Relatório da implementação de Processo de Negócio

Caio Silvestre Almeida da Silva
Guilherme Estevam Ferreira Putzeys
João Pedro Rodrigues Camargo
Tsuyoshi Yodogawa

São Paulo
2021

Introdução

BPMS (Business Process Management Systems, ou ainda, Sistemas de Gestão de Processos de Negócios) é um sistema criado para o gerenciamento de processos, onde os usuários conseguem gerenciar seus processos. O BPMS é um sistema cujo intuito é ser flexível no gerenciamento dos processos.

Similar ao ERP (Enterprise Resource Planning), o BPMS tem o objetivo de gerenciar processos pertencentes aos sistemas, com a diferença de ser mais técnico/específico.

Abordagem

O BPMS é dividido em quatro subgrupos:

1. Groupware system - Softwares para gestão de grupos, mais especificamente para a comunicação dos usuários.
2. Ad hoc workflow sytems (Ad hoc - caso a caso) - Para cada caso, uma implementação, não havendo um padrão.
3. Production workflow systems - Trabalho roteado estritamente com base em modelos de processo.
4. Case management systems - Oferece suporte a processos que não são especificados de maneira rígida nem completa.

Dentre as quatro formas de se orquestrar processos de serviços, ou seja, efetuar a automação dos processos de negócio, o grupo utilizou a terceira, Production workflow systems. Há um plano claro para o fluxo de sua implementação, já que é baseado estritamente nos modelos de processo, i.e., uma vez que o processo de negócio foi modelado, analisado e refatorado, sua implementação é basicamente linear. E a outra vantagem, é que por ser a forma mais predominante em BPMS, há um vasto material para consultas (com guias e discussões em fóruns).

Tecnologias utilizadas

- Para a codificação do sistema utilizamos a linguagem Java
- Para a construção do BPMN utilizamos o Camunda Modeler
- Para a execução do modelo BPMN utilizamos o Camunda Run

Modelagem BPMN

Para que fosse possível realizar a execução do BPMN tivemos que realizar algumas alterações no nosso modelo para que o mesmo pudesse ser aceito pelo "Camunda Run", plataforma a qual utilizamos para realizar a execução. Alteramos algumas partes do nosso BPMN que estão em anexo na ModelagemFinal.bpmn, para que o nosso modelo pudesse rodar, dividimos as piscinas em arquivos diferentes e deployando simultaneamente, assim gerando o fluxo do modelo.

Seguindo a divisão de arquivos implementadas no camunda, nossa modelagem dentro código java, também dividimos as mensagens cada uma delas definidas por suas ações, então temos o Cliente para a Corretora, Cliente para o banco, Corretora para o Cliente, Corretora para o Banco, Banco para o Cliente e Banco para a Corretora. As mensagens realizam as conversações entre as piscinas, para isso utilizamos o "java class" para que fosse possível a utilização de uma classe java.

Para a passagem sobre os gateways XOR, utilizamos os próprios formulários que existem hoje na plataforma do camunda, com eles conseguimos definir palavras chaves que farão com que o usuário responda e defina o caminho do XOR ao qual ele irá seguir.

Ferramenta de modelagem e execução

Utilizamos a plataforma "Camunda" tanto para modelar como para executar o processo de negócio.

O Camunda Run é o sistema que faz a execução do modelo BPMN, o sistema permite que as tarefas sejam executadas de diferentes formas, sendo elas:

1. Internal Service tasks: Que realizam a chamada do código juntamente com o processo do modelo BPMN, de forma síncrona. Sendo apenas necessário a realização do deploy de ambos.
2. External tasks: Fornece uma lista de atividades que podem ser consultadas pelos workers (programas que realizam a execução de alguma atividade).

O fluxo de execução de tarefas externas pode ser conceitualmente separado em três etapas, conforme ilustrado na imagem a seguir:

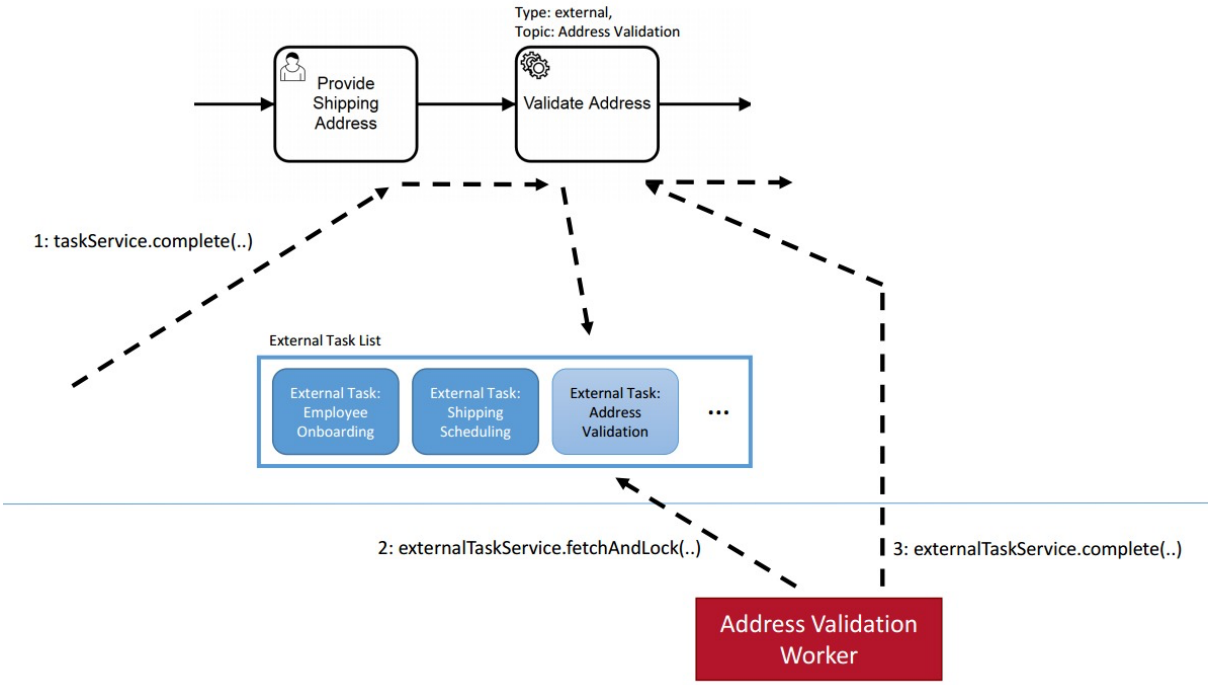

Figura 1

Process Engine é a Criação de uma instância de tarefa externa, *Trabalhador externo* é buscar e bloquear tarefas externas, *Trabalhador externo e mecanismo de processo* é a instância de tarefa externa completa.

Quando o mecanismo de processo encontra uma tarefa de serviço que está configurada para ser tratada externamente, ele cria uma instância de tarefa externa e a inclui em uma lista de tarefas externas (etapa 1).

A instância da tarefa recebe um tópico que identifica a natureza do trabalho a ser executado. Em um momento no futuro, um trabalhador externo pode buscar e bloquear tarefas para um conjunto específico de tópicos (etapa 2).

Para evitar que uma tarefa seja buscada por vários trabalhadores ao mesmo tempo, uma tarefa tem um bloqueio baseado em carimbo de data/hora que é definido quando a tarefa é adquirida. Somente quando o bloqueio expira, outro trabalhador pode buscar a tarefa novamente. Quando um trabalhador externo conclui o trabalho desejado, ele pode sinalizar ao mecanismo de processo para continuar a execução do processo após a tarefa de serviço (etapa 3).

As tarefas externas são conceitualmente muito semelhantes às tarefas do usuário. Ao tentar entender o padrão de tarefa externa pela primeira vez, pode ser útil pensar sobre ele em analogia às tarefas do usuário.

As tarefas do usuário são criadas pelo mecanismo de processo e adicionadas a uma lista de tarefas. O mecanismo de processo então espera que um usuário humano consulte a lista, solicite uma tarefa e a conclua. As tarefas externas são semelhantes, uma tarefa externa é criada e adicionada a um tópico. Um aplicativo externo então consulta o tópico e bloqueia a tarefa. Depois que a tarefa é bloqueada, o aplicativo pode trabalhar nela e concluí-la.

Para nossa modelagem, utilizamos a segunda opção (External tasks), que no nosso código será implementado como 'Delegation Code' ou como 'Script'

Como dito previamente, a tecnologia utilizada para a codificação do nosso projeto foi a linguagem Java, por isso exemplificamos como ficaria a Delegação com Java.

Delegação com Java

Para a implementação do código, a classe responsável deve implementar (especificamente o método 'execute(DelegateExecution dE)') a interface JavaDelegate, encontrada na lib 'org.camunda.bpm.engine.delegate'. Dado um processo que será executado fará a chamada do método (seu código) e a atividade irá ficar na forma padrão do BPMN 2.0.

```java
@Override
public void execute(DelegateExecution delegateExecution) throws Exception {

    delegateExecution.getProcessEngineServices().getRuntimeService()
            .createMessageCorrelation( s: "TratamentoBanco")
            .correlate();
}
```

Figura 2

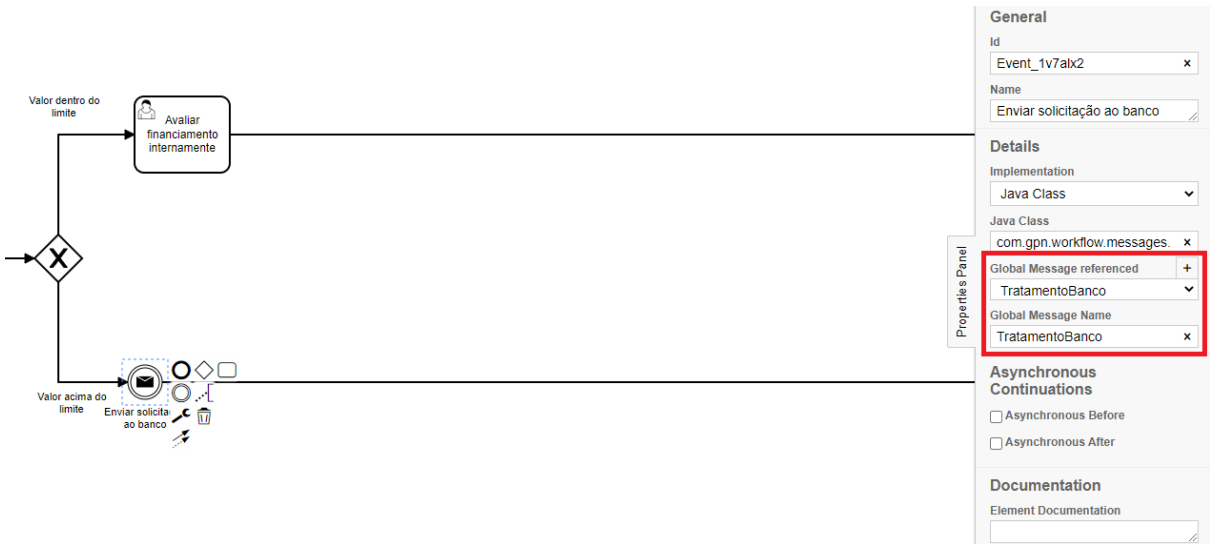

Figura 3

Essa lib também é utilizada para criar o código responsável para ligar os eventos de piscinas diferentes através dos eventos de envio e recebimento de mensagem. Logo, organizamos o código de modo que cada piscina consiga se comunicar com as outras.

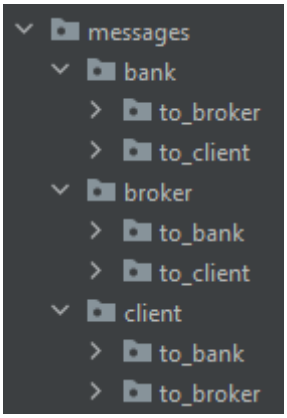

Figura 3

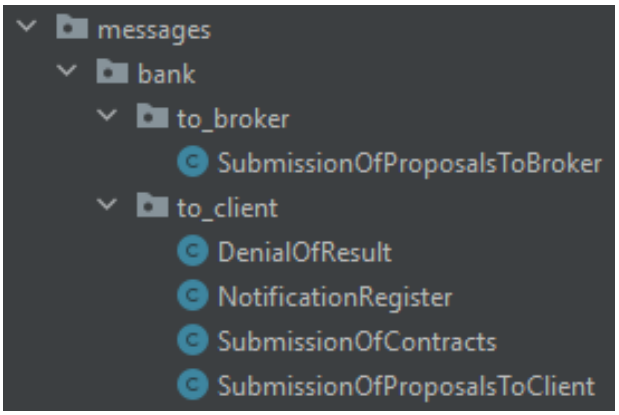

Figura 4

Essa correlação é feita através do método createMessageCorrelation() (figura 2)

Decisão do algoritmo (ProcessServer)

A corretora pode operar dentro do limite de R$ 50.000,00, caso o valor do financiamento ultrapassar o limite, o cliente é redirecionado a operar diretamente com o Banco.

A base do cálculo para aprovação do financiamento pelo banco é que o ***custo anual*** precisa ser inferior ou igual à diferença entre o ***rendimento anual*** com ***10% do valor de financiamento***. Logo:

CustoAnual ≤ RendimentoAnual - (0,1 x ValorFinanciamento)

	Cenário de aprovação	Cenário de rejeição
Rendimento anual	50.000,00	50.000,00
Valor do financiamento	200.000,00	200.000,00
Custo anual	30.000,00	31.000,00

Solução Arquitetural:

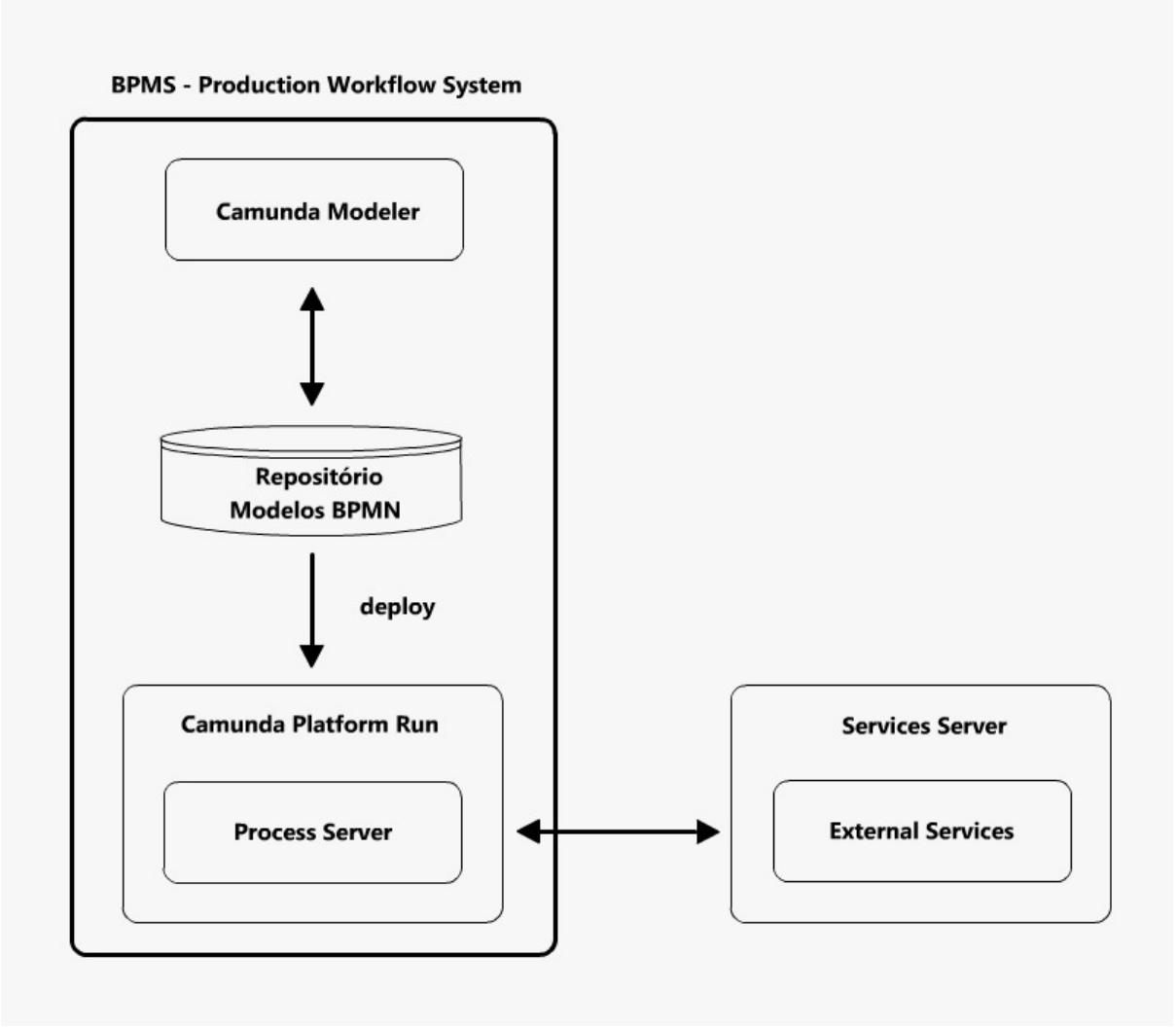

Figura 5

Serviços remotos usados (para quais tarefas)
Servidores
Diagrama (uml por exemplo)
Limitações encontradas

Cliente

Tentativas de interface de cliente foram feitas, mas não conseguimos finalizar. As implementações acarretaram em um uso muito grande do tempo. A primeira tentativa foi pelo Java RMI:

- ClientFinancingRequest
    - Java RMI
    - JDK 11
    - JavaFX

Ilustração do módulo ClientFinancingRequest por Java RMI

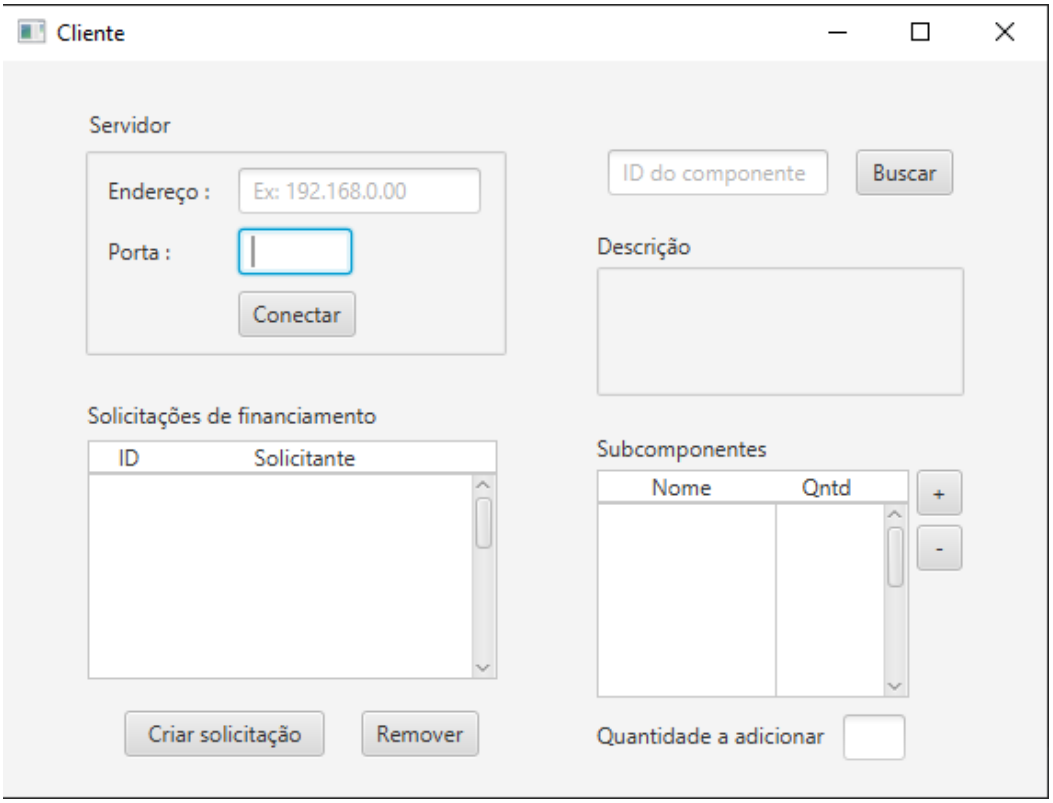

Figura 6

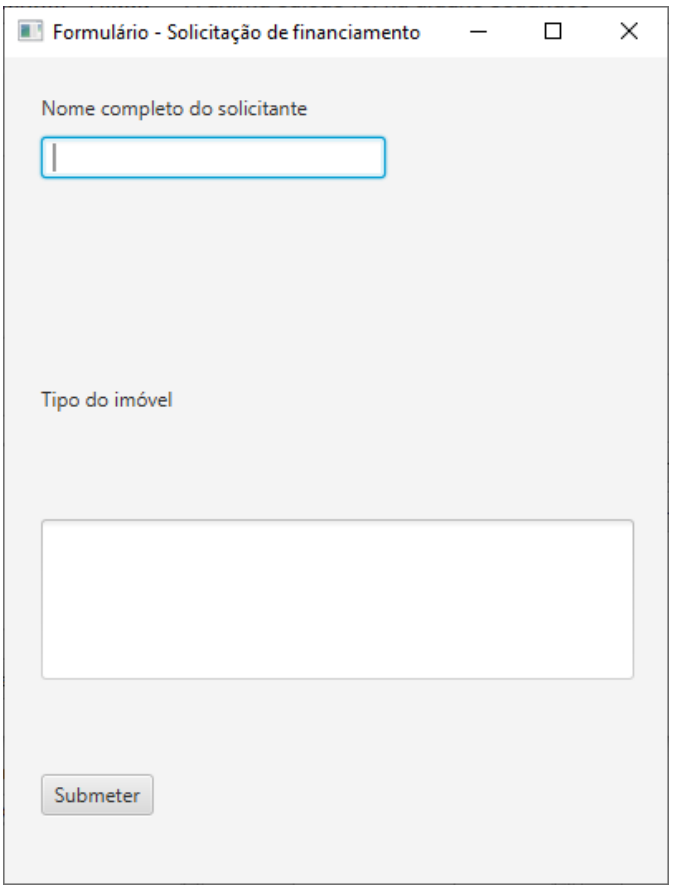

Figura 7

Outra tentantiva foi com o uso do framework Spring Boot, com as configurações:

- Client (FinancingRequest)
    - Configuração do projeto: Spring Initializr (start.spring.io)
    - Spring Boot: versão 2.6.1
    - JDK: versão 11.0.12
    - Sistema de build: Maven 4.0.0
    - Gerenciador de dependências: Maven 4.0.0

FinancingRequest

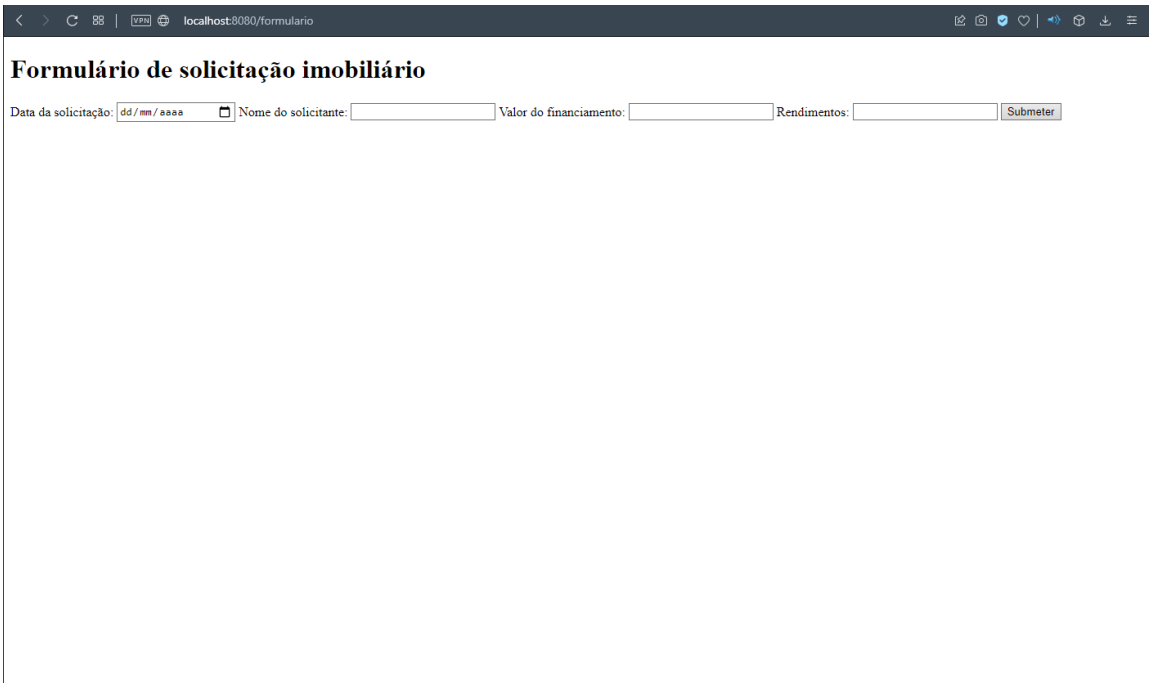

Figura 8

Referências:

- https://docs.camunda.org/get-started/quick-start/service-task/#configure-the-service-task
- https://docs.camunda.org/manual/latest/user-guide/process-engine/external-tasks/