Documento de Arquitetura

Sistema de Automação de Formulários da Unichristus - FORMCHRISTUS

**Versão 01.00** 30 de maio de 2014

Histórico de Revisões

|  |  |  |  |
| --- | --- | --- | --- |
| Data | Versão | Descrição | Autor |
| 29/04/2014 | 1.0 | Versão inicial do documento | Ari, Giancarlo, Glaydson |
|  |  |  |  |

**SUMÁRIO**

[1. Introdução 4](#_Toc388352783)

[2. Arquitetura da Aplicação 4](#_Toc388352784)

[2.1 Objetivos, Restrições Arquiteturais e Tecnológicas 4](#_Toc388352787)

[2.1.1 Plataforma 4](#_Toc388352788)

[2.1.2 Projeto de Iniciação Científica 5](#_Toc388352789)

[2.2 Descrição da arquitetura 5](#_Toc388352790)

[2.2.1 Biblioteca Primefaces 5](#_Toc388352791)

[2.2.2 Framework EJB e CDI 6](#_Toc388352792)

[2.2.3 Framework Hibernate 7](#_Toc388352793)

[2.2.4 Frameworks Jasper e IReports 8](#_Toc388352794)

[3. Visões Arquiteturais 9](#_Toc388352795)

[3.1 Visão de Casos de Uso 9](#_Toc388352797)

[3.1.1 Visão Geral 9](#_Toc388352798)

[3.1.2 Mecanismos Arquiteturais 10](#_Toc388352799)

[3.2 Visão Lógica 10](#_Toc388352800)

[3.2.1 Visão Geral 10](#_Toc388352801)

[3.3 Visão de Implantação 10](#_Toc388352802)

[3.4 Visão de Implementação 10](#_Toc388352803)

[3.4.1 Componentes Reutilizados 10](#_Toc388352804)

[3.5 Visão de Dados 10](#_Toc388352805)

[4. Padrões Adotados 11](#_Toc388352806)

[5. Referências 11](#_Toc388352807)

[6. Arquitetura 11](#_Toc388352808)

[6.1. Elementos que compõe a Arquitetura 12](#_Toc388352809)

[6.2. Modelos 12](#_Toc388352810)

[4 13](#_Toc388352811)

[6.3. Infraestrutura 13](#_Toc388352812)

[6.4. Domínio 13](#_Toc388352813)

[6.5. Visão 14](#_Toc388352814)

[Desenho GERAL da A rquitetura 15](#_Toc388352815)

[7. Padrões de Projeto 16](#_Toc388352816)

[8. Objetivos e Restrições Arquiteturais 16](#_Toc388352817)

[8.1. Requisitos básicos 16](#_Toc388352818)

[8.2. Estratégias de implementação 16](#_Toc388352819)

# Introdução

Neste documento iremos detalhar as principais partes da arquitetura proposta para o desenvolvimento do Sistema de Automação de Formulários da Unichristus, denominado *FormChristus*. A arquitetura é formada por diversos padrões de projeto, principalmente, padrões Orientados a Objetos com destaque no mercado. Iremos destacar em cada parte da arquitetura o motivo da sua criação e o qual a sua influência para a criação de sistemas de alta coesão, mas com baixo acoplamento.

O Documento de Arquitetura do Software provê uma visão geral da arquitetura, usando um conjunto de visões arquiteturais para tratar aspectos diferentes do software.

Este documento serve como um meio de comunicação entre o Arquiteto de Software e outros membros da equipe de projeto sobre as decisões significativas que forem tomadas durante o projeto.

# Arquitetura da Aplicação



## Objetivos, Restrições Arquiteturais e Tecnológicas

### Plataforma

O sistema *FormChristus* será desenvolvido na plataforma Java e executará em ambiente Web. A aplicação será implantada em servidor de aplicações GlassFish utilizando a infraestrutura já existente no ambiente do Centro Universitário Unichristus.

O sistema será autocontido, não possuindo quaisquer interações com outros sistemas já existentes na Unichristus, em particular o sistema “aluno online” que já é utilizado pela instituição para registros acadêmicos. Poderão ser carregados inicialmente para fins de teste, dados disponíveis em outros sistemas, mas sem comunicação entre eles. O *FormChristus* possuirá sua própria base de dados, não estando também previstas quaisquer exportações de dados para sistemas externos.

### Projeto de Iniciação Científica

O projeto e desenvolvimento do sistema *FormChristus* faz parte do Programa de Iniciação Científica desenvolvido na Unichristus no período 2013/2014, tendo como objetivo promover a automatização dos formulários que atualmente são preenchidos à mão dentro da instituição. O foco inicial do projeto é a automatização do preenchimento e geração dos formulários, embora outras funcionalidades possam ser acrescidas no futuro.

Para este projeto, foram selecionados pela coordenação do curso de Sistemas de Informação os quatro formulários abaixo:

* A1 – Programa de Atividade Complementar
* A3 – Relatório de Atividade Complementar
* A4 – Inscrição em Atividade Complementar
* E1 – Programa de Extensão

## Descrição da arquitetura

A arquitetura do sistema utiliza um modelo em camadas, composto de:

* Uma camada de **Apresentação**, que utiliza o *Primefaces*, uma implementação do framework *Java Server Faces* (JSF).
* Uma camada de **Negócio**, que utiliza o framework *EJB* e *CDI* (*Context and Dependency Injection*) para injeção de dependências.
* Uma camada de **Persistência**, que utiliza o *Hibernate* como framework de Mapeamento Objeto-Relacional, a biblioteca *Ehcache* para prover cache de segundo nível e injeção de dependência através do *CDI* para prover a conexão com o banco de dados juntamente com aplicação de um Pool de conexões implementado pelo servidor *GlassFish*.
* Uma camada utilitária adicional para visualização e impressão dos formulários em seu formato real, em papel ou em arquivo de formato PDF. Para tal, utilizou-se os frameworks *Jasper/IReports*.
* Uma camada de segurança para fornecimento de autenticação e autorização do acesso às funcionalidades do sistema. Para esse fim, utilizou-se o framework *Spring Security*.

### Biblioteca Primefaces

O *Primefaces* é uma biblioteca de componentes baseada no framework JSF (*Java Server Faces*). Tem como principal objetivo adicionar componentes à interface da aplicação, tais como janelas de diálogo, campos de texto configuráveis, painéis com abas, menus, etc. Entre suas principais características, destacam-se:

* Biblioteca extensa de componentes visuais;
* Compatibilidade completa com a versão 2 do framework *JSF*;
* Disponível de forma gratuita e com código-fonte aberto;
* Baixo “*overhead*” para adição de componentes a uma aplicação *JSF*;

*Java Server Faces*, por sua vez, é um framework para aplicações web baseado na linguagem Java, projetado para simplificar o desenvolvimento de interfaces de usuário baseadas na web. É um framework web orientado por requisições, que implementa o padrão *MVC* (*Model-View-Controller*)[[1]](#footnote-1) para construção de interfaces utilizando componentes. JSF é uma tecnologia padronizada que foi formalizada em uma especificação que faz parte da Plataforma Java, em sua versão *Enterprise* (Java EE).

### Framework EJB e CDI

*Enterprise Java Beans (EJB)* é um modelo de componentes especificado pela plataforma Java EE, projetado para resolver problemas complexos encontrados em aplicações corporativas. EJBs atuam no servidor e normalmente são classificados como componentes de negócio. Podem ser utilizados em diferentes situações, tais como: desenvolvimento distribuído, integração/conectividade com legado, processamento assíncrono, controle transacional, entre outros.

*Contexts and Dependency Injection* (*CDI*) define um conjunto de serviços para o ambiente *Java EE*, com o objetivo de tornar as aplicações mais simples de serem desenvolvidas. *CDI* fornece uma arquitetura que permite a componentes *Java EE* tais como *servlets*, *enterprise beans* e *JavaBeans* serem integrados ao ciclo de vida de uma aplicação com escopos bem definidos.

*CDI* também ajuda a reduzir o “*gap*” entre a camada enterprise e a camada web, permitindo, por exemplo, que uma página *JSF* acesse diretamente componentes *EJB*, ao invés de passar por *Managed Beans*. Isto se tornou possível após a especificação *Java EE* 6, que definiu o conceito de *Managed Beans* para unificar os vários tipos de *beans* existentes em uma aplicação *Java EE*, tais como *Managed Beans JSF*, *enterprise beans* e *CDI Beans*.

Dessa forma, é possível utilizar *CDI* para definir o componente *Managed Bean* que é utilizado pelas páginas *JSF* para fornecer e resgatar dados da aplicação, bem como para resolver as dependências desse *Managed Bean* com os componentes *EJB*, responsáveis pelas regras de negócio da aplicação (controle transacional e persistência).

### Framework Hibernate

*Hibernate* é uma solução de mapeamento objeto-relacional (ORM) para Java, surgindo em 2001 como um framework de persistência *open source*. Fornece um serviço de consultas e persistência de alta performance para qualquer aplicação Java.

O framework mapeia classes Java para tabelas de bancos de dados e tipos de dados Java para tipos de dados SQL, retirando do programador grande parte das tarefas de programação relacionadas a persistência de dados.

*Hibernate* se posiciona entre os objetos Java tradicionais e o servidor de banco de dados para gerenciar todo o trabalho de persistir esses objetos, baseado em mecanismos e padrões apropriados de mapeamento. A Figura 1 mostra o funcionamento genérico do framework.

*Figura 1 – Funcionamento Genérico do Hibernate*



Fonte: http://www.tutorialspoint.com/hibernate/hibernate\_overview.htm

Entre as principais vantagens do uso do framework, pode-se destacar:

* Mapeamento de classes Java para tabelas de bancos de dados usando anotações ou arquivos XML, sem necessidade de escrever linhas de código;
* Fornece APIs para armazenar e recuperar objetos Java diretamente do banco de dados;
* A qualquer mudança no banco de dados ou em qualquer tabela, é necessário mudar somente a anotação ou arquivo XML;
* Abstrai tipos SQL não familiares, aproximando-se da nomenclatura Java mais utilizada;
* Não requer um servidor de aplicações para funcionar;
* Manipula associações complexas entre objetos;
* Minimiza o acesso ao banco de dados com estratégias inteligentes de “fetching”;
* Fornece mecanismos mais simples para consultas de dados.

### Frameworks Jasper e IReports

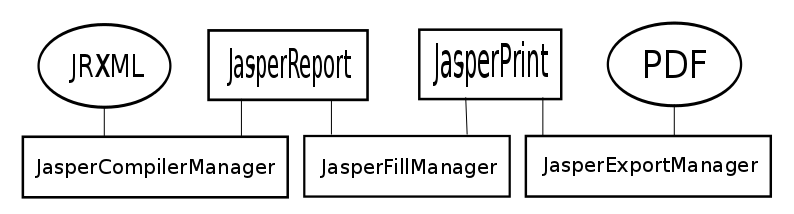
O *JasperReports* é um framework para a geração de relatórios. É uma ferramenta totalmente open source e gratuita, sendo uma das mais utilizadas atualmente com esse propósito. Entre as funcionalidades do *JasperReports* pode-se destacar:

* Capacidade 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 um banco de dados, uma sessão do *Hibernate*, etc.
* Permite o uso de diagramas, gráficos e até códigos de barras.

Um aspecto importante do *JasperReports* é que o *layout* do relatório é definido em um arquivo XML, geralmente com a extensão .*jrxml*. Este arquivo possui as informações de formatação do relatório e, além disso, possui os campos que serão preenchidos posteriormente, de acordo com a fonte de dados (data source) utilizada.

Para gerar um relatório, o arquivo XML é compilado, resultando em um objeto do tipo JasperReport. Em seguida, o relatório é preenchido com os dados desejados, ficando então armazenado em um objeto do tipo JasperPrint. Esse objeto já representa o relatório finalizado, que pode ser enviado diretamente para impressão ou exportado para outro formato, como o PDF por exemplo. A Figura 2 mostra o funcionamento desse processo.

Figura 2 – Funcionamento do JasperReports



*Fonte:*

* commons-beanutils
* commons-collections
* commons-digester
* commons-logging
* groovy-all
* iText
* poi
* jasperreports (é claro)

O *iReport* é um aplicativo gráfico, desenvolvido pela *JasperForge* (mesma desenvolvedora do *JasperReports*), que permite a criação do *layout* de um relatório, utilizando componentes visuais. Ao ser salvo, o *layout* gera automaticamente um arquivo com extensão .*jrxml* que poderá ser utilizada na aplicação em desenvolvimento. Para ser utilizada, a aplicação não requer conhecimento de XML, economizando tempo de desenvolvimento. O *iReport* disponibiliza ainda um conjunto de *templates* para serem utilizados diretamente, embora também seja possível criar os próprios *templates*.

O *iReport* geralmente é utilizado em conjunto com o *JasperReports*, uma vez que se torna complicado definir um *layout* de relatório escrevendo-se código em XML.

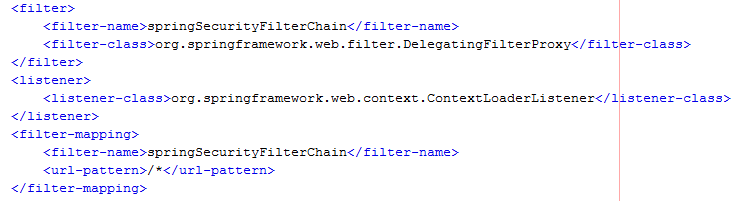
### Segurança

Um dos requisitos básicos da aplicação é o controle do acesso de cada um dos possíveis usuários do sistema. Inicialmente foram definidos quatro possíveis tipos de usuário para o sistema: Aluno, Professor, Coordenador e Administrador. Cada um desses usuários deve ter à sua disposição somente as funcionalidades permitidas para a sua função. Além disso, o sistema deve também garantir a autenticação de cada usuário com seu próprio login e senha.

Para prover essas funcionalidades, foi utilizado o *Spring Security*, que é um framework *Java EE*, cujo objetivo é fornecer autenticação, autorização e outros serviços de segurança para aplicações corporativas. “Autenticação” é o processo de estabelecer se um “principal” é quem realmente diz ser (um “principal” geralmente significa um usuário, dispositivo ou algum outro sistema que pode executar alguma ação dentro da aplicação). “Autorização” se refere ao processo de decidir se um “principal” tem permissão para executar uma ação dentro da aplicação. Para se chegar ao ponto onde uma decisão de autorização é necessária, a identidade do “principal” já deve ter sido estabelecida pelo processo de autenticação.

A configuração do framework é feita através de um arquivo XML (*applicattionContext*), ao lado de algumas entradas no descritor da aplicação web (arquivo *web.xml*). Uma configuração típica para que o framework intercepte todas as requisições vindas do cliente é vista na figura abaixo:

Figura 3 – Configuração do spring framework no descritor da aplicação



Observa-se que o framework é configurado na forma de um filtro, presente na especificação *Java EE*, que neste caso está programado para interceptar todas as requisições oriundas do browser (“/\*).

Feita essa configuração, basta configurar o arquivo “applicationContext” com os parâmetros de segurança da aplicação. Nesse arquivo são definidos papéis (“roles”), qual a página de login da aplicação, onde estão armazenadas as senhas de cada usuário, quais recursos são permitidos para acesso por cada papel, etc.

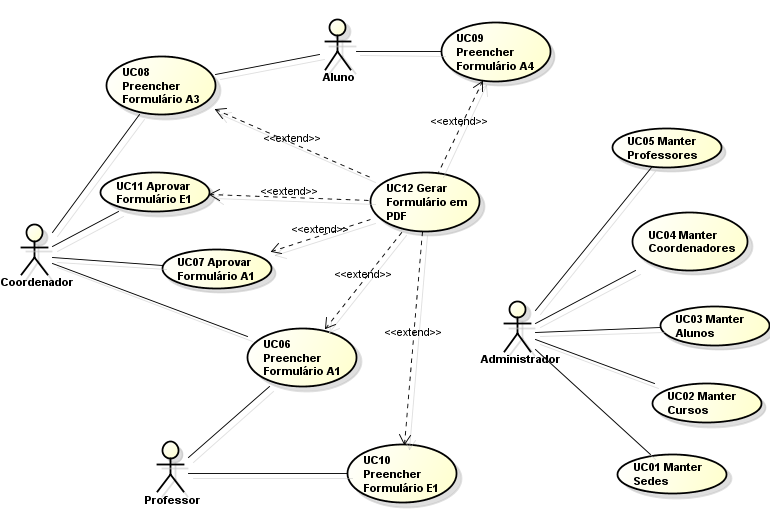
# Visões Arquiteturais



## Visão de Casos de Uso

A pretensão inicial da equipe do projeto, conforme solicitação da Unichristus foi a automatização do preenchimento dos formulários, definindo-se assim as seguintes funcionalidades, apresentadas no Diagrama de Casos de Uso mostrado na Figura 4.

*Figura 4 – Diagrama de Casos de Uso*



Fonte: Modelagem do Sistema FormChristus

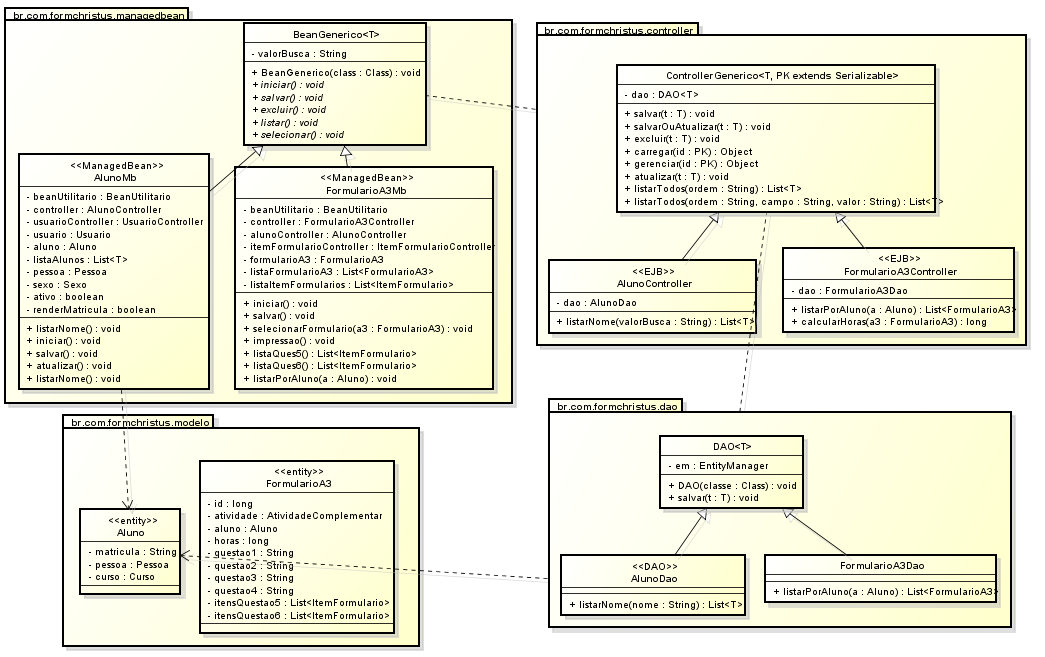
Todos os casos de uso mostrados na figura são considerados significativos e foram implementados para este projeto. Os formulários escolhidos foram modelados usando a ferramenta *IReports*, sendo para cada um deles gerado um arquivo com a extensão jrxml, conforme comentado na seção 2.2.4.

## Visão Lógica

### Visão Geral

O projeto e a divisão da aplicação em pacotes seguem a definição arquitetura exposta no item 2.2. A divisão em pacotes objetiva facilitar o entendimento, o desenvolvimento e manutenção da aplicação. A Figura 5 mostra uma visão parcial dos pacotes do sistema, demonstrando o uso da arquitetura em camadas para as entidades Aluno e FormularioA3.

*Figura 5 – Diagrama de Pacotes Parcial*



Fonte: Modelagem do Sistema FormChristus

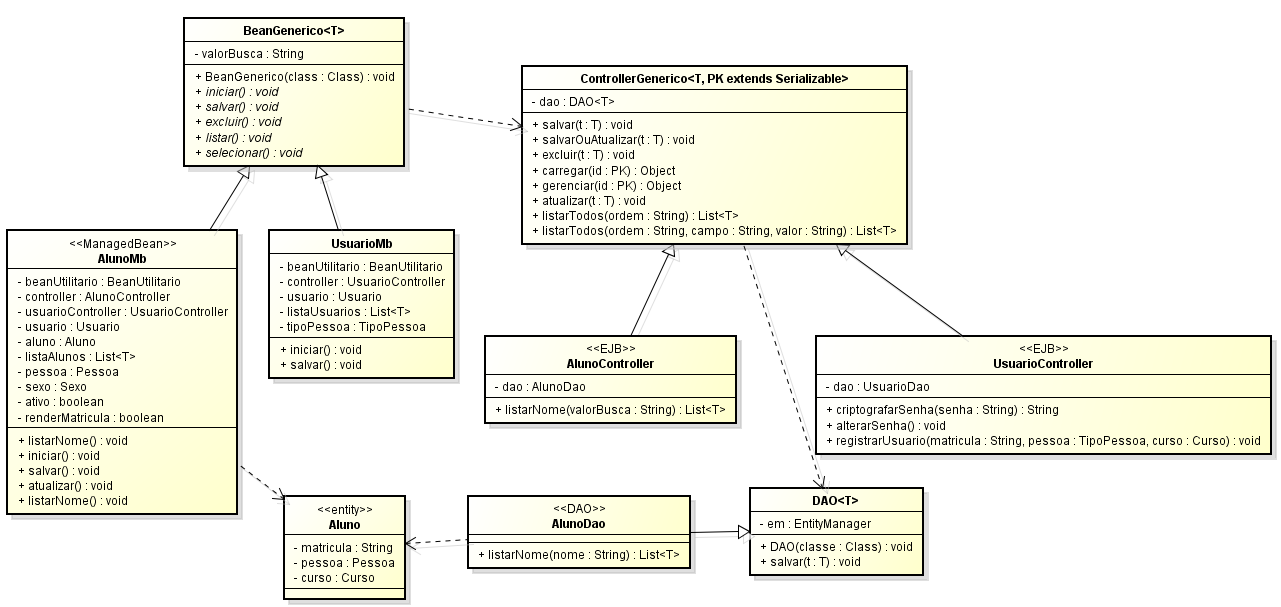
### Realização dos Casos de Uso

Nesta seção demonstraremos a realização de dois casos de uso da aplicação, através de Diagramas de Classe e Sequencia UML.

#### UC03 – Manter Alunos

Este caso de uso é responsável pelo cadastramento, consulta, alteração e exclusão de alunos do sistema. Ao ser cadastrado, o aluno é também registrado como um usuário do sistema, sendo criado seu login e senha. A Figura 6 apresenta as principais classes envolvidas na realização desse caso de uso.

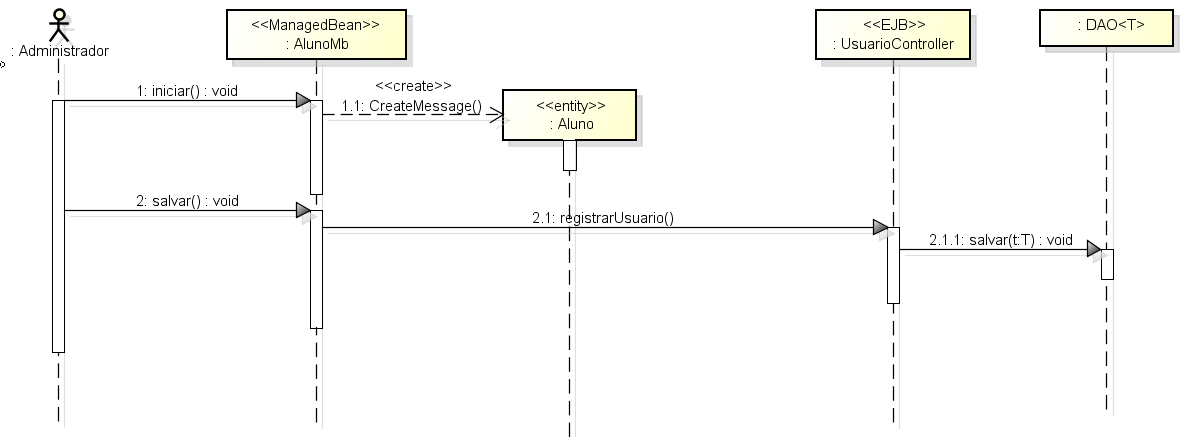
*Figura 6 – UC03 – Manter Alunos (Cadastro) – Principais Classes Envolvidas*



Fonte: Modelagem do Sistema FormChristus

Na realização desse UC, a partir da página de cadastramento, é acionado o *Managed* *Bean* AlunoMb, que cria uma entidade Aluno e a devolve para a apresentação. Ao serem preenchidos os campos do aluno e escolhida a opção salvar, o *Managed Bean* é novamente acionado, passando a requisição para o Controlador UsuarioController, que por sua vez aciona o DAO para gravação do aluno. Esse processo é mostrado na Figura 7.

*Figura 7 – Diagrama de Sequência para o Cadastro de Aluno (UC01)*

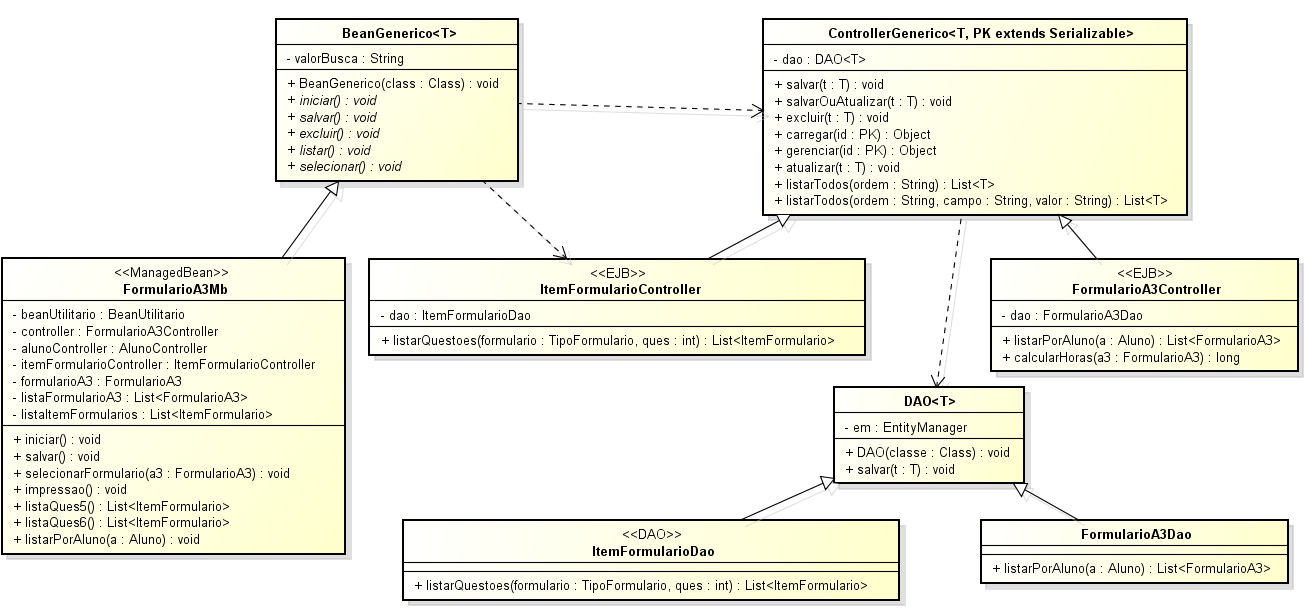


Fonte: Modelagem do Sistema FormChristus

#### UC08 – Preencher Formulário A3

Este UC é responsável pelo preenchimento do formulário A3, que se constitui no relatório de Atividade Complementar executada pelo aluno. As principais classes envolvidas nesse caso de uso são mostradas na Figura 8.

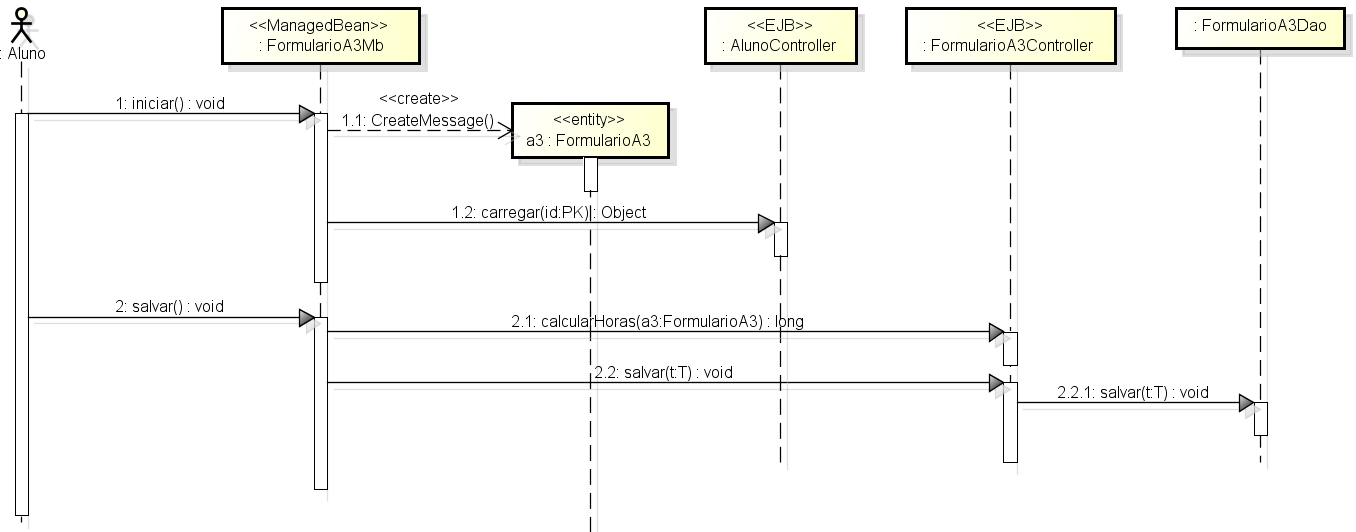
*Figura 8 – UC08 – Preencher Formulário A3 – Principais Classes Envolvidas*



Fonte: Modelagem do Sistema FormChristus

Na realização desse caso de uso, a partir da página de preenchimento do formulário na camada de apresentação, é acionado o *Managed Bean* FormularioA3Mb, que inicialmente obtém os dados do aluno a partir de seu controlador (AlunoController). Em seguida, uma instância da classe que representa o formulário (FormularioA3) é criada e devolvida à apresentação para preenchimento. Após o preenchimento do formulário, o Managed Bean é novamente chamado, repassando os dados ao controlador (FormularioA3Controller) e daí ao DAO para gravação, conforme mostrado na Figura 9.

*Figura 9 – Diagrama de Sequência para o UC08 – Preencher Formulário A3*



Fonte: Modelagem do Sistema FormChristus

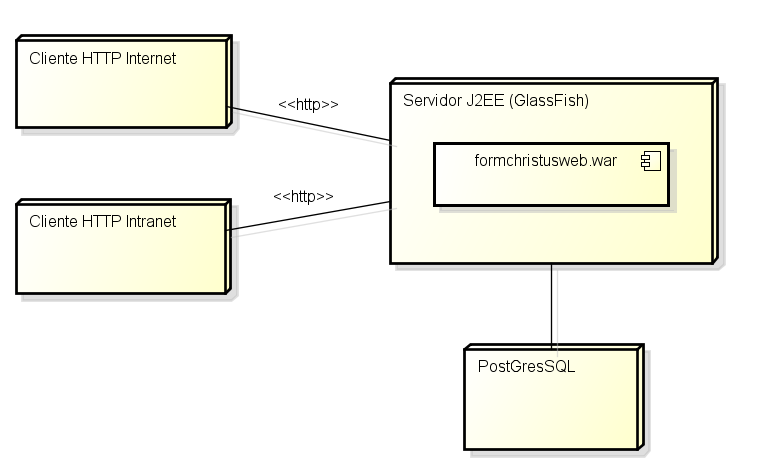
## Visão de Implantação

A aplicação foi desenvolvida em dois módulos distintos, a saber:

* FormChristusModelo – contém os pacotes contendo as classes do modelo e as enumerações utilizadas por esse modelo.
* FormChristusWeb – contém a aplicação web propriamente dita com suas devidas configurações. Contempla as configurações de segurança e armazena os arquivos contendo os modelos dos formulários que são gerados. Contém também pacotes contendo as classes *Managed Bean*, Controladores EJB e DAO.

Durante o processo de compilação e montagem da aplicação, o módulo FormChristusModelo e compilado em um arquivo JAR (Java Archive), que é utilizado como biblioteca para o módulo FormChristusWeb. O arquivo WAR (Web Archive) resultante é publicado no servidor de aplicações GlassFish, ficando assim disponível para utilização pelos usuários. Esse módulo se comunica com um servidor de banco de dados PostGres também rodando no servidor. A Figura 10 mostra essa configuração de implantação do sistema.

*Figura 10 – Diagrama de Implantação*



## Visão de Implementação

A aplicação possui um único módulo web, publicado no servidor de aplicações conforme descrito no item anterior.

### Componentes Reutilizados

A seguir são listados os componentes utilizados pelo sistema – que são suas dependências. As bibliotecas estão todas disponibilizadas em arquivos JAR. Cada biblioteca também tem suas dependências, que não são listadas aqui.

|  |  |
| --- | --- |
| **Nome do Componente** | **Versão** |
| Primefaces | 4.0 |
| Hibernate | 4.1.2 |
| Hibernate Validator | 4.3.1 |
| Spring Security | 3.0.7 |
| Jasper | 5.1.0 |

## Visão de Dados

# Padrões Adotados

O Arquiteto de Software é o responsável por elaborar este documento e por manter a integridade do mesmo durante o processo de desenvolvimento do software. Ele deve:

* Aprovar todas as mudanças arquiteturais significativas e documentá-las.
* Fazer parte do comitê que decide sobre os problemas que tenham algum impacto arquitetural.

# Referências

1. <http://community.jaspersoft.com/project/jasperreports-server>
2. <http://community.jaspersoft.com/project/ireport-designer>
3. <http://mahtabrasheed.wordpress.com/2013/01/25/what-is-java-server-faces-jsf-and-primefaces/>
4. <http://www.devmedia.com.br/ejb-3-1-versus-cdi-revista-java-magazine-105/25040>
5. <http://www.mastertheboss.com/cdi/using-cdi-to-manage-ejb-dependencies>
6. <http://www.yaw.com.br/open/projetos/cdi-jsf-ejb3/>
7. <http://docs.spring.io/spring-security/site/docs/3.0.x/reference/introduction.html>

# Arquitetura

O que é? E como é composta?

A arquitetura foi desenvolvida para ser totalmente de alta coesão e baixo acoplamento, e que ao mesmo tempo fosse independente de tecnologia de solução existentes no mercado.Utilizando o padrão MVC(Model View Controler).

* 1. Elementos que compõe a Arquitetura

Quais são os principais elementos da arquitetura?

A arquitetura é composta por alguns elementos, entenda-se classes, que em conjunto produzem o efeito desejado pela arquitetura como um produto final para o desenvolvimento.

Elementos pertencentes à arquitetura:

Modelo

Infraestrutura

Dominio

Visão

JSF

Nos próximos capítulos iremos discutir cada componente listado anteriormente e qual o seu papel dentro da arquitetura como um todo, além de discutir de forma sucinta a tecnologia e/ou o padrão adotado para a implementação do mesmo.

* 1. Modelos

O que é essa camada? E pra que serve?

No desenvolvimento de sistemas precisamos em muitas vezes fazer acesso a uma determinada base de dados. Em algumas linguagens de programação, principalmente as estruturadas, a separação real de funcionalidades na programação é muito complexa de forma que acessos a dados, consultas, entre outros elementos do sistema, acabam se misturando, o que ocasiona um alto acoplamento para termos alta coesão. Com a programação orientada a objetos isso já não ocorre em modelagens mais elaboradas. No caso da arquitetura desenvolvida temos uma camada apenas com classes que espelham o banco de dados ou camada de modelo. Nessa camada estão todas as classes mapeadas com as entidades do banco de dados.

# 

* 1. Infraestrutura

Qual a sua principal finalidade?

Quando estamos elaborando uma arquitetura para o desenvolvimento de sistemas, principalmente orientado a objetos, temos que nos preocupar com a separação real das camadas pertencentes à arquitetura. É nesse contexto que começamos a discutir os principais elementos da arquitetura, sendo que agora iremos começar detalhar a camada de infraestrutura e seu contexto dentro da arquitetura.

A camada de infraestrutura é composta por um conjunto de classe pertencentes à camada de acesso a dados, responsável por tratar as toda a requisição de acesso a dados provenientes da camada de regra de negocio, feita através do uso de uma interface DAO entre a camada de negócios e a camada de modelo. Dessa forma, as principais funcionalidades implementadas são: inserção, exclusão, alteração e consulta.

Com a criação do DAO, separamos o acesso a elementos de persistência, assim como o controle e a manipulação dos dados que são retornados de um banco de dados.

* 1. Domínio

Onde é implementada a regra de negócio?

Todo sistema é formado por um conjunto de regras de negócio, ou seja, um fluxo lógico que deve ser processado para que tenhamos o resultado desejado. Uma regra de negócio é representada na UML (linguagem de modelagem de sistemas orientada a objetos) através de um caso de uso (modelagem da regra de negócio). Com a análise orientada a objetos a preocupação com a separação real da regra de negócio das demais funcionalidades do sistema é constante e essencial para termos um sistema de baixo acoplamento e de fácil manutenção e extensão.

Por esse motivo foi criado o elemento chamado controle dentro da camada de domínio na arquitetura. Um controlador é responsável pela execução de um ou mais fluxos de execução que são modeladas em um caso de uso, ou seja, podemos dizer que o controlador é em si a implementação da regra de negócio. O mesmo pode ser modularizado, quando existem algumas particularidades dentro da implementação das regras, em classes que chamamos de RN (regras de negócio). O controlador faz uso da infraestutura para obter as informações necessárias para o seu processamento.

Com isso temos para cada caso de uso existente no sistema um controlador responsável por implementá-lo, assim temos um controle transacional muito mais robusto (por caso de uso ou pela interação entre eles), por exemplo, cada método dentro do controlador estará sempre sobre o mesmo contexto transacional.

* 1. Visão

Como é dividida? Quais seus principais elementos?

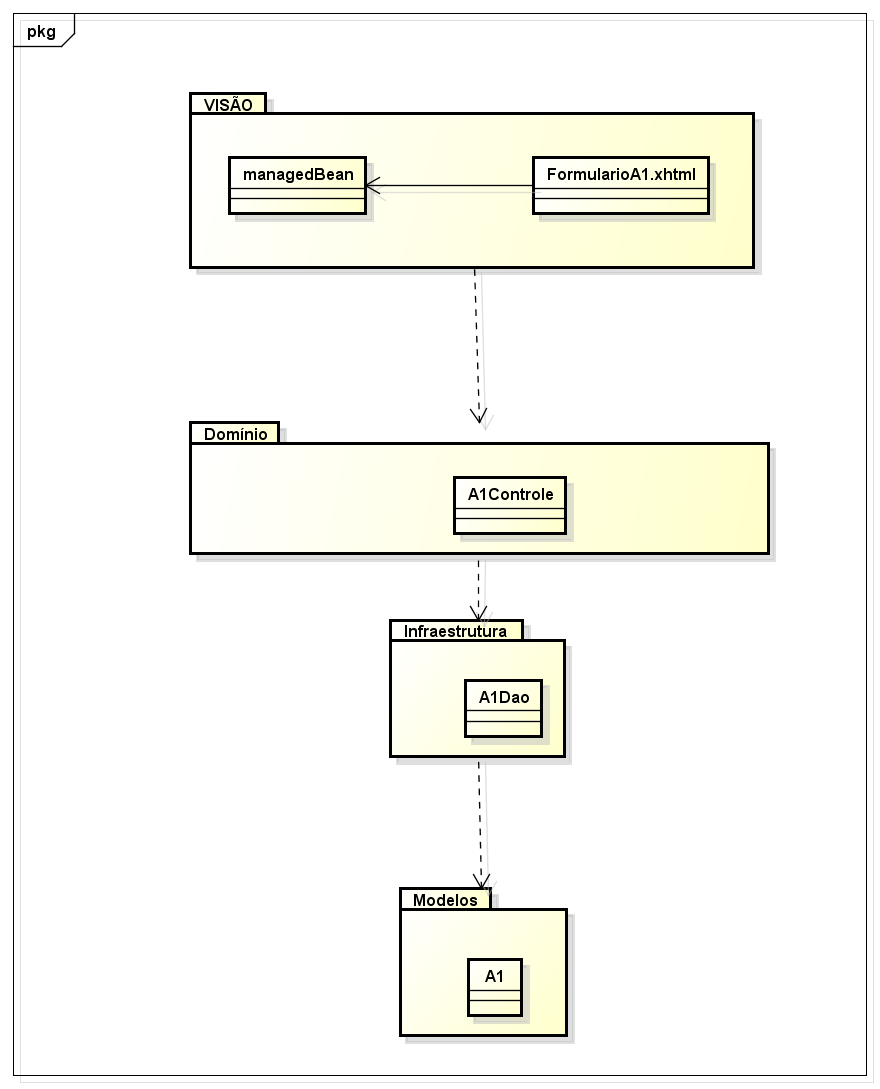
Nós já detalhamos bem a camada de regra de negócio e persistência de dados, agora iremos detalhar a camada de interface com o usuário. Nesse primeiro momento iremos falar sobre interfaces web, como ela foi dividida e o que foi envolvido na criação da mesma.

O GlassFish é um servidor de aplicações Java para web. É oficialmente endossado pela Sun como a Implementação de Referência (RI) para as tecnologias Java Servlet e JavaServer Faces (JSF). O GlassFish é robusto e eficiente o suficiente para ser utilizado mesmo em um ambiente de produção.

Parte da plataforma corporativa Java Enterprise Edition (J2EE ou Java EE) que abrange as tecnologias Servlet e JSP,JSF, incluindo tecnologias de apoio relacionadas como Realms e segurança, JNDI Resources e JDBC DataSources.

A interface web foi desenvolvida utilizando-se JSF.

Desenho GERAL da A rquitetura



# Padrões de Projeto

# Objetivos e Restrições Arquiteturais

* 1. Requisitos básicos
  2. Estratégias de implementação

1. Modelo de Arquitetura de Software que separa a representação da informação da interação do usuário com ele. O modelo (*model*) consiste nos dados da aplicação, regras de negócio, lógica e funções. Uma visão (*view*) pode ser qualquer saída de representação dos dados. O controlador (*controller*) faz a mediação das entradas e saídas, convertendo-as em comandos para o modelo ou a visão. [↑](#footnote-ref-1)