****

**Documento de Arquitetura**

**Versão 1.0**

**Histórico de Revisão**

|  |  |  |  |
| --- | --- | --- | --- |
| Data | Versão | Descrição | Autor |
| 27/03/2013 | 0.1 | Criação do Documento | Mariana Victor |
| 09/04/2013 | 1.0 | Primeira Versão do Documento | Mariana Victor |

Sumário

[**1.** **Introdução** 4](#_Toc353362988)

[**1.1.** **Visão geral deste documento** 4](#_Toc353362989)

[**2.** **Estilo Arquitetural** 4](#_Toc353362990)

[**3.** **Padrões de Projeto** 4](#_Toc353362991)

[**3.1.** **Singleton** 5](#_Toc353362992)

[**3.2.** **Facade** 5](#_Toc353362993)

[**3.3.** **Transfer Object** 6](#_Toc353362994)

[**4.** **Representação Arquitetural** 6](#_Toc353362995)

[**4.1.** **Visão** 6](#_Toc353362996)

[**4.1.1.** **JSF (JavaServer Faces)** 7](#_Toc353362997)

[**4.1.2.** **XHTML** 7](#_Toc353362998)

[**4.1.3.** **PrimeFaces** 7](#_Toc353362999)

[**4.2.** **Fachada** 7](#_Toc353363000)

[**4.3.** **Controlador** 7](#_Toc353363001)

[**4.4.** **Repositório** 8](#_Toc353363002)

[**4.5.** **Hibernate** 8](#_Toc353363003)

[**4.6.** **Banco de Dados** 9](#_Toc353363004)

[**4.7.** **Classes Básicas** 9](#_Toc353363005)

[**4.8.** **Objetos de Transferência** 10](#_Toc353363006)

1. **Introdução**

Este documento tem como objetivo apresentar toda a arquitetura com os padrões de desenvolvimento e os frameworks que foram definidos para o desenvolvimento do SGCONT – Sistema Gerencial de Contabilidade.

* 1. **Visão geral deste documento**

Os próximos capítulos deste documento estão divididas conforme a tabela abaixo:

|  |  |
| --- | --- |
| Sessão | Descrição |
| 2 - Estilo Arquitetural | É apresentado o estilo arquitetura definido na solução. |
| 3 - Padrões de Projeto | São apresentados os padrões selecionados para a solução. |
| 4 - Representação Arquitetural | São apresentados todos os elementos que compõem a arquitetura da solução. |

1. **Estilo Arquitetural**

Deverá ser utilizada a Arquitetura em Camadas (MVC) como estilo arquitetural para o sistema. A arquitetura em camadas deverá ser utilizada com o objetivo de separar a lógica de negócio da lógica de apresentação, permitindo o desenvolvimento, teste e manutenção isolado de ambos.

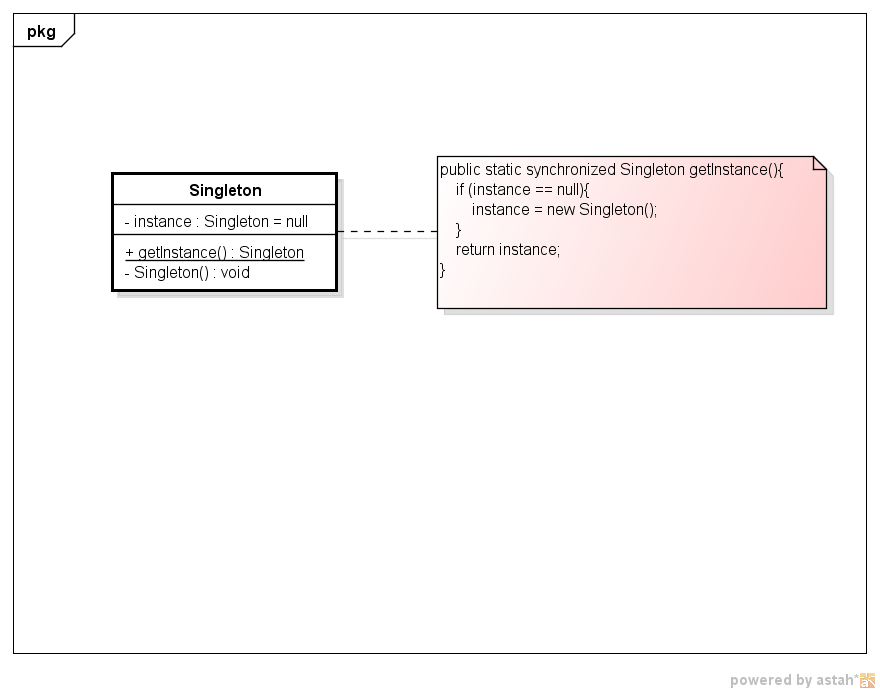
1. **Padrões de Projeto**

Esta seção apresenta as definições relacionadas aos padrões de projeto definidos para as atividades de projeto e de implementação.

* + Singleton;
  + Facade;
  + Transfer Object;
  1. **Singleton**

Este padrão deverá ser utilizado para garantir que seja criada apenas uma instância de uma *classe*, fornecendo um ponto de acesso global a essa instancia única.

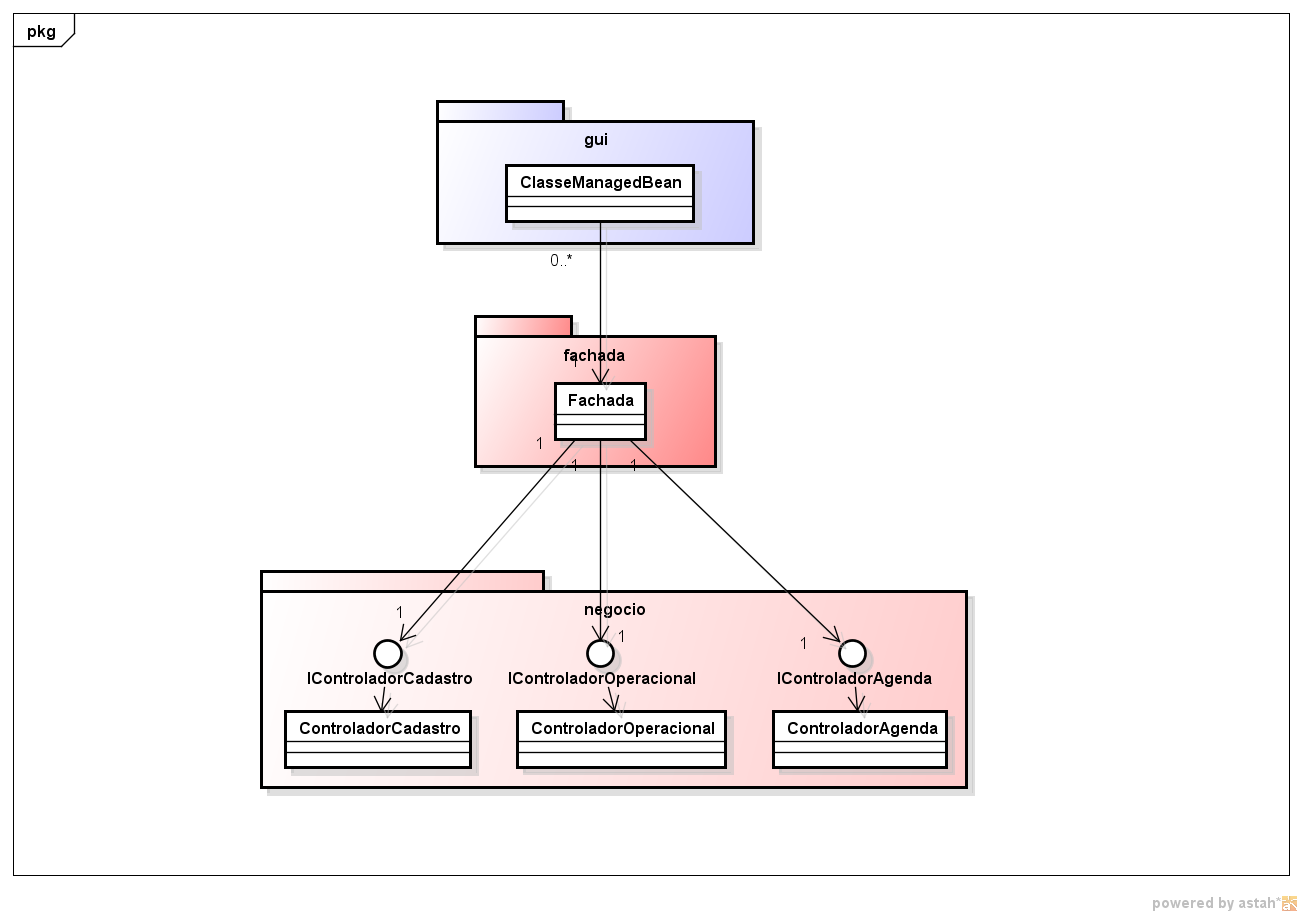
As *classes* referentes a *Fachada*, *Controlador* e *Repositório* deverão utilizar este padrão de acordo com as definições a seguir:

****

* 1. **Facade**

Será utilizado padrão *Facade* para instanciar todas as *classes* do pacote de negócios. Para implementar esse padrão, será criada a *classe* *Fachada*, que será responsável oferecer uma interface única para as interfaces definidas no pacote de negócios, delegando as requisições aos devidos objetos.

A implementação deste padrão deverá ser feita de acordo com as definições a seguir:



* 1. **Transfer Object**

Este padrão deverá ser utilizado para aumentar a granularidade de acesso e transferência de dados entre as camadas do sistema. Para a implementação desse padrão, nos cenários em que são enviados muitos dados entre as camadas do sistema, deverá ser criada uma *classe* com o intuito de agrupar todos os dados.

1. **Representação Arquitetural**

A arquitetura do SGCONT é composta pelos elementos apresentados a seguir:

Visão

Fachada

Controlador

Repositório

Classes Básicas

Hibernate

Banco de Dados

Objetos de Transferência

Cada um desses elementos serão descritos nos próximos tópicos.

* 1. **Visão**

A camada de Visão é composta pelos elementos responsáveis pela apresentação dos dados ao usuário. Esses elementos são: JSF, XHTML, PrimeFaces e ManagedBeans.

* + 1. **JSF (JavaServer Faces)**

O SGCONT utilizará a versão 2.2 do JavaServer Faces (JSF). O JSF é um framework MVC de aplicações Web baseado em Java com o objetivo de simplificar o desenvolvimento de interfaces de usuário baseadas em web.

* + 1. **XHTML**

Para utilizar a tecnologia do JSF, o padrão adotado para a interface gráfica foi o formato padrão adotado pelo Facelets: o XHTML. O Facelets é uma linguagem de descrição de páginas (PDL – Page Description Language) criada especificamente para JSF. Ele estabelece uma linguagem de templates que suporta a criação da árvore de componentes das telas JSF, o que permite o reuso de padrões de telas e a composição de componentes JSF para formar novos componentes. Quando compilado no servidor, o usuário recebe a interface em puro HTML.

* + 1. **PrimeFaces**

Para o desenvolvimento dos componentes gráficos, o SGCONT utilizará o framework PrimeFaces (versão 3.5) que é uma suite open-source de componentes customizados para o JavaServer Faces. Esses componentes foram construídos para trabalhar com AJAX por padrão de forma que não é necessário nenhum esforço extra por parte do desenvolvedor para realização de chamadas assíncronas ao servidor.

O Primefaces oferece diversos temas para os seus componentes para que possa ser utilizado o que mais se adéqua ao sistema. Dessa forma, o SGCONT deverá utilizar o tema “Redmond” como padrão para todos os componentes do PrimeFaces utilizados na aplicação.

* 1. **Fachada**

Para implementação do padrão *Facade* descrito anteriormente, será criada a *classe* *Fachada*, que deverá prover uma interface única para as interfaces de cada *Controlador*.

Dessa forma, ela irá centralizar todas as chamadas de métodos da camada de negócio de forma que outras aplicações ou outras camadas superiores possam usá-las.

A *classe Fachada* também deverá implementar o padrão *Singleton,* garantindo que a mesma possua uma instância única e fornecendo um ponto de acesso global a essa instância.

* 1. **Controlador**

Todas as regras de negócio do sistemas deverão estar centralizadas nas *classes* de *Controle*. Para melhor separação das regras de negócio, serão criados os seguintes controladores:

|  |  |
| --- | --- |
| Controlador | Descrição |
| ControladorCadastro | Responsável pelas regras de negócio do módulo de cadastro |
| ControladorOperacional | Responsável pelas regras de negócio do módulo operacional |
| ControladorAgenda | Responsável pelas regras de negócio do módulo da agenda |

As *classes* de *Controle* também deverão implementar o padrão *Singleton,* garantindo que cada uma possua uma instância única e fornecendo um ponto de acesso global a essa instância.

* 1. **Repositório**

Serão criadas *classes* de *Repositório* que serão responsáveis por todas as operações no banco de dados (consultas, inserções, remoções e atualizações de dados). Para melhor separação da operações, serão criados os seguintes repositórios:

|  |  |
| --- | --- |
| Repositório | Descrição |
| RepositórioCadastro | Responsável pelas operações do módulo de cadastro |
| RepositórioOperacional | Responsável pelas operações do módulo operacional |
| RepositórioAgenda | Responsável pelas operações do módulo da agenda |

As *classes* de *Repositório* também deverão implementar o padrão *Singleton,* garantindo que cada uma possua uma instância única e fornecendo um ponto de acesso global a essa instância.

* 1. **Hibernate**

A persistência ao banco de dados deverá ser feita través do framework *Hibernate* (versão 4) que permite a persistência dos objetos Java.

As informações referentes à configuração do *Hibernate* e ao banco de dados a ser utilizado deverão estar presentes no arquivo “hibernate.cfg.xml”.

Será criada a classe *HibernateUtil* com o objetivo de fornecer sessões para a realização de operações nos objetos do banco de dados a partir das *classes* de *Repositório*.

* 1. **Banco de Dados**

Para armazenar as informações das entidades relacionadas no SGCONT, será utilizado o banco de dados PostgreSQL.

* 1. **Classes Básicas**

Para cada tabela do banco de dados que deverá ser manipulada pela aplicação, será criada uma classe. Esta classe deverá utilizar o recurso *Annotation* do *Hibernate* para fazer o mapeamento da tabela. A classe deverá seguir o seguinte exemplo:

@Entity

@Table(name = "cadastro.usuario")

**public** **class** Usuario {

@Id

@GeneratedValue(strategy = GenerationType.*AUTO*)

@Column(name="cdusuario", nullable=**false**)

**private** Integer codigo;

@Column(name="nmusuario", nullable=**false**, length=50)

**private** String nome;

//...

**public** Integer getCodigo() {

**return** codigo;

}

**public** **void** setCodigo(Integer codigo) {

**this**.codigo = codigo;

}

**public** String getNome() {

**return** nome;

}

**public** **void** setNome(String nome) {

**this**.nome = nome;

}

}

* 1. **Objetos de Transferência**

Para as chamadas de métodos entre as camadas do sistema que envolvam muitos dados, deverá ser criada uma *classe* utilizando o padrão *Transfer Object* (Objeto de Transferência ou *TO*) com o intuito de agrupar esses dados passados como parâmetros.

Por exemplo, se for necessário enviar os dados de cadastro do cliente do tipo pessoa física, deverá ser criada uma classe similar ao código a seguir.

**public** **class** ClientePessoaFisicaTO **implements** Serializable {

**private** **static** **final** **long** *serialVersionUID* = 1L;

**private** String nome;

p**rivate** Integer cpf;

//...

**public** String getNome() {

**return** nome;

}

**public** **void** setNome(String nome) {

**this**.nome = nome;

}

**public** Integer getCpf() {

**return** cpf;

}

**public** **void** setCpf(Integer cpf) {

**this**.cpf = cpf;

}

}