Arquitetura de Plataforma

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 delimita o escopo das especificações relativas às tecnologias envolvidas na arquitetura de plataforma referencial para o desenvolvimento de aplicações Web no Poder Judiciário do Estado do Rio de Janeiro.

2. Terminologia

Termo	Descrição
EJB	*Enterprise JavaBeans* é um componente executado em um *container*, cujo objetivo principal consiste em encapsular a lógica de negócio de uma aplicação.
JDBC	*Java Database Connectivity* é um conjunto de classes e interfaces (*API*) destinado ao envio de instruções *SQL* para o banco de dados.
JMS	*Java Message Service* é uma *API* destinada ao envio de mensagens.
JPA	A *API* de Persistência Java (*JPA*) é responsável pelo mapeamento objeto-relacional. Este mapeamento facilita o desenvolvimento de aplicações, cujas informações estejam persistidas em bancos de dados relacionais.
JSP	*JavaServer Pages* é a tecnologia que permite a criação de páginas

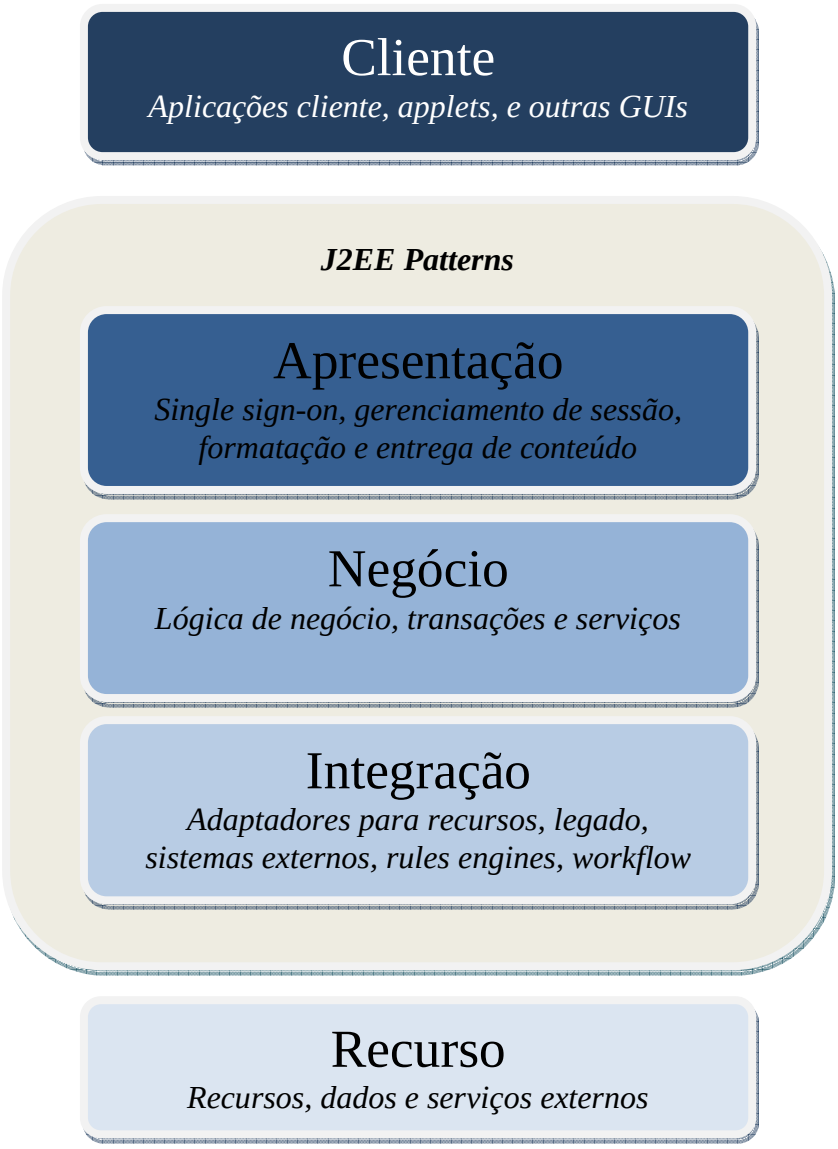
	dinâmicas em *HTML*, *XML* ou outros tipos de documentos.
JTA	A *Java Transaction API* disponibiliza interface destinada para transações.
REST	Transferência de estado representacional, cuja arquitetura consiste em restrições aplicadas a componentes, conectores e outros elementos.
Servlet	Classes *Java* utilizada para a geração dinâmica de conteúdo *HTML*.
SOAP	Padrão para troca de informações estruturadas de modo descentralizado e distribuído.

3. Arquitetura em Camadas

De modo a propiciar a interoperabilidade de diversos componentes, assim como permitir a possibilidade de disponibilização de diferentes interfaces de acesso às aplicações, o Poder Judiciário adotará o modelo arquitetural em *n* camadas.

Neste modelo de arquitetura, cada camada existente é equivalente a um dos particionamentos lógicos dentre os variados aspectos tratados em um sistema. Deste modo, cada camada possui uma atribuição bem delimitada e suas respectivas responsabilidades. Deste modo, cada camada é logicamente distinta, possuindo um fraco acoplamento com sua adjacente.

Uma vez que a presente proposta sugere aderência ao padrão arquitetural JEE para o desenvolvimento de seus softwares, recomenda-se a adoção estrutural das camadas conforme dispostas abaixo (ALUR, CRUPI, & MALKS, 2002):



3.1. Camada Cliente

Representa todos os possíveis dispositivos ou sistemas clientes que possam acessar a aplicação ou sistema. Este cliente pode ser desde um browser web, uma aplicação Java, ou uma aplicação para dispositivos móveis.

3.2. Camada de Apresentação

Possui o encapsulamento de toda a lógica necessária para os clientes que acessarão o sistema. A camada de apresentação é responsável pela interceptação das requisições oriundas do cliente, provisão de single sign-on, gerenciamento de sessão, elaboração e entrega das respostas ao cliente e controle de acesso ao barramento de negócios. Servlets e JSPs são alocados nesta camada.

3.3. Camada de Negócio

Disponibiliza os serviços necessários relativos aos dados e a parte lógica pertinente ao negócio. Deste modo, provê acesso ao cliente aos serviços referentes aos negócios. Há de se ressaltar que em sistemas legados os serviços supraditos podem ser acessados através da camada de recursos. Tipicamente, a arquitetura de componentes Enterprise almeja objetos de negócio em sua respectiva e adequada camada.

3.4. Camada de Integração

Mantém as comunicações com recursos externos e outros sistemas. A camada de negócio é acoplada a esta camada sempre que os objetos de negócio necessitem de dados ou serviços que estejam alocados na camada de recurso. A título de exemplificação, componentes podem usar JDBC, um conector J2EE ou um middleware para trabalhar com a camada de recurso.

3.5. Camada de Recurso

Contém os dados de negócio e recursos externos como sistemas legados e integração de sistemas.

4. Web Frameworks

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. AngularJS

AngularJS 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 AngularJS.

Estrutura de diretórios de uma aplicação típica construída em AngularJS:

```
app/                    --> all of the source files for the application
  app.css               --> default stylesheet
  components/           --> all app specific modules
    version/              --> version related components
      version.js                --> version module declaration and basic "version" value servi
      version_test.js           --> "version" value service tests
      version-directive.js      --> custom directive that returns the current app version
      version-directive_test.js  --> version directive tests
      interpolate-filter.js     --> custom interpolation filter
      interpolate-filter_test.js --> interpolate filter tests
  view1/                --> the view1 view template and logic
    view1.html            --> the partial template
    view1.js              --> the controller logic
    view1_test.js         --> tests of the controller
  view2/                --> the view2 view template and logic
    view2.html            --> the partial template
    view2.js              --> the controller logic
    view2_test.js         --> tests of the controller
  app.js                --> main application module
  index.html            --> app layout file (the main html template file of the app)
  index-async.html      --> just like index.html, but loads js files asynchronously
karma.conf.js         --> config file for running unit tests with Karma
e2e-tests/            --> end-to-end tests
  protractor-conf.js    --> Protractor config file
  scenarios.js          --> end-to-end scenarios to be run by Protractor
```

Figura 1 Referência: https://github.com/angular/angular-seed

5. Negócio

Responsável pelos serviços relativos aos dados e a parte lógica pertinente ao negócio. Tipicamente, a arquitetura de componentes JEE almeja objetos de negócio em sua respectiva e adequada camada.

Enterprise JabaBeans (EJB) é uma especificação (JSR 345) que compõe a estrutura JEE. Alguns autores apontam que esta padronização (RUBINGER & BURKE, 2010) propiciará sua permanência por tempo suficiente de modo a garantir segurança quanto a atualizações e manutenção. Através de suas especificações, um desenvolvedor não precisa conhecer os detalhes de implementação EJB, podendo mover sua aplicação entre servidores de aplicação compatíveis com a versão JEE, uma vez que soluções

específicas do servidor de aplicações não sejam incorporadas. Deste modo, recomenda-se a utilização de EJB.

6. Integração

Responsável por manter comunicações com recursos externos e outros sistemas. A camada de negócio é acoplada a esta camada sempre que os objetos de negócio necessitem de dados ou serviços que estejam alocados na camada de recurso. Ao longo desta seção, diversos aspectos e tecnologias serão abordados e assinalados o padrão a ser utilizado no desenvolvimento de software no Poder Judiciário do Estado do Rio de Janeiro.

6.1. Persistência

6.1.1. JDBC

Java Database Connectivity é uma API, baseada em Java, para o provimento de acesso a banco de dados através de SQL, através do driver JDBC apropriado. Vislumbra-se a utilização de JDBC nativo (Tipo 4) nas aplicações em que couber o seu uso.

6.1.2. 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.

6.1.3. Frameworks de persistência (JPA)

O EclipseLink é a implementação de referência JPA e possui, dentre outras vantagens, o grande suporte para stored procedures (SEMBERA, 2012). Deste modo, elege-se o EclipseLink como framework JPA.

6.1.4. 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.

6.2. Services

6.2.1. JAX-WS

É uma API Java destinada para a construção de web services que se comuniquem através de XML. Com esta API, clientes e serviços possuem a independência de plataforma de linguagem de programação, uma vez que a API se baseia em tecnologias definidas pelo W3C: HTTP, SOAP e WSDL. Recomenda-se a utilização desta API para os projetos que necessitem de SOAP em sua estrutura.

6.2.2. 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.

6.3. JMS

API Java para MOM (Message Oriented Middleware) utilizada para o envio de mensagens, permitindo a comunicação entre diversos componentes de uma aplicação distribuída. Recomendo a utilização de desta API caso a proposta arquitetural necessite de algum requisito intrínseco. Para outras necessidades deste porte, sugiro a utilização de REST, inclusive à análise arquitetural (JACOBI & RADUL, 2010).

7. Recurso

O Tribunal de Justiça desenvolve suas aplicações, atualmente, utilizando o banco de dados Oracle. Deste modo, sugerimos a continuidade do uso deste produto. 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).

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, 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.

9. Adendos

9.1. Acesso aos dados

As aplicações desenvolvidas na plataforma Java EE, proposta neste documento, serão executadas em Servidores de Aplicação. As regras de negócio estarão implementadas na própria aplicação, mais especificamente na camada de negócio, em componentes EJB3. Essa centralização trará ganhos no sentido de facilitar a manutenção e evolução dos sistemas.

Por esse motivo, o uso de Stored Procedures é desencorajado pela Arquitetura de Software, pois, além do fato de serem escritas em linguagem proprietária do SGBD, sua implementação implica na violação da separação de conceitos descritos neste documento. As Stored Procedures devem ser usadas apenas em casos excepcionais,

onde o desempenho alcançado pelos mecanismos de ORM não alcance os requisitos do cliente.

Cabe ressaltar que a comunicação entre as aplicações JAVA EE e o banco de dados é executada pelo Servidor de Aplicações, assim como o controle de abertura, fechamento e gerenciamento do Pool de Conexões. O mecanismo de Pool de Conexões deve ser utilizado para evitar que as conexões com o banco de dados precisem ser abertas e fechadas a cada nova requisição dos usuários das aplicações, uma vez que a abertura de conexões com é um processo custoso, que tende a degradar a performance da aplicação. Além disso, a plataforma Java EE prevê o uso de conexões compartilhadas como padrão, de modo que muitos recursos da plataforma seriam perdidos caso as conexões tenham que ser gerenciadas pela aplicação, e não pelo Servidor de Aplicação. Por estes motivos o uso de Connection Pooling é indispensável.

9.2. 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.

9.2.1. 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.

9.3. 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. Os arquivos serão configurados de modo a permitir auditorias no formato *dia_mês_ano* ou similar. Apenas as exceções consideradas essências serão encaminhadas para os e-mails configurados através do componente desenvolvido pelo TJRJ para o envio de e-mails. Estas exceções, assim como as especificações para a criação de exceções próprias constarão em documento apropriado anexo.

9.4. 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