# **Fluxo Funcional da Aplicação JANUS 2 – Visão Geral da Arquitetura Operacional e de Interface**

## **🎯 Objetivo do Sistema**

O **JANUS 2** é uma aplicação web que permite a usuários-chave do tribunal visualizar, lançar e acompanhar **automações processuais inteligentes**, com foco na integração com o **PJe**, utilizando **requisições HTTP como via principal**, e **Playwright como fallback** quando a automação por requisição falhar.

A plataforma é altamente observável, interativa e responsiva, fornecendo logs em tempo real, histórico completo e controle total do processo de automação.

## **🧱 Stack Tecnológica da Solução**

| **Camada** | **Tecnologia** |
| --- | --- |
| **Frontend** | Angular + UIKIT |
| **Backend API** | Java + Spring Boot |
| **Mensageria/Fila** | RabbitMQ |
| **Banco de Dados** | PostgreSQL |
| **Automação Fallback** | Playwright (Java) |
| **Observabilidade** | OpenTelemetry + ELK Stack + Grafana + Jaeger |
| **Orquestração** | Docker + Kubernetes |
| **CI/CD** | GitLab + gitlab-ci |

## **🖥️ Descrição Funcional Detalhada do Fluxo da Interface Web**

A aplicação web é projetada para usuários-chave com o seguinte fluxo principal:

### **🔷 Tela Inicial – Painel de Servidores**

* Apresenta **blocos por servidor ativo**.
* Cada bloco mostra:  
  + Nome do servidor
  + Status atual (online, executando, inativo)
  + Total de automações em execução
  + Botão: ▶ Iniciar automação

### **🔍 Clique sobre um servidor**

Exibe os **detalhes do servidor**:

* Fila atual de automações agendadas ou em execução (via RabbitMQ).
* Lista de automações anteriores com:  
  + Nome do robô
  + Status final (sucesso, erro, fallback)
  + Duração
  + Botão para 📄 Ver logs em tempo real
  + Botão para ⬇️ Download de logs históricos

## **🤖 Lançamento de Automação**

### **🧩 Ação: Iniciar automação**

1. **Modal de Seleção:**
   * Escolher o **servidor de destino**
   * Escolher o **robô (automação)**
2. **Resumo da Automação:**
   * Exibido de forma elegante e descritiva.
   * Exemplo:  
       
       
      "Robô Sentença Final – Lança automaticamente a movimentação 874 no PJe após validar minuta, documentos obrigatórios e se o processo está apto. Prioriza via API e utiliza Playwright como fallback."
3. **Botão Confirmar Execução:**
4. Ao clicar, o frontend envia uma **mensagem estruturada via API REST para o backend**, que publica na **fila RabbitMQ**:  
     
    json  
   CopiarEditar  
   {
5. "servidor": "janus-salvador-01",
6. "robot": "sentenca\_final",
7. "usuario": "joao.silva",
8. "timestamp": "2025-06-23T10:32:00Z",
9. "preferencia": "HTTP",
10. "fallback": "PLAYWRIGHT"
11. }

## **📦 Execução da Automação via Worker**

Cada worker, ao consumir uma tarefa da fila:

1. **Tenta executar via requisição HTTP (API externa ao PJe ou Codex)**
   * Exemplo: POST /pje/lancar-movimento
2. **Se falhar (timeout, status != 200, exceção de negócio)**
   * A automação **cai automaticamente para o fallback Playwright**
   * O robô abre a interface do PJe no navegador, simula o fluxo e executa a tarefa
3. **Ambas as execuções (HTTP ou Playwright):**
   * Geram logs em tempo real enviados via WebSocket ou long polling ao frontend
   * Logs estruturados com níveis (INFO, WARN, ERROR)
   * Cada log possui timestamp e contexto da automação

## **📊 Logs e Histórico**

1. Durante a execução, o usuário acompanha logs diretamente na interface:  
     
   [10:32:05] INFO - Validando processo via HTTP
2. [10:32:06] ERROR - API retornou erro 400: Processo Bloqueado
3. [10:32:07] WARN - Iniciando fallback via Playwright...
4. [10:32:10] INFO - Robô acessou PJe com sucesso
5. ...
6. [10:32:30] INFO - Movimento lançado com sucesso

* Após finalização:  
  + Logs ficam disponíveis para download em .txt ou .json
  + Estado final é registrado no PostgreSQL

## **🧠 Visão da Arquitetura Executora (Execução Típica)**

ascii

CopiarEditar

1. Usuário (Web Angular)
2. ↓
3. [Confirma execução]
4. ↓
5. [Backend Spring Boot]
6. ↓ (envia mensagem)
7. [RabbitMQ]
8. ↓
9. [Worker Executor Java]
10. ↓
11. [1] Tenta Requisição HTTP → OK? → Fim ✔
12. ↓
13. [2] Se falha → Executa Playwright (Java)
14. ↓
15. [3] Logs em tempo real → Frontend
16. ↓
17. [4] Logs persistidos + histórico → PostgreSQL + Elastic

## **📑 Objetivo da Arquitetura**

Garantir:

* **Robustez**: fallback automático em caso de falha
* **Rastreabilidade**: logs completos, detalhados e auditáveis
* **Escalabilidade**: múltiplos workers e servidores com filas independentes
* **Autonomia para o usuário**: lançamento sob demanda
* **Observabilidade total**: via ELK, Grafana e Jaeger