# **Proposta de Migração – JANUS 2 (Arquitetura Web Padrão CNJ)**

## **Componentes Tecnológicos Atualizados**

1. **Backend**: Java + Spring Boot
2. **Automação**: Requisições HTTP como principal + Playwright (Java) como fallback
3. **Filas e Orquestração**: RabbitMQ
4. **Logs e Observabilidade**: OpenTelemetry + Elastic Stack + Grafana + Jaeger
5. **Frontend**: Angular + UIKIT com consumo via API REST

## **🔄 Fluxo de Migração Proposto**

### **1. Documentação da Situação Atual**

* Mapeamento completo dos fluxos de automação.
* Identificação dos pontos manuais no sistema.
* Levantamento das regras e validações existentes no JANUS atual.

### **2. Desenvolvimento da API REST com Spring Boot**

* Refatoração e criação dos endpoints necessários.
* Aplicação de boas práticas RESTful.
* Integração com Codex e outros sistemas via HTTP.
* Segurança com autenticação JWT (CNJ).

### **3. Criação da Interface Angular (UIKIT)**

* Construção de painel para:  
  + Acompanhamento da fila de tarefas.
  + Monitoramento dos status de execução.
  + Execução manual de tarefas em casos críticos.
* Visualização de logs e status em tempo real via REST.

### **4. Implementação da Automação**

* **Prioridade: execução via requisições HTTP REST**.
* **Fallback: execução via Playwright (Java)** para simular interação na interface do PJe.
* Conversão gradual dos scripts atuais em Selenium para arquitetura modular com Page Object Model no Playwright.

### **5. Setup de Filas e Workers (RabbitMQ)**

* Toda automação (HTTP ou fallback) será enviada como mensagem para o RabbitMQ.
* Workers Java assíncronos processarão a fila, com:  
  + Tenta via HTTP.
  + Se falhar: executa Playwright.
* Suporte a reprocessamento, dead-letter e priorização.

### **6. Logs e Observabilidade**

* Instrumentação com OpenTelemetry.
* Armazenamento de logs estruturados no Elastic Stack.
* Dashboards com:  
  + **Kibana**: análise textual de eventos e exceções.
  + **Grafana**: métricas de execução e performance.
  + **Jaeger**: tracing distribuído por processo/tarefa.

## **📚 Documentação Técnica do Projeto JANUS 2**

### **1. Visão Geral**

O **JANUS 2** é um sistema de automações jurídicas com integração ao **PJe** e **Codex**, utilizando inteligência artificial e automação para auxiliar julgadores e servidores na tramitação de processos eleitorais.

#### **Módulos:**

* **Automação**: comandos HTTP ou interface via Playwright.
* **Extrator**: backend em Spring Boot responsável pelas regras de decisão.

### **2. Nova Arquitetura Tecnológica**

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

### **3. Mapeamento dos Fluxos Atuais (BPMN e UML)**

#### **BPMN – Fluxo Simplificado**

scss

CopiarEditar

Início

↓

[Obter Lista de Processos]

↓

[Filtrar por Classe/Tarefa]

↓

[Validar Aptidão]

↓

(É apto?)

↙️ Sim ↘️ Não

[Lançar via HTTP] [Logar Exceção]

↓

(Falhou?)

↙️ Sim ↘️ Não

[Fallback Playwright] [Logar Resultado]

↓

Fim

#### **UML – Diagrama de Classes**

**Automation Module**

* AutomationController  
  + recuperarProcessos()
  + lancarMovimento()
* PageObjects  
  + LoginPage
  + TarefaPage
  + MovimentoPage

**Extrator Module**

* ProcessoController
* DocumentoProcessoService
* CodexController

### **4. Documentação dos Endpoints REST (Spring Boot)**

| **Endpoint** | **Método** | **Descrição** |
| --- | --- | --- |
| /janus/processos/recuperarPorNumero/{numero} | GET | Recupera processo específico |
| /janus/processos/aptoMinutarSentenca | POST | Valida aptidão para minuta |
| /janus/processos/aptoLancarMovimentoSentenca | POST | Valida aptidão para lançar movimento |
| /janus/codex/processos/{numero} | GET | Recupera ID do Codex |
| /janus/codex/processoDocumentos/{numero}/{tipo} | GET | Recupera documentos do processo |

### **5. DTOs e Serviços Refatorados**

| **Classe Atual** | **DTO Proposto** | **Serviço Proposto** |
| --- | --- | --- |
| Processo | ProcessoDTO | ProcessoService |
| Documento | DocumentoProcessoDTO | DocumentoProcessoService |
| MovimentoTPU | MovimentoDTO | MovimentoService |
| Assunto | AssuntoProcessoDTO | AssuntoService |
| Candidato | CandidatoDTO | CandidatoService |
| TipoDocumento | TipoDocumentoDTO | DocumentoService |

### **6. Regras de Negócio Capturadas**

* **Bloqueio de Processo**: impedir execução se estiver bloqueado.
* **Validação de Documentos**: exige pré-condições documentais.
* **Movimentação Condicional**: depende de aptidão REST.
* **Minuta de Sentença**: liberada apenas com todos os documentos corretos.

### **7. Regras Técnicas Avançadas**

* **Enums e Classes TPU/Sinapses**: controle rigoroso de tipagem.
* **Tratamento de Exceções HTTP**: padrão para logs e auditoria.
* **Sincronização Codex ↔ PJe**: confirmação de integridade documental.
* **Automação Condicional por Zona**: escalonamento geográfico por tribunal.

### **8. Recomendações Finais para Refatoração**

* Separar totalmente automação (Playwright) e backend (Spring Boot).
* Utilizar filas RabbitMQ para desacoplamento e performance.
* Instrumentar com OpenTelemetry desde o início.
* Manter arquitetura orientada a DTOs e serviços.
* Automatizar testes com JUnit, Mockito e Playwright.
* Integrar análise estática (SonarQube) e CI/CD com GitLab.

### **✅ Próximos Passos**

* Montar protótipo inicial (Spring Boot + Playwright).
* Testar orquestração com Docker + Kubernetes.
* Validar fluxo de fallback com RabbitMQ.
* Criar primeiro dashboard com logs e tracing.
* Iniciar piloto com processo real do Codex.