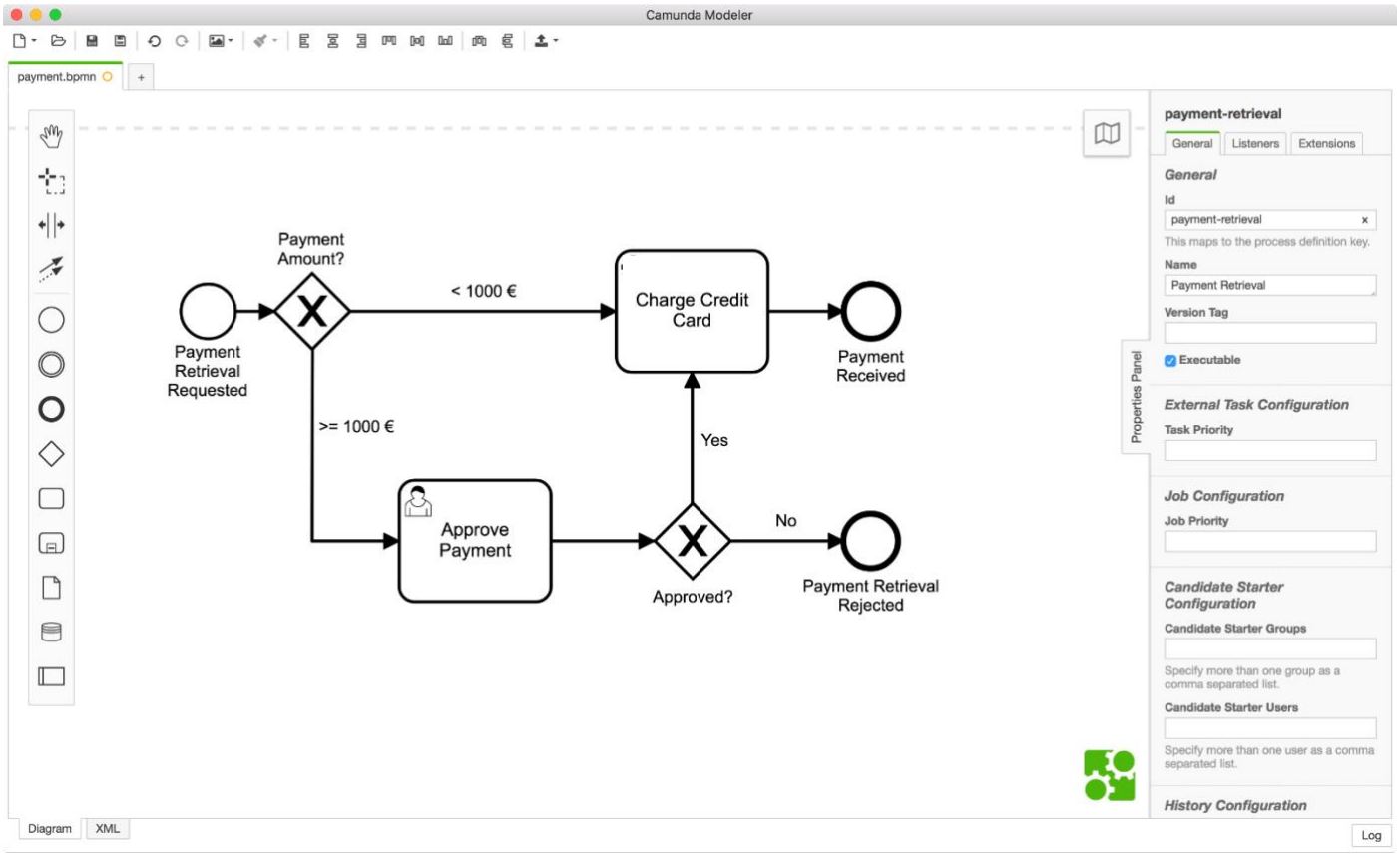
F.1. Add Two Gateways

We want to modify our process so that it is more dynamic. To do so, open the process in the Camunda Modeler.

Next, from the Modeler's left-hand menu, select the gateway shape (diamond) and drag it into position between the Start Event and the Service Task. Move the User Task down and add another Gateway after it. Lastly, adjust the Sequence Flows so that the model looks like this:

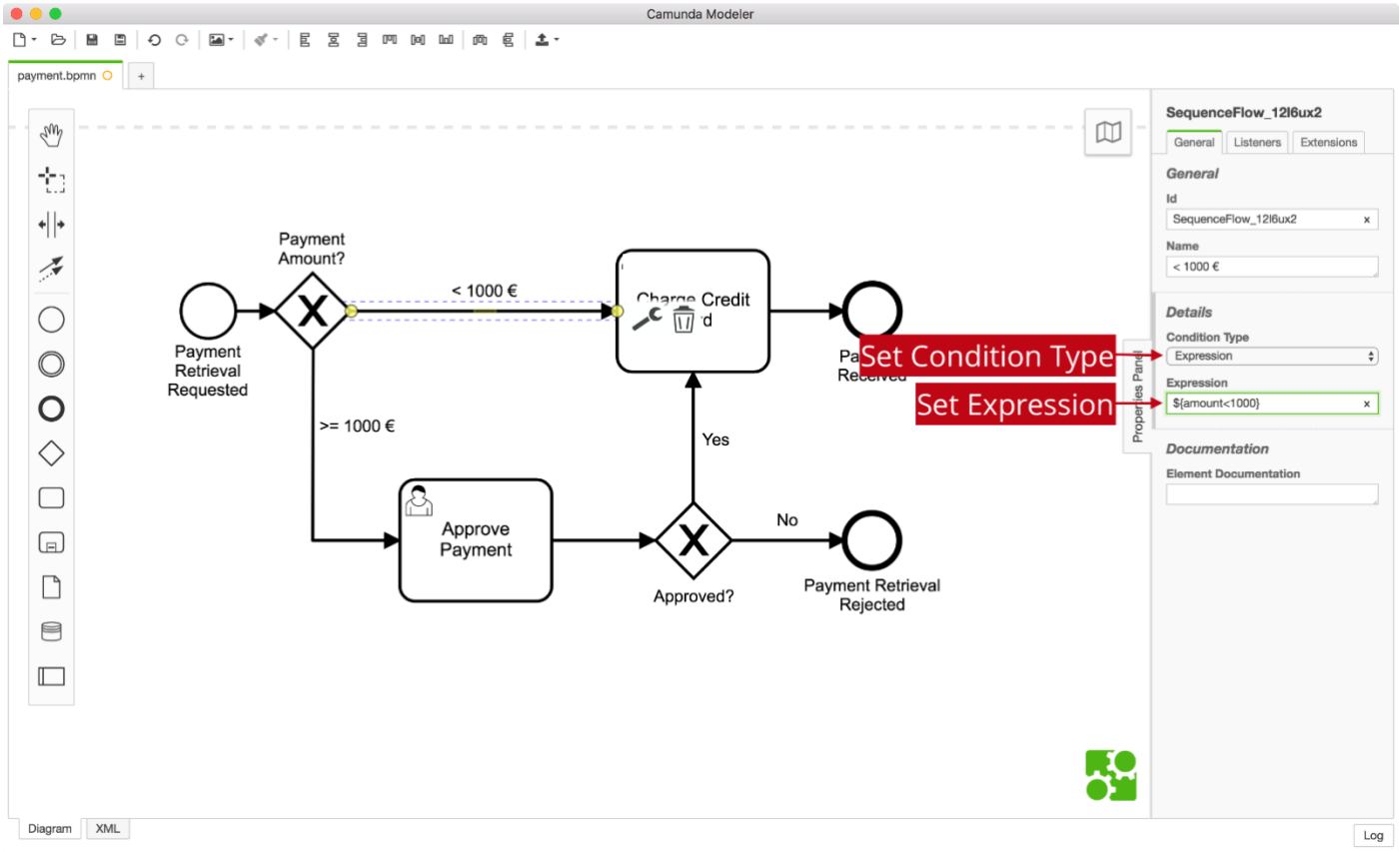


Now also name the new elements accordingly:



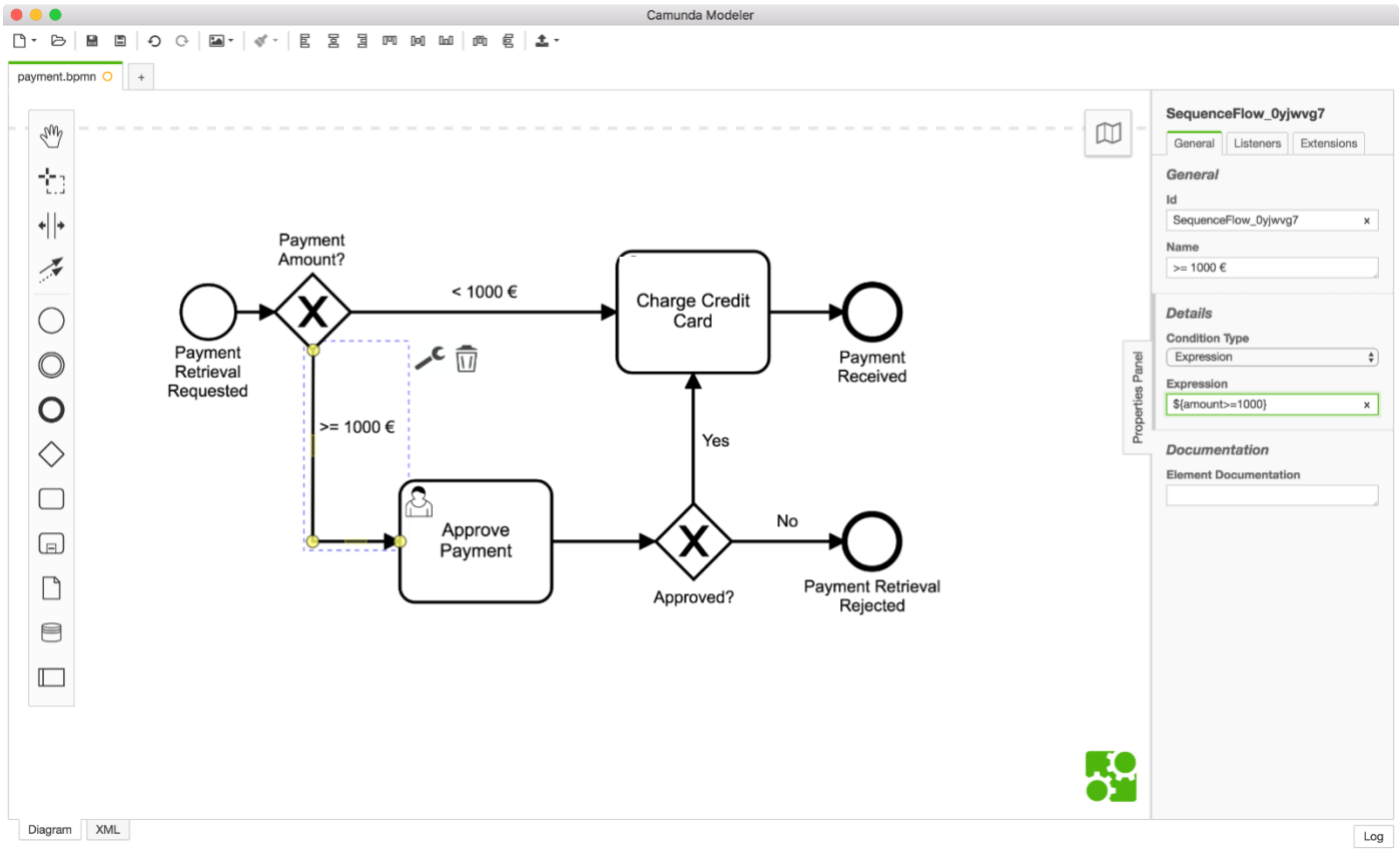
F.2. Configure the Gateways

Next, open the properties panel and select the `<1000 €` Sequence Flow after the Gateway on the canvas. This will update the selection in the properties panel. Scroll to the property named `Condition Type` and change it to `Expression`. Then input `${amount<1000}` as the Expression. We are using the [Java Unified Expression Language] to evaluate the Gateway.

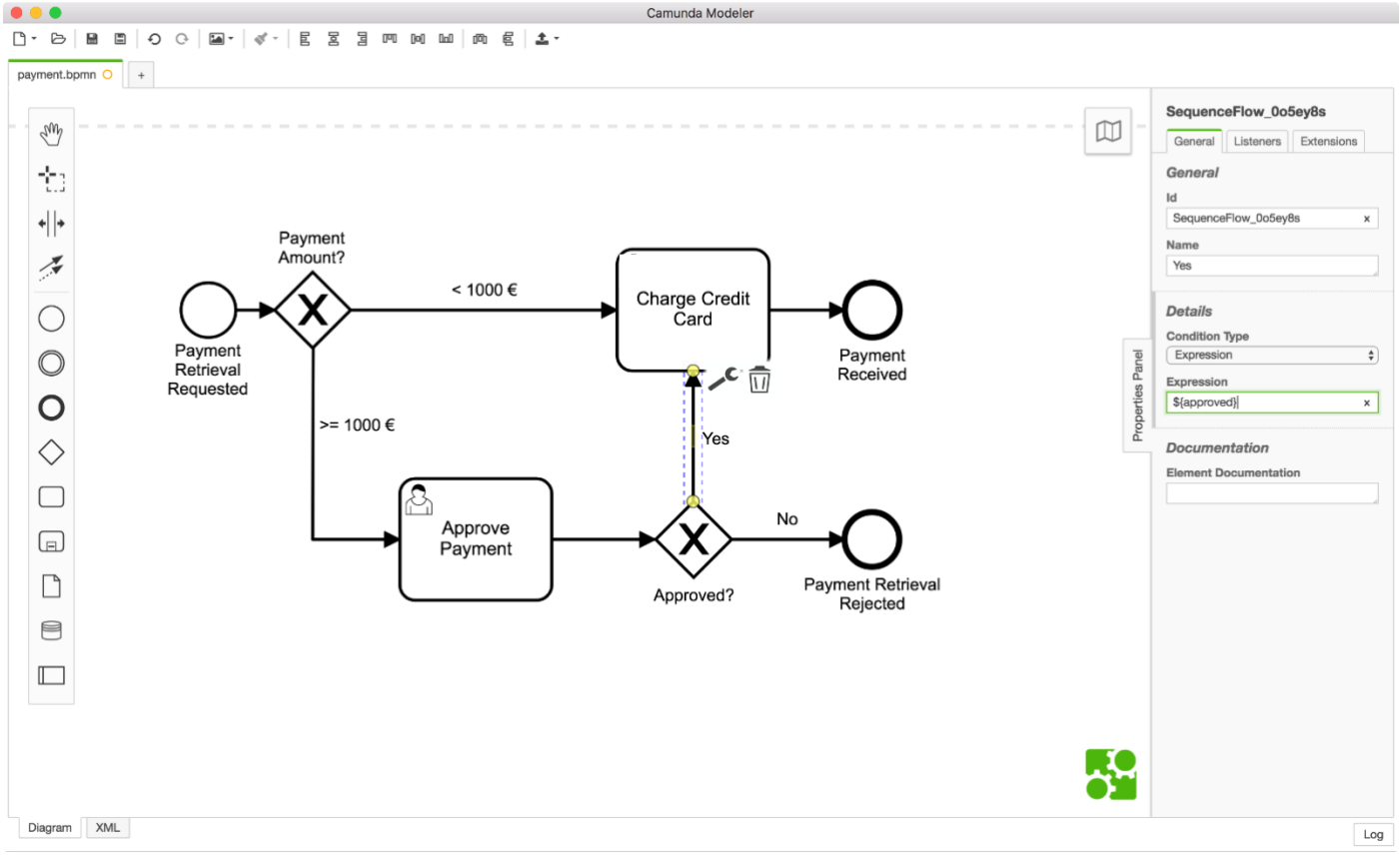


Next, change the Expressions for the other Sequence Flows, too.

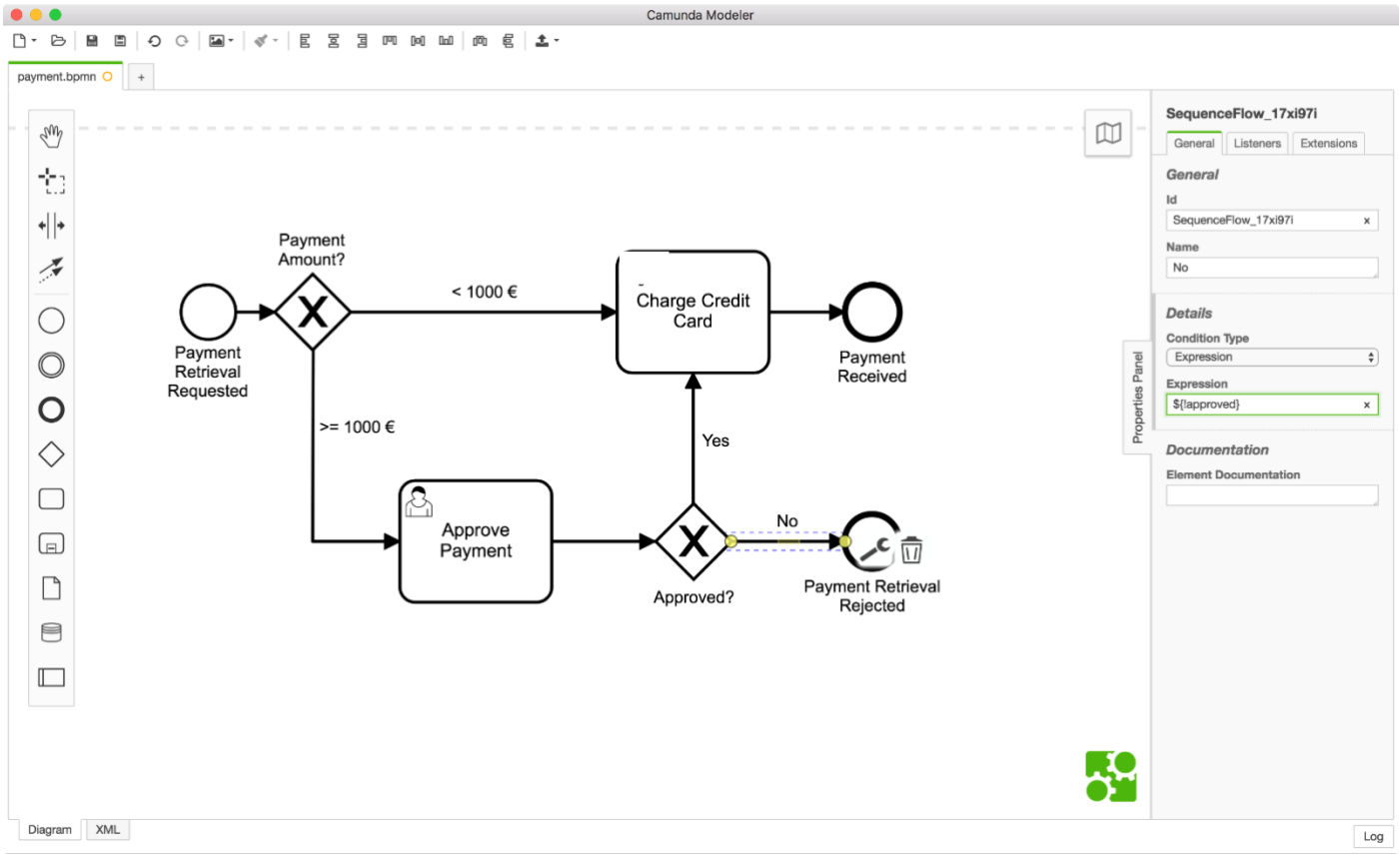
For the `>=1000 €` Sequence Flow, use the Expression `${amount>=1000}`:



For the `Yes` Sequence Flow, use the Expression `${approved}`:



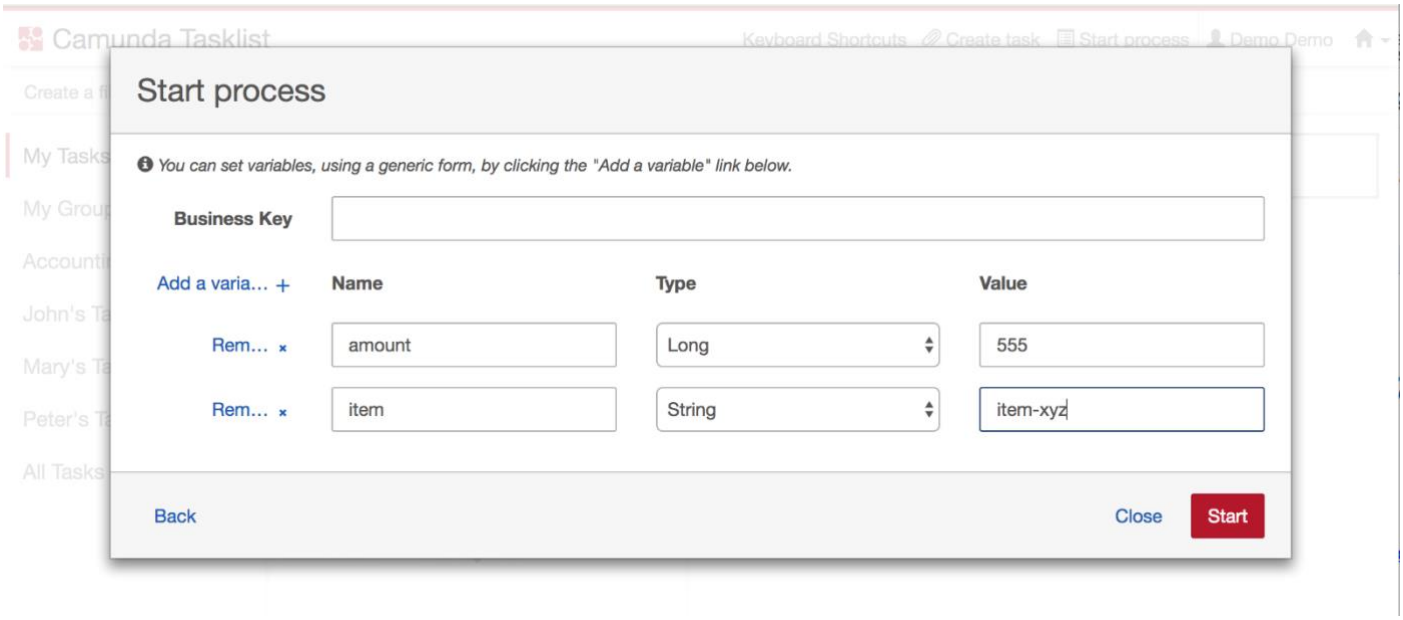
For the `No` Sequence Flow, use the Expression `${!approved}`:



F.3. Deploy the Process

F.4. Work on the Task

Go to Tasklist and log in with the credentials "demo / demo". Click on the "Start process" button to start a process instance for the *Payment Retrieval* Process. Next, set variables for the process instance using the generic form as we learned in the *User Tasks* section.



Fill in the form as shown in the screenshot and make sure you use an amount that is larger or equal to 1000 in order to see the User Task *Approve Payment*. When you are done, click *Start*.

You should see the *Approve Payment* task when you click on *All Tasks*. In this quick start, we're logged into Tasklist as an admin user, and so we can see all tasks associated with our processes. However, it's possible to create filters in Tasklist to determine which users can see which tasks based on user authorization as well as other criteria.

To work on the task, select the *Form* tab and check the *approved* checkbox so that our payment retrieval gets approved. We should see that our worker prints something to the console.

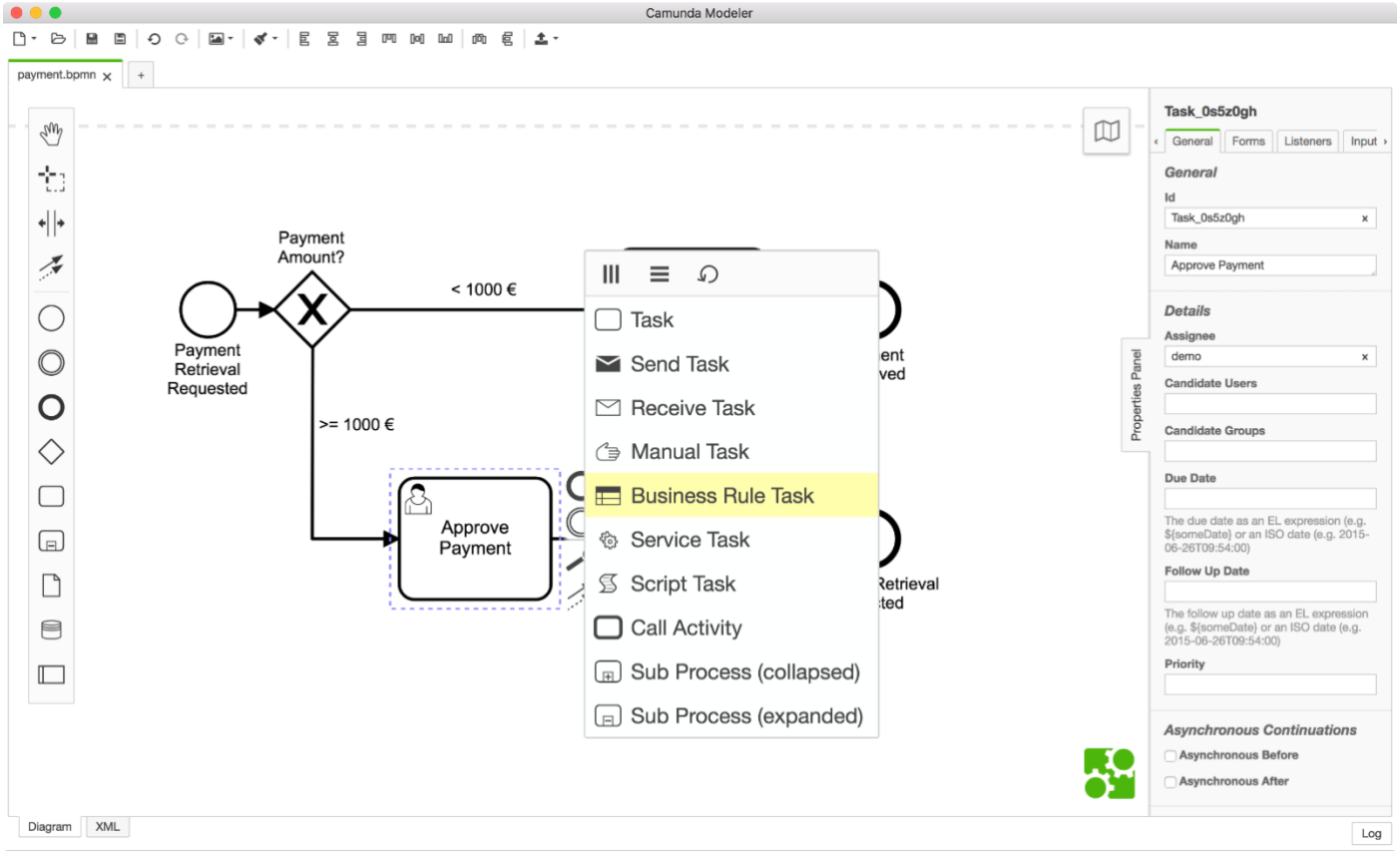
Next, repeat the same steps, and this time, reject the payment. You should also create one instance with an amount less than 1000 to confirm that the first gateway works correctly.

F. Business rule enforcement

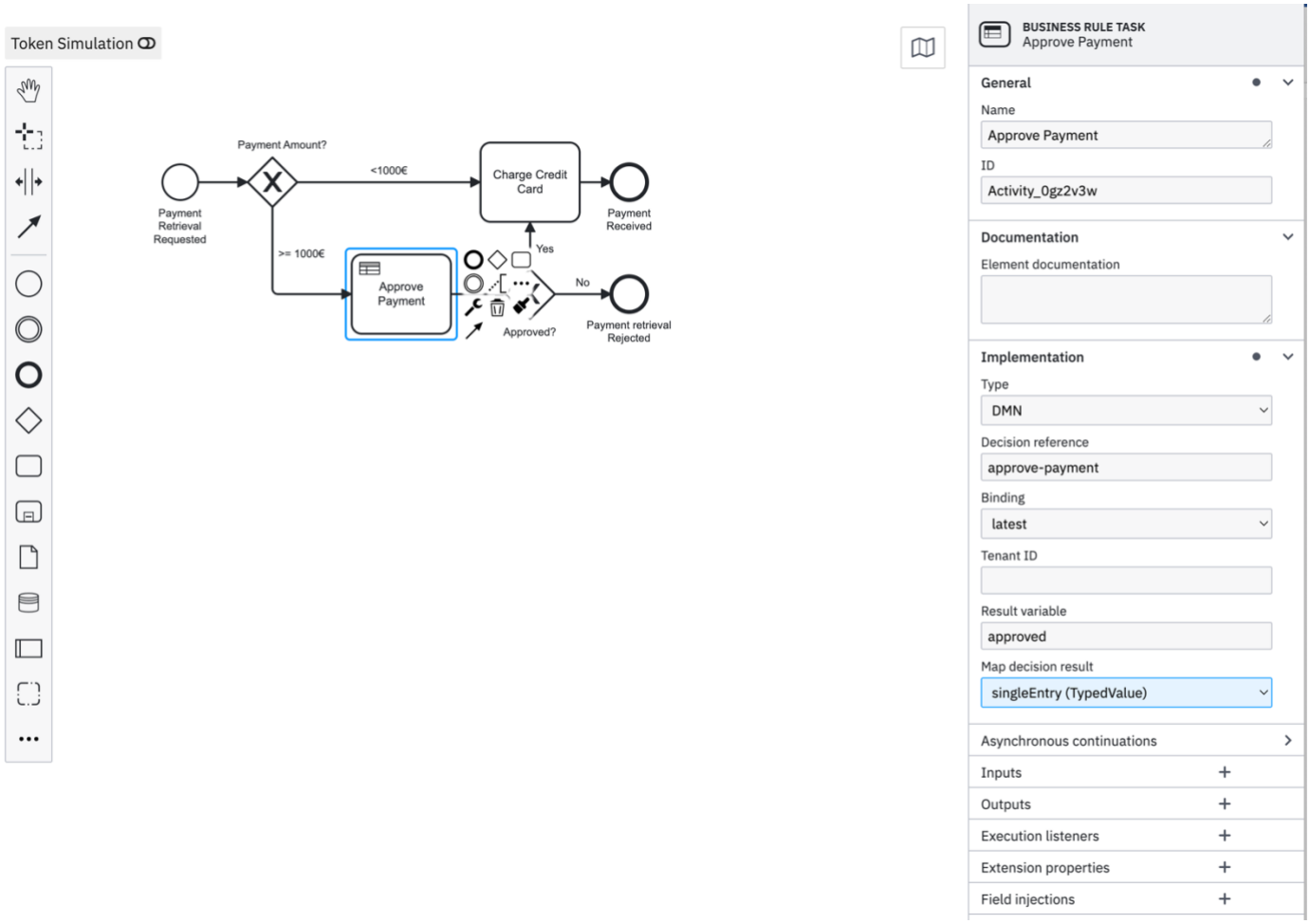
This example is based in the Camunda documentation available at https://docs.camunda.org/get-started/quick-start/decision-automation/. The goal of this example is to implement business rules using the DMN 1.1 standard.

G.1. Add a Business Rule Task to the Process

Use the Camunda Modeler to open the Payment Retrieval process then click on the Approve Payment Task. Change the activity type to *Business Rule Task* in the wrench button menu.



Next, link the Business Rule Task to a DMN table by changing `Implementation` to `DMN` and `Decision Ref` to `approve-payment` in the properties panel. In order to retrieve the result of the evaluation and save it automatically as a process instance variable in our process, we also need to change the `Result Variable` to `approved` and use `singleEntry` as the `Map Decision Result` in the properties panel.



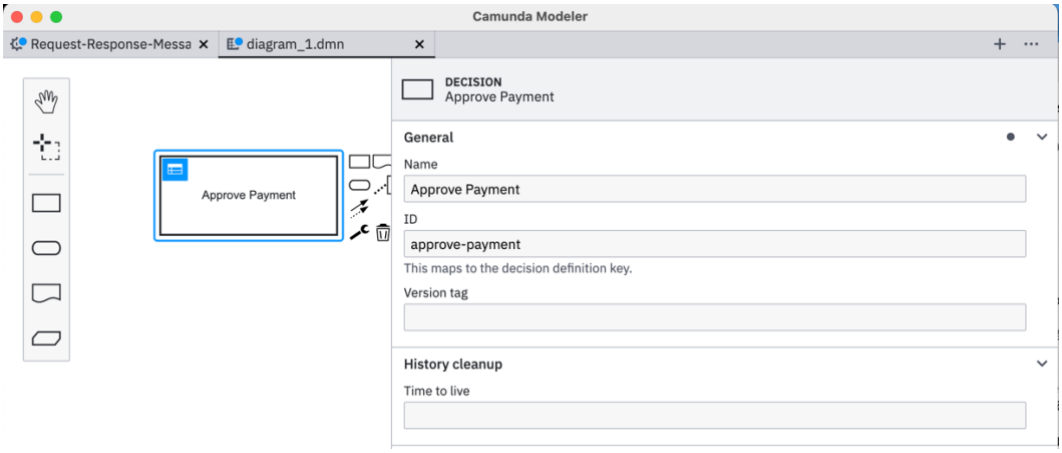
Save your changes and deploy the updated process using the `Deploy` Button in the Camunda Modeler.

G.2. Create a DMN table using the Camunda Modeler

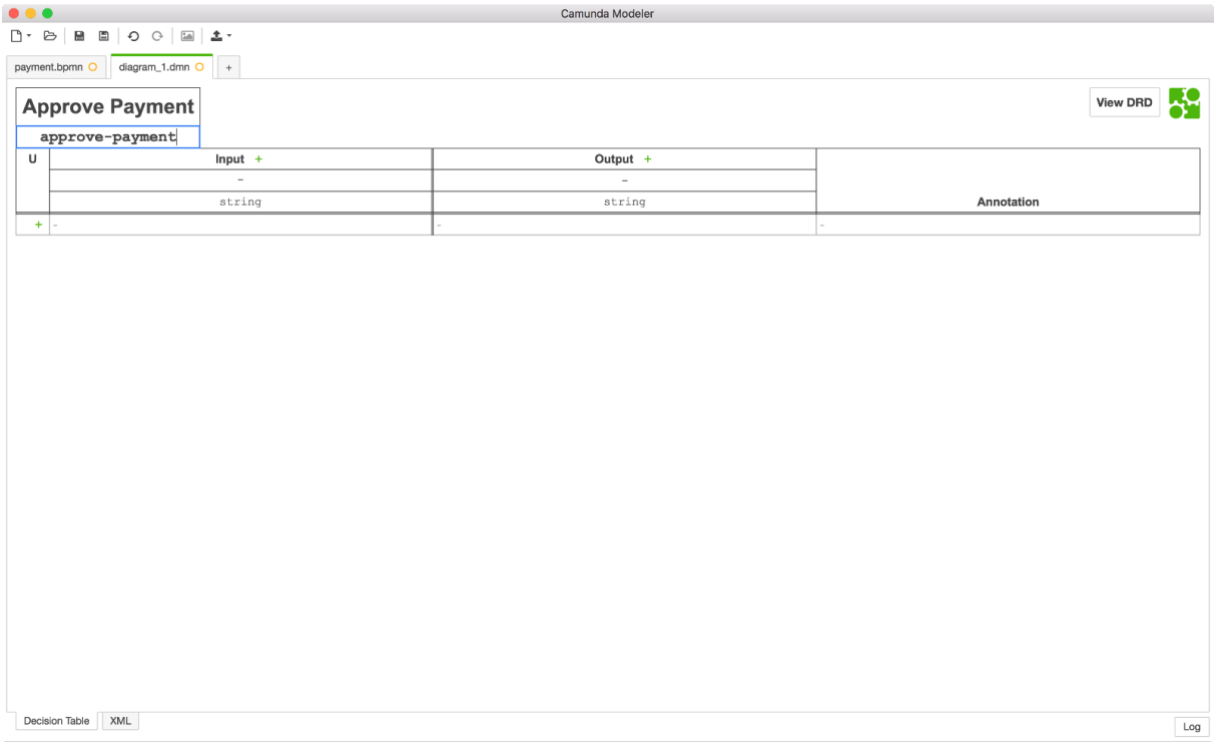
Create a new DMN table by clicking *File > New File > DMN Diagram (Camunda Platform 7)*.

G.3. Specify the DMN table

First, give the DMN table the name Approve Payment and the ID approve-payment, and then double click on it, as following.

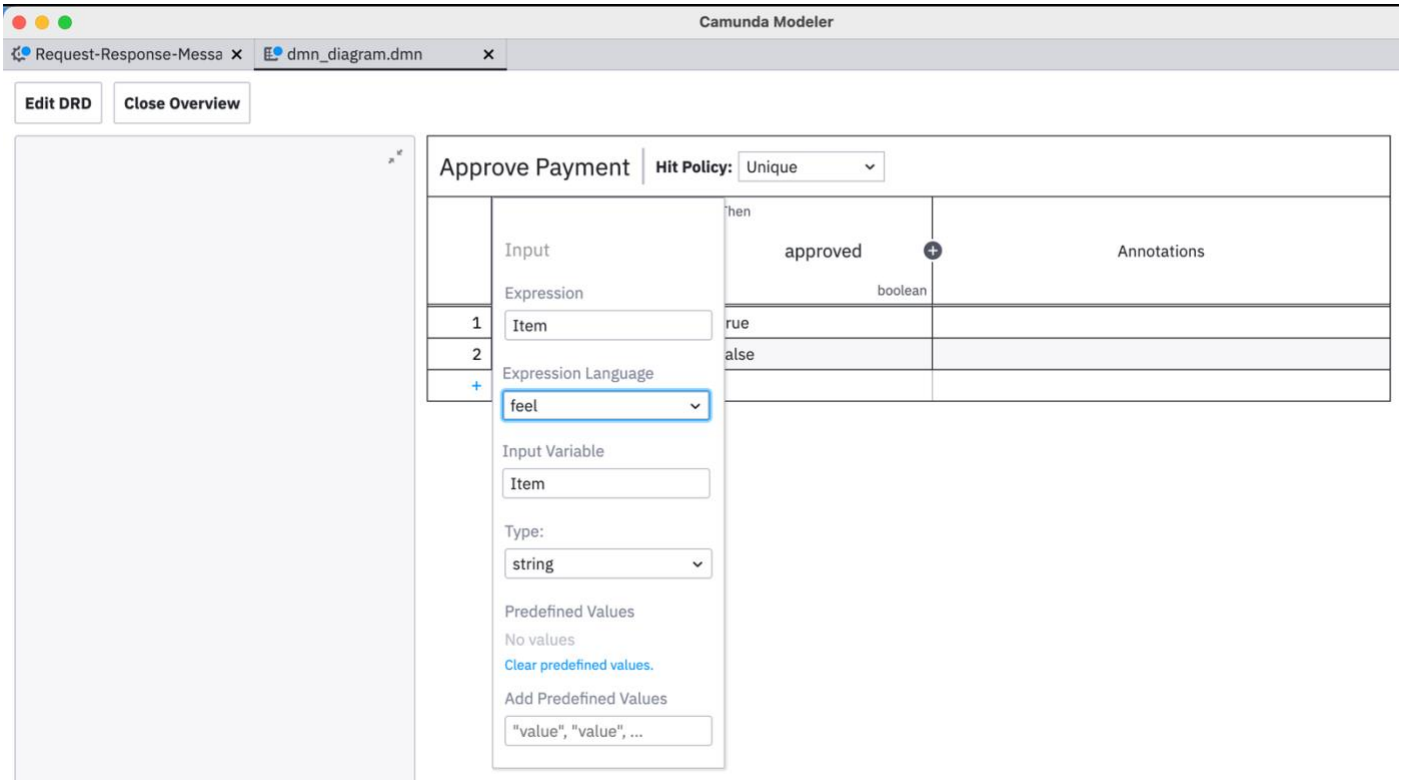


The DMN table ID will match the `Decision Ref` in your BPMN process.

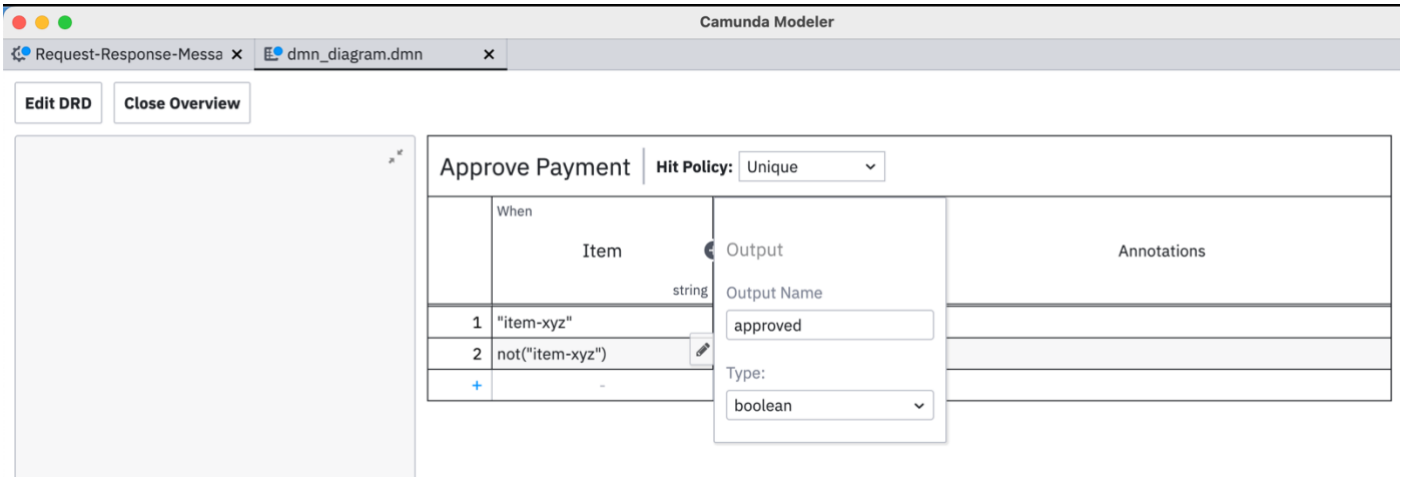


Next, specify the input expressions for the DMN table. In this example, we'll decide whether a payment is approved based on the item name. Your rules can also make use of the FEEL Expression Language, JUEL or Script. If you like, you can read more about Expressions in the DMN Engine.

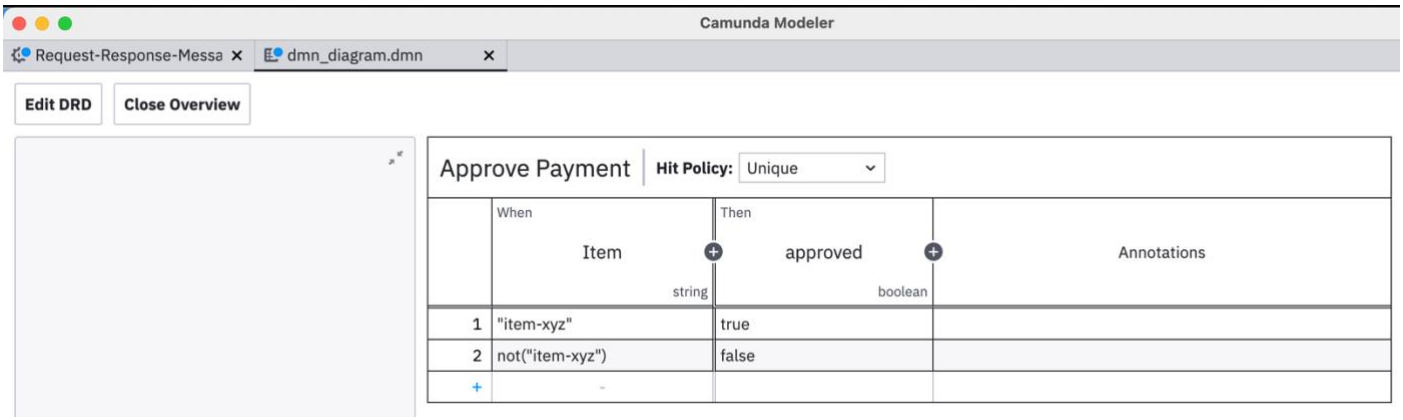
For the input column, use the following configuration:



Next, set up the output column, as following:

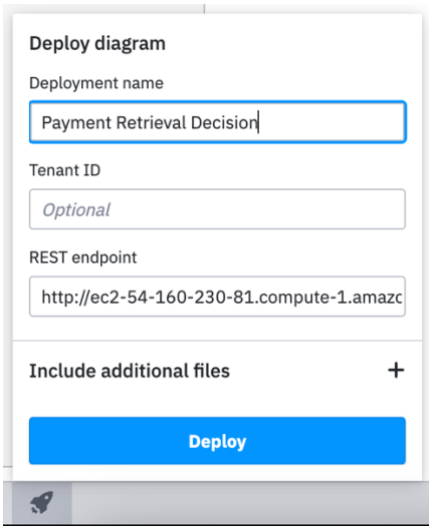


Let's create some rules by clicking on the plus icon on the left side of the DMN table. After setup, your DMN table should look like this:



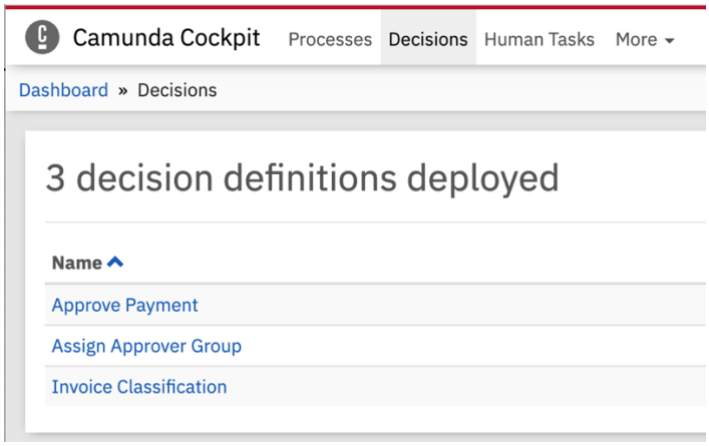
G.4. Deploy the DMN table

To deploy the Decision Table, click on the Deploy button in the Camunda Modeler, give it Deployment Name "**Payment Retrieval Decision**", then hit the Deploy button.



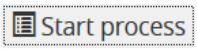
G.5. Verify the Deployment with Cockpit

Now, use Cockpit to see if the decision table was successfully deployed. Log in with the credentials *demo / demo*. Navigate to the "Decision definitions" section. Your decision table *Approve Payment* should be listed as deployed decision definition.

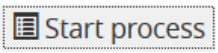


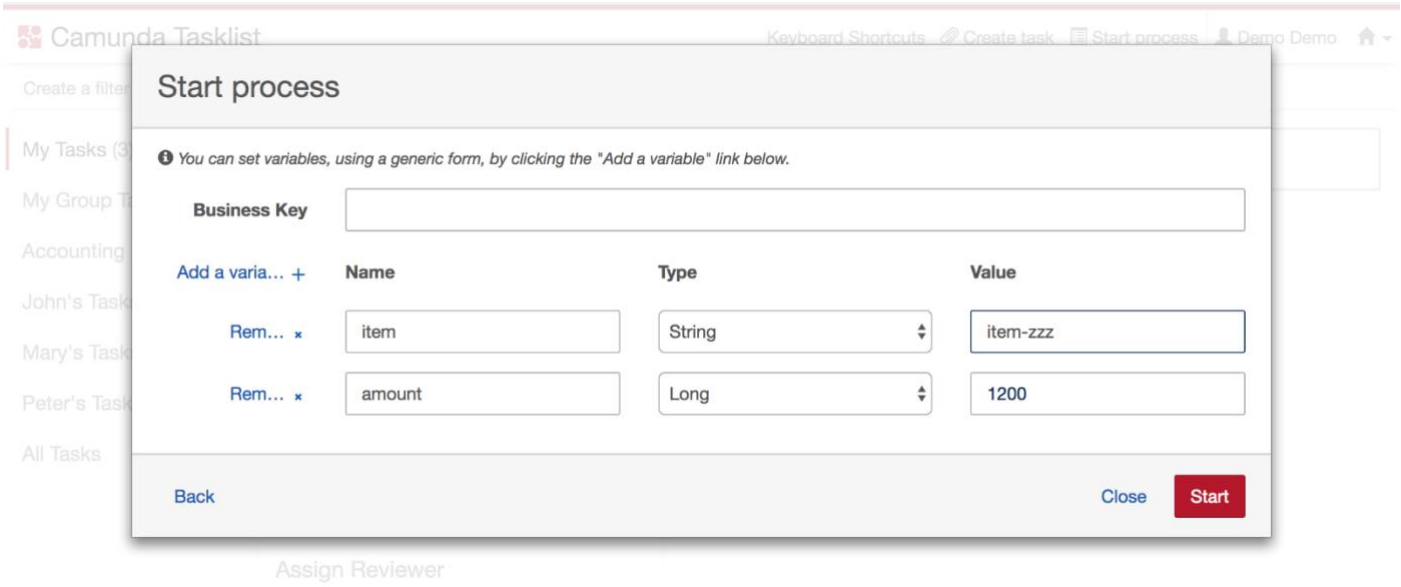
G.6. Inspect using Cockpit and Tasklist

Next, use Tasklist to start two new Process Instances and verify that depending on your input, the Process Instance will be routed differently.

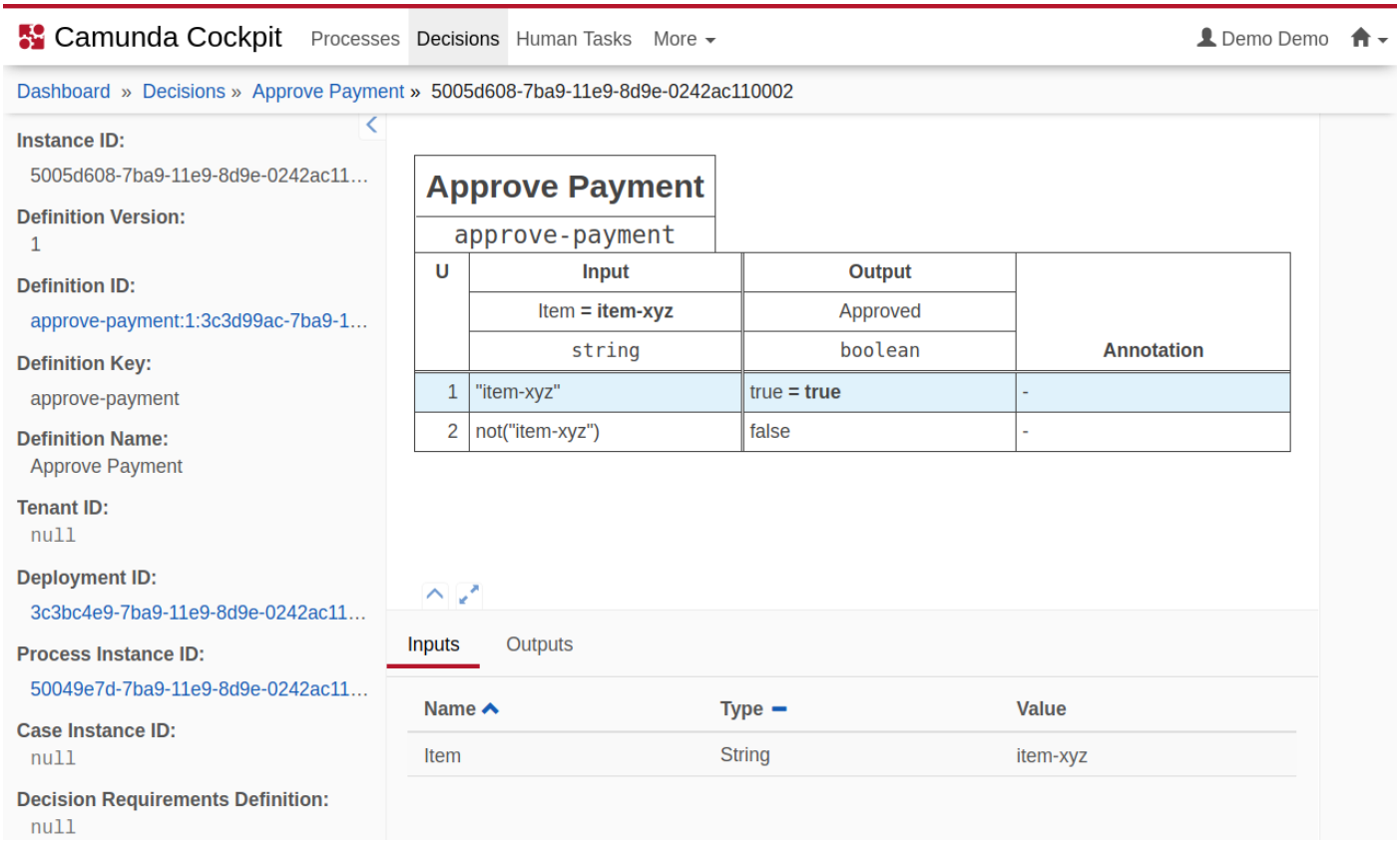
Click on the  button to start a process instance and choose the `Payment` process. Use the generic form to add the variables as follows:

Hit the Start Instance button.

Next, click again on the  button to start another process instance and choose the `Payment` process. Use the generic form to add the variables as follows:



You will see that depending on the input, the worker will either charge or not charge the credit card. You can also verify that the DMN tables were evaluated by using Camunda Cockpit. Log in with the credentials *demo / demo*. Navigate to the "Decisions" section and click on Approve Payment. Check the different Decision Instances that were evaluated by clicking on the ID in the table. A single DMN table that was executed could look like this in Camunda Cockpit:



G. Integrate with a REST API installed remotely using Camunda http-connector – REQUEST ONLY

This example shows how to invoke a REST API from a **BPMN service task using the Camunda connector if indications for installation in section B were followed**. You need to create the service task element in the BPMN model as explain previously in this tutorial. Then, to invoke a REST API during the BPMN execution the following configuration in the Camunda modeller is required.

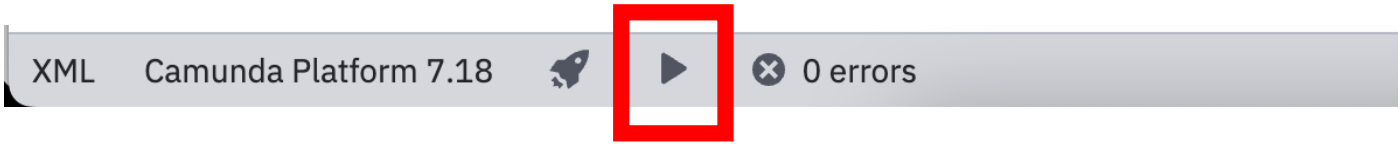
H.1. Configure your task as a service task and choose the "**Connector**" in the implementation details, the connector ID should be "**http-connector**".

H.2. Then, add three "**Connector inputs**":

a. "**method**": of type String or expression with value = POST
b. "**payload**": of type String or expression with value containing the JSON to be send with the request
c. "**url**": of type String or expression with value = endpoint previously available, *e.g.*, Kong API or AWS Gateway API, or other

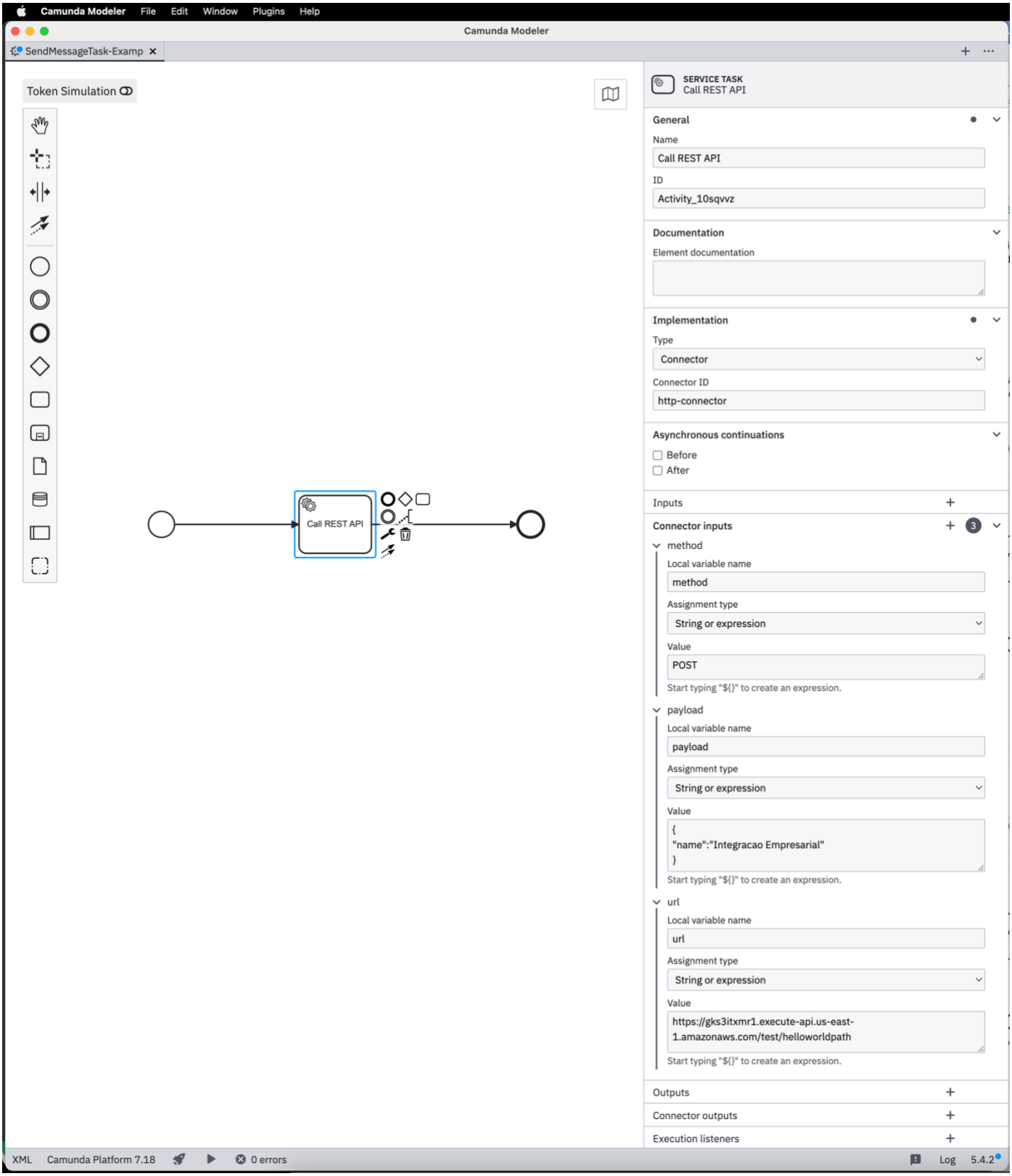
H.3. Deploy the BPMN model

H.4. Start a process instance, using the **`Start instance`** Button in the Camunda Modeler to deploy the updated process to Camunda.



H.5. Verify in the REST API endpoint that it was invoked. For instance, in AWS CloudWatch.

The previous configuration is also depicted in the following figure:


Camunda Modeler File Edit Window Plugins Help
Camunda Modeler
SendMessageTask-Examp
Token Simulation
Call REST API
SERVICE TASK
Call REST API
General
Name
Call REST API
ID
Activity_10sqvvz
Documentation
Element documentation
Implementation
Type
Connector
Connector ID
http-connector
Asynchronous continuations
Before
After
Inputs
Connector inputs
3
method
Local variable name
method
Assignment type
String or expression
Value
POST
Start typing "${}" to create an expression.
payload
Local variable name
payload
Assignment type
String or expression
Value
{
"name":"Integracao Empresarial"
}
Start typing "${}" to create an expression.
url
Local variable name
url
Assignment type
String or expression
Value
https://gks3itxmr1.execute-api.us-east-1.amazonaws.com/test/helloworldpath
Start typing "${}" to create an expression.
Outputs
Connector outputs
Execution listeners
XML
Camunda Platform 7.18
0 errors
Log
5.4.2

H. Integrate with a REST API installed remotely using Camunda http-connector – REQUEST with variable

This example shows how to invoke a REST API from a **BPMN service task using the Camunda connector if indications for installation in section B were followed** and that includes some data previously processed in the BPMN process flow into the REST API Request. You need to create the service task element in the BPMN model as explain previously in this tutorial. Then, to invoke a REST API during the BPMN execution the following configuration in the Camunda modeller is required.

If you need to add any process variable to the web service call you may add it, using the payload, as follows (*CourseName* is used to only for demonstration purposes).

Firstly, a user task is created to allow the input of the course name as text description from the end user. For that, you need to add a Task and choose the **"User Task"** option. Then, assign a user to the task. In this example is used the "demo" user. After that, choose a form of type "Generated Task Forms", and create a Form field:

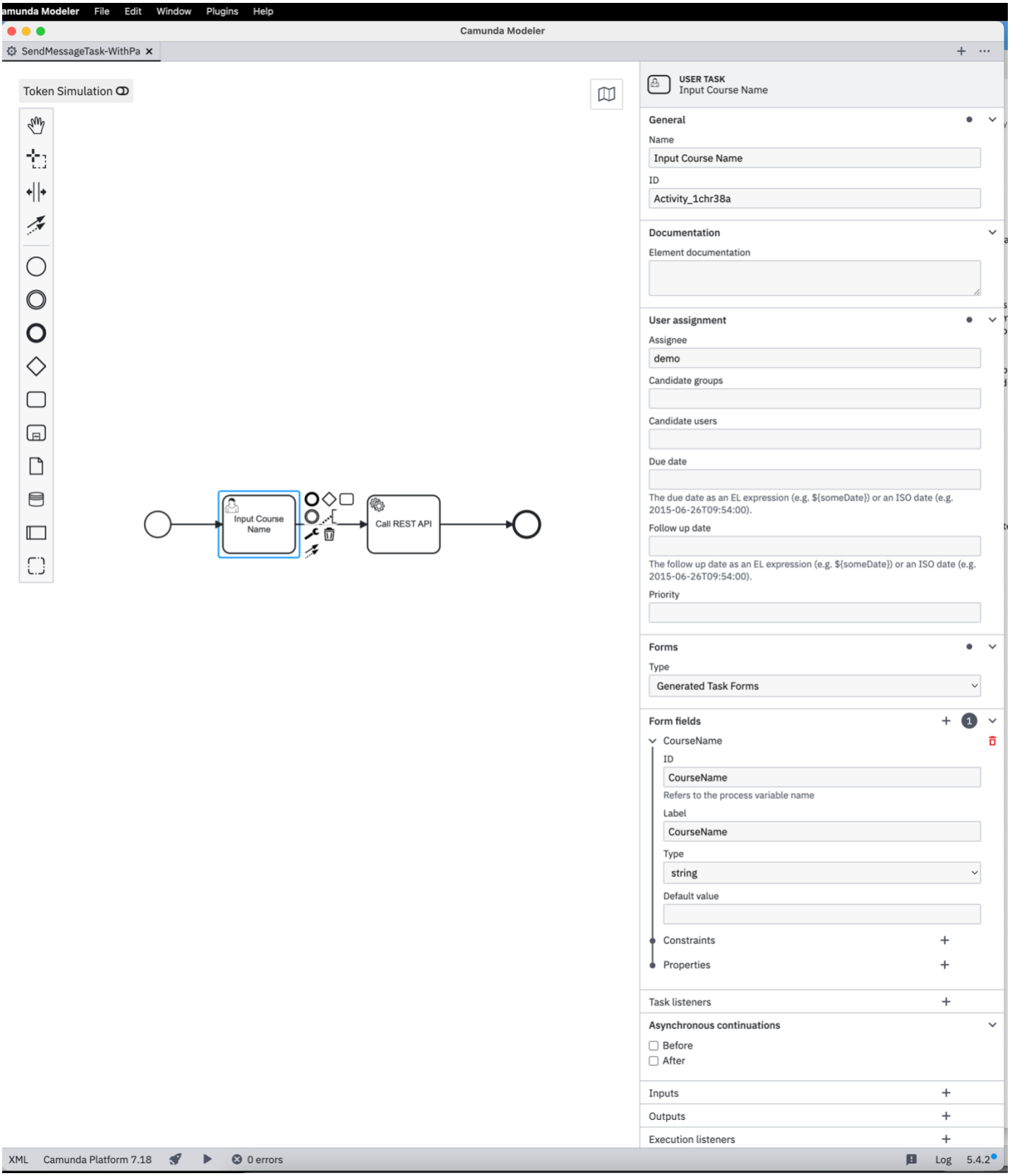
- ID = CourseName
- Label = CourseName
- Type = string

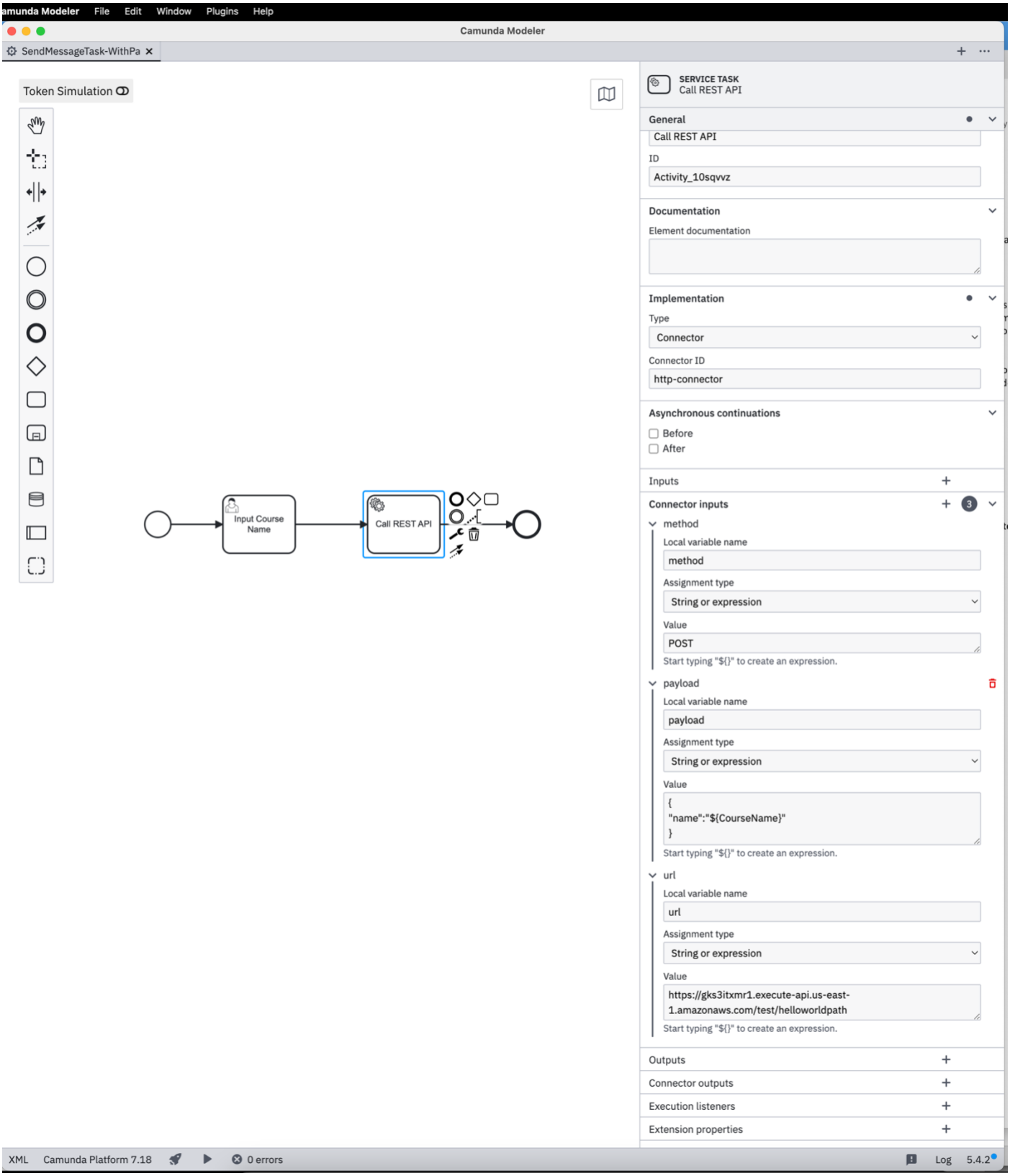
With this configuration you are now accepting the coursename from the user and allocate a variable that could be used afterwards.

Secondly, choose the "Call REST API" task and change the payload to be dynamically created on the previous **CourseName**, using the following syntax:

```
{
"name":"${CourseName}"
}
```

The previous configuration is also depicted in the following figures:

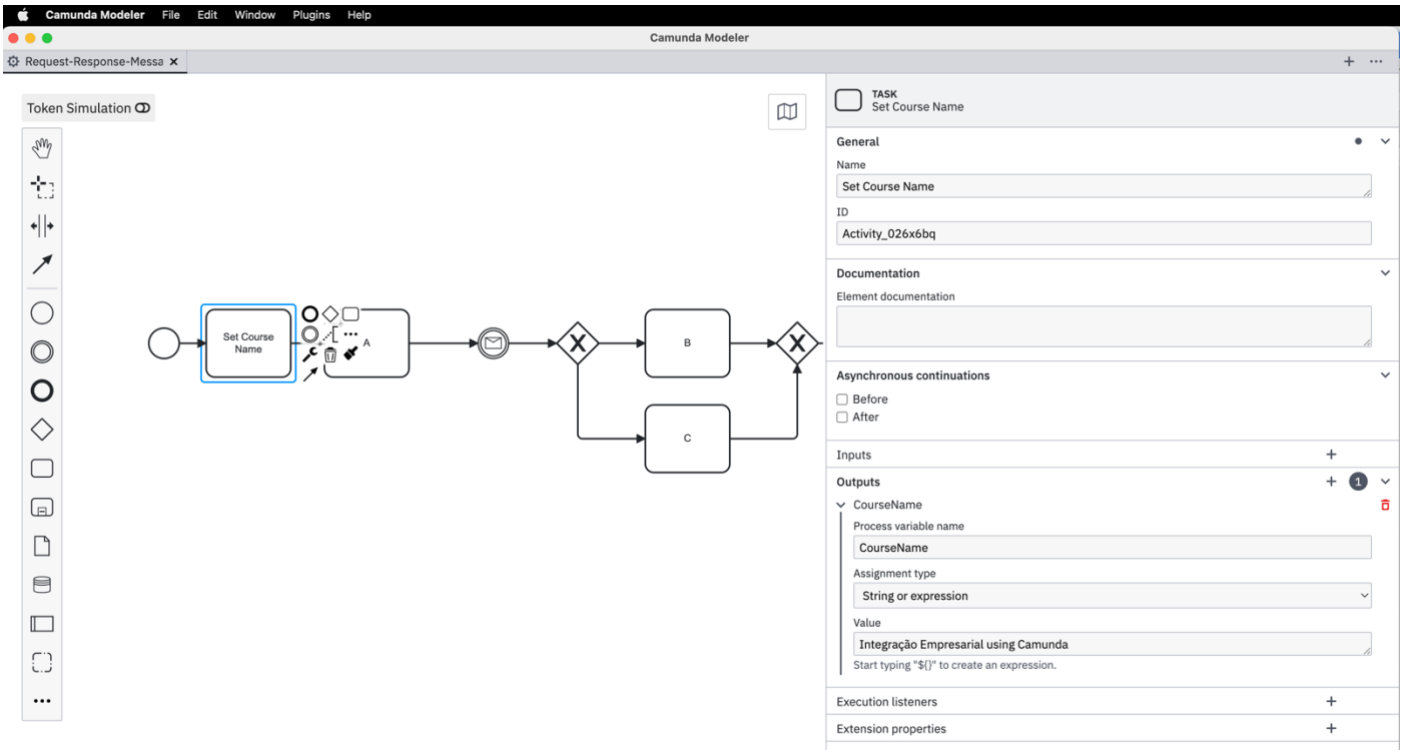

amunda Modeler File Edit Window Plugins Help
Camunda Modeler
SendMessageTask-WithPa
Token Simulation
Input Course Name
Call REST API
USER TASK
Input Course Name
General
Name
Input Course Name
ID
Activity_1chr38a
Documentation
Element documentation
User assignment
Assignee
demo
Candidate groups
Candidate users
Due date
The due date as an EL expression (e.g. ${someDate}) or an ISO date (e.g. 2015-06-26T09:54:00).
Follow up date
The follow up date as an EL expression (e.g. ${someDate}) or an ISO date (e.g. 2015-06-26T09:54:00).
Priority
Forms
Type
Generated Task Forms
Form fields
CourseName
ID
CourseName
Refers to the process variable name
Label
CourseName
Type
string
Default value
Constraints
Properties
Task listeners
Asynchronous continuations
Before
After
Inputs
Outputs
Execution listeners
XML
Camunda Platform 7.18
0 errors
Log
5.4.2


amunda Modeler File Edit Window Plugins Help
Camunda Modeler
SendMessageTask-WithPa
Token Simulation
Input Course Name
Call REST API
SERVICE TASK
Call REST API
General
Call REST API
ID
Activity_10sqvvz
Documentation
Element documentation
Implementation
Type
Connector
Connector ID
http-connector
Asynchronous continuations
Before
After
Inputs
Connector inputs
3
method
Local variable name
method
Assignment type
String or expression
Value
POST
Start typing "${}" to create an expression.
payload
Local variable name
payload
Assignment type
String or expression
Value
{
"name":"${CourseName}"
}
Start typing "${}" to create an expression.
url
Local variable name
url
Assignment type
String or expression
Value
https://gks3itxmr1.execute-api.us-east-1.amazonaws.com/test/helloworldpath
Start typing "${}" to create an expression.
Outputs
Connector outputs
Execution listeners
Extension properties
XML
Camunda Platform 7.18
0 errors
Log
5.4.2

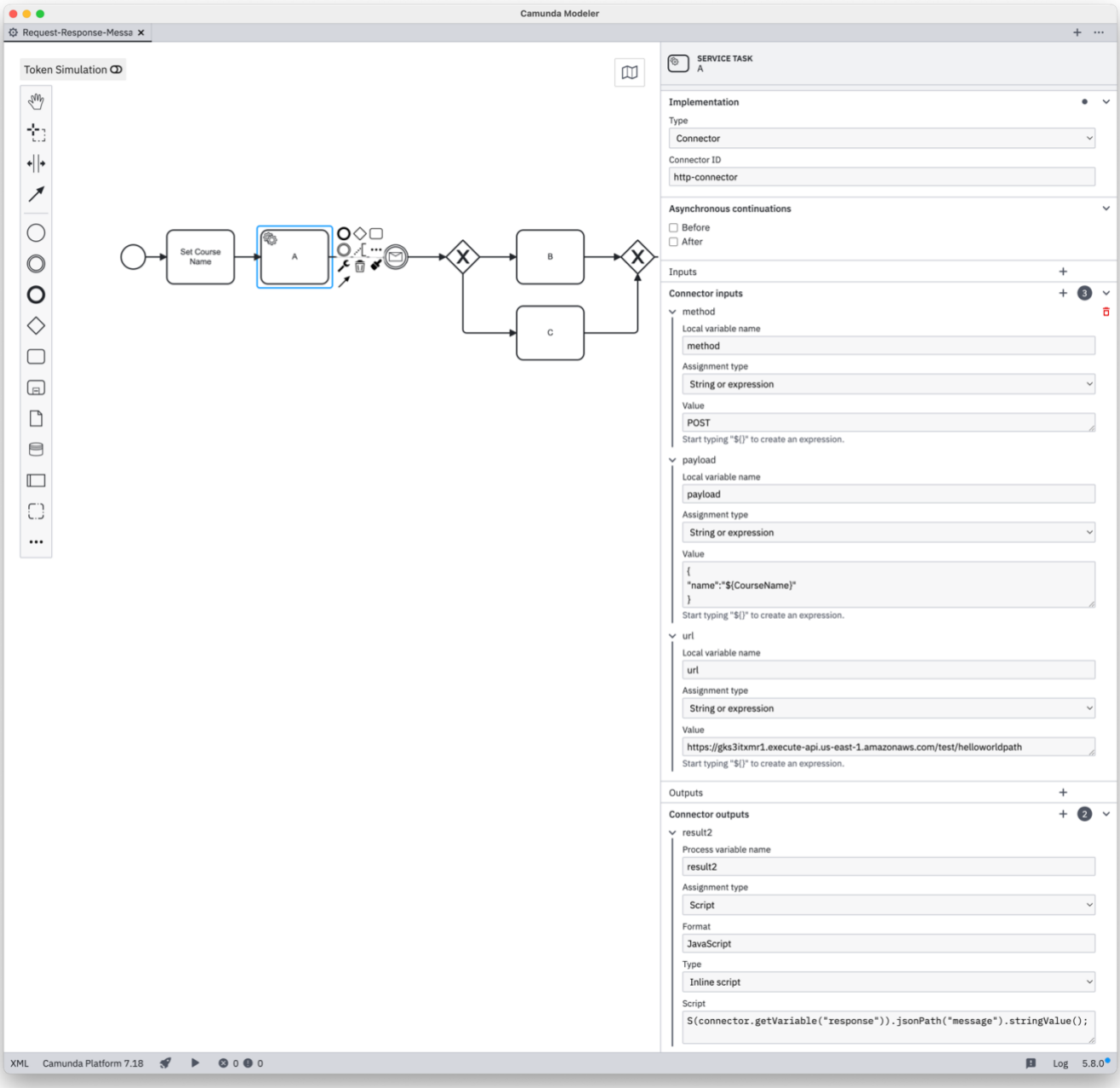
I. Integrate with a REST API installed remotely using Camunda http-connector – REQUEST & RESPONSE

For configuring the parsing of a REST API response, you may use inline *javascript* for parsing the received message, as depicted in the next figures. Many other JSON queries could be programmed, check reference manual at https://docs.camunda.org/manual/7.4/reference/spin/json/03-querying-json/.

Firstly, in this example, instead of receiving the course name from the user, we are adding the process variable "CourseName" as the output of first task, as depicted in the next figure.



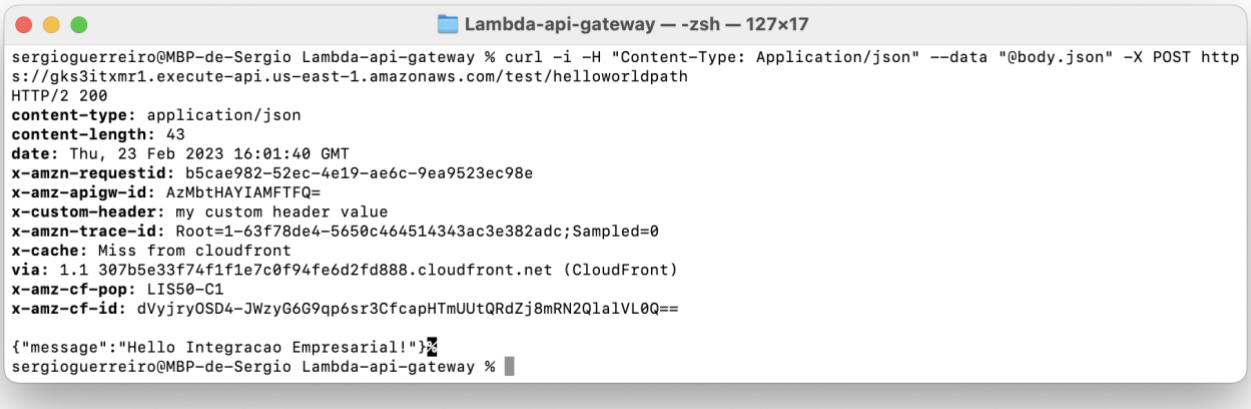
Secondly, the process variable "CourseName" is used in the payload of the connector inputs. Then, in the connector output you decode the fields that you like to receive from the response.



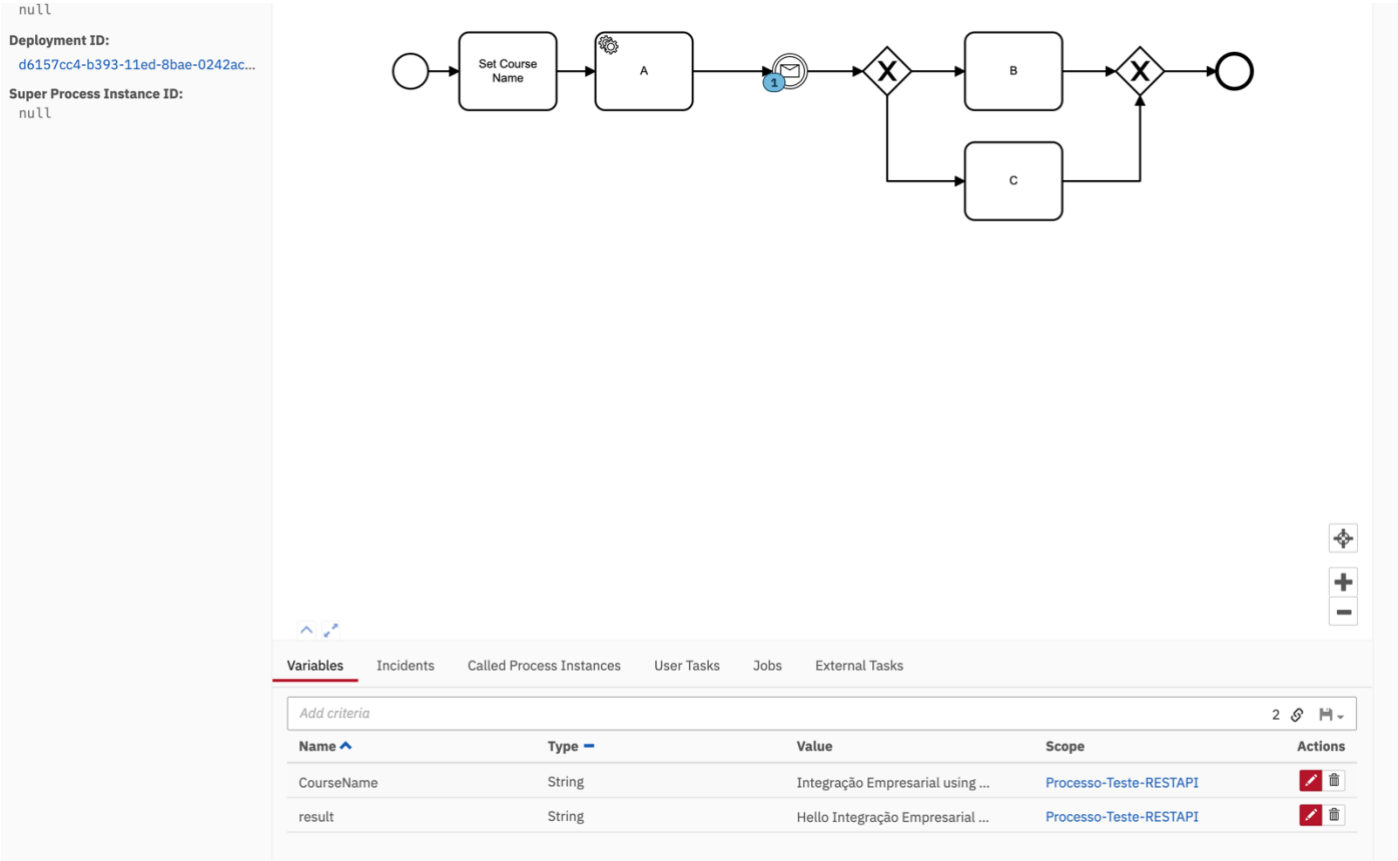
Notice that the inline script used in the connector output is reading "**message**" field from response and writing that to the **"result"** process variable:

```
S(connector.getVariable("response")).jsonPath("message").stringValue();
```

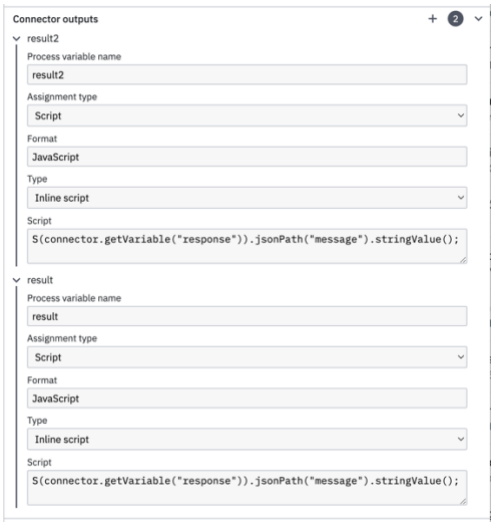
If you try to invoke your REST API using the curl command, you see the matching between the response and the name of the field "**message**".



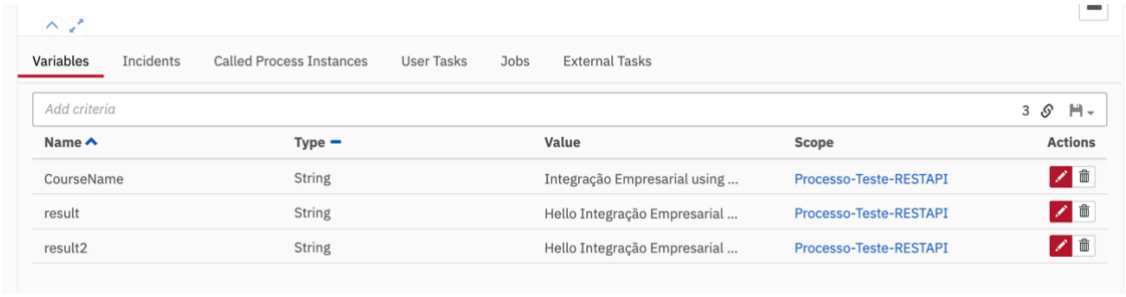
Moreover, in the Camunda cockpit, choose the process instance (an intermediate catch message event has been deliberately added to the process to hold it!), and verify that your instance now has the "result" process variable.



If you have many variables to retrieve from REST API response you only need to add more connector outputs with different process variable names, as depicted in the following figure.

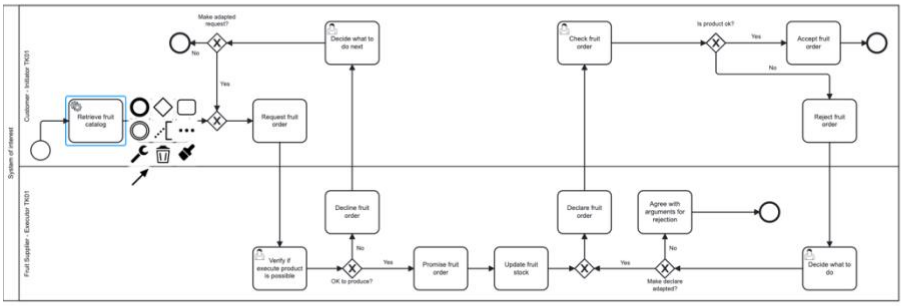


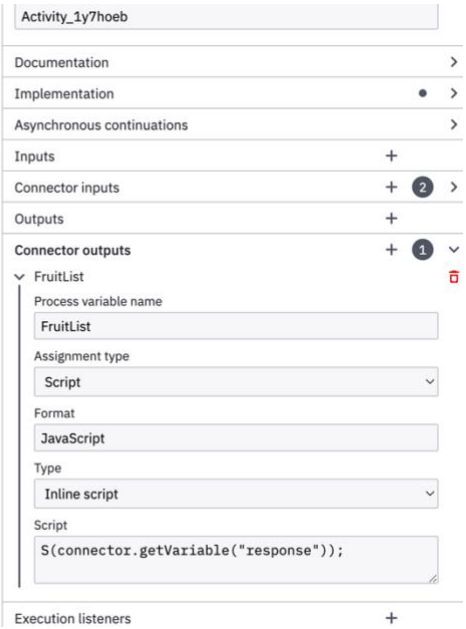
After that, you can check, in Camunda cockpit, that both variables are available.



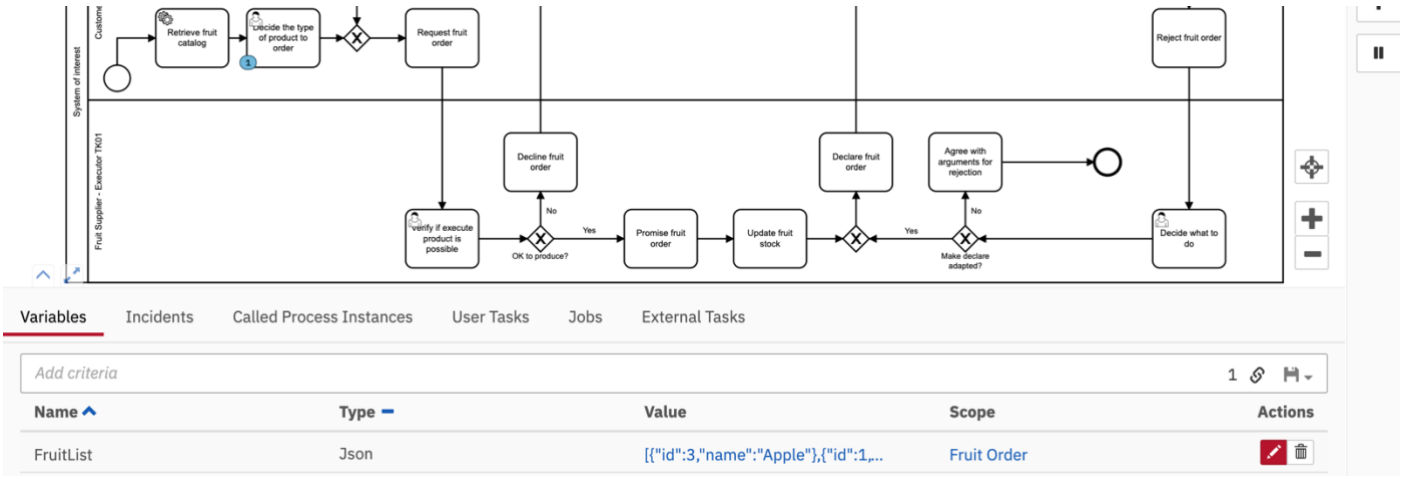
--*--

Hint: If you would like to debug a json message that has been received by a service task, you can use the following source code in the connector output:





And then, start an instance and check in the Camunda cockpit the message has been received correctly.

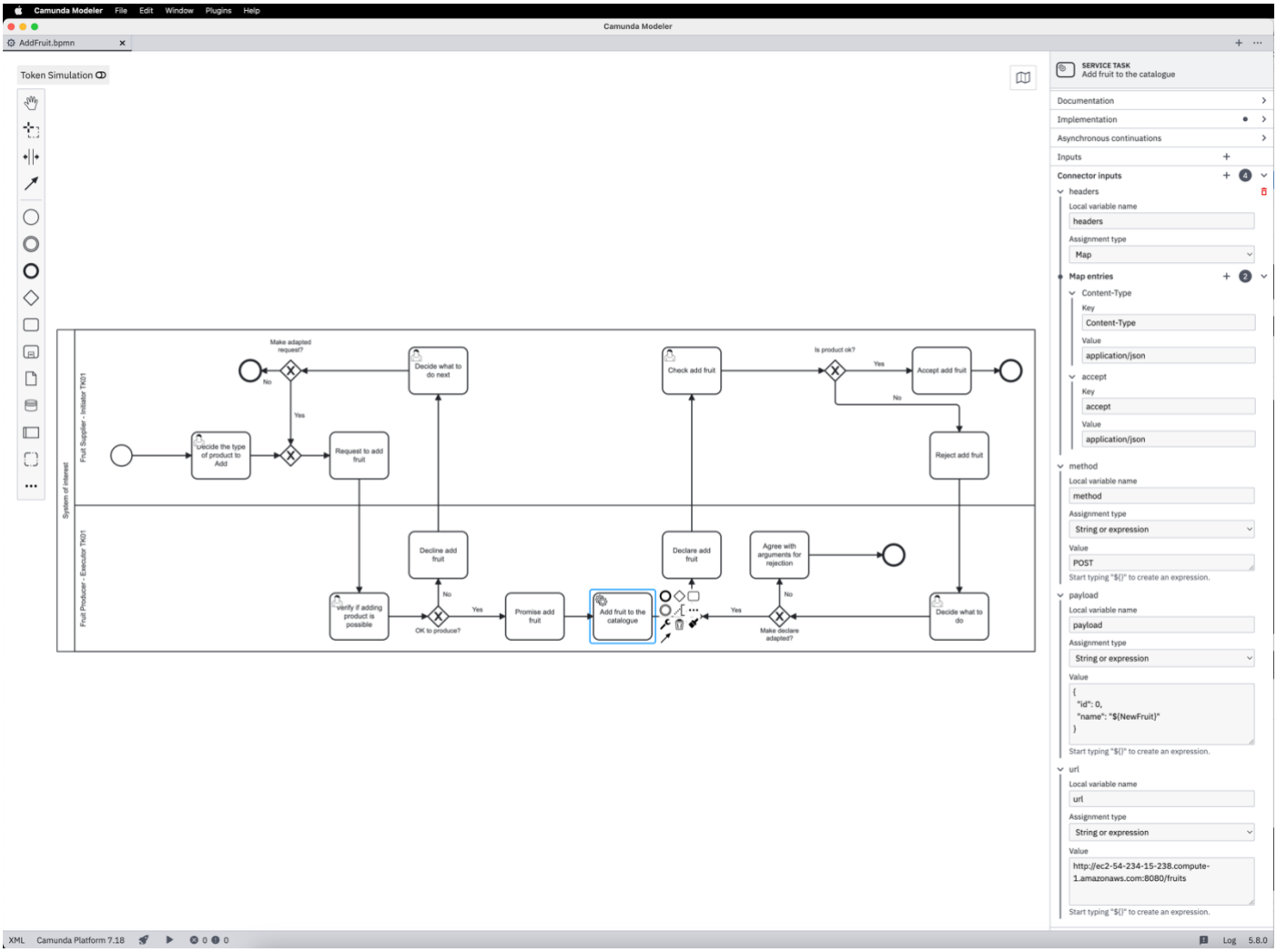


J. Integrate with a REST API installed remotely using Camunda http-connector – REQUEST & RESPONSE & Headers fields

When needed more information can be added to an API REST. For instance, if you need to specify the messages media, Headers can be included in the connector inputs. Considering the following POST request:

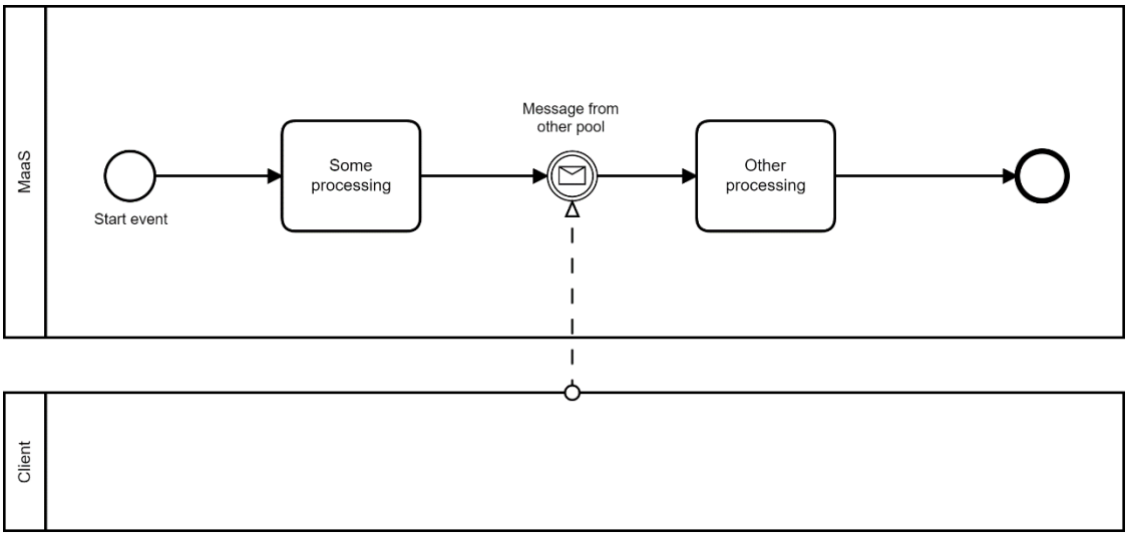
```
curl -X 'POST' \
  'http://ec2-54-234-15-238.compute-1.amazonaws.com:8080/fruits' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "id": 0,
  "name": "sd"
}'
```

The same can be configured in a Camunda service task accordingly:

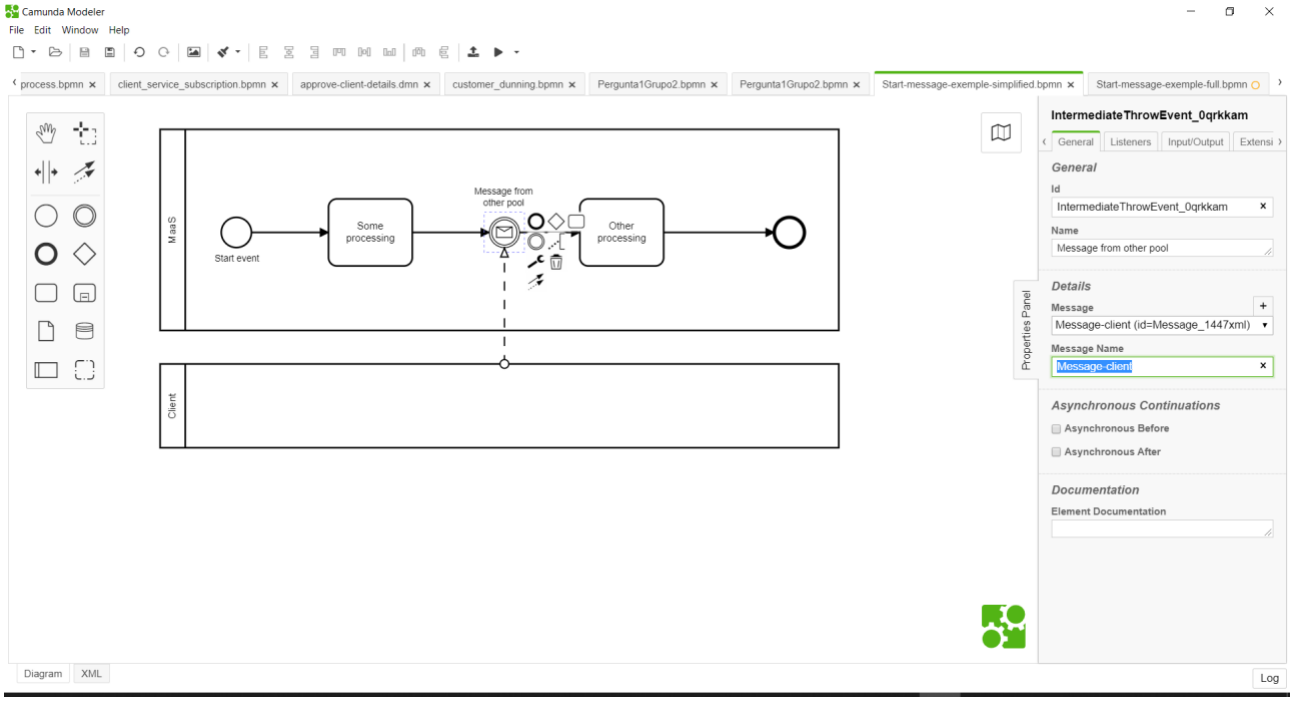


K. Using an intermediate message catch event inside a process.

L.1. Design the following collaboration in Camunda Modeller:

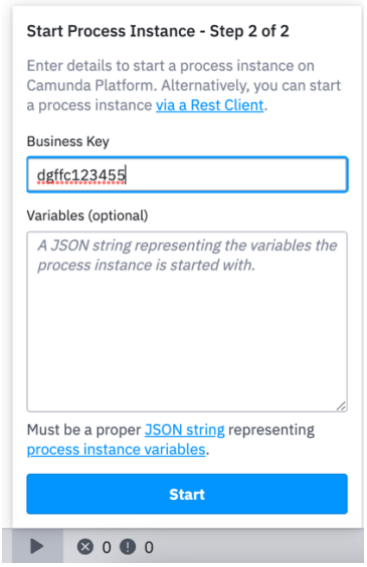


Where the intermediate message catch event should have a configuration similar with the following:

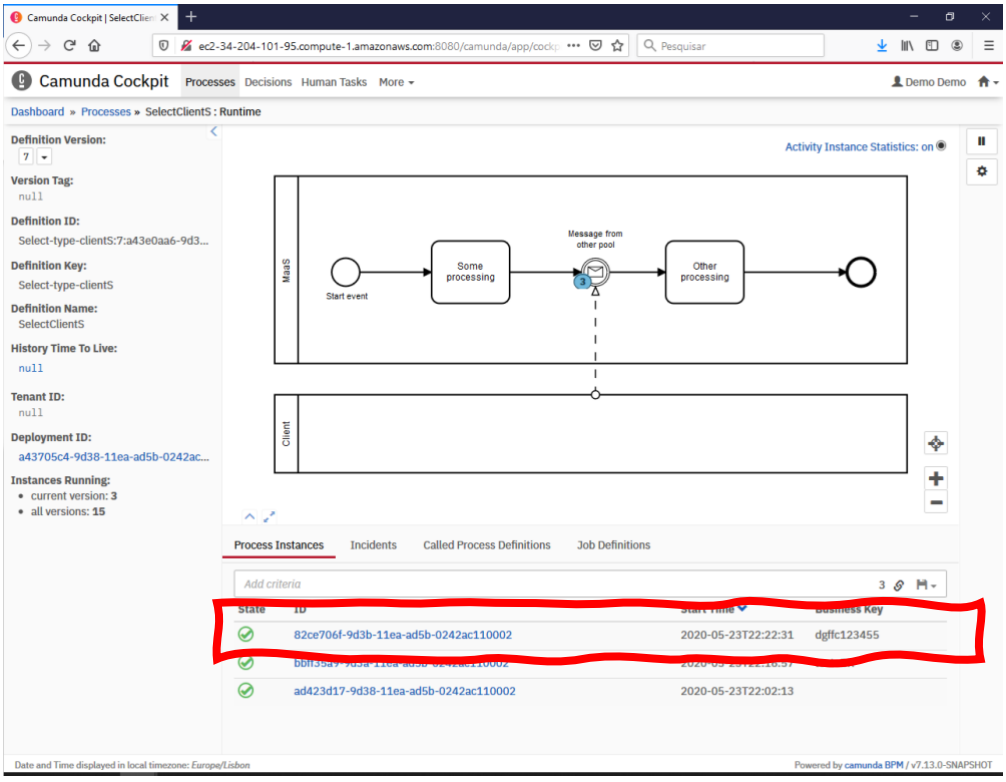


L.2. Deploy the model in your Camunda engine.

L.3. Start a new process instance with new configuration, where the **businessKey** is filled with any unique identifier.



You can verify the creation of the instance in the camunda cockpit. You will obtain something similar with the following business key as defined:



L.4. Then, the instance is expecting a message. To test the catch of a REST message, use a similar curl command:

```
curl -H "Content-Type: application/json" --data "@body.json" -X POST http://<YOUR CAMUNDA EC2 DNS>:8080/engine-rest/message
```

where bodyStart.json defines the message listener and the correlation with the key for your instance and sending some process variables:

```
{
  "messageName" : "Message-client",
  "businessKey" : "dgffc123455",
  "processVariables": {"amount": {"value":240,"type":"long"}},
  "resultEnabled" : true
}
```

References

Full official documentation:

- https://docs.camunda.org/manual/
- https://docs.camunda.org/javadoc/camunda-bpm-platform/7.12/

Testing the basics of the Camunda:

- https://docs.camunda.org/get-started/quick-start/
- Videos tutoriais: https://camunda.com/learn/videos/

URLs with other resources

- https://camunda.com/products/
- https://github.com/camunda/camunda-bpm-examples
- Send Task - https://docs.camunda.org/manual/7.7/reference/bpmn20/tasks/send-task/
- Receive task - https://docs.camunda.org/manual/7.7/reference/bpmn20/tasks/receive-task/
- https://github.com/camunda/camunda-bpm-examples/tree/master/servicetask/rest-service
- https://github.com/rob2universe/camunda-http-connector-example
- https://docs.camunda.org/manual/7.3/guides/user-guide/#process-engine-connectors
- **https://docs.camunda.org/manual/7.4/reference/spin/json/03-querying-json/**