# 1. Introdução

A utilização de um padrão arquitetural de plataforma enseja a redução de tempo destinado ao desenvolvimento de software, bem como majora a qualidade do mesmo.

O presente documento apresenta o escopo de especificações relativas às tecnologias envolvidas na arquitetura de plataforma referencial para o desenvolvimento de novas aplicações Web no Poder Judiciário do Estado do Rio de Janeiro.

A escolha arquitetural sobre os itens apresentados será evidenciada em função das características intrínsecas ao projeto a ser desenvolvido.

# 2. Microserviços

Uma arquitetura de microserviços é uma abordagem para construção de softwares que decompõem modelos de domínio de negócios em contextos delimitados e consistentes, implementados por serviços que operam isolados e autônomos, através de mecanismos eficientes de comunicação para o fornecimento integral das funcionalidades de negócio. Os microserviços são tipicamente implementados e operados por pequenas equipes com autonomia suficiente para mudar sua implementação interna com impacto mínimo para o resto do sistema.

Deste modo, podem ser evoluídos ou substituídos de modo independente pois cada serviço executa seu próprio processo, que deve ser independente e possuir uma interface bem definida diretamente sobre o protocolo HTTP.

Este conjunto de serviços, em uma atuação integrada, denota as funcionalidades desejadas para a aplicação como um todo (FOWLER-LEWIS, 2014). A independência entre os serviços permite o lançamento de novas versões com maior frequência, contendo um número menor de novas funcionalidades, evitando desta forma o acúmulo de inúmeras funcionalidades que acabam gerando dificuldades na validação de uma versão (APPLE, 2014). Esta arquitetura propicia naturalmente o uso de integração contínua no ciclo de desenvolvimento de software.

## 2.1. Independência tecnológica

Neste modelo arquitetural os serviços podem ser desenvolvidos em diferentes plataformas tecnológicas. Desta forma, cada serviço será implementado com base na tecnologia mais propícia para a resolução do problema. (HOFF, 2014)

## 2.2. Comunicação

Para que os serviços possam se comunicar se faz necessário um acordo de interface padronizado para a troca de informações. Recomenda-se a utilização de um padrão capaz de operar diretamente sobre o protocolo HTTP, como os Web Services REST (FOWLER-LEWIS, 2014).

## 2.3. Resiliência

Atingida através da replicação, contingência, isolamento e delegação. As falhas podem ser isoladas em serviços e isso garante que as partes do software que apresentem problemas possam se recuperar sem ocasionar o comprometimento de todo o sistema (BONÉR et al, 2014).

## 2.4. Vantagens

Desacoplamento das funcionalidades de um software, desenvolvimento de módulos independentes, escalabilidade distinta para cada serviço, maior confiabilidade dos serviços, refletem integralmente as funcionalidades do sistema devido à separação lógica dos serviços, contém inúmeras tecnologias e podem ser substituídas sem grande impacto (SIMONE, 2014).

## 2.5. Limitações

A arquitetura de microserviços possui também algumas limitações ou desafios que impõem dificuldades na implementação, manutenção, instalação e monitoramento. Alguns exemplos (HOFF, 2014; DVORKIN, 2015):

Custo operacional – serviços representam um custo extra para instalação das aplicações nos servidores, comparando-se à arquitetura monolítica. Crescimento potencial com o aumento dos serviços. Necessidade de recursos de orquestração, um processo complexo demandante de um conhecimento prévio de ferramentas para esta tarefa.

Monitoramento - Monitorar dezenas de serviços, por vezes, torna-se bastante complexo devido às inúmeras tecnologias empregadas.

# 3. Arquitetura Monolítica

As principais plataformas destinadas ao desenvolvimento de aplicações oferecem abstrações para diluir a complexidade dos sistemas em módulos. Entretanto, são projetadas para a disponibilização de uma única aplicação monolítica, onde toda a modularização utilizada é executada em uma mesma máquina. Os módulos compartilham recursos de processamento, memória, bancos de dados e arquivos.

## 3.1. Vantagens

A implantação é simples de se realizar, pois há apenas um ponto de deploy a ser feito. Provavelmente não haverá duplicidade de código entre os diferentes módulos, uma vez que fazem parte de uma mesma unidade.

## 3.2. Desvantagens

Proporciona um grande ponto central de falha, pois erros em um fragmento da aplicação podem desestabilizar o software como um todo. Seu código-fonte torna-se extenso e complexo, principalmente para novos desenvolvedores alocados no projeto.

# 4. Frontend

O desenvolvimento de software envolve não apenas um compêndio de padrões de projeto isolados, mas necessita de melhores práticas acerca das relações entre os mesmos, de modo a propiciar uma atuação conjunta de forma coesa para que soluções maiores sejam disponibilizadas. Assim, a combinação dos padrões disponibilizados no catálogo JEE, cuja vinculação pode ensejar uma solução que os aborde pode constituir um framework web.

Este processo, atrelado aos padrões, necessita de identificação dos cenários envolvidos e seus respectivos padrões aplicáveis em cada uma de suas camadas, fornecendo desta maneira um conjunto estratégico de implementações para cada função desejada.

## 4.1. Angular

Angular é um framework estrutural para aplicações web dinâmicas. Permite a utilização de HTML como a linguagem modelo, assim como a extensão de sua sintaxe para expressar componentes de forma clara e sucinta. A correlação dos dados com os componentes fornecidos pelo framework e injeção de dependência propicia a eliminação de boa parte do código. Usa diretivas, que são tags especiais que definem uma certa ligação a um elemento de uma página.

O conceito de duas vias de ligação implica em atualização do modelo automaticamente, ou se necessário quando os usuários incluírem dados em formulários. Este acoplamento entre os valores de entrada e sua representação variável no modelo facilita a manipulação dos dados, sem necessitar prestar atenção aos eventos orientados para o usuário. Este conceito não é inovador para outras plataformas como .NET e Java,

porém para frameworks Javascript é uma novidade. O framework também provê nativamente injeção de dependência.

A dissociação da manipulação DOM da lógica da aplicação é esperada, uma vez que há uma melhora perceptível na capacidade de testes do código, assim como a dissociação do lado cliente de uma aplicação do lado servidor, pois há a permissão do trabalho de desenvolvimento em paralelo, permitindo a reutilização em ambos os lados, a título de exemplificação.

Para a camada de visão das aplicações do Poder Judiciário do Estado do Rio de Janeiro, recomenda-se a utilização do framework Angular.

### 4.2. React

Framework Javascript idealizado pelo Facebook com o objetivo de facilitar o desenvolvimento de frontends.

## 5. Backend

### 5.1. Spring Cloud

O Spring Cloud fornece ferramentas para desenvolvedores criarem rapidamente alguns dos padrões comuns em sistemas distribuídos (gerenciamento de configuração, descoberta de serviço, disjuntores, roteamento inteligente, micro-proxy, barramento de controle, tokens únicos, bloqueios globais, eleição de liderança, distribuição sessões, estado de cluster).

### 5.2. Netflix Eureka

Eureka é um serviço baseado em REST (Representational State Transfer) que é usado principalmente na nuvem da AWS para localizar serviços com o objetivo de balanceamento de carga e failover de servidores de camada intermediária.

## 5.3. Netflix Zuul

O Zuul é a porta de entrada de todas as solicitações de dispositivos e sites para o backend da aplicação. Como um aplicativo de serviço de borda, o Zuul é construído para permitir roteamento dinâmico, monitoramento, resiliência e segurança. Ele também tem a capacidade de encaminhar solicitações para vários grupos do *Amazon Auto Scaling*, conforme apropriado.

## 5.4. Netflix Hystrix

O Hystrix é uma biblioteca de tolerância a falhas e latência projetada para isolar pontos de acesso a sistemas remotos, serviços e bibliotecas de terceiros, interromper falhas em cascata e permitir a resiliência em sistemas distribuídos complexos onde a falha é inevitável.

- Latência e tolerância a falhas – Interrupção de falhas em cascata.
- Operações em tempo real – Monitoramento em tempo real e alterações de configuração. Seja alertado, tome decisões, afete a mudança e veja os resultados em segundos.
- Concorrência – Execução paralela. Armazenamento em cache de solicitação de concorrência. Agrupamento automatizado por meio do recolhimento de solicitação.

## 5.5. Spring Framework

Framework destinado ao fornecimento de CDI e AOP, dentre outras especificidades.

### 5.5.1. Spring Boot

O Spring Boot facilita a criação de aplicativos baseados em Spring Framework, de modo a propiciar o início do projeto com o mínimo de trabalho. A maioria dos aplicativos Spring Boot precisa de uma configuração de Spring muito pequena.

## 5.6. RabbitMQ

O RabbitMQ é o mais popular message broker de código aberto. O RabbitMQ é leve e fácil de implementar no local e na nuvem, suportando vários protocolos de mensagens. Pode ser implementado em configurações distribuídas e federadas para atender a requisitos de alta disponibilidade e alta escala.

## 5.7. OAuth 2

O OAuth 2 é uma estrutura de autorização que permite que os aplicativos obtenham acesso limitado às contas de usuários em um serviço HTTP. Seu funcionamento ocorre através de delegação da autenticação de usuário ao serviço que hospeda a conta do usuário, e autorizando aplicações de terceiros a acessar a conta do usuário. O OAuth 2 fornece fluxo de autorização para aplicações web e desktop, e para dispositivos móveis.

## 5.8. ELK Stack

É o acrônimo para três projetos de código aberto: Elasticsearch, Logstash e Kibana.

### 5.8.1. Elasticsearch

Motor de buscas open source distribuído baseado no Apache Lucene.

### 5.8.2. Logstash

Motor central de fluxo de dados do Elastic Stack para coletar, enriquecer e unificar todos os seus dados independentemente do formato ou esquema

### 5.8.3. Kibana

Plataforma de análise e visualização de dados, projetada para trabalhar com Elasticsearch, possibilitando a compreensão de grandes volumes de dados, sendo possível compartilhar dashboards dinâmicos que exibem alterações em tempo real.

### 5.8.4. JPA

Java Persistence API destinada a prover uma interface comum para frameworks de persistência de dados bem como uma abstração destinada ao mapeamento objeto-relacional. Objetiva-se que as aplicações deste órgão utilizem, majoritariamente, esta API.

### 5.8.5. Java Transaction API (JTA)

Especificação padrão para a interface entre um gerenciador de transações e os componentes envolvidos em uma transação distribuída, tais como gerenciador de recursos, servidor de aplicações e aplicações transacionais. Recomenda-se o uso desta especificação em projetos que necessitem deste controle.

## 5.9. Services

### 5.9.1. JAX-RS

Java API for RESTful Web Services prove suporte para o desenvolvimento de web services de acordo com a arquitetura REST (Representational State Transfer). A API JAX-RS utiliza anotações de modo a simplificar a construção dos clientes e endpoints.

Após análises conceituais envolvendo a utilização de SOAP ou REST, conclui-se que REST demonstra ser uma melhor alternativa para a construção de web services, uma vez que estudos demonstram um melhor desempenho, escalabilidade, interoperabilidade, dentre outros itens. (MULLIGAN & GRACANIN, 2009) (MUMBAIKAR & PADIYA, 2013) (POTTI, 2011). Deste modo, recomenda-se a utilização majoritária desta tecnologia.

## 6. Desenvolvimento para dispositivos móveis

Em função das características da aplicação, principalmente às não-funcionais, tais como desempenho, acesso ao hardware, etc, os projetos utilizarão as tecnologias abaixo citadas:

*Java*
*Kotlin*
*Swift*
*Ionic*
*React Native*

## 7. Recurso

O Tribunal de Justiça desenvolve suas aplicações, atualmente, utilizando o banco de dados Oracle. Todavia, recomenda-se uma análise mais profunda acerca da disponibilização da tecnologia NoSQL em soluções específicas, uma vez que a mesma

demonstra uma grande escalabilidade e performance (ZAKI, 2014) e o banco de dados PostgreSQL.

# 8. Configuração

Além dos itens supracitados, as seguintes tecnologias foram avaliadas de forma a sustentar o ambiente destinado ao desenvolvimento:

## 8.1. IDE

Recomenda-se a utilização da IDE Eclipse (Spring Suite Tools), por se tratar de ferramenta já utilizada por este órgão e atender plenamente às demandas de desenvolvimento.

## 8.2. Build

Após análises relativas às ferramentas Maven, Ant e Gradle, recomenda-se a adoção do Maven, uma vez que a referida tecnologia possui ampla utilização no mercado, assim como vasta documentação e comunidade (zeroturnaround, 2014).

## 8.3. Análise de código

Recomenda-se a utilização de ferramenta destinada à análise de código. Neste quesito, assinalamos o uso do SonarQube devido a sua utilização frente ao mercado, uma vez que a mesma possui maturidade e inteligência para integração com outras ferramentas de análise, caso seja necessário.

## 8.4. Integração Contínua

A Integração Contínua é uma prática de desenvolvimento destinada à integração do código em um repositório compartilhado por parte dos desenvolvedores. Deste

modo, cada check-in incide em uma verificação automatizada no build do produto, permitindo à equipe a localização de possíveis problemas rapidamente. A ferramenta indicada para o ambiente de Integração Contínua é o Jenkins.

## 8.5. Versionamento

Atualmente, o nosso órgão faz uso do Subversion para o versionamento de seus códigos. Porém, recomendamos a adoção do Git para os novos projetos, devido às suas características distribuídas, bem como a alta performance frente ao Subversion (SPANDEL & KJELLGREN, 2014).

## 8.6. Repository Manager

Um gerenciador de repositório consiste de um servidor dedicado ao gerenciamento do repositório dos binários. A utilização de um repositor manager é considerada uma boa prática essencial para o uso do Maven, e desta forma utilizaremos o Nexus.

## 8.7. Testes

Sugerimos a adoção de boas práticas relativas ao desenvolvimento de software referente aos frameworks destinados aos testes. Deste modo, a solução arquitetural possuirá ferramentas destinadas aos testes unitários, mocking, testes automatizados em browser e carga e desempenho. Assim, assinalamos as seguintes tecnologias para este item: JUnit, Mockito, Selenium, JMeter, Karma e Protractor.

## 8.8. Conhecimentos aplicáveis a novos projetos

Os conceitos abaixo relacionadas poderão ser aplicados em função das especificidades oriundas da solução.

*Docker*

*Kubernetes*

*OKD*

*TypeScript*

*Ionic, React Native, Kotlin e Swift*

*React*

*PrimeNg*

*Ehcache*

*JCR*

*Swagger*

*PGAdmin*

*Flyway*

Técnicas de Machine Learning, Deep Learning e processamento de linguagem natural utilizando a plataforma Python, R ou Java.

# 9. Adendos

## 9.1. Segurança

O controle relativo à autenticação e autorização será realizado através dos mecanismos fornecidos pela API padrão Java Authentication and Authorization Service (JAAS) de forma a abranger usuários, grupos e papéis. Outros protocolos de segurança, como HTTPS, que permite sigilo na comunicação entre navegadores web e servidores, podem ser utilizados nas aplicações, sempre que necessário.

### 9.1.1. Autenticação e Autorização

Para evitar a replicação desnecessária de cadastros de usuários e perfis de acesso nas aplicações do Poder Judiciário do Estado do Rio de Janeiro, orienta-se o uso dos

softwares Sistema de Controle de Usuários – SISTUSU e Sistema de Segurança - SISTSEG.

O SISTUSU e o SISTSEG são sistemas desenvolvidos pelo nosso órgão de modo a prover um mecanismo de acesso a uma base de dados centralizada, contendo os usuários e as suas respectivas permissões. Todas as aplicações web desenvolvidas utilizarão o componente de segurança SEGWEB, destinado a prover os mecanismos de autenticação e autorização.

Novos projetos necessitarão de estudos quanto à viabilidade de utilização do Keycloak.

## 9.2. Tratamento de Exceções

As exceções lançadas serão despachadas para a camada imediatamente superior até chegar à camada de apresentação, que exibirá uma mensagem de erro para o usuário. Ressalta-se a importância de que todas as mensagens de erro deverão ser detalhadas e compreensíveis.

Todas as exceções deverão ser persistidas em arquivos de log, utilizando a ferramenta Log4j ou similar. Os arquivos serão configurados de modo a permitir auditorias no formato *dia_mês_ano* ou similar. Apenas as exceções consideradas essenciais serão encaminhadas para os e-mails configurados através do componente desenvolvido pelo TJRJ para o envio de e-mails.

## 9.3. Padronização de Código

Será estritamente observado as convenções de código Java aprovadas e disponibilizadas através da url: http://www.oracle.com/technetwork/java/codeconvtoc-136057.html

Recomenda-se a observância ao Google JavaScript Style Guide em relação ao código escrito em JavaScript e a documentação pertinente a cada linguagem disponibilizada neste documento.

## 9.4. Referências

APPLE, Lauria. Making Architecture Work in Microservice. Disponível em: < http://tech.gilt.com/post/102628539834/making-architecturework-in-microservice>. Acesso em: 24/05/2014.

BONÉR, Jonas; FARLEY, Dave; KUHN, Roland; THOMPSON, Martin. The Reactive Manifesto. Disponível em: < http://www.reactivemanifesto.org>. Acesso em: 29/05/2015.

DVORKIN, Eugene. Seven micro-services architecture problems and solutions. Disponível em: 92 . Acesso em: 01/06/2015

FOWLER, Martin. Patterns of enterprise application architecture. Addison-Wesley Longman Publishing Co., Inc., 2003. 533 p. ISBN 0321127420

FOWLER, Martin, LEWIS, James. Microservices. Disponível em: . Acesso em: 05/05/2015.

SIMONE, Sergio De. Practical Implications of Microservices in 14 Tips. Disponível em: . Acesso em: 10/05/2015.