**Documento de Arquitetura**

**Projeto SISSOFIA**

# Apresentação

Nesse documento serão apresentadas as tecnologias e ferramentas bem como a arquitetura utilizada no sistema. Esse documento utiliza termos técnicos, visto que se destina principalmente a equipe de desenvolvimento.

# Tecnologias Utilizadas

Essa sessão apresenta as tecnologias utilizadas no projeto e a sua importância.

## Maven

O Apache Maven é uma ferramenta de automação de compilação utilizada primariamente em projetos Java.

Será utilizado nesse projeto com o objetivo de automatizar o processo de build e deploy da aplicação no servidor. Também utilizaremos o Maven para fazer o gerenciamento de dependências do projeto.

Além desses motivadores, o Maven é um grande framework de automação e podemos aproveitar bem esse poder do Maven à medida que o sistema evoluir.

## JPA

A JPA é a especificação oficial da Oracle para persistência de dados com JAVA, foi adotada pela JSR pelas seguintes questões:

* Agilidade e menor tempo gasto na camada de persistência;
* Paradigma ORM - o programador não precisa programar em dois paradigmas distintos, o Objeto e o Relacional, ele programa só em um.
* Desacoplamento com o banco de dados – Inicialmente o projeto utilizará um banco do fabricante X. Se no futuro o cliente optar por mudar de fabricante, a aplicação não sofrerá nenhum impacto.

Além desses, existem outras vantagens em utilizar JPA. A JPA atende perfeitamente as necessidades desse projeto, por isso da sua escolha.

## Spring Framework (MVC)

O Spring MVC é um framework que encapsula a API de Servlet em uma camada de abstração garantindo ao programador um ganho considerável de produtividade. Além disso, é um framework *action-base* o que garante melhor performance do lado do servidor, quando comparado a frameworks *componente-base*, como o JSF por exemplo.

Outro motivador na escolha do Spring framework é que a equipe é composta por um programador front-end.

## JSP e JSTL

JavaServer Pages (JSP) é uma tecnologia que ajuda os desenvolvedores de software a criarem páginas web geradas dinamicamente baseadas em HTML, XML ou outros tipos de documentos. Lançada em 1999 pela Sun Microsystems, JSP é similar ao PHP, mas usa a linguagem de programação Java.

A JSTL é a API que encapsulou em tags simples toda a funcionalidade que diversas páginas Web precisam, como controle de laços (fors), controle de fluxo (if-else), manipulação de dados XML e a internacionalização de sua aplicação. Com a API JSTL, evitamos criar códigos blocos de código Java dentro das JSPs, ou seja, essa API evita o uso de Scriptlets.

## Banco de Dados MySQL

A escolha do provedor de banco de dados é tão importante quanto à escolha da linguagem de programação para o Projeto.

Abaixo destacamos os principais motivos para a escolha do banco de dado MySQL neste projeto:

* **O MySQL e o SaaS**

O SaaS (Software como Serviço) é essencialmente software por demanda: É um modelo onde a aquisição e/ou utilização de um software não está relacionado a compra de licenças, ou seja, você utiliza algum software e paga por sua utilização.

Hoje, a grande maioria das empresas de hospedagem já oferecem o MySQL como opção de banco de dados, diferente de outros provedores, com isso, a equipe não precisa se preocupar em fazer instalações/configurações do banco no servidor.

* **Desenvolvimento e Tempo de Lançamento no Mercado Mais Rápidos**

A facilidade de uso do MySQL tem sido uma meta desde sua concepção, esse é um dos principais fatores de sua adoção e popularidade. Existem poucos desenvolvedores que ainda não estão familiarizados com o MySQL. Os desenvolvedores podem ganhar velocidade usando a documentação de qualidade do MySQL e os fóruns de usuários ativos, além do Suporte por Consultoria do MySQL.

* **Desempenho e Escalabilidade Altos**

O MySQL é reconhecido por sua capacidade de ser executado e dimensionado horizontalmente em todo o hardware. Isso o tornou o banco de dados preferencial para os aplicativos mais exigentes e das maiores empresas na Web, incluindo o Facebook, que tem milhares de servidores MySQL e escalou o MySQL para gerenciar mais de 1 bilhão de usuários ativos.

## JasperReports

O JasperReports é um framework para a geração de relatórios em Java. É uma ferramenta totalmente open source e gratuita, e a mais utilizada com esse propósito atualmente. Entre as funcionalidades do JasperReports podemos destacar:

* É capaz de exportar relatórios para diversos formatos diferentes, tais como PDF, HTML, XML, XLS, etc.
* Aceita diversas formas de entrada de dados, tais como um arquivo XML ou CSV, conexão com o banco de dados, uma sessão do Hibernate, uma coleção de objetos em memória, etc.
* Permite o uso de diagramas, gráficos, e até códigos de barras.

## Log4J

APIs de Logging servem para controlar o fluxo de execução da nossa aplicação, ou seja, controlar eventuais exceções que possam ocorrer e essas informações podem ser persistidas em algum lugar (arquivos, BD etc.) e ainda podemos controlar o nível de informações que queremos ver (desde bem detalhadas a ver apenas os erros que acontecem). Há várias bibliotecas de logging no mercado. O log4j da Apache é o mais conhecido e o mais usado no mercado hoje.

# Ferramentas

Essa sessão apresenta as ferramentas utilizadas no projeto e a sua importância.

## SQL Yog

SQLyog é um programa desenvolvido pela WEByog Enterprise que possibilita a edição de bancos de dados Mysql, que baseados na linguagem SQL. Utilizado na criação, edição, sincronização de banco de dados internos e em servidores.

## Eclipse

IDE utilizada para programar na linguagem Java. É a IDE Java mais popular atualmente.

## GitHub

GitHub é um Serviço de Web Hosting Compartilhado para projetos que usam o controle de versionamento Git. Além de versionar código fonte, também possui ferramentas de gerenciamento e acompanhamento de projetos de software como controle de issues, pulls requests, etc.

## Jaspersoft studio

É o novo gerador de relatório baseado em Eclipse para JasperReports e JasperReports Server. É uma reescrita completa do iReport Designer, disponível como Eclipse plugin, e como um aplicativo independente. Será utilizado para fazer o design (layout) dos relatórios / arquivos pdf.

# Padrões de Projetos e Boas Práticas

Design Patterns (Padrões de Projetos) são elementos da programação orientada a objetos pelo(s) qual(is) nos permite abstrair de maneira mais eficiente e eficaz a recuperação de dados através de uma simples codificação própria para cada elemento de um Design Pattern.

Padrões de Projetos são soluções predefinidas para problemas recorrentes no desenvolvimento de software.

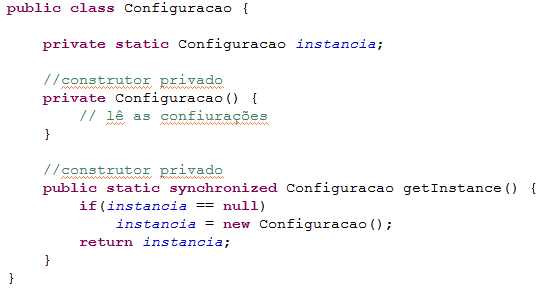
## DAO

O DAO (Data Access Object ou Objeto de acesso a dados) é um dos padrões de projeto mais utilizados no desenvolvimento de aplicações de softwares e tem como finalidade separar as regras de acesso a banco de dados das regras de negócios e de interface com usuário ou qualquer outro tipo de classe que não tenha relevância alguma com as ações de persistência. Ou seja, tudo o que estiver relacionado ao acesso ao banco será isolado no DAO.

## Singleton

A ideia desse padrão é que tenhamos uma classe-objeto capaz de ser instanciada (criada em memória pronta para seu uso) uma única vez e com visibilidade e acessibilidade global dessa instância em um determinado escopo de projeto. Com isso evitamos que a aplicação crie diversas instancias da mesma entidade no sistema.

Veja abaixo exemplo de uma classe usando o padrão Singleton:



Para utilizar a classe acima basta instanciá-la da seguinte forma:

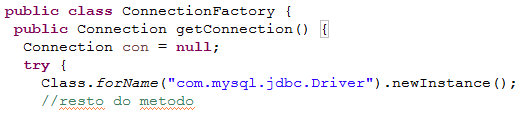


O Padrão Singleton será utilizado na camada de DAO, ou seja, todas as entidades DAOImpl deverão usar Singleton.

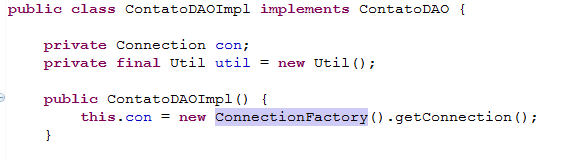
## Factory Method

Esse padrão de projeto permite às classes delegar para subclasses decidirem, isso é feito através da criação de objetos que chamam o método fabrica especificado numa interface e implementada por uma classe filha ou implementado numa classe abstrata e opcionalmente sobrescrito por classes derivadas.

Exemplo de Factory Method:



Exemplo de uso:



Esse padrão será utilizado basicamente na fábrica de conexões com o banco de dados, a Connection Factory.

## Factory e APIs

A utilização de APIs se dará por uma Classe Factory. Imagine o seguinte cenário:

Em nosso sistema precisamos passar diversas informações para XML, em vários pontos da aplicação. Para atender nossa necessidade vamos utilizar a API Xstream. Porém chamamos essa API em diversos pontos da aplicação. Se no futuro precisarmos tirar a Xstream e colocar outra biblioteca, vamos ter grandes problemas de manutenção.

Por esse motivo, para facilitar a manutenibilidade da aplicação, vamos isolar cada biblioteca e uma única Classe Factory, e sempre que precisar usar a biblioteca, é só chamar essa fábrica.

## Boas Práticas - Coesão

Coesão trata do *princípio da responsabilidade única*, que foi introduzido por Robert C. Martin no inicio dos anos 2000 e diz que **uma classe deve ter apenas uma única responsabilidade e realizá-la de maneira satisfatória**, ou seja, uma classe não deve assumir responsabilidades que não são suas. Logo as classes devem ser objetivas e pequenas, tratando apenas de um único assunto. Classes grandes, complexas e que fazem muitas coisas fogem das boas práticas da orientação a objetos.

Os métodos também devem seguir o mesmo conceito de coesão, ou seja, os métodos também devem ter apenas uma única responsabilidade, curtos e objetivos. Não se pode colocar dentro do mesmo método regras diferentes, validações diferentes, isso deve ser separado em vários métodos.

## Boas Práticas - Acoplamento

Acoplamento significa o quanto uma classe depende da outra para funcionar. E quanto maior for esta dependência entre ambas, dizemos que estas classes elas estão fortemente acopladas.

Alguns conceitos da orientação a objetos devem ser pensados e repensados antes de serem aplicados a fim de diminuir o acoplamento entre as classes. Um exemplo comum é usar **SEMPRE HERANÇA**, mas onde realmente a herança é necessária? A herança é considerado um recurso de alto acoplamento, pois as classes filhos são totalmente dependente das classes pais. Na orientação a objetos avançadas, existe uma regra que diz: “procure dar preferencia ao **Polimorfismo** e a **Composição** ao invés da Herança”, em outras palavras, o baixo acoplamento deve ser mantido e respeitado sempre.

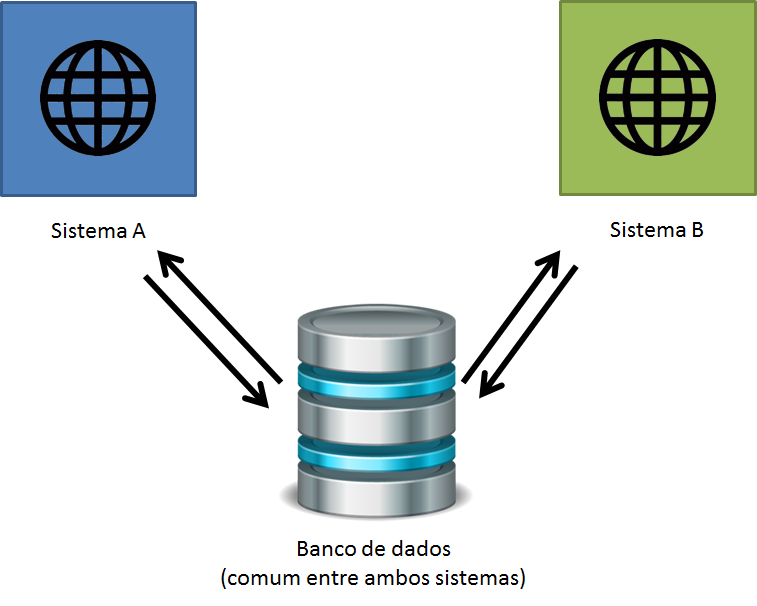
# Arquitetura

## Arquitetura MVC

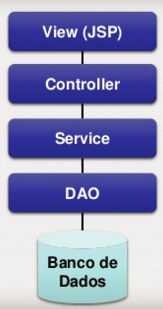
A aplicação utilizará o padrão arquitetural MVC. Esse padrão separa a representação da informação da interação do usuário. O modelo (model) consiste nos dados da aplicação, regras de negócios, lógica e funções. Uma visão (view) pode ser qualquer saída de representação dos dados, como uma tabela ou um diagrama, JSP, HTML, imagens, etc. O controlador (controller) faz a mediação da entrada, convertendo-a em comandos para o modelo ou visão. As ideias centrais por trás do MVC são a reusabilidade de código e separação de conceitos.

### Visão do projeto:

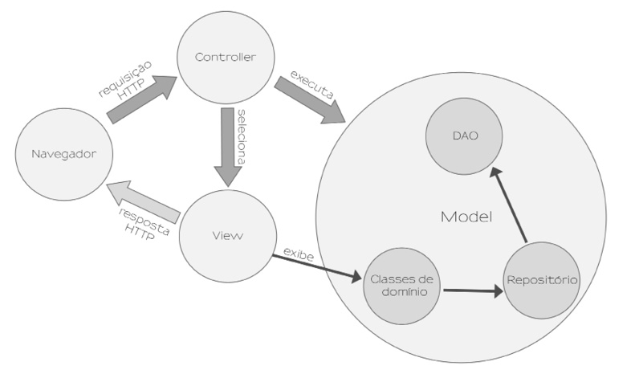
O projeto será dividido em **DOIS SISTEMAS**. O primeiro será o portal, utilizado pela empresa para dar entrada ao processo de alvará e acompanha-lo. Já o segundo sistema, será responsável por avaliar o pedido de alvará e conceder ou não a liberação, logo toda a tramitação por parte da prefeitura será feita no segundo sistema. Ambos os sistemas irão utilizar a mesma base de dados, garantindo a integridade dos dados.



### Visão geral da arquitetura do sistema:

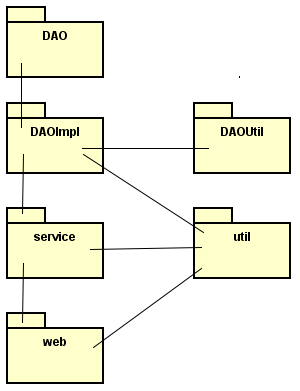


### Visão sistêmica da arquitetura MVC proposta:



### Estrutura de pacotes:

Todos os pacotes devem seguir o padrão de nomenclatura, devendo sempre começar com **br.com.manguetecnologia.sissofia.nome\_da\_camada.** Onde nome\_da\_camada, é uma das camadas do sistema, de acordo com a imagem a seguir:



Definição dos pacotes:

* **DAO** – Deve ficar apenas a Interface DAO.
* **DAOImpl** – Todas as classes que implementam a Interface DAO.
* **DAOUtil –** Nesse pacote, devem ficar as classes utilitárias da camada DAO, como por exemplo a Fabrica de conexão,uma classe para armazenar códigos SQL, etc.
* **Util –** Todas as classes utilitárias (verificadores, validadores, conectores, etc.) do sistema e pode ser chamado em qualquer camada. Nesse pacote também poderão ser incluídas classes utilitárias para frameworks.
* **Service –** Todas as classes de regra de negócio da aplicação, é aqui que fica concentrada toda a lógica do sistema.
* **Web** – Esse pacote vai conter apenas o controle (**Controller**) entre a camada de View e de Modelo do Sistema.

# Documentação do projeto

## JAVADOC

Javadoc é um gerador de documentação criado pela Sun Microsystems para documentar a API dos programas em Java, a partir do código-fonte. O resultado é expresso em HTML.

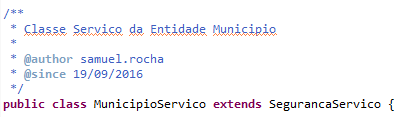
Este sistema é o padrão de documentação de classes em Java, e muitas dos IDEs desta linguagem irão automaticamente gerar um Javadoc em HTML.

Ele também provê uma API para a criação de doclets e taglets, que permitem a análise da estrutura de um aplicativo Java. É assim, por exemplo, que o JDiff consegue gerar relatórios de alterações feitas entre duas versões de uma API.

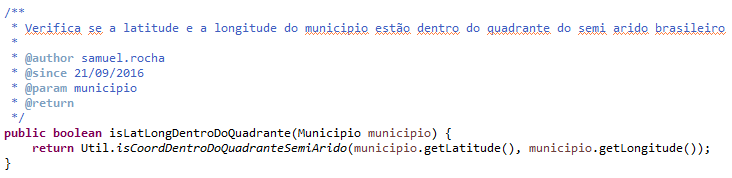
## Padrões Adotados

Será utilizado *JAVADOC* para a documentação de Classes e de métodos do sistema.

Exemplo de documentação de Classe:



Exemplo de documentação de método:



## Comentários

Os comentários são importantes na aplicação, porém devem ser usados sempre com moderação, ou seja, você só precisa fazer comentários em trechos de código realmente complexos para facilitar a manutenção no futuro. Há uma linha de pensamento que diz se o código é muito comentado, logo esse código não presta!

Exemplo de mau uso de comentários:

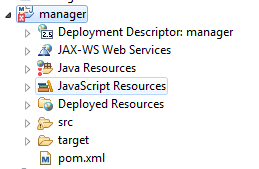


Observe que no exemplo acima o programador comentou cada parte do seu método, cada passo foi comentado, não existe necessidade disso é apenas lixo no código fonte.

# Estrutura do Projeto

Utilizaremos um Maven Project do Eclipse.

## Estrutura do Projeto no Eclipse



# Configurações

## Banco de Dados (Ambiente de Desenvolvimento)

|  |  |
| --- | --- |
| Host |  |
| Banco |  |
| Usuário |  |
| Senha |  |
| Porta |  |

## Repositório

|  |  |
| --- | --- |
| Repositório |  |
|  |  |

# Referencias

## Apache Maven

Site Oficial

<https://maven.apache.org/>

Wikipedia Apache Maven

<https://pt.wikipedia.org/wiki/Apache_Maven>

Conceitos Importantes

<http://luizricardo.org/2014/06/instalando-configurando-e-usando-o-maven-para-gerenciar-suas-dependencias-e-seus-projetos-java/>

## MySQL

10 Motivos para usar MySQL

<http://www.mysql.com/why-mysql/white-papers/10-principais-motivos-para-usar-o-mysql-como-um-banco-de-dados-incorporado/>

## JasperReports

Como funciona o JasperReports

<http://www.k19.com.br/artigos/relatorios-em-java-jasperreports-e-irepor/>

## JUnit

Site Oficial

<http://junit.org/junit4/>

Wikipedia Apache Maven

<https://pt.wikipedia.org/wiki/JUnit>

TDD

<http://tdd.caelum.com.br/>

## Log4J

Site Oficial

<http://logging.apache.org/log4j/2.x/>

Wikipedia Log4J

<https://pt.wikipedia.org/wiki/Log4j>

Usando logs

<http://www.caelum.com.br/apostila-java-testes-xml-design-patterns/apendice-logging-com-log4j/#12-1-usando-logs-log4j>

## JAVADOC

Javadoc Tool

<http://www.oracle.com/technetwork/articles/java/index-jsp-135444.html>

Ferramentas: jar e javadoc

<https://www.caelum.com.br/apostila-java-orientacao-objetos/ferramentas-jar-e-javadoc/>

Wikipedia Javadoc

<https://pt.wikipedia.org/wiki/Javadoc>

## Coesão e Acoplamento

Entendendo a Coesão e Acoplamento

<http://www.devmedia.com.br/entendendo-coesao-e-acoplamento/18538>

Uma outra perspectiva sobre o acoplamento

<http://blog.caelum.com.br/orientacao-a-objetos-uma-outra-perspectiva-sobre-o-acoplamento/>

# Histórico de alterações

|  |  |  |
| --- | --- | --- |
| Data | Autor | Versão |
| 27/11/2016 | Samuel Rocha | 1.0 |
| 29/11/2016 | Samuel Rocha | 1.1 |