

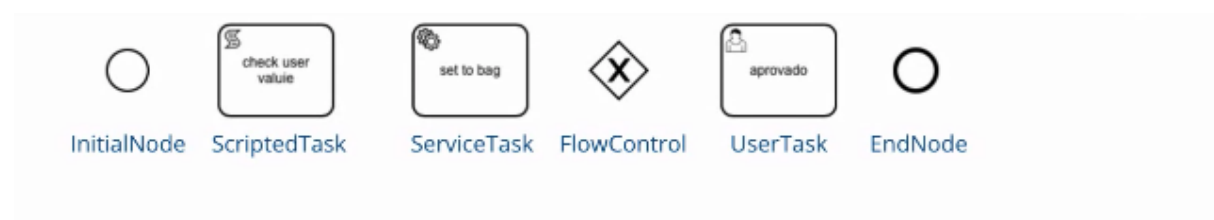
# Modelagem

## Blueprints

### Introdução às Blueprints

Blueprint é uma palavra de tradução simples, que pode assumir significados diferentes de acordo com o contexto. No nosso contexto, da utilização do FlowBuild para o desenvolvimento de software, podemos entender a blueprint como esquema. Um esquema que representa um processo. O processo desenhado a partir da necessidade de negócio. Este processo pode ter sido desenhado utilizando, por exemplo, a notação BPMN<sup>1</sup>. Assim, **a Blueprint é o artefato que descreve o processo de negócio para o Flowbuild Engine interpretar**. Ou seja, é um arquivo do tipo JSON<sup>2</sup> passível de interpretação pelo FlowBuild Engine. A Blueprint é responsável por especificar todos os elementos de um processo e orquestrar os canais e serviços.

[[ Recorte dos elementos do BPMN utilizados na blueprint



Para desenhar os fluxos, utilizamos alguns dos elementos do BPMN. A notação prevê muitos outros elementos que não são necessários no contexto do FlowBuild.]]

Assim como o BPMN, as blueprints seguem um padrão de escrita que identificam raias e nós que serão executados. Diferente do BPMN alguns nós serão utilizados para que haja consistência do sistema.

---

<sup>1</sup> A título de curiosidade, existe um padrão de notação em texto para BPMN que é baseado em XML ao invés de JSON ( <https://www.omg.org/spec/BPMN/2.0/> ) Este padrão é utilizado pela Camunda, desenvolvedora do Cawemo.

<sup>2</sup> JSON (JavaScript Object Notation), <https://www.json.org/json-en.html>

Para executar uma aplicação utilizando o Flowbuild é necessário que tenha pelo menos uma blueprint. No modo de execução, a blueprint é instanciada em um processo, este instanciamento garante que uma vez iniciado o Flowbuild executa o processo da blueprint até o final. Mesmo que essa blueprint seja atualizada. E a cada mudança de nós é gerado um process states. Ou seja, temos um processo de auditoria completa.

ATENÇÃO: TODAS AS STRINGS DA BLUEPRINT SÃO CASE SENSITIVE!

## Estrutura das Blueprints

A estrutura de uma blueprint tem os seguintes itens:

- name: texto com o nome da blueprint
- description: descrição sobre o processo que ela representa
- blueprint\_spec
  - lanes: raias, ou conjunto de nodes de um determinado ator;
  - nodes: nós, ou menor parte do processo;
  - prepare:
  - requirements:

Exemplo:

```
{  
  "name": "nome da blueprint",  
  "description": "descrição da blueprint",  
  "blueprint_spec": {  
    "lanes": {},  
    "nodes": {},  
    "prepare": {},  
    "environment": {},  
    "requirements": {}  
  }  
}
```

## Recomendação

A visão de negócio orientado a processos sugere que os processos em si documentam o negócio, assim todos os parâmetros de documentação devem ser utilizados para que tenhamos um melhor entendimento das blueprints. Os itens name e description servem muito mais a um olhar humano no que ao Flowbuild Engine.

Numa perspectiva de projeto com muitas blueprints, é recomendável que os nomes e descrições sejam de certa forma padronizados, para que todos os profissionais que escrevem blueprints sigam o mesmo racional.

## Raias ou Lanes

Uma raia é a expressão do controle de acesso a um conjunto de nós. Uma raia define quais regras devem ser atendidas para que o nó seja executado. A rigor as raias definem os atores, ou seja, um conjunto de características. Exemplo: raia Anônimo, qualquer um pode executar os nós.

Na prática a raia é relevante em apenas duas situações: no start e nas tarefas de usuário. O start verifica se as regras da raia estão sendo respeitadas. Os nós do tipo tarefa de usuário devem respeitar a raia que ele pertence, ou seja, para executar deve preencher as regras definidas na raia. Todos os nós devem estar numa raia. No caso dos nós de sistema não faz diferença qual raia ele está. Apesar disso, a representação do diagrama da blueprint ou mesmo seu entendimento em código é facilitado quando organizamos as raias com as devidas regras de quais atores podem executar.

Numa blueprint a raia é definida pelos parâmetros:

- id: identificador da raia, é uma string, não precisa ser numérico;

- name: nome da raia, tem a função de descrição para quem precisar ler a blueprint;
- rule: regra que define quem pode executar os nós daquela raia.

A rule, assim como o script da scriptTask, recebe código na sintaxe LISP. Deve ser uma função que retorna true/false dizendo se um dado usuário tem permissão de executar nodes dentro da lane. A rule é executada contra o token do usuário. A função da raia é sempre executada em tempo de execução.

Exemplo:

1 - Qualquer usuário pode acessar:

```
"lanes": [  
  {  
    "id": "1",  
    "name": "everyone",  
    "rule": ["fn", ["&", "args"], true];  
  }  
]
```

2 - Apenas usuários que criaram o processo podem acessar. Nesta raia não deve estar o startNode, já que as informações do usuário estão na bag e a bag só existe após o startNode.

```
"lanes": [  
  {  
    "id": "2",  
    "name": "creatorOnly",  
    "rule": ["fn", ["actor_data", "bag"],  
      ["-", ["get", "bag", ["`", "creatorid"]],  
        ["get", "actor_data", ["`", "actor_id"]]]]  
  }  
]
```

As Lanes existem para controlar o acesso de usuários ao processo, portanto são usadas para:

- Abrigar StartNodes, portanto controlar quem pode ativar o processo.
- Abrigar UserTasks, portanto controlar quem pode realizar aquela tarefa – seja um usuário específico ou um grupo de usuários.

Atualmente não existe controle para distinguir política de acesso (leitura ou ação). Não é recomendável criar lanes para representar setores ou simplesmente para agrupar tarefas de sistema ou flowNodes.

Lembre-se que as condições da lane dependem de:

- Dados do ator
- Regra da Lane
- Contexto do Processo (quando aplicável) → é importante que o contexto, uma vez definido no processo, não seja alterado.

Nomeie lanes usando a CONDIÇÃO. Caso a lane utilize algum parâmetro de contexto do processo, utilize o nome do atributo utilizado como condição. Mesmo que um processo possa ser executado por qualquer ator, é importante que a raia seja identificada como esta condição, um ator anônimo.

Em um projeto em que mais de um processo se relacionam é aconselhável que as raia de um mesmo ator utilizem o mesmo nome e a mesma definição. Assim facilitamos o entendimento e diminuimos a possibilidade de inconsistências no todo.

<precisamos incluir a explicação de claims>

## Nós ou Nodes

Um nó é a menor unidade do processo. Um nó é uma tarefa que deve ser executada. Toda vez que um nó executado ele gera um estado e grava esse estado numa estrutura de dados.

Existem vários tipos de nós.

Numa blueprint o nó deve ser definido pelos parâmetros:

- id: identificador do nó, deve ser único no contexto da blueprint;

- **name:** nome do nó, tem uma função descritiva para quem lê a blueprint, é a informação logada para análise futura, ou seja, na análise de logs a informação visível é o name;
- **lane\_id:** identifica a raia ou lane que contém o nó atual.
- **next:** indica qual o próximo nó será executado após a execução do nó atual. O valor do parâmetro next deve ser o id de outro nó. No caso de um nó de finalização este parâmetro deve ser null.
- **type:** define o comportamento que a aquele node terá, ou seja, os tipos de tarefas. Os tipos utilizados são: Start, Finish, UserTask, SystemTask, ScriptTask e Flow.
- **parameters:** dados de input para execução do nó.

Exemplo:

```
{  
  "id": "12",  
  "name": "Finish node",  
  "type": "Finish",  
  "lane_id": "1",  
  "next": null  
}
```

## IDs e Names

No FlowBuild, cada nó do fluxo tem um atributo name e um id. O atributo name não é utilizado em nenhum momento pelo motor de execução. Sua função é trazer entendimento para o público de negócio e para o designer. O id, por sua vez, é utilizado pelo motor de execução para identificar a sequência de tarefas. Isso não impede, contudo, que este campo não possa também ser utilizado para facilitar o designer no entendimento do fluxo de negócio.

A única restrição do ponto de vista do motor de execução é que o campo id seja único no fluxo (condição que atualmente é validada pelo FlowBuild no momento da criação do fluxo). Fora isso, é permitido utilizar qualquer tipo de texto neste campo.

É recomendado utilizar termos curtos no campo de id e, o uso de uma codificação é uma boa prática. Abaixo estão apresentadas algumas codificações utilizadas, com seus prós e contras.

Regra	Descrição	Prós	Contras
Sequência Numérica		Fácil entendimento do sequenciamento de tarefas	Não funciona em fluxos com flowNodes Pode gerar confusão com o step_number
ID do Caminho + Sequência Numérica	Similar ao caso anterior, com a inclusão de um prefixo para identificar o caminho feliz e as possíveis derivações	Resolve o problema de fluxos com FlowNodes. Identifica de forma clara o caminho feliz.	Em caso de inclusão de tarefas intermediárias, gera uma demanda de re-identificação de todos os nós subsequentes.
ID do tipo de nó + Sequência Numérica	Nesse caso o prefixo prioriza o tipo de nó e a sequência numérica está relacionada ao tipo de nó e não ao processo.	Funciona com fluxos muito ramificados, onde não existem caminhos bem definidos ou há muito entrelaçamento de ramos.	O sequenciamento por tipo de nó pode causar confusão de negócio.
Id de Lane + ID do tipo de Nó + Sequência Numérica	Busca criar, através do id uma coordenada cartesiana do nó (lane = Y, sequencia = X)	Funciona para fluxo pouco ramificados. Identificação rápida quando lido pela blueprint.	A codificação é longa e bastante vinculada a características da tarefa.

Toda vez que um nó termina ele coloca as informações no result, para que tres nós a frente consiga usar é só colocar na bag.

### Eventos (Start e Finish)

StartNodes e FinishNodes são os dois tipos de eventos existentes no FlowBuild para iniciar e finalizar um processo. A recomendação para os nomes desses nodes é que reflitam o objetivo do processo. Assim estes eventos podem ser nomeados usando o modelo ENTIDADE + STATUS, onde:

- StartNodes é nomedo em função das condições de ativação do processo.
  - FinishNodes é nomeado em função da condição de saída do processo.
- o Caso o processo tenha estados finais alternativos, crie um finishNode para cada estado.
- o Evite criar 2 finishNodes para representar um mesmo estado.

Exemplos:

- Usuário Cadastrado
- Contrato Criado
- Senha atualizada

## StartNode

StarNode ou nó de início. É o primeiro nó do processo a ser executado.

Precisamos falar sobre o inputSchema, que é um parâmetro que pode ser enviado no startNode para controlar os dados de entrada para início do workflow.

Exemplo:

```
{  
  "id": "1",
```



```
{  
  "name": "Start Node",  
  "lane_id": "1",  
  "next": "2",  
  "type": "Start",  
}
```

O StartNode gera dois estados: um quando é criado e outro quando é executado. Ele é o único tipo de nó que gera mais de um estado.

## FinishNode

FinishNode ou nó de finalização. É o último nó a ser executado no processo. Indica que o processo foi encerrado. Neste caso o parâmetro next deve ser null.

Exemplo:

```
{  
  "id": "12",  
  "type": "Finish",  
  "name": "Finish node",  
  "lane_id": "1",  
  "next": null,  
}
```

## UserNode

UserNode ou nó de usuário. Este tipo de nó descreve uma interação com um canal, que normalmente é uma interface com um humano, e requer que uma ação seja executada para continuar o fluxo. Ou seja, os nós de usuário executam uma ação que solicita uma resposta interativa e aguardam por esta interação.

- parameters: parâmetros necessários para executar a action;
- input: são os inputs necessários para executar a action;
- action: nome da action. Esse nome deve ser combinado com o desenvolvedor Front-end;

Exemplo:

```
{
  "id": "2",
  "name": "user task",
  "lane_id": "1",
  "next": "3",
  "type": "UserTask",
  "parameters": {
    "input": {
      "purposes": {
        "$ref": "bag.purposes"
      },
      "occupations": {
        "$ref": "bag.occupations"
      }
    },
    "action": "FillFormSelects"
  }
}
```

Dica:

Se você está na equipe Lendico, tem uma página do Confluence com todas as actions listadas:  
<https://lendicobr.atlassian.net/wiki/spaces/TL/pages/182747141/A+es+do+Workflow+para+o+Frontend>

## SystemNode

SystemNode ou nó de sistema. Ele é base para todos os tipos de nodes de tarefas automáticas, como: ServiceNode, DecisionNode, SetToBagNode e ScriptNode. Assim não existe na blueprint um nó apenas com o type SystemTask, este tipo de nó usará um novo parâmetro category para indicar o subtipo. Estes nós executam o que está descrito nos parâmetros e não bloqueiam o fluxo de execução.

As categorias são:

- HTTP

- SetToBag
- StartProcess
- Timer

## ServiceNode

ServiceNode ou nó de serviço é o nó que executa um serviço.

- category: indica o sub-tipo de SystemNode.
- input:
- request:
  - url:
  - verb:
  - headers:

Exemplo:

```
{
  "id": "2",
  "name": "Call occupations",
  "next": "3",
  "lane_id": "1",
  "type": "SystemTask",
  "category": "http",
  "parameters": {
    "input": {},
    "request": {
      "url": "https://lists-v2.lendico.com.br/api/occupations",
      "verb": "GET",
      "headers": {
        "ContentType": "application/json"
      }
    }
  }
}
```

## SetToBagNode

SetToBagNode ou nó de memória. A bag é capaz de armazenar dados que podem ser acessados pelos nós a qualquer momento no fluxo. Este nó armazena o dado na bag. É uma forma de persistir dados.

Se pensarmos a blueprint como uma linguagem de programação que será executada pelo FlowBuild, o setToBagNode é equivalente a sintaxe de declaração de variáveis.

- category: indica o sub-tipo de SystemNode.

Exemplo:

```
{
  "id": "8",
  "name": "Set register response to bag",
  "next": "9",
  "type": "SystemTask",
  "category": "setToBag",
  "lane_id": "1",
  "parameters": {
    "input": {
      "registerUserResponse": {
        "$ref": "result"
      }
    }
  }
}
```

## StartProcess

É o nó que inicializa outro workflow. O próximo nó deverá ser o nó de finalização.

Exemplo:

```
{
  "id": "22",
  "name": "Start process node",
  "next": "99",
```

```

    "type": "SystemTask",
    "lane_id": "1",
    "category": "startProcess",
    "parameters": {
      "input": {
        "cpf": {
          "$ref": "bag.cpf"
        },
        "name": {
          "$ref": "bag.name"
        },
        "income_id": {
          "$ref": "bag.income_id"
        }
      },
      "actor_data": {
        "$ref": "actor_data"
      },
      "workflow_name": "ATV_UPLOAD_PAYSLIP"
    },
  },

```

## Timer

timeout especifica o tempo que será esperado para executar o nó.

Exemplo:

```

{
  "id": "6",
  "type": "SystemTask",
  "category": "timer",
  "name": "Timer node",
  "parameters": {
    "input": {},
    "timeout": 60
  },

```

```
"next": "4",
"lane_id": "1"
},
```

## ScriptNode

ScriptNode ou nó de script é o nó capaz de executar um script.

Idealmente os serviços devem ser escritos fora do Flowbuild, mas é possível avaliar ou testar um processo executando scripts menos complexos escritos na blueprint.

- input:
- script:

Exemplo:

```
{
  "id": "4",
  "name": "Script task node",
  "next": "5",
  "type": "ScriptTask",
  "lane_id": "1",
  "parameters": {
    "input": {
      "features": {"$ref": "result.data.features"}
    },
    "script": {
      "function": ["fn", ["input", "&", "args"],
        ["js", ["str", ["`", "let features = ",
          ["get", "input", ["`", "features"]],
          ["`", "; let result = {}; let is_istantor_on =
features.find(feature => feature.name === 'instantor'); result.is_istantor_on =
Boolean(is_istantor_on); result;"]]]]
        ]
      ]
    }
  }
}
```

## FlowNode

FlowNode ou nó de fluxo. Este tipo de nó direciona o fluxo para diferentes nós de acordo com os dados processados pelo nó antecessor.

- next: o parâmetro next indica as opções de nós para seguimento. Para cada opção deve ser identificado o id do nó que deverá seguir o fluxo;
- parameters
  - input
    - decision

Exemplo:

```
{
  "id": "9",
  "type": "Flow",
  "name": "Is User Registered successfully",
  "lane_id": "1",
  "next": {
    "default": "10",
    "201": "11",
    "206": "11"
  },
  "parameters": {
    "input": {
      "decision": {
        "$ref": "bag.registerUserResponse.status"
      }
    }
  }
}
```

O input de um FlowNode tem que ser uma string. Normalmente é a resposta do nó anterior. Essa entrada em forma de texto é comparada com as opções descritas no Next. O nó que será executado é o resultado desta comparação. Se o parâmetro do input não for reconhecido ou for um objeto vazio, ele será convertido em "undefined".

Nesta versão do Flowbuild não existe o conceito de paralelismo, ou seja, um nó iniciar dois nós na sua seqüência.

## Prepare

O campo prepare tem por objetivo descrever funções a serem executadas antes da inicialização do processo.

## Environment

O campo enviroment tem por objetivo trazer para a blueprint as variáveis de ambiente da aplicação. É possível definir quantas variáveis forem necessárias.  
Como estas variáveis são utilizadas depois???

```
"environment": {  
  "BASE_URL": "BASE_URL"  
},
```

## Requirements

O campo requirements tem por objetivo listar pacotes de scripts para utilização em scriptTasks. É uma forma de importar para a blueprint pacotes específicos de execução.

```
"requirements": [  
  "core"  
]
```

## Sobre o \$

\$ indica que um comando será executado.

\$Ref: Referencia, busca o valor de uma variável;

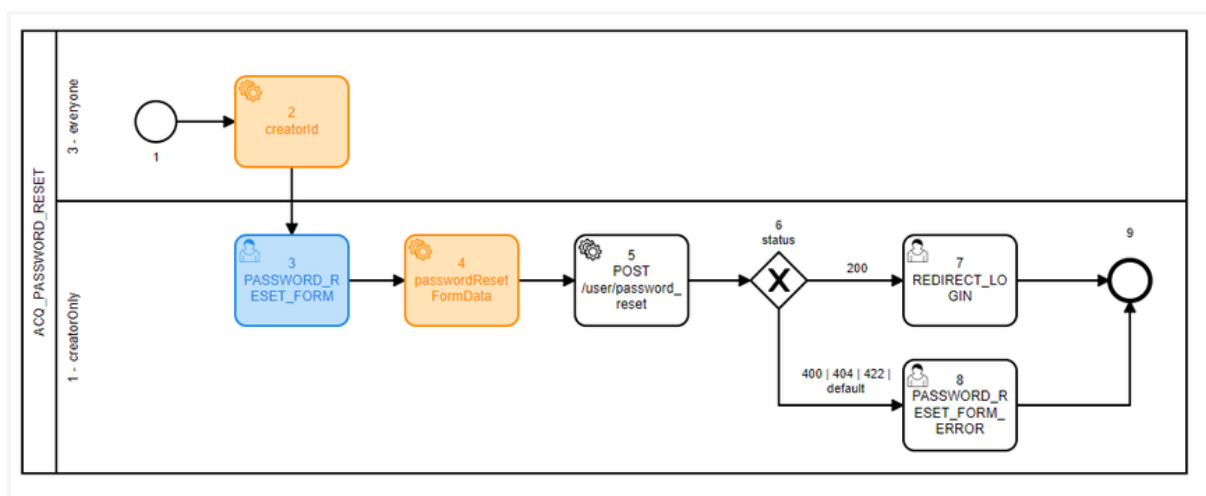
\$JS: JavaScript, executa um comando javascript; <https://www.javascript.com/>

\$Mustache: insere uma variável na string; <http://mustache.github.io/>

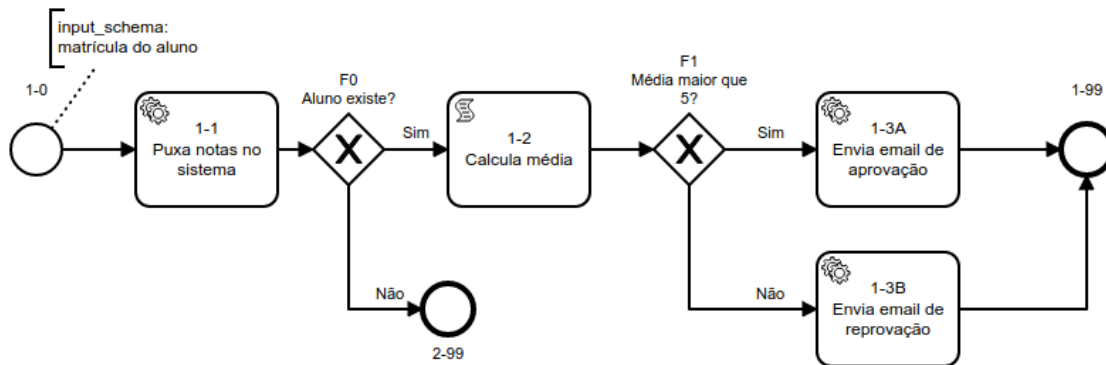


## Cases

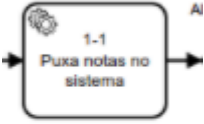
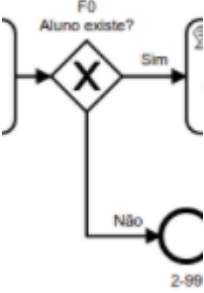
Antes de escrever uma blueprint, é desenhado um fluxo para representar o negócio. Este fluxo segue a notação BPMN, como na figura abaixo. Esta representação gráfica facilita a comunicação da equipe uma vez que é de mais fácil compreensão.


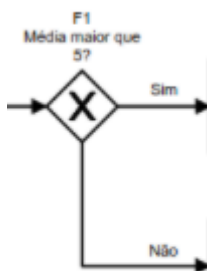
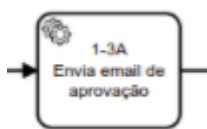


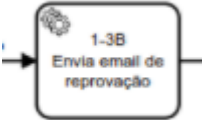
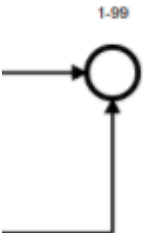
## Exemplo 1



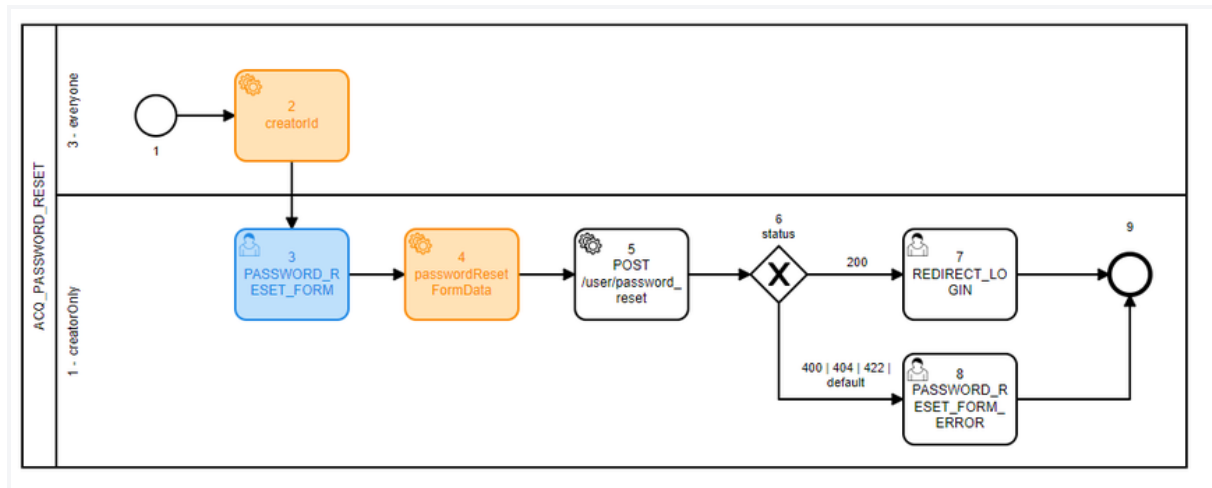
	<pre>const blueprint_spec = {   environment: [],   requirements: ['core'],   prepare: [],   lanes: [ {     id: '1',     name: 'default',     rule: ['fn', ['&amp;', 'args'], true], },   ],   nodes: [</pre>
	<pre>{   id: '1-0',   type: 'Start',   name: 'Start node',   parameters: {     input_schema: {       matricula: { type: 'string' },       required: ['matricula'],     },   },   next: '1-1',   lane_id: '1', },</pre>

<p>chama: la do aluno</p> 	<pre> {   id: '1-1',   next: 'F0',   lane_id: '1',   type: 'SystemTask',   category: 'setToBag',   name: 'Salvar collections na bag',   parameters: {     input: {       // url: { \$mustache: 'http://localhost:1337/api/v0/notas/{{bag.matricula}}' },       notas: { \$ref: 'result' },     },     request: {       verb: 'GET',       url: { \$mustache: 'http://localhost:1337/api/v0/notas/{{bag.matricula}}' },       headers: {         ContentType: 'application/json',       },     },   }, } </pre>
	<pre> {   id: 'F0',   type: 'Flow',   name: 'A matrícula é válida?',   next: {     success: '1-2',     default: '2-99',   },   lane_id: '1',   parameters: {     input: {       decision: {         \$ref: 'result.data.status',       },     },   }, } </pre>

	<pre>     },   }, </pre>
	<pre> {   id: '1-2',   next: 'F1',   lane_id: '1',   type: 'SystemTask',   category: 'setToBag',   name: 'Calcula média',   parameters: {     input: {       media: { \$js: '({ result }) =&gt; result.data.message.reduce((t, n, i) =&gt; (t * i + n) / (i + 1), 0)', },     },   }, }, </pre>
	<pre> {   id: 'F1',   type: 'Flow'   name: 'O aluno foi aprovado?',   next: {     true: '1-3A',     false: '1-3B',     default: '1-99',   },   lane_id: '1',   parameters: {     input: { decision: { \$js: '({bag}) =&gt; bag.media &gt; 5', }, },   }, }, </pre>
	<pre> {   id: '1-3A',   next: '1-99',   lane_id: '1',   type: 'SystemTask',   category: 'setToBag',   name: 'Envia email de aprovação', } </pre>

	<pre>parameters: {   input: {     email: 'Você foi aprovado!',   }, }, },</pre>
	<pre>{   id: '1-3B',   next: '1-99',   lane_id: '1',   type: 'SystemTask',   category: 'setToBag',   name: 'Envia email de reprovação',   parameters: {     input: {       email: 'Você foi reprovado!',     },   }, },</pre>
	<pre>{   id: '1-99',   type: 'Finish',   name: 'Finish node',   next: null,   lane_id: '1', },</pre>
	<pre>], }</pre>

## Exemplo 2



```

{
  "name": "ACQ_REQUEST_PASSWORD_RESET",
  "description": "Request password flow",
  "blueprint_spec": {
    "lanes": [
      {
        "id": "3",
        "name": "everyone",
        "rule": [
          "fn",
          [
            "&",
            "args"
          ],
          true
        ]
      },
      {
        "id": "1",
        "name": "creatorOnly",

```

```
    "rule": [  
      "fn",  
      [  
        "actor_data",  
        "bag"  
      ],  
      [  
        "=",  
        [  
          "get",  
          "bag",  
          [  
            "\",",  
            "creatorId"  
          ]  
        ],  
        [  
          "get",  
          "actor_data",  
          [  
            "\",",  
            "actor_id"  
          ]  
        ]  
      ]  
    ],  
    "nodes": [  
      {  
        "id": "1",  
        "name": "Start Node",
```

```
"next": "2",
"type": "Start",
"lane_id": "3",
"parameters": {
  "input_schema": {}
},
{
  "id": "2",
  "name": "Set to bag node",
  "next": "3",
  "type": "SystemTask",
  "lane_id": "3",
  "category": "setToBag",
  "parameters": {
    "input": {
      "creatorId": {
        "$ref": "actor_data.actor_id"
      }
    }
  },
  {
    "id": "3",
    "name": "wait for token to be sent",
    "next": "4",
    "type": "UserTask",
    "lane_id": "1",
    "parameters": {
      "input": {
        "current_user": {
          "$ref": "bag.creatorId"
```



```

    }
  },
  "action": "REQUEST_PASSWORD_RESET_FORM"
}
},
{
  "id": "4",
  "name": "Validate url token",
  "next": "5",
  "type": "SystemTask",
  "lane_id": "1",
  "category": "HTTP",
  "parameters": {
    "input": {
      "$ref": "result.activities[0].data"
    },
    "request": {
      "url": {
        "$mustache":
"https://api.{{environment.BASE_URL}}/user/password_reset"
      },
      "verb": "POST",
      "headers": {
        "ContentType": "application/json"
      }
    }
  }
},
{
  "id": "5",
  "name": "Check if the url token is valid",
  "next": {

```

```
"200": "6",
"400": "7",
"403": "7",
"404": "7",
"409": "7",
"default": "7"
},
"type": "Flow",
"lane_id": "1",
"parameters": {
  "input": {
    "decision": {
      "$ref": "result.status"
    }
  }
},
},
{
  "id": "6",
  "name": "Redirect user to denied approval page",
  "next": "8",
  "type": "UserTask",
  "lane_id": "1",
  "parameters": {
    "input": {
      "current_user": {
        "$ref": "bag.creatorId"
      }
    }
  },
  "action": "REQUEST_PASSWORD_RESET_FORM_SUCCESS"
},
},
```

```
{
  "id": "7",
  "name": "Redirect user to allowed approval page",
  "next": "4",
  "type": "UserTask",
  "lane_id": "1",
  "parameters": {
    "input": {
      "current_user": {
        "$ref": "bag.creatorId"
      }
    }
  },
  "action": "REQUEST_PASSWORD_RESET_FORM_ERROR"
},
{
  "id": "8",
  "name": "Finish node",
  "next": null,
  "type": "Finish",
  "lane_id": "1"
}
],
"prepare": [],
"environment": {
  "BASE_URL": "BASE_URL "
},
"requirements": [
  "core"
]
}
```

## Anexos

<https://documenter.getpostman.com/view/2387160/TVRrW5Jw>

<https://codebeautify.org/jsonviewer>