

UEM – Universidade Estadual de Maringá

Aplicações para WEB utilizando EJB

Eliezer Gomes Parangaba Filho

Maringá – 2003

Aplicações para WEB utilizando EJB

Eliezer Gomes Parangaba Filho

Monografia apresentada ao Departamento de Informática, como parte dos requisitos para obtenção do título de especialista em Tecnologia de Desenvolvimento para Web.

Orientador: Profa. Dra. Elisa Hatsue M. Huzita

DEDICATÓRIA

Dedico esta pesquisa a minha esposa Luciana, sempre companheira e amiga, que com sabedoria e fortaleza me guiou em vários momentos, para que eu não desviasse de meus objetivos. Inigualável mulher.

AGRADECIMENTOS

Em primeiro lugar a Deus, divino, absoluto, e aos meus amigos de viagem, Aldo, César, Cris, Eduardo e Mário, pelos bons momentos que passamos e por terem tornado agradável e possível a conclusão dessa longa jornada.

RESUMO

Cada vez mais existe a necessidade de desenvolvimento de aplicações escaláveis, robustas, seguras e distribuídas considerando a construção de componentes que sejam executados do lado do servidor. Para suprir estas necessidades surgiu *Enterprise JavaBeans* - EJB, que é uma arquitetura para componentes do lado servidor que possibilita e simplifica o processo de construir aplicações de objetos distribuídos em *Java*. Usando EJB pode-se escrever aplicações com essas características desejadas sem que seja necessário escrever sua própria infra-estrutura complexa para objetos distribuídos. EJB é desenvolvido para prover portabilidade e reusabilidade a qualquer que seja o fornecedor de serviços corporativos do *middleware*.

Este trabalho apresenta um estudo da tecnologia *Enterprise JavaBeans*, realizado com o intuito de adquirir e compartilhar experiências e conhecimentos. Como parte do trabalho é apresentado um estudo de caso da implementação de um protótipo usando EJB. Espera-se que este trabalho possa ser usado como fonte de informação e referência para profissionais da área da informática interessados nessa tecnologia.

ABSTRACT

Each more time there is the necessity of development of scaled, robust, safe, distributed applications through construction of components on the side server. With this, there is one architecture that propose to supply these necessities called Enterprise JavaBeans - EJB, that is an architecture for components on the side server and makes possible and simplifies the process of building applications of distributed objects in Java. Using EJB it can be written applications with these characteristics without writing the complex infrastructure for distributed objects. EJB is developed to provide portability and reusability to any middleware corporative supplier.

This work shows a study about the Enterprise JavaBeans technology and was make to aquire and to share experiences and knowledges. In the present work it is showed a case study of the prototype implementation using EJB. We hope that this work can be used like information reference for informatic's professionals and others.

SUMÁRIO

1	INTRODUÇÃO.....	10
2	REVISÃO BIBLIOGRÁFICA.....	12
2.1	COMPONENTES DE SOFTWARE.....	12
2.2	ARQUITETURA DE COMPONENTES.....	13
2.2.1	AS NECESSIDADES DO LADO DO SERVIDOR.....	14
2.3	SOLUÇÕES DE ARQUITETURAS DE COMPONENTES DO LADO DO SERVIDOR.....	15
2.3.1	<i>SUN MICROSYSTEMS'S JAVA 2 PLATFORM, ENTERPRISE EDITION (J2EE)</i> ...	15
2.4	AS TECNOLOGIAS J2EE.....	20
2.5	<i>ENTREPRISE JAVABEAN - EJB</i>	21
2.6	OS SEIS ELEMENTOS DE EJB.....	24
2.7	ELEMENTOS DA ARQUITETURA EJB.....	25
2.8	A ARQUITETURA EJB.....	26
2.8.1	CONTÊINER DE EJB.....	27
2.8.2	COMPONENTE EJB.....	29
2.8.3	TIPOS DE EJBS.....	30
2.9	RESUMO.....	35
3	ESTUDO DE CASO.....	36
3.1	A INSTALAÇÃO DO AMBIENTE.....	36
3.2	O MODELO DE DADOS.....	38
3.3	A SUBDIVISÃO DA APLICAÇÃO.....	38
3.4	OS SESSION BEANS.....	39
3.4.1	<i>SESSION BEAN AREA</i>	39
3.4.2	<i>SESSION BEAN SUBAREA</i>	40
3.4.3	<i>SESSION BEAN TIPO_USUARIO</i>	42
3.4.4	<i>SESSION BEAN TOPICOS</i>	43
3.4.5	<i>SESSION BEAN USUARIOS</i>	44
3.4.6	<i>SESSION BEAN RELEVANCIA</i>	46
3.4.7	<i>SESSION BEAN DIFICULDADE</i>	48
3.4.8	<i>SESSION BEAN QUESTAO</i>	49
3.4.9	<i>SESSION BEAN RESPOSTA</i>	51
3.4.10	<i>SESSION BEAN AVALIACAO</i>	52
3.4.11	<i>SESSION BEAN QUESTOES_AVALIACAO</i>	53
3.4.12	<i>SESSION BEAN RESOLUCAO</i>	55
3.4.13	<i>SESSION BEAN RESPOSTAS_AVALIACAO</i>	57
3.5	COMPILANDO CADA <i>SESSION BEAN</i>	58
3.6	PUBLICANDO O PROTÓTIPO NO JBOSS.....	59
3.7	O APLICATIVO CLIENTE EM JSP.....	60
3.8	RESUMO.....	63
4	RESULTADOS OBTIDOS.....	65
4.1	RESUMO.....	69
5	CONCLUSÃO E TRABALHOS FUTUROS.....	70
6	REFERÊNCIAS BIBLIOGRÁFICAS.....	71
7	ANEXOS.....	73
7.1	Classe para Manipulação e Conversão de Data.....	74
7.2	Descritor de Distribuição.....	76
7.3	Exemplo de dois dos treze <i>session beans</i> do projeto.....	79
7.4	Exemplo de duas das quatorze páginas JSP da aplicação cliente.....	90

LISTA DE FIGURAS

Figura 1 – Arquitetura de aplicativo EJB	26
Figura 2 – Arquitetura para clientes servlets ou páginas JSP	27
Figura 3 – Isolamento das classes <i>bean</i>	28
Figura 4 – Funcionamento de um contêiner EJB	28
Figura 5 – Esquema de 4 camadas	36
Figura 6 – Modelo de dados	38
Figura 7 – Estrutura de pastas para criação de uma aplicação EJB	58
Figura 8 – Visão Geral do Filesystem no NetBeans.....	59
Figura 9 – Estrutura de pastas para publicação da aplicação cliente JSP no Tomcat.....	63
Figura 10 – Tela principal do site	65
Figura 11 – Tela de login	66
Figura 12 – Tela principal após login	67
Figura 13 – Tela de cadastro de questões	67

LISTA DE ABREVIATURAS

API	Application Program Interface
BMP	Bean Manager Persistence
CMP	Container Manager Persistence
CORBA	Common Object Request Broker Architecture
DBMS	Database Manager System
EJB	Enterprise JavaBeans
ENC	Environment Naming Context
GUI	Graphic User Interface
HTML	Hyper Text Markup Language
IDE	Integrated Development Environment
IDL	Interface Definition Language
IIOB	Internet Inter-ORB Protocol
J2EE	Java 2 Platform Enterprise Edition
J2ME	Java 2 Platform Micro Edition
J2SDK	Java 2 Software Development Kit
J2SE	Java 2 Platform Standard Edition
JDBC	Java Database Connectivity
JDK	Java Development Kit
JMS	Java Messaging Service
JNDI	Java Naming and Directory
JSP	Java Server Pages
JTA	Java Transaction API
JTS	Java Transaction Service
LDAP	Lightweight Directory Access Protocol
MDB	Message Drive Bean
MVC	Model View Controller
OMG	Object Management Group
SQL	Structured Query Language
RMI	Remote Method Invocation
URL	Uniform Resource Locator
WORA	Write Once Run Anywhere
XML	Extensible Markup Language

1 INTRODUÇÃO

No mundo globalizado em que vivemos e com o advento da Internet e das redes de computadores, tem crescido a necessidade por aplicações distribuídas que possam ser acessadas através da rede. Com isso, é necessário ter uma arquitetura que dê suporte a criação desse tipo de aplicação, sendo de grande importância que essas sejam robustas e seguras.

Enterprise JavaBeans (EJB) é um componente do lado do servidor que encapsula a lógica de negócio. Esta precisa estar de acordo com as especificações EJB e só são distribuídas e podem executar apenas em um contêiner EJB. Escrevendo aplicativos EJB, podemos usufruir alguns benefícios oferecidos pelo contêiner EJB:

- Aplicativos EJB são fáceis de desenvolver, pois o desenvolvedor pode se concentrar na lógica de negócio;
- EJB são componentes, ao comprá-los de terceiros, se evita precisar ter de desenvolver seus próprios *beans*, o que significa desenvolvimento mais rápido do aplicativo;
- O contêiner EJB gerencia transações, detalhes de gerenciamento de estado, múltiplas seqüências e gerenciamento de conexões, tornando desnecessário conhecer detalhes destas implementações para desenvolver a aplicação;
- O contêiner EJB oferece segurança para os aplicativos como, por exemplo, a proteção das classes *beans* não permitindo que o cliente as acesse diretamente.

O objetivo deste trabalho é o estudo da tecnologia *Enterprise JavaBeans*, com o intuito de adquirir e compartilhar experiências e conhecimentos, onde como metodologia será desenvolvido um sistema de apoio a aplicação de avaliações através da WEB. O material resultante servirá como um roteiro para pessoas interessadas nessa tecnologia.

Esta monografia está organizada em 7 capítulos. O capítulo 2 apresenta conceitos sobre componentes de software, linguagem *Java* para desenvolvimento de componentes, a plataforma J2EE e EJB, elementos de sua arquitetura, funcionalidade e tipos fundamentais. O capítulo 3 apresenta o estudo de caso, todos os passos executados durante a implementação do protótipo. O capítulo 4 apresenta o resultado

obtido, bem como algumas de suas telas. Finalmente, o capítulo 5 apresenta as conclusões e sugestões de trabalhos futuros.

2 REVISÃO BIBLIOGRÁFICA

2.1 COMPONENTES DE SOFTWARE

Grady Booch (apud ROMÃO, 2000) define componente de software como "fragmento de um sistema, físico e substituível, que disponibiliza e está conforme a realização de um conjunto de interfaces". Por sua vez D'Souza e Wills (apud ROMÃO, 2000) definem-no como "um pacote coerente de artefactos de software que pode ser desenvolvido e disponibilizado de forma independente e autônoma e que, sem alterações, pode ser acoplado a outros componentes para construir algo maior". Uma definição mais completa chega-nos de Szyperski (apud ROMÃO, 2000) "(...) é uma unidade de composição com interfaces contratualmente especificadas e só com dependências explícitas do contexto. Um componente de software pode ser disponibilizado de forma independente e é passível de composição por terceiras partes".

Um componente de software é um código que implementa um conjunto bem definido de interfaces. Ele é um pequeno pedaço de lógica. Componentes não são aplicações completas, eles não podem ser executados sozinhos. Ao invés disto, eles devem ser utilizados como peças de um quebra cabeças para resolver uma variedade de problemas.

A idéia de componentes de software é muito poderosa. Uma companhia pode comprar um módulo bem definido que resolve um problema e combiná-lo com outros componentes para resolver um problema mais complexo. Componentes reusáveis são bem atraentes, pois promovem um rápido desenvolvimento da aplicação.

Assim que o desenvolvedor disponibilizar novas versões do componente, os clientes deste fornecedor irão querer um upgrade. Um desenvolvimento orientado a objeto introduz uma ótima prática de programação que auxilia a resolver este problema. Esta boa prática de programação consiste em separar a interface do componente de sua implementação:

- A interface de um componente: É o contrato do componente com o código que o chama. A interface contém os métodos e parâmetros que o componente aceita. A interface mascara a implementação para clientes do componente, então estes manipulam apenas o retorno da operação.
- A implementação de um componente: é o núcleo da lógica de negócio que um cliente necessita. Ela possui alguns algoritmos específicos, lógica e

dados. Estes dados são privados do componente e devem ser ocultados de qualquer código cliente que chama o componente.

Para que a separação entre interface e implementação seja efetiva, desenvolvedores devem escrever o código dos clientes apenas para a interface dos componentes, isto é chamado de programação baseada em interfaces. Estando escrevendo componentes, deve-se forçar os desenvolvedores a desenvolver dentro deste paradigma, publicando apenas a interface para os componentes, não sua implementação.

Separando a interface da implementação, podemos variar a lógica de negócio do componente sem modificar qualquer código do cliente.

A linguagem *Java* suporta a separação entre interface e implementação no nível sintático via a palavra-chave *interface* e *classe* (SUN, 2003). E porque ela é uma linguagem interpretada, a separação de código em arquivos diferentes assegura que clientes não tenham que recompilar seu código se for lançada uma nova versão do componente.

Em adição à separação da interface da implementação, *Java* é uma linguagem orientada a objeto que foi construída com independência de plataforma e é suportada por uma grande variedade de fornecedores. Isto faz a linguagem *Java* uma tecnologia ideal em que se possa basear para desenvolver componentes (ROMAN, 2001).

2.2 ARQUITETURA DE COMPONENTES

Segundo Roman (1999), para facilitar o processo de desenvolvimento de componentes, deve haver uma maneira padrão para construir, manipular e manter os componentes. Este padrão consiste no seguinte:

- Ferramentas para construir componentes: O processo de construir componentes deve ser eficiente, permitindo que o desenvolvedor de componentes se focalize em desenvolver a lógica interna do componente sem se preocupar com detalhes de infra-estrutura. Isto promove um rápido desenvolvimento da aplicação e é essencial para que os componentes possam ser utilizados com sucesso.
- Um compartimento que gerencie os componentes disponibilizados: Este compartimento é denominado de contêiner. Provê: um ambiente de execução para que os componentes desenvolvidos e disponibilizados

possam ser executados; um conjunto de serviços comuns que um conjunto de componentes poderão utilizar. Por exemplo, o contêiner pode automaticamente instanciar novos componentes quando for necessário, aliviando assim esta carga de trabalho do desenvolvedor de componentes. Para combinar qualquer contêiner com qualquer componente, deve existir uma regra bem definida entre eles. Esta regra permite a qualquer contêiner gerenciar qualquer componente.

- Ferramentas para disponibilizar e gerenciar componentes: Quando uma organização compra componentes de fornecedores, deve haver um conjunto de ferramentas para auxiliar na disponibilização e manutenção destes componentes. Uma arquitetura de componentes bem definida provê o padrão necessário para que diferentes fornecedores escrevam componentes, contêiner de componentes e ferramentas que se comuniquem com harmonia.

2.2.1 AS NECESSIDADES DO LADO DO SERVIDOR

Segundo Roman (1999), uma completa arquitetura de componentes deve disponibilizar de alguma maneira os seguintes benefícios:

- Possibilitar aos desenvolvedores criar componentes reusáveis;
- Possibilitar aos fornecedores prover contêiners para os componentes que forneçam um ambiente de execução robusto para os componentes;
- Possibilitar aos fornecedores prover ferramentas para gerenciar, desenvolver e disponibilizar os componentes.

Estes benefícios permitem que os desenvolvedores disponibilizem um conjunto de serviços comuns que a maioria dos componentes irão necessitar, salvando assim um precioso tempo dos desenvolvedores para criar e disponibilizar os componentes. Ao invés de reinventar a roda, o desenvolvedor pode simplesmente utilizar os recursos que ele necessita de outros fornecedores. Os profissionais que forem especialistas em determinado serviço podem disponibilizar um componente que implemente este serviço. E assim os desenvolvedores podem salvar um tempo precioso comprando componentes ao invés de desenvolvê-los.

2.3 SOLUÇÕES DE ARQUITETURAS DE COMPONENTES DO LADO DO SERVIDOR

Já faz alguns anos desde que a idéia do desenvolvimento de aplicações organizadas e divididas em múltiplas camadas surgiu. Desde então, várias espécies de servidores de aplicação surgiram no mercado. Estes servidores disponibilizam um ambiente de execução onde componentes podem ser executados e disponibilizam os serviços necessários para a aplicação (como repositório de recursos e segurança). Por causa disto, cada servidor de aplicação tem fornecido serviços para componentes de uma maneira não uniforme, ou seja, de um modo proprietário. Isto significa, que uma vez escolhido um servidor de aplicação, sua aplicação está amarrada a um fornecedor. Isto reduz fortemente a portabilidade, o que é incompatível com o paradigma do mundo *Java*, que defende a portabilidade. Isto também atrasa a comercialização de componentes porque um cliente não pode combinar um componente escrito em um servidor de aplicação com outro componente escrito em um outro servidor de aplicação diferente.

Por causa disto surgiu a necessidade de um padrão de arquitetura para componentes do lado do servidor. Esta arquitetura necessita que se escreva uma interface bem formulada entre o servidor de aplicação, o qual contém os componentes, e os componentes propriamente ditos. Estas interfaces permitem que os componentes sejam manipulados de uma maneira portátil, ao invés de uma maneira proprietária. Os fornecedores de componentes estarão habilitados a manter o foco na lógica da aplicação e não se preocupar com questões como repositório de recursos, segurança, e assim por diante. O objetivo é o rápido desenvolvimento da aplicação do lado do servidor. Isto permitirá que componentes sejam trocados em diferentes servidores de aplicação, sem ter que alterar o código ou recompilar os componentes.

Para solucionar esta necessidade, avanços tecnológicos têm acontecido para compensar a ausência de uma arquitetura de componentes para o lado do servidor, sendo o EJB a proposta da Sun Microsystems.

2.3.1 *SUN MICROSYSTEMS'S JAVA 2 PLATFORM, ENTERPRISE EDITION* (J2EE)

Sun Microsystems também percebeu a necessidade de uma arquitetura do

lado do servidor. Muitos fornecedores de componentes *Java* tem clamado por uma arquitetura de n-camadas do lado do servidor que fosse baseada em *Java*; a linguagem *Java* em si é bem adaptada para o servidor. O lado do cliente em *Java* tem muitos problemas, como inconsistência com interfaces de usuários entre plataformas, baixa velocidade destas interfaces com o usuário, e versões erradas da máquina virtual *Java* executando nas máquinas dos clientes. Mas do lado do servidor, *Java* é uma linguagem ideal. Desenvolvimentos do lado do servidor executam em um ambiente controlado, o que significa que a versão correta da linguagem de programação *Java* estará sempre sendo utilizada. A velocidade com que *Java* é executado é o motivo menos significativo do lado do servidor porque freqüentemente 80% ou mais do tempo de uma aplicação de n-camadas é gasto com operações com o banco de dados e um nível de rede.

Assim, para habilitar a computação do lado do servidor em *Java*, a Sun produziu uma plataforma completa de desenvolvimento chamada de *Java 2 Platform, Enterprise Edition* (J2EE). A missão da J2EE é prover um padrão de arquitetura corporativo que seja independente de plataforma, portátil, multiusuário, e totalmente escrito na linguagem *Java*. A pedra fundamental do J2EE é o *Enterprise JavaBeans* (EJB), um padrão para construir componentes do lado do servidor em *Java*.

J2EE simplifica muito a complexidade de construir uma aplicação baseada em componentes do lado do servidor que seja escalável, além do mais, J2EE é uma especificação e não um produto. Este especifica as regras de compromisso que as pessoas devem concordar quando estiverem escrevendo softwares corporativos. É de sua natureza não estar amarrado a um fornecedor; e também suporta desenvolvimento independente de plataforma. Isto encoraja os fornecedores a competir, oferecendo produtos cada vez melhores. Fornecedores então implementam essa especificação com seus produtos compatíveis a J2EE.

Para compreender porque o J2EE é necessário, é preciso ver um pouco da história da evolução da plataforma *Java*.

A Sun Microsystem's primeiro se focalizou em desenvolver um robusto pacote de desenvolvimento *Java*, o *Java Development Kit* (JDK). O JDK tornou-se a referência de implementação de fato da plataforma *Java*. Até este ponto, não foram realizados muitos esforços para o desenvolvimento da *middleware* do lado do servidor.

A Sun Microsystem's então reconheceu o poder da linguagem *Java* do lado do servidor e começou a desenvolver várias Application Program Interface (APIs) corporativas que provesses serviços de níveis corporativos do lado do servidor. Estes

serviços incluem os serviços *naming* e *lookup*, serviços de transação, e a API do *Enterprise JavaBeans* (EJB) 1.0. A Sun também começou a desenvolver elementos para os clientes *Java*, que possam oferecer funcionalidades para que os dispositivos dos clientes pudessem interagir com esses serviços no servidor.

Muitos meses depois que o EJB 1.0 foi concluído, os primeiros servidores de aplicação baseados em EJB, começaram a surgir no mercado (o BEA's *WebLogic* foi o primeiro). Estes servidores de aplicação tiveram algumas vantagens em relação a outros servidores de aplicação, como os serviços de *naming* e *lookup*, de transações e para operações com banco de dados. Os recém lançados servidores de aplicação serviram como uma importante fonte de informações para a Sun, pois eles trouxeram à luz alguns problemas com as APIs corporativas. Estes problemas incluíam o seguinte:

- Ambigüidades: as seções não tratadas, consideradas na especificação da API corporativa (particularmente o EJB 1.0) atrapalhava a portabilidade dos componentes. Isto não é desejável, pois violava o princípio da Sun de “Escreva uma vez, e execute em qualquer lugar”.
- Baixo sincronismo entre as APIs corporativas: Cada uma das APIs corporativas da Sun foi relatada, desenvolvida separada, independentemente.
- Controle incorreto de versões: Cada uma das APIs corporativas da Sun estava envolvida separadamente e tinha novos números de versões chegando a toda hora. Isto torna difícil programar usando EJB porque EJB depende destas APIs corporativas. Qual versão da API deveria ser utilizada? Isto não estava especificado, levando a códigos não portáveis entre servidores de aplicação.
- Ausência de mecanismos para testar a compatibilidade do servidor de aplicação: quando um fornecedor escreve um produto para o padrão EJB 1.0, não existe maneira para testar se este produto está compatível com a especificação da Sun. Similarmente, o cliente não tinha informação se um fornecedor era compatível com a especificação EJB 1.0.
- Ausência implementação de referência: As APIs corporativas eram simplesmente especificações, a Sun não proveu implementação alguma que fosse padrão de referência. Assim, os desenvolvedores não tinham uma implementação de referência para testar seus códigos. Apenas para

uma comparação, as APIs corporativas necessitavam de algo semelhante ao JDK para as APIs corporativas.

Percebendo os problemas com as APIs corporativas, a Sun Microsystem recentemente tomou um passo maior a frente na resolução destes problemas provendo três diferentes plataformas de desenvolvimento *Java*. Cada uma delas é um superconjunto das plataformas anteriores:

- *Java 2 Platform, Micro Edition (J2ME)*: é uma plataforma de desenvolvimento para dispositivos habilitados para *Java*, como *Palm Pilots*, *paggers*, *watches*, e assim por diante. Esta é uma forma muito restrita da linguagem *Java* para prover desempenho e atender a capacidade limitada destes pequenos dispositivos.
- *Java 2 Platform, Standard Edition (J2SE)*: contém serviços padrões *Java* para applets e aplicações, como facilidades para realizar entrada e saída, interface gráfica e outros
- *Java 2 Platform, Enterprise Edition (J2EE)*: contém as APIs corporativas *Java* e juntam todas em uma completa plataforma para classes corporativas de desenvolvimento do lado do servidor em *Java*.

A chegada do J2EE é significativa pois solucionou os problemas surgidos quando a Sun desenvolveu as APIs corporativas, incluindo o seguinte:

- Menor Ambigüidade: A Sun solucionou a maior quantidade das incompatibilidades com a especificação do EJB 1.1; o que permitiu corrigir os erros que restringiam a portabilidade.
- Sincronismo das APIs Corporativas: cada uma das APIs corporativas têm um regra clara no J2EE, como está definido no documento da Sun chamado *J2EE Application Programming Model*. Isto tornou claro como utilizar as APIs corporativas juntas para disponibilizar um ambiente do lado do servidor.
- Controle Conciso de Versões: A Sun Microsystems agora bloqueia as versões de cada especificação para API corporativas e agrupa todas as APIs formando uma versão de fato da J2EE. Isto proporciona portabilidade de código entre fornecedores de produtos porque cada fornecedor suporta exatamente a mesma revisão da API.
- Suporte a Testes: Como foi mencionado, não havia maneira de saber se um fornecedor estava implementando o EJB 1.0 apropriadamente. A Sun

fixou isto com J2EE, provendo um conjunto de testes a fornecedores para assegurar que seu produto implementa a especificação.

- Implementação de Referência: Para habilitar os desenvolvedores a escreverem código dentro do padrão J2EE assim como eles vinham desenvolvendo com o JDK.

Segundo Roman (1999), para que um componente resolva um problema de negócio com sucesso, tanto o desenvolvedor do componente quanto o cliente que usa o componente devem concordar com a sintaxe e semântica para chamar os métodos do componente. Assim, o fornecedor do componente deve publicar a interface do componente para o código do cliente.

Java também é uma linguagem muito conveniente para escrever componentes do lado do servidor, por um motivo muito importante: as aplicações do lado do servidor são dominadas por um ambiente de máquinas UNIX e mainframes. Isto significa que uma linguagem independente de plataforma para escrever componentes do lado do servidor acrescenta um grande valor porque um desenvolvedor pode escrever um componente uma vez e disponibilizá-lo em diversos clientes existente no ambiente do servidor. Isto significa que clientes com um legado de aplicativos (como programas escritos em COBOL e sistemas de mainframes) terão uma migração bem definida para o e-commerce e outro moderno processo de negócio.

2.4 AS TECNOLOGIAS J2EE

A especificação *Java 2 Platform, Enterprise Edition* é um conjunto robusto de serviços da *middleware* que facilita a vida dos desenvolvedores de aplicações do lado do servidor. As tecnologias que estão incluídas na plataforma J2EE são as seguintes (Sun, 2003):

- *Enterprise JavaBeans* (EJB): Define como os componentes do lado do servidor são escritos e provê um contrato padrão entre componentes e servidores de aplicação que os gerenciam.
- *Java Remote Method Invocation* (RMI): RMI permite a comunicação entre processos. RMI-IIOP é uma extensão portátil do RMI que pode usar o Internet Inter-ORB Protocol (IIOP) e pode ser utilizado para a integração com CORBA.
- *Java Naming and Directory Interface* (JNDI): Identifica a localização de componentes ou outros recursos através da rede.
- *Java Database Connectivity* (JDBC): É uma ponte com banco de dados relacional que permite, com relativa portabilidade, efetuar operações com banco de dados.
- *Java Transaction API* (JTA) e *Java Transaction Service* (JTS): as especificações JTA e JTS permitem que componentes tenham suporte a transações.
- *Java Messaging Service* (JMS): Oferece suporte à comunicação assíncrona entre objetos distribuídos.
- *Java Servlets* e *Java Server Pages* (JSPs): Servlets e JSP são componentes do lado do servidor que foram, idealmente, adaptados para uma computação baseada em requisição e resposta, como a iteração com clientes HTTP.
- *Java IDL*: *Java IDL* é uma implementação de CORBA baseada em *Java* provida pela Sun. *Java IDL* permite a integração com outras linguagens, e também permite que objetos distribuídos usufruam dos serviços CORBA. Assim, J2EE como toda a plataforma, é totalmente compatível com CORBA.
- *Java Mail*: O serviço *Java Mail* permite enviar mensagens de e-mail de uma maneira independente de plataforma e de protocolo para programas *Java*. Por exemplo, em um desenvolvimento J2EE do lado do servidor, pode-se

usar o *Java Mail* para confirmar a compra feita em um site de e-commerce na Internet enviando um e-mail para o cliente.

- *Extensible Markup Language (XML)*: Muitas tecnologias J2EE (assim como EJB 1.1 e JSP) dependem de XML como uma linguagem descritiva de seus conteúdos.

A *plataforma Java 2 Platform, Enterprise Edition (J2EE)*, é construída na tecnologia existente na plataforma *Java 2 Platform, Standard Edition (J2SE)*. J2EE inclui o suporte básico Java e várias bibliotecas (.awt, .net, .io), tanto a *applets* quanto a aplicações.

EJB, não se propõe a manipular o lado do cliente, mas sim componentes do lado do servidor (ROMAN, 1999). Eles tem em vista realizar operações do lado do servidor, como executar complexos algoritmos ou realizar um grande volume de transações de negócio. O lado do servidor tem necessidades bem diferentes daquelas expostas anteriormente, ou seja, tratar de apresentações de interfaces com o usuário. Componentes do lado do servidor necessitam ser executados em um ambiente robusto, tolerante a falhas, que suporte transações, e seguro quanto a acesso de vários usuários. O servidor de aplicação disponibiliza este ambiente para os EJB, e também o contêiner de tempo de execução necessário para gerenciá-los.

Finalmente, note que *JavaBeans*, *applets*, *servlets* e EJB não são tecnologias concorrentes. Pode-se usar *JavaBeans* como um componente para a construção de blocos de estruturas e combiná-las para construir um EJB que pode ser disponibilizado. E pode-se também provêr uma interface gráfica para os EJB com *applets* ou *servlets*.

2.5 ENTREPRISE JAVABEAN - EJB

Os EJB foram criados em Março de 1998, na primeira especificação da Sun com o intuito de promover uma arquitetura de objetos distribuídos pela Internet.

EJB é uma especificação de componentes *Java* para serem executados do lado do servidor que encapsula lógica de negócio, precisa estar de acordo com as especificações EJB, são distribuídos e podem executar em um contêiner EJB (servidor de aplicação), idêntico a como os *servlet* só podem ser executados dentro de seu contêiner. Um contêiner *servlet* oferece serviços tais como gerenciamento de sessão e segurança. Da mesma forma, um contêiner EJB oferece serviços em nível de sistema.

EJB é uma tecnologia que define um modelo para desenvolvimento e implantação de componentes de aplicação que executam em um servidor de aplicação. Segue a definição de WORA: “*Write Once, Run Anywhere*” e segue a especificação *Enterprise Java*.

Aplicativos EJB são fáceis de desenvolver, pois o desenvolvedor de aplicativos pode se concentrar na lógica de negócio e ao mesmo tempo, pode usar serviços fornecidos pelo contêiner EJB, tais como transações e conexão em conjunto. A parte mais difícil é o processo de aprendizagem.

EJB são componentes, o que faz com que possa ser comprado de terceiros, evitando o desenvolvimento dos seus próprios *beans*, o que significa que o desenvolvimento de seu aplicativo é mais rápido. A especificação assegura que os *beans* desenvolvidos por outros podem ser usados em seu aplicativo.

Esta tecnologia pode parecer complicada, mas pode simplificar o desenvolvimento de aplicações por resolver alguns requisitos de serviços que são importantes. (Manhães, 2002)

Esses requisitos são:

- Gerenciamento da Persistência. Muitas aplicações manipulam dados persistentes. Um *entity bean* (seção 2.8.3.1) provê gerenciamento automático de dados persistentes, inclusive dados legados.
- Acesso a dados concorrentemente. Aplicações empresarias multi-usuárias devem prover acesso concorrente a dados sem sacrificar a consistência. EJB são projetados para gerenciar acessos concorrentes automaticamente e possuem recursos multi-thread.
- Acesso a dados remotamente. Aplicações empresariais tipicamente acessam dados a partir de múltiplos recursos remotos. As interfaces remotas (de negócio) fornecem transparência de localização o que permite aumentar a disponibilidade.
- Modelo desenvolvido baseado em componentes. Componentes de software podem prover reusabilidade, portabilidade e uma clara separação da interface com a implementação. EJB provê estes benefícios uma vez que são componentes de software.

- Portabilidade. Uma aplicação que é portátil através de múltiplas plataformas de hardware e sistemas operacionais faz melhor uso dos recursos da empresa, principalmente em se tratando de sistemas heterogêneos e provê flexibilidade e integração. A portabilidade diminui os riscos de uma aplicação tornar-se obsoleta quando sistemas operacionais são atualizados ou hardware são substituídos.
- Controle Transacional e de Segurança. Dados Empresariais devem ser atualizados consistentemente e somente por aqueles que estiverem autorizados. EJB provê um controle declarativo das configurações de segurança e de transações, simplificando a customização e aumentando a flexibilidade. EJB podem também controlar as transações e a segurança programaticamente.
- Alta disponibilidade. Sistemas de missão crítica frequentemente necessitam estar disponíveis todo tempo. Implementações de EJB podem prover controle a falhas automaticamente quando um servidor sai do ar por falhas ou para manutenções, quando a rede está indisponível ou não confiável. EJB também gerencia uma religação de dados quando acontece uma falha de sistema.
- Escalabilidade. Aplicações freqüentemente necessitam manipular o aumento de demanda e novos requerimentos. Como o ciclo de vida é gerenciado pelo contêiner, as instâncias dos *beans* que servem as requisições podem ser colocadas em um repositório para maximizar a eficiência de recursos. Componentes podem ser migrados para balancear.
- Neutralidade do Cliente. Algumas aplicações requerem acesso por muitos tipos de cliente. Objetos de negócios como EJB provêm acesso para um modelo de aplicação para qualquer tipo de cliente. Clientes *Java* podem acessar os EJBs através das interfaces padrões (*Home* e *Remote*). Clientes não *Java* podem acessar EJBs usando *Common Object Request Broker Architecture* (CORBA) ou interfaces *Web services*.

Segundo Manhães (2002), no passado as empresas construíram seu próprio *middleware* com o intuito de se ter uma solução que fosse adequada às realidades da empresa e esbarraram nos seguintes inconvenientes:

- Custo de manter um time de desenvolvimento e de manutenção de código.
- Um *middleware* de alto nível é extremamente complicado de construir.
- Pouca reusabilidade da solução, porque estes sistemas muitas vezes não eram adequados para a realidade WEB e desta forma o percentual de reconstrução era muito alto e principalmente quando era necessário a construção de pontes (bridges), porque as diferentes tecnologias utilizadas não conversavam satisfatoriamente o custo aumentava ainda mais.
- Caso fosse necessário conectar a um outro DBMS que não fosse o do projeto original exigia uma reprogramação muito grande principalmente por que poucos sistemas eram organizados em três camadas com um foco orientado a *Model-View-Controller* (MVC). (Almeida, 2003)

Assim que o EJB se tornar maduro, este cenário se tornará cada vez mais uma realidade. Assumindo que os fornecedores irão seguir fielmente o contrato do EJB, um verdadeiro mundo *plug-and-play* com características corporativas se tornará uma realidade. Mesmo se isto não for inteiramente uma verdade hoje, a visão de componente EJB portáveis através de servidores de aplicações heterogêneos se solidificará nos próximos anos.

2.6 OS SEIS ELEMENTOS DE EJB

Os fundamentos da portabilidade do EJB são os contratos e regras que cada parte deve seguir. Guiado pelo tema de dividir e conquistar, o EJB particionou as responsabilidades de um desenvolvimento em EJB em seis diferentes papéis (KURNIAWAN, 2002): Desenvolvedor *Bean*, Assembler de aplicativo, Distribuidor, Administrador de sistema, Provedor de servidor EJB e Provedor de contêiner EJB.

Pode ser estranho como são necessários tantos participantes para prover um desenvolvimento em EJB. A resposta para isto é que EJB habilita pessoas ou companhias a se especializarem em tarefas e esta divisão do trabalho conduz a um bom desenvolvimento do trabalho. Por exemplo, um disponibilizador de EJB não

precisa ser um especialista em desenvolvimento de softwares corporativos mas precisa apenas ser um especialista em disponibilizar uma solução já desenvolvida em EJB em um domínio particular.

A especificação do EJB distingue cada papel muito claramente, possibilitando especialistas em diferentes áreas a participar de um desenvolvimento sem perder a interoperabilidade. Observe que também existe a possibilidade de um ou mais destes papéis serem combinados. Por exemplo, um servidor de EJB e um contêiner de EJB podem realmente vir de um mesmo fornecedor. Para as outras partes, EJB meramente sugere as responsabilidades que cada parte deve assumir, assim um administrador de sistema pode também muito bem ser um disponibilizador do sistema. Para as outras partes, como o provedor e o contêiner de *beans*, EJB define um conjunto rígido de interfaces e guias que devem ser rigorosamente seguidos por todos. Para claramente definir o papel de cada parte, EJB define os fundamentos de uma arquitetura de componentes distribuída e escalável onde muitos fornecedores de produtos podem interagir.

2.7 ELEMENTOS DA ARQUITETURA EJB

Servidor EJB: Servidor de aplicação genérico que fornece um ambiente compatível com a especificação da arquitetura EJB. Fornece um ou mais contêiners para os componentes nele implantados. Responsável pelo gerenciamento e coordenação da alocação de recursos como: *Threads*, processos, memória, conexões a banco de dados.

Contêiner EJB: Fornece o contexto de execução e o contexto transacional aos componentes, registra o componente no serviço de *namings*, cria e destrói instâncias, fornece interface remota para o componente, gerencia transações, estado e persistência.

Componentes EJB:

Interface Home: Define os métodos de ciclo de vida do componente, criação, remoção e busca. Através dessa interface, os clientes vêem componentes EJB como uma coleção homogênea de instâncias.

Interface Remote: Define os métodos funcionais do componente, representa a visão que o cliente terá do componente. Expõe todas as interfaces relacionadas à aplicação.

Classe do Componente: Implementa os métodos funcionais.

Tipos de componentes EJB:

Componentes Entidades (*Entity Beans*)

Componentes Sessão (*Session Beans*)

2.8 A ARQUITETURA EJB

A arquitetura EJB define um modelo de sistema distribuído, baseado em componentes. O modelo de programação define um conjunto de perfis, através de um conjunto de contratos, que definem uma plataforma comum de desenvolvimento. O principal objetivo destes contratos é assegurar a portabilidade entre os fornecedores de servidores de EJB ou de componentes enquanto oferecem apoio a um rico conjunto de funcionalidades.

A arquitetura de aplicativo EJB, Figura 1, estende a arquitetura de aplicativos web, acrescentando uma outra camada.

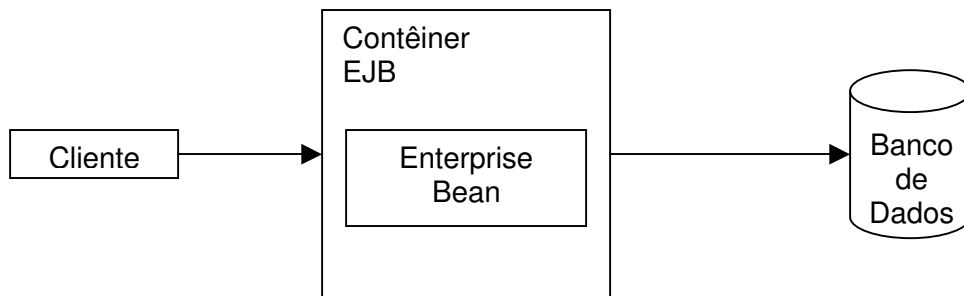


Figura 1 – Arquitetura de aplicativo EJB

Os clientes de um EJB podem ser um aplicativo tradicional *Java*, um *applet*, uma página JSP ou *servlet* ou um outro EJB.

Os clientes nunca chamam diretamente seus métodos. A comunicação entre clientes e *beans* é realizada através do contêiner EJB. Quando um cliente é um *servlet* ou uma página JSP, a estrutura de um aplicativo EJB se parece com a da Figura 2.

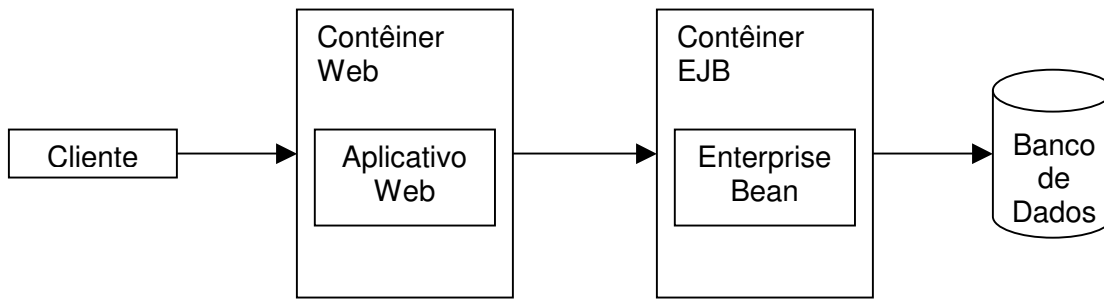


Figura 2 – Arquitetura para clientes servlets ou páginas JSP

2.8.1 CONTÊINER DE EJB

O conceito de Contêiner na arquitetura J2EE define um conjunto de componentes que atendem uma especificação. Existem diversos tipos de contêiner como *Applet Contêiner*, *WEB Contêiner*, *EJB Contêiner*, em que um componente para executar nestes ambientes tem que atender às especificações descritas nos mesmos.

Baseado na definição de EJB, poderíamos definir um contêiner acrescentando que os componentes executam em um programa que define o funcionamento *runtime* (tempo de execução) da especificação. Se há o tipo de contêiner que define o padrão de objetos distribuídos então estaremos falando em EJB Contêiner.

Um EJB não pode ser executado fora de um contêiner, porque o mesmo gerencia todo o seu aspecto em um ambiente de execução, como por exemplo: o acesso remoto ao *bean*, segurança, persistência, transações, concorrência, acesso aos recursos.

Através da Figura 3, podemos ver que o contêiner isola o EJB do acesso direto pelas aplicações cliente. Quando essas invocam uma chamada remota a um EJB, o contêiner, inicialmente, intercepta a invocação para assegurar persistência, transação e segurança pelas propriedades do *Bean* para toda a operação que o cliente executa, conforme demonstrado na Figura 4. Como o contêiner gerencia todos esses fatores automaticamente para o *bean*, então o desenvolvedor não precisa implementar este tipo de lógica para o código do *bean*. O desenvolvedor pode focar nas regras de negócio enquanto o contêiner cuida dos serviços solicitados pelo *bean* e que são definidos em um arquivo chamado *deployment descriptor*.

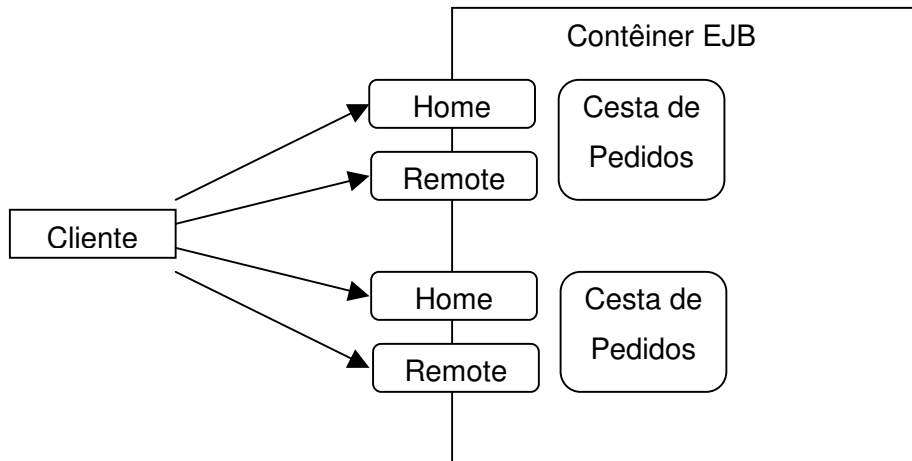


Figura 3 – Isolamento das classes *bean*

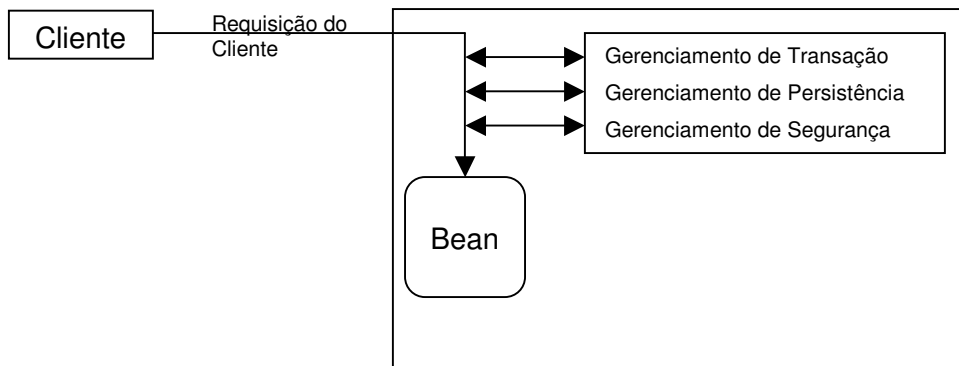


Figura 4 – Funcionamento de um contêiner EJB

O contêiner gerenciará muitos *beans* simultaneamente do mesmo modo que o *Web Server* gerencia muitos *servlets*. Para reduzir o consumo de memória e processamento, o repositório de recursos do contêiner gerencia os ciclos de vida de todos os *beans* com muito cuidado. Quando um *bean* não está sendo usado, o contêiner o colocará no repositório para ser reutilizado por outro cliente, ou possivelmente eliminará da memória quando o contêiner decidir que ele não é mais necessário, o que pode ser definido, por exemplo, de acordo com o número de acessos. Devido ao fato de que aplicações cliente não têm acesso direto aos *beans*, o contêiner vive entre o cliente e o *bean*, portanto as aplicações cliente desconhecem completamente as atividades do contêiner.

Um *bean* que não está em uso, por exemplo, pode ser armazenado a partir da memória em algum outro mecanismo que varia entre tipos de EJB (*Session*, *Entity* e *MDB*) no servidor, enquanto a referência remota fica intacta. Quando o cliente invoca

um método na interface remota o contêiner simplesmente revitaliza o *bean* para servir a requisição, deixando o processo totalmente transparente para a aplicação cliente.

Um EJB depende do contêiner para qualquer serviço que necessite. Se ele necessita acessar uma conexão com outro *Bean* tudo é feito da mesma forma que um cliente comum passando pelo contêiner.

Para isto ele interage com o contêiner através de três mecanismos:

- *Métodos de Callback*: Todo EJB implementa uma *Interface* EJB e esta interface as quais definem vários métodos, chamados métodos de *callback*. Cada método *callback* alerta o *bean* de um diferente evento no seu ciclo de vida e o contêiner invocará estes métodos para notificar o *Bean* quando o mesmo deva ser criado, removido, persistido na base de dados, etc. Os métodos de *callback* dão chance ao *bean* de fazer algum processamento antes ou depois de algum evento gerado pelo contêiner.
- *EJBContext Interface*: Todo EJB obtém um objeto de contexto de EJB, o qual é uma referência direta para o contêiner. A interface *EJBContext* provê métodos para interagir com o contêiner tanto que o *bean* pode requisitar a informação sobre o seu ambiente como a identificação do cliente ou o status de uma transação ou pode obter referências remotas para ele mesmo.
- *Java Naming and Directory Interface (JNDI)*: JNDI é uma extensão da plataforma *Java* para acesso a sistemas como *Lightweight Directory Access Protocol* (LDAP), sistemas de arquivo, etc. Todo *bean* automaticamente tem acesso para um sistema especial chamado *Environment Naming Context* (ENC). O ENC é gerenciado pelo contêiner e acessa os *beans* usando JNDI. O JNDI ENC permite um *bean* acessar recursos como conexões JDBC, outros *EJB*, e propriedades específicas para aquele *bean*. Isto é feito internamente, porém como já dissemos a programação é a mesma que de um cliente comum.

2.8.2 COMPONENTE EJB

Para implementar um EJB, é necessário definir duas interfaces e uma ou duas classes:

- *Remote Interface:*

A interface *remote* de um EJB define os métodos de negócio, ou seja, os métodos presentes para o mundo externo que irá utilizar o *bean*. Ela estende `javax.ejb.EJBObject`, que estende `Java.rmi.Remote`.

- *Home Interface:*

A interface *home* define o ciclo de vida do *bean* com métodos de criação, remoção ou localização de *beans*. Esta interface herda de `javax.ejb.EJBHome`, a qual estende de `Java.rmi.Remote`.

- *Bean Class:*

A classe *bean* é a responsável por implementar seus métodos de negócios. É importante notar que a classe usualmente não implementa a Interface *Home* nem a Interface *Remote*. Entretanto na classe *Bean* deve-se ter métodos que implementam as assinaturas definidas na interface *remote* e métodos que correspondem, não são iguais, a métodos na interface *home*.

- *Primary Key:*

A chave primária é uma classe muito simples que provê um ponteiro para a base de dados. Somente *entity beans* necessitam de uma chave primária, o único requerimento desta classe é que implemente `java.io.Serializable`.

2.8.3 TIPOS DE EJBS

Um componente EJB possui três tipos fundamentais: *entity beans*, *session beans* e *message drive beans* (KURNIAWAN, 2002). Como regra geral, um *entity bean* modela conceitos de negócios que podem ser representados por nomes, como um comprador, um equipamento, um item de inventário. Em outras palavras *entity beans* modelam objetos do mundo real. *Session beans* são uma extensão da aplicação cliente e são responsáveis por modelar processos ou tarefas (ações) e *Message Drive*

Beans que são um tipo de *bean* que se assemelha ao *Session Bean* e que são acessados a partir de um sistema de mensagens pelo *Java Message Service (JMS)*.

2.8.3.1 ENTITY BEANS

Entity beans modelam conceitos de negócios que podem ser expressos por nomes e esta é uma regra importante quando na sua modelagem de sistema você precisa eleger um candidato a ser um. Estes tipos de *beans* modelam realmente um dado no Banco de Dados onde uma instância representa uma linha na tabela de um Banco de Dados. A partir disto podemos ter o questionamento: Porque não acessar o banco de dados diretamente? Existem muitas vantagens em se usar *Entity Beans* ao invés de acessar a base de dados diretamente. Estes *beans* promovem um simples mecanismo de acesso e alteração de dados. É muito mais fácil, por exemplo, de executar um método para atualizar um campo de um *entity* do que fazer um comando SQL para isto. Quando um novo *bean* é criado, um novo registro deve ser inserido na base de dados e uma instância do *bean* associada a este dado. Conforme o *bean* é usado e seu estado é alterado estas mudanças devem estar sincronizadas com a base de dados. Este processo de coordenação dos dados do *database* com a instância do *bean* denomina-se persistência.

Um *entity bean* pode ser considerado como de vida longa (*long-lived*) porque sobrevive a uma falha no servidor de aplicativos e por sua característica em um modelo N-tier ficaria mais próximo do DBMS (*Database Manager System*).

Com relação aos *session beans* os *entity beans* são muito diferentes porque ao invés de se modelar um fluxo de controle (*workflow*), que é papel do *session bean*, eles são responsáveis pelos dados de negócios (*Core Business Data*). Com relação ao mapeamento com as tabelas, um *entity bean* não necessita estar mapeado para uma única tabela, pode ser representado por várias porque na fase de design nem toda tabela de seu esquema de persistência necessita estar mapeada para um *entity bean*.

O conjunto de classes básicas que compõem um *entity bean* seria:

- *Remote Interface*, onde seriam executados os métodos de alteração de dados propriamente ditos.
- *Home Interface*, onde existem métodos de criação, Localização e remoção de dados.

- *Bean Class*, que implementa a persistência propriamente dita.
- *Primary Key*, classe que identifica unicamente um *entity bean* de acordo com o tipo de *bean*, a *interface home* e o contexto do contêiner no qual está sendo usado.
- As interfaces de pesquisa que definem os SQLs que serão executados para pesquisas específicas, excetuando a pesquisa pela chave primária, e que a partir da definição do método de pesquisa na interface *home* promovem a opção de definir métodos personalizados de busca além do método *findByPrimaryKey* que já é padrão.

De um modo geral, o que temos que entender sobre *entity beans* é que:

- É uma representação em memória de um dado persistido em um meio de armazenamento;
- Sabe quando ler o dado de um armazenamento e salvar os campos neste meio de armazenamento;
- Um Objeto pode ser modificado em memória para mudar os valores do dado.

2.8.3.1.1 PERSISTÊNCIA GERENCIADA PELO CONTÊINER

A persistência de *beans* gerenciada pelo contêiner é o meio mais simples de se desenvolver porque se permite focar na lógica de negócio, delegando a responsabilidade da persistência para o contêiner. Quando se faz a publicação do *bean* em um *Application Server* que possua um Contêiner de EJBs, você identifica quais são os campos do *entity bean* que precisam ser gerenciados pelo contêiner, como eles são mapeados para o *database*. Uma vez que você tem definido os campos que serão gerenciados, o contêiner gera a lógica necessária para salvar o estado do *bean* automaticamente.

Os campos que são gerenciados pelo contêiner podem ser um tipo primitivo *Java* ou objetos serializados. A maioria dos *beans* que fazem persistência usam tipos primitivos *Java* quando persistem para os bancos de dados relacionais pois é mais fácil mapear tipos primitivos nestes bancos dados.

A vantagem de se usar persistência gerenciada pelo contêiner é que o *bean* pode ser definido independentemente da base de dados usada para armazenar o estado. *Beans* gerenciados pelo contêiner tomam vantagem de ser publicados tanto

em Servidores de Aplicativos que suportam bancos de dados relacionais quanto Servidores de Aplicativos que suportam banco de dados orientado a objetos. O estado do *bean* é definido independentemente, o qual faz o *bean* mais reusável e flexível através das aplicações. Não se considerando alguns tipos de dados como datas e *timestamps*, que de banco para banco existem padrões diferentes, quando se faz um *entity bean* bem projetado a flexibilidade na troca de um banco de dados é muito grande e não há problemas de reprogramação.

A desvantagem de persistência gerenciada pelo contêiner é que eles requerem uma ferramenta de mapeamento sofisticada para definir como os campos dos *beans* são gerenciados pelo contêiner. Em alguns casos isto pode ser, de uma maneira simples, como o mapeamento direto de tipos diretamente para uma coluna de uma tabela de um banco de dados relacional ou a serialização de um *bean* para um arquivo.

2.8.3.1.2 PERSISTÊNCIA GERENCIADA PELO BEAN

Persistência gerenciada pelo *bean* é mais complicada do que a persistência gerenciada pelo contêiner porque você deve explicitamente escrever a lógica de persistência dentro da classe *bean*. Com a finalidade de escrever a manipulação da persistência na classe do *bean* você deve inserir no código qual o tipo de base de dados está sendo usada e como seus campos necessitam ser mapeados para a base de dados.

A persistência gerenciada pelo *bean* promove uma maior flexibilidade na forma como o estado é manipulado entre a sua instância e a base de dados. *Entity bean*, nos quais são definidos por *joins* complexas, combinação de diferentes bancos de dados ou outros recursos entre os sistemas legados são beneficiados pela persistência gerenciada pelo *bean*. Essencialmente, esse tipo de persistência é a alternativa quando as ferramentas de publicação são inadequadas para mapear a instância do *bean* para a base de dados.

A desvantagem é que mais linhas de código são necessárias para defini-lo. O desenvolvedor tem que entender a estrutura da base de dados e desenvolver a lógica para criar, atualizar e remover o dado associado com o *entity bean*. Isto requer que haja um controle maior com inserção de código nos métodos que são usados pelo contêiner na persistência gerenciada automaticamente como *ejbLoad()* e *ejbStore()*, principalmente além de *ejbActivate()* e *ejbPassivate()*. O desenvolvedor também deve

prover a programação dos métodos de busca de dados (*ejbFind...*) que são definidos pela interface *home*.

Outra desvantagem é que a sua construção define explicitamente a base de dados e o tipo de banco de dados. Quaisquer mudanças na base de dados ou na estrutura de dados requerem mudanças na definição da instância do *bean* e estas mudanças podem não ser triviais. Em resumo esse tipo de persistência não é independente da base de dados como a persistência gerenciada pelo contêiner e é por isto que as especificações EJB estão focando cada vez mais na melhoria dos serviços nos *entity beans* com a persistência gerenciada pelo contêiner. Por exemplo, na especificação EJB 2.0 a persistência *Container Manager Persistence* (CMP) foi bastante melhorada no sentido de melhorar a relação de *joins* complexos e integridades referenciais entre tabelas que no caso da especificação 1.1 deixa bastante a desejar, obrigando os desenvolvedores a desenvolverem o modelo *Bean Manager Persistence* (BMP).

2.8.3.2 SESSION BEANS

Os *session beans*, representam as ações de negócios. Por isto eles atendem mais o lado da aplicação cliente que necessita acionar um serviço. Especificando este serviço, eles podem representar o *workflow* que descreve todos os passos para realizar uma tarefa tais como reserva de passagem ou uma cesta de pedidos em um site B2C ou ainda um conjunto de operações financeiras que necessitam manter uma integridade transacional, ou seja, tudo ou nada.

Session beans podem gerenciar as iterações entre *entity beans*, descrevendo como os *entities* trabalham em conjunto para se realizar uma determinada tarefa.

A relação entre eles pode ser comparada a uma peça de teatro onde os atores representam os *Entity Beans* e o script o *Session Beans*.

Os *Session Beans* são divididos em 2 tipos básicos: *stateless* e *statefull*. Um *session bean stateless* é uma coleção de serviços cada qual sendo representado por um método específico. Quando uma aplicação cliente, que pode ser até um outro EJB, invoca um de seus métodos, o método é executado e retorna o resultado sem nenhum compromisso de se manter o estado do Objeto entre as chamadas para este mesmo EJB que está no repositório. Pense nele como um conjunto de procedimentos ou programas *batch* que executam uma requisição baseada em alguns parâmetros e

retornam um resultado. Eles tendem para ser de propósitos gerais ou reusáveis como um serviço de software.

Um *session bean statefull* é uma extensão da aplicação cliente. Quando um cliente obtém uma conexão com ele o seu estado é mantido entre as chamadas do mesmo cliente até o cliente remover esta conexão. Isto significa que o estado dos atributos de sua classe permanece inalterado enquanto o cliente mantém a conexão. Internamente o Contêiner executa um *ejbPassivate()* neste estado, ou seja, persiste em um armazenamento temporário quando o cliente fica sem usar por algum tempo ou o número de chamadas é maior que os *beans* que estão disponíveis no repositório.

Esta característica se chama estado conversacional porque representa uma conversa contínua entre o mesmo e o cliente. Eles representam uma lógica que pode ser capturada uma aplicação cliente entre chamadas e temos como exemplo uma cesta de pedidos de livros em um site como o Amazon.

2.9 RESUMO

Neste capítulo, foram apresentados conceitos sobre componentes de software, o porque a linguagem *Java* é ideal para o desenvolvimento de componentes, as tecnologias incluídas na plataforma J2EE, conceitos sobre EJB, os elementos de sua arquitetura, sua funcionalidade e seus tipos fundamentais.

No próximo capítulo, serão apresentados todos os passos executados na implementação do protótipo, desde a instalação de todo o ambiente de desenvolvimento e execução, até a compilação e publicação dos pacotes.

3 ESTUDO DE CASO

O objetivo desse projeto é desenvolver um sistema exemplo (protótipo) para WEB, utilizando *Java J2EE* (JSP) na camada de apresentação, EJB/JBoss como servidor de aplicações para a camada de negócios e MySQL como banco de dados. O protótipo a ser desenvolvido consistirá em um “Sistema de apoio a aplicação de Avaliações através da WEB” que possibilite ao professor testar o conhecimento de seus alunos. A Figura 5 representa todo o esquema gráfico das quatro camadas envolvidas:

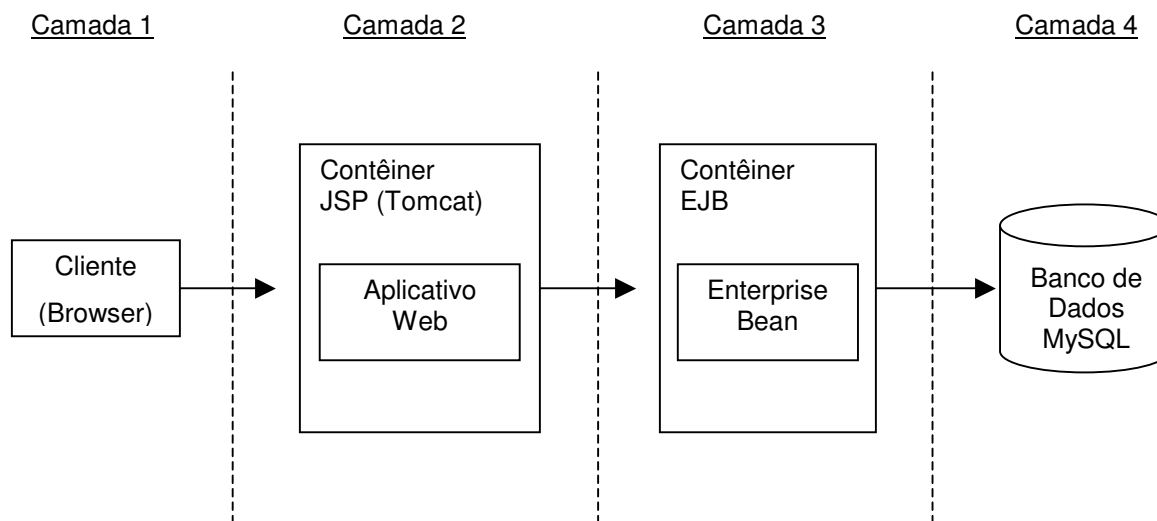


Figura 5 – Esquema de 4 camadas

3.1 A INSTALAÇÃO DO AMBIENTE

Para que o protótipo funcione, é necessário realizar a instalação do J2SDK, JBoss e Tomcat. O protótipo foi desenvolvido sobre o ambiente instalado no sistema operacional Windows 2000.

A primeira instalação necessária é do J2SDK, disponível para download em <http://java.sun.com/j2se/1.4.2/download.html>. A versão utilizada nesse trabalho foi a 1.4.2_02, vá até a seção *Download J2SE v 1.4.2_02* de Windows Installation (SDK).

Para o JBoss, pode-se fazer o download diretamente em <http://prdownloads.sourceforge.net/jboss/JBoss-2.4.4.zip> ou vá a página de download em <http://www.jboss.org/index.html?module=html&op=userdisplay&id=downloads>.

Após o download, descompacte o arquivo na raiz da unidade, ex: C:\JBoss-2.4.4. Pronto, a instalação está concluída. Agora, para que o JBoss possa ser iniciado, é necessário criar uma variável de ambiente chamada JAVA_HOME, onde o seu conteúdo é a do caminho da pasta onde o J2SDK está instalado. Ex: C:\J2SDK1.4.0_03.

Para o Tomcat, pode-se fazer o download em <http://jakarta.apache.org/site/binindex.cgi> na seção *Release Builds*. Após o download, descompacta-se o arquivo na raiz da unidade, ex: C:\jakarta-tomcat-4.1.24. Pronto, a instalação está concluída. Agora, para que o Tomcat possa ser iniciado, é necessário criar uma variável de ambiente chamada CATALINA_HOME, onde o seu conteúdo é o caminho da pasta onde o Tomcat foi instalada. Ex: C:\jakarta-tomcat-4.1.24.

Estando tudo instalado, pode-se inicializar o JBoss e o Tomcat. Para isso, vá à pasta onde foi instalado o JBoss, abra a subpasta Bin e execute o arquivo RUN.BAT. Para a inicialização do Tomcat, vá à pasta onde foi instalado, abra a subpasta Bin e execute o arquivo STARTUP.BAT.

Para verificar se o Tomcat está funcionando, abra o browser na seguinte URL: <http://localhost:8080/>. A porta padrão do servidor é 8080, mas pode ser alterada facilmente para outra dentro do arquivo C:\jakarta-tomcat-4.1.24\conf\server.xml:

```
<!-- Define a non-SSL Coyote HTTP/1.1 Connector on port 8080 -->
<Connector className="org.apache.coyote.tomcat4.CoyoteConnector"
    port="8080" minProcessors="5" maxProcessors="75"
    enableLookups="true" redirectPort="8443"
    acceptCount="100" debug="0" connectionTimeout="20000"
    useURIVValidationHack="false" disableUploadTimeout="true" />
```

Após a alteração desse arquivo, finalize e inicialize novamente o Tomcat.

Posteriormente, para que o nosso protótipo possa ser executado, o pacote completo com todas as classes do protótipo e o pacote mm.mysql-208-bin.jar (conexão com MySQL) deverá ser publicado dentro do JBoss e alguns pacotes *Java* (JBoss.jar, JBoss-J2EE.jar, JNPServer.jar) e as interfaces *Home* e *Remote* do protótipo deverão ser publicadas dentro do Tomcat em locais específicos. Tudo isso será visto em detalhes adiante.

3.2 O MODELO DE DADOS

Com base no modelo de dados apresentado na Figura 6, foram criados diversos *sessions beans* relacionados às tabelas do modelo de dados. Cada um destes contém métodos que fazem acesso aos dados contidos nas tabelas do banco de dados. Assim os *session beans* encapsulam o acesso às informações armazenadas no banco de dados de forma que todo acesso a essas precisa, obrigatoriamente, acontecer através do acesso aos seus métodos.

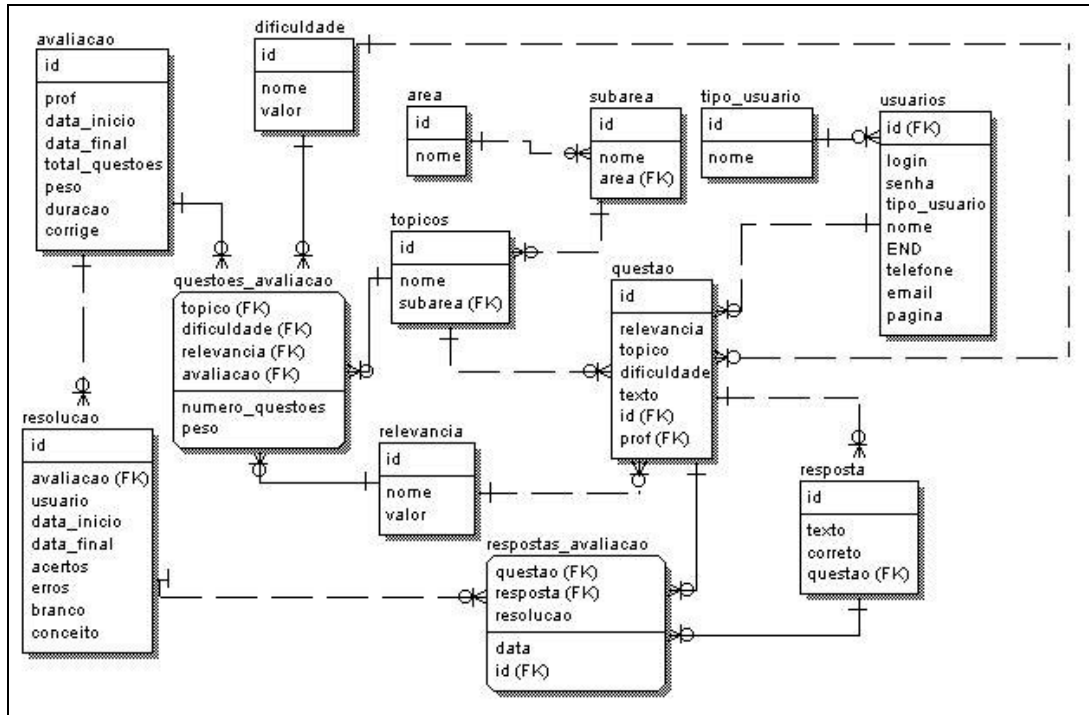


Figura 6 – Modelo de dados

3.3 A SUBDIVISÃO DA APLICAÇÃO

Como dito anteriormente, os *session beans* são responsáveis por armazenar a lógica de negócio. Uma aplicação, sempre que possível, pode ser subdividida em unidades lógicas, cada uma responsável por uma determinada funcionalidade. Sendo assim, para Implementação do protótipo foram desenvolvidos treze *session beans*. Cada um é responsável pela manipulação de uma tabela no banco de dados.

3.4 OS SESSION BEANS

A seguir, segue a descrição de todos os *session beans* e as assinaturas dos métodos contidos no protótipo.

3.4.1 SESSION BEAN AREA

Contém a lógica para manipular a tabela AREA, é composto pelos seguintes métodos:

- `public boolean inclui(int Id, String Nome) throws RemoteException;`

Realiza a inclusão de novas áreas. O retorno dessa função indica se essa operação foi realizada com sucesso.

- `public boolean altera(int Id, String Nome) throws RemoteException;`

Realiza a alteração do nome de uma área. O retorno dessa função indica se essa operação foi realizada com sucesso.

- `public boolean exclui(int Id) throws RemoteException;`

Realiza a exclusão de uma área. O retorno dessa função indica se essa operação foi realizada com sucesso.

- `public String[] localizaId(int Id) throws RemoteException;`

Realiza consulta em áreas através da chave primária. O retorno dessa função é um *array* de *strings* contendo o registro encontrado ou *null* caso contrário.

- `public ArrayList localizaNome(String Nome) throws RemoteException;`

Realiza consulta em áreas através do nome. O retorno dessa função é uma coleção de *array* de *Strings* contendo os registros encontrados, caso nenhum registro seja encontrado será retornado uma coleção com tamanho zero.

- `public ArrayList localizaTudo() throws RemoteException;`

Realiza consulta em todas as áreas. Retorna uma coleção de *array* de *strings* contendo todos os registros encontrados, caso nenhum registro seja encontrado será retornado uma coleção com tamanho zero.

Esse *session bean* é composto pelos arquivos `Area.java`, `AreaHome.java` e `AreaBean.java`.

3.4.2 SESSION BEAN SUBAREA

Contém a lógica para manipular a tabela SUBAREA, é composto pelos seguintes métodos:

- `public boolean inclui(int Id, String Nome, int Area) throws RemoteException;`

Realiza a inclusão de novas subáreas. O retorno dessa função indica se essa operação foi realizada com sucesso.

- `public boolean altera(int Id, String Nome, int Area) throws RemoteException;`

Realiza a alteração dos dados de uma subárea. O retorno dessa função indica se essa operação foi realizada com sucesso.

- `public boolean exclui(int Id) throws RemoteException;`

Realiza a exclusão de uma subárea. O retorno dessa função indica se essa operação foi realizada com sucesso.

- `public String[] localizaId(int Id) throws RemoteException;`

Realiza consulta em subáreas através da chave primária. O retorno dessa função é um *array* de *strings* contendo o registro encontrado ou *null* caso contrário.

- `public ArrayList localizaNome(String Nome) throws RemoteException;`

Realiza consulta em subáreas através do nome. O retorno dessa função é uma coleção de *array* de *strings* contendo os registros encontrados, caso nenhum registro seja encontrado será retornado uma coleção com tamanho zero.

- `public ArrayList localizaArea(int Area) throws RemoteException;`

Realiza consulta em subáreas através da área. O retorno dessa função é uma coleção de *array* de *strings* contendo os registros encontrados, caso nenhum registro seja encontrado será retornado uma coleção com tamanho zero.

- `public ArrayList localizaTudo() throws RemoteException;`

Realiza consulta em todas as subáreas. Retorna uma coleção de *array* de *strings* contendo todos os registros encontrados, caso nenhum registro seja encontrado será retornado uma coleção com tamanho zero.

Esse *session bean* é composto pelos arquivos `SubArea.java`, `SubAreaHome.java` e `SubAreaBean.java`.

3.4.3 *SESSION BEAN TIPO_USUARIO*

Contém a lógica para manipular a tabela TIPO_USUARIO, é composto pelos seguintes métodos:

- `public boolean inclui(int Id, String Nome) throws RemoteException;`

Realiza a inclusão de novos tipos de usuários. O retorno dessa função indica se essa operação foi realizada com sucesso.

- `public boolean altera(int Id, String Nome) throws RemoteException;`

Realiza a alteração do nome de um tipo de usuário. O retorno dessa função indica se essa operação foi realizada com sucesso.

- `public boolean exclui(int Id) throws RemoteException;`

Realiza a exclusão de um tipo de usuário. O retorno dessa função indica se essa operação foi realizada com sucesso.

- `public String[] localizaId(int Id) throws RemoteException;`

Realiza consulta em tipos de usuários através da chave primária. O retorno dessa função é um *array* de *strings* contendo o registro encontrado ou *null* caso contrário.

- `public ArrayList localizaNome(String Nome) throws RemoteException;`

Realiza consulta em tipos de usuários através do nome. O retorno dessa função é uma coleção de *array* de *strings* contendo os registros encontrados, caso nenhum registro seja encontrado será retornado uma coleção com tamanho zero.

- `public ArrayList localizaTudo() throws RemoteException;`

Realiza consulta em todos os tipos de usuários. Retorna uma coleção de *array* de *strings* contendo todos os registros encontrados, caso nenhum registro seja encontrado será retornado uma coleção com tamanho zero.

Esse *session bean* é composto pelos arquivos `Tipo_Usuário.java`, `Tipo_UsuárioHome.java` e `Tipo_UsuarioBean.java`.

3.4.4 SESSION BEAN TOPICOS

Contém a lógica para manipular a tabela TOPICOS, é composto pelos seguintes métodos:

- `public boolean inclui(int Id, String Nome, int SubArea) throws RemoteException;`

Realiza a inclusão de novos tópicos. O retorno dessa função indica se essa operação foi realizada com sucesso.

- `public boolean altera(int Id, String Nome, int subArea) throws RemoteException;`

Realiza a alteração dos dados de um tópico. O retorno dessa função indica se essa operação foi realizada com sucesso.

- `public boolean exclui(int Id) throws RemoteException;`

Realiza a exclusão de um tópico. O retorno dessa função indica se essa operação foi realizada com sucesso.

- `public String[] localizaId(int Id) throws RemoteException;`

Realiza consulta em tópicos através da chave primária. O retorno dessa função é um *array* de *strings* contendo o registro encontrado ou *null* caso contrário.

- `public ArrayList localizaNome(String Nome) throws RemoteException;`

Realiza consulta em tópicos através do nome. O retorno dessa função é uma coleção de *array* de *strings* contendo os registros encontrados, caso nenhum registro seja encontrado será retornado uma coleção com tamanho zero.

- `public ArrayList localizaTudo() throws RemoteException;`

Realiza consulta em todos os tópicos. Retorna uma coleção de *array* de *strings* contendo todos os registros encontrados, caso nenhum registro seja encontrado será retornado uma coleção com tamanho zero.

Esse *session bean* é composto pelos arquivos `Topico.java`, `TopicoHome.java` e `TopicoBean.java`.

3.4.5 **SESSION BEAN USUARIOS**

Contém a lógica para manipular a tabela `USUARIOS`, é composto pelos seguintes métodos:

- `public boolean inclui(int Id, String Login, String Senha, int Tipo_Usuario, String Nome, String End, String Telefone, String Email, String Pagina) throws RemoteException;`

Realiza a inclusão de novos usuários. O retorno dessa função indica se essa operação foi realizada com sucesso.

- `public boolean altera(int Id, String Login, String Senha, int Tipo_Usuario, String Nome, String End, String Telefone, String Email, String Pagina) throws RemoteException;`

Realiza a alteração dos dados de um usuário. O retorno dessa função indica se essa operação foi realizada com sucesso.

- `public boolean exclui(int Id) throws RemoteException;`

Realiza a exclusão de um usuário. O retorno dessa função indica se essa operação foi realizada com sucesso.

- `public String[] localizaId(int Id) throws RemoteException;`

Realiza consulta de usuários através da chave primária. O retorno dessa função é um *array* de *strings* contendo o registro encontrado ou *null* caso contrário.

- `public ArrayList localizaLogin(String Login) throws RemoteException;`

Realiza consulta de usuários através do login. O retorno dessa função é uma coleção de *array* de *strings* contendo os registros encontrados, caso nenhum registro seja encontrado será retornado uma coleção com tamanho zero.

- `public ArrayList localizaNome(String Nome) throws RemoteException;`

Realiza consulta de usuários através do nome. O retorno dessa função é uma coleção de *array* de *strings* contendo os registros encontrados, caso nenhum registro seja encontrado será retornado uma coleção com tamanho zero.

- `public ArrayList localizaTipo_Usuario(int Tipo_Usuario) throws RemoteException;`

Realiza consulta de usuários através do tipo do usuário. O retorno dessa função é uma coleção de *array* de *strings* contendo os registros encontrados,

caso nenhum registro seja encontrado será retornado uma coleção com tamanho zero.

➤ `public ArrayList localizaTudo() throws RemoteException;`

Realiza consulta de todos os usuários. Retorna uma coleção de *array* de *strings* contendo todos os registros encontrados, caso nenhum registro seja encontrado será retornado uma coleção com tamanho zero.

➤ `public String[] UsuarioValido(String login, String Senha) throws RemoteException;`

Verifica se o login e a senha de um usuário é válido. O retorno dessa função é um *array* de *strings* contendo o registro encontrado ou *null* caso contrário.

Esse *session bean* é composto pelos arquivos `Usuarios.java`, `UsuariosHome.java` e `UsuariosBean.java`.

3.4.6 SESSION BEAN RELEVANCIA

Contém a lógica para manipular a tabela RELEVANCIA, é composto pelos seguintes métodos:

➤ `public boolean inclui(int Id, String Nome, double Valor) throws RemoteException;`

Realiza a inclusão de novas relevâncias. O retorno dessa função indica se essa operação foi realizada com sucesso.

- `public boolean altera(int Id, String Nome, double Valor) throws RemoteException;`

Realiza a alteração dos dados de uma relevância. O retorno dessa função indica se essa operação foi realizada com sucesso.

- `public boolean exclui(int Id) throws RemoteException;`

Realiza a exclusão de uma relevância. O retorno dessa função indica se essa operação foi realizada com sucesso.

- `public String[] localizaId(int Id) throws RemoteException;`

Realiza consulta em relevâncias através da chave primária. O retorno dessa função é um *array* de *strings* contendo o registro encontrado ou *null* caso contrário.

- `public ArrayList localizaNome(String Nome) throws RemoteException;`

Realiza consulta em relevâncias através do nome. O retorno dessa função é uma coleção de *array* de *strings* contendo os registros encontrados, caso nenhum registro seja encontrado será retornado uma coleção com tamanho zero.

- `public ArrayList localizaTudo() throws RemoteException;`

Realiza consulta em todas as relevâncias. Retorna uma coleção de *array* de *strings* contendo todos os registros encontrados, caso nenhum registro seja encontrado será retornado uma coleção com tamanho zero.

Esse *session bean* é composto pelos arquivos `Relevancia.java`, `RelevanciaHome.java` e `RelevanciaBean.java`.

3.4.7 *SESSION BEAN* DIFICULDADE

Contém a lógica para manipular a tabela DIFICULDADE, é composto pelos seguintes métodos:

➤ `public boolean inclui(int Id, String Nome, double Valor) throws RemoteException;`

Realiza a inclusão de novas dificuldades. O retorno dessa função indica se essa operação foi realizada com sucesso.

➤ `public boolean altera(int Id, String Nome, double Valor) throws RemoteException;`

Realiza a alteração dos dados de uma dificuldade. O retorno dessa função indica se essa operação foi realizada com sucesso.

➤ `public boolean exclui(int Id) throws RemoteException;`

Realiza a exclusão de uma dificuldade. O retorno dessa função indica se essa operação foi realizada com sucesso.

➤ `public String[] localizaId(int Id) throws RemoteException;`

Realiza consulta em dificuldades através da chave primária. O retorno dessa função é um *array* de *strings* contendo o registro encontrado ou *null* caso contrário.

➤ `public ArrayList localizaNome(String Nome) throws RemoteException;`

Realiza consulta em dificuldades através do nome. O retorno dessa função é uma coleção de *array* de *strings* contendo os registros encontrados, caso nenhum registro seja encontrado será retornado uma coleção com tamanho zero.

- `public ArrayList localizaTudo() throws RemoteException;`

Realiza consulta em todas as dificuldades. Retorna uma coleção de *array* de *strings* contendo todos os registros encontrados, caso nenhum registro seja encontrado será retornado uma coleção com tamanho zero.

Esse *session bean* é composto pelos arquivos `Dificuldade.java`, `DificuldadeHome.java` e `DificuldadeBean.java`.

3.4.8 **SESSION BEAN QUESTAO**

Contém a lógica para manipular a tabela `QUESTAO`, é composto pelos seguintes métodos:

- `public boolean inclui(int Id, int Relevancia, int Prof, int Topico, int Dificuldade, String Texto) throws RemoteException;`

Realiza a inclusão de novas questões. O retorno dessa função indica se essa operação foi realizada com sucesso.

- `public boolean altera(int Id, int Relevancia, int Prof, int Topico, int Dificuldade, String Texto) throws RemoteException;`

Realiza a alteração dos dados de uma questão. O retorno dessa função indica se essa operação foi realizada com sucesso.

- `public boolean exclui(int Id) throws RemoteException;`

Realiza a exclusão de uma questão. O retorno dessa função indica se essa operação foi realizada com sucesso.

➤ `public String[] localizaId(int Id) throws RemoteException;`

Realiza consulta em questões através da chave primária. O retorno dessa função é um *array* de *strings* contendo o registro encontrado ou *null* caso contrário.

➤ `public ArrayList localizaRelevancia(int Relevancia) throws RemoteException;`

Realiza consulta em questões através da relevância. O retorno dessa função é uma coleção de *array* de *strings* contendo os registros encontrados, caso nenhum registro seja encontrado será retornado uma coleção com tamanho zero.

➤ `public ArrayList localizaProf(int Prof) throws RemoteException;`

Realiza consulta em questões através do professor. O retorno dessa função é uma coleção de *array* de *strings* contendo os registros encontrados, caso nenhum registro seja encontrado será retornado uma coleção com tamanho zero.

➤ `public ArrayList localizaTopico(int Topico) throws RemoteException;`

Realiza consulta em questões através do tópico. O retorno dessa função é uma coleção de *array* de *strings* contendo os registros encontrados, caso nenhum registro seja encontrado será retornado uma coleção com tamanho zero.

➤ `public ArrayList localizaDificuldade(int Dificuldade) throws RemoteException;`

Realiza consulta em questões através da dificuldade. O retorno dessa função é uma coleção de *array* de *strings* contendo os registros encontrados, caso nenhum registro seja encontrado será retornado uma coleção com tamanho zero.

- `public ArrayList localizaTudo() throws RemoteException;`

Realiza consulta em todas as questões. Retorna uma coleção de *array* de *strings* contendo todos os registros encontrados, caso nenhum registro seja encontrado será retornado uma coleção com tamanho zero.

Esse *session bean* é composto pelos arquivos `Questao.java`, `QuestaoHome.java` e `QuestaoBean.java`.

3.4.9 SESSION BEAN RESPOSTA

Contém a lógica para manipular a tabela RESPOSTA, é composto pelos seguintes métodos:

- `public boolean inclui(int Id, int Questao, String Texto, boolean Correto) throws RemoteException;`

Realiza a inclusão de novas respostas. O retorno dessa função indica se essa operação foi realizada com sucesso.

- `public boolean altera(int Id, int Questao, String Texto, boolean Correto) throws RemoteException;`

Realiza a alteração dos dados de uma resposta. O retorno dessa função indica se essa operação foi realizada com sucesso.

- `public boolean exclui(int Id) throws RemoteException;`

Realiza a exclusão de uma resposta. O retorno dessa função indica se essa operação foi realizada com sucesso.

- `public String[] localizaId(int Id) throws RemoteException;`

Realiza consulta em respostas através da chave primária. O retorno dessa função é um *array* de *strings* contendo o registro encontrado ou *null* caso contrário.

➤ `public ArrayList localizaQuestao(int Questao) throws RemoteException;`

Realiza consulta em dificuldades através da questão. O retorno dessa função é uma coleção de *array* de *strings* contendo os registros encontrados, caso nenhum registro seja encontrado será retornado uma coleção com tamanho zero.

➤ `public ArrayList localizaTudo() throws RemoteException;`

Realiza consulta em todas as respostas. Retorna uma coleção de *array* de *strings* contendo todos os registros encontrados, caso nenhum registro seja encontrado será retornado uma coleção com tamanho zero.

Esse *session bean* é composto pelos arquivos Resposta.java, RespostaHome.java e RespostaBean.java.

3.4.10 SESSION BEAN AVALIACAO

Contém a lógica para manipular a tabela AVALIACAO, é composto pelos seguintes métodos:

➤ `public boolean inclui(int Id, int Prof, String Data_Inicio, String Data_Final, int Total_Questoes, int Peso, int Duracao, boolean corrige) throws RemoteException;`

Realiza a inclusão de novas avaliações. O retorno dessa função indica se essa operação foi realizada com sucesso.

➤ `public boolean altera(int Id, int Prof, String Data_Inicio, String Data_Final, int Total_Questoes, int Peso, int Duracao, boolean corrige) throws RemoteException;`

Realiza a alteração dos dados de uma avaliação. O retorno dessa função indica se essa operação foi realizada com sucesso.

➤ `public boolean exclui(int Id) throws RemoteException;`

Realiza a exclusão de uma avaliação. O retorno dessa função indica se essa operação foi realizada com sucesso.

➤ `public String[] localizaId(int Id) throws RemoteException;`

Realiza consulta em avaliações através da chave primária. O retorno dessa função é um *array* de *strings* contendo o registro encontrado ou *null* caso contrário.

➤ `public ArrayList localizaProf(int Prof) throws RemoteException;`

Realiza consulta em avaliações através do professor. O retorno dessa função é uma coleção de *array* de *strings* contendo os registros encontrados, caso nenhum registro seja encontrado será retornado uma coleção com tamanho zero.

➤ `public ArrayList localizaTudo() throws RemoteException;`

Realiza consulta em todas as avaliações. Retorna uma coleção de *array* de *strings* contendo todos os registros encontrados, caso nenhum registro seja encontrado será retornado uma coleção com tamanho zero.

Esse *session bean* é composto pelos arquivos Avaliacao.java, AvaliacaoHome.java e AvaliacaoBean.java.

3.4.11 SESSION BEAN QUESTOES_AVALIACAO

Contém a lógica para manipular a tabela QUESTOES_AVALIACAO, é composto pelos seguintes métodos:

- `public boolean inclui(int Avaliacao, int Topico, int Relevancia, int Dificuldade, int Numero_Questoes, int Peso) throws RemoteException;`

Realiza a inclusão de novas questões de avaliação. O retorno dessa função indica se essa operação foi realizada com sucesso.

- `public boolean altera(int Avaliacao, int Topico, int Relevancia, int Dificuldade, int Numero_Questoes, int Peso) throws RemoteException;`

Realiza a alteração dos dados de uma questão de avaliação. O retorno dessa função indica se essa operação foi realizada com sucesso.

- `public boolean exclui(int Avaliacao, int Topico, int Relevancia, int Dificuldade) throws RemoteException;`

Realiza a exclusão de uma questão de avaliação. O retorno dessa função indica se essa operação foi realizada com sucesso.

- `public String[] localizaId(int Avaliacao, int Topico, int Relevancia, int Dificuldade) throws RemoteException;`

Realiza consulta em questões de avaliação através da chave primária. O retorno dessa função é um *array* de *strings* contendo o registro encontrado ou *null* caso contrário.

- `public ArrayList localizaAvaliacao(int Avaliacao) throws RemoteException;`

Realiza consulta em questões de avaliação através da avaliação. O retorno dessa função é uma coleção de *array* de *strings* contendo os registros encontrados, caso nenhum registro seja encontrado será retornado uma coleção com tamanho zero.

- `public ArrayList localizaTudo() throws RemoteException;`

Realiza consulta em todas as questões de avaliação. Retorna uma coleção de *array* de *strings* contendo todos os registros encontrados, caso nenhum registro seja encontrado será retornado uma coleção com tamanho zero.

Esse *session bean* é composto pelos arquivos `Questoes_Avaliacao.java`, `Questoes_AvaliacaoHome.java` e `Questoes_AvaliacaoBean.java`.

3.4.12 SESSION BEAN RESOLUCAO

Contém a lógica para manipular a tabela RESOLUCAO, é composto pelos seguintes métodos:

- `public boolean inclui(int Id, int Avaliacao, int Usuario, String Data_Inicio, String Data_Final, int Acertos, int Erros, int Branco, int Conceito) throws RemoteException;`

Realiza a inclusão de novas resoluções. O retorno dessa função indica se essa operação foi realizada com sucesso.

- `public boolean altera(int Id, int Avaliacao, int Usuario, String Data_Inicio, String Data_Final, int Acertos, int Erros, int Branco, int Conceito) throws RemoteException;`

Realiza a alteração dos dados de uma resolução. O retorno dessa função indica se essa operação foi realizada com sucesso.

➤ `public boolean exclui(int Id) throws RemoteException;`

Realiza a exclusão de uma resolução. O retorno dessa função indica se essa operação foi realizada com sucesso.

➤ `public String[] localizaId(int Id) throws RemoteException;`

Realiza consulta em resoluções através da chave primária. O retorno dessa função é um *array* de *strings* contendo o registro encontrado ou *null* caso contrário.

➤ `public ArrayList localizaAvaliacao(int Avaliacao) throws RemoteException;`

Realiza consulta em resoluções através da avaliação. O retorno dessa função é uma coleção de *array* de *strings* contendo os registros encontrados, caso nenhum registro seja encontrado será retornado uma coleção com tamanho zero.

➤ `public ArrayList localizaUsuario(int Usuario) throws RemoteException;`

Realiza consulta em resoluções através do usuário. O retorno dessa função é uma coleção de *array* de *strings* contendo os registros encontrados, caso nenhum registro seja encontrado será retornado uma coleção com tamanho zero.

➤ `public ArrayList localizaTudo() throws RemoteException;`

Realiza consulta em todas as resoluções. Retorna uma coleção de *array* de *strings* contendo todos os registros encontrados, caso nenhum registro seja encontrado será retornado uma coleção com tamanho zero.

Esse *session bean* é composto pelos arquivos `Resolucao.java`, `ResolucaoHome.java` e `ResolucaoBean.java`.

3.4.13 *SESSION BEAN* RESPOSTAS_AVALIACAO

Contém a lógica para manipular a tabela RESPOSTAS_AVALIACAO, é composto pelos seguintes métodos:

- `public boolean inclui(int Resolucao, int Resposta, String Data) throws RemoteException;`

Realiza a inclusão de novas respostas de avaliação. O retorno dessa função indica se essa operação foi realizada com sucesso.

- `public boolean altera(int Resolucao, int Resposta, String Data) throws RemoteException;`

Realiza a alteração dos dados de uma resposta de avaliação. O retorno dessa função indica se essa operação foi realizada com sucesso.

- `public boolean exclui(int Resolucao, int Resposta) throws RemoteException;`

Realiza a exclusão de uma resposta de avaliação. O retorno dessa função indica se essa operação foi realizada com sucesso.

- `public String[] localizaId(int Resolucao, int Resposta) throws RemoteException;`

Realiza consulta em respostas de avaliação através da chave primária. O retorno dessa função é um *array* de *strings* contendo o registro encontrado ou *null* caso contrário.

- `public ArrayList localizaResolucao(int Resolucao) throws RemoteException;`

Realiza consulta em respostas de avaliação através da resolução. O retorno dessa função é uma coleção de *array* de *strings* contendo os registros encontrados, caso nenhum registro seja encontrado será retornado uma coleção com tamanho zero.

➤ `public ArrayList localizaTudo() throws RemoteException;`

Realiza consulta em todas as respostas de avaliação. Retorna uma coleção de *array* de *strings* contendo todos os registros encontrados, caso nenhum registro seja encontrado será retornado uma coleção com tamanho zero.

Esse *session bean* é composto pelos arquivos `Respostas_Avaliacao.java`, `Respostas_AvaliacaoHome.java` e `Respostas_AvaliacaoBean.java`.

3.5 COMPILANDO CADA *SESSION BEAN*

Para o desenvolvimento e compilação das classes, foi utilizado a versão 4.3.1 beta do NetBeans, e para que a compilação das classes ocorra com sucesso, é necessário montar um filesystem apontando para o arquivo `C:\JBoss-2.4.10\client\jboss-j2ee.jar`. As classes fazem parte de um pacote denominado `com.projeto.ejb`, sendo assim, é necessário a criação de uma estrutura de pastas como mostrado na figura abaixo:



Figura 7 – Estrutura de pastas para criação de uma aplicação EJB

Todos os arquivos de cada *session bean* deverão ficar dentro da pasta *ejb*. A pasta denominada *META-INF* é a pasta onde deve ser armazenado o descritor de distribuição, que será descrito na seção 3.6.

Com base na estrutura de pastas representada na Figura 7, dentro do NetBeans deve-se montar outro filesystem apontando para a pasta *Avaliacao*. A visão geral de todos os arquivos envolvidos no projeto pode ser verificada na Figura 8:

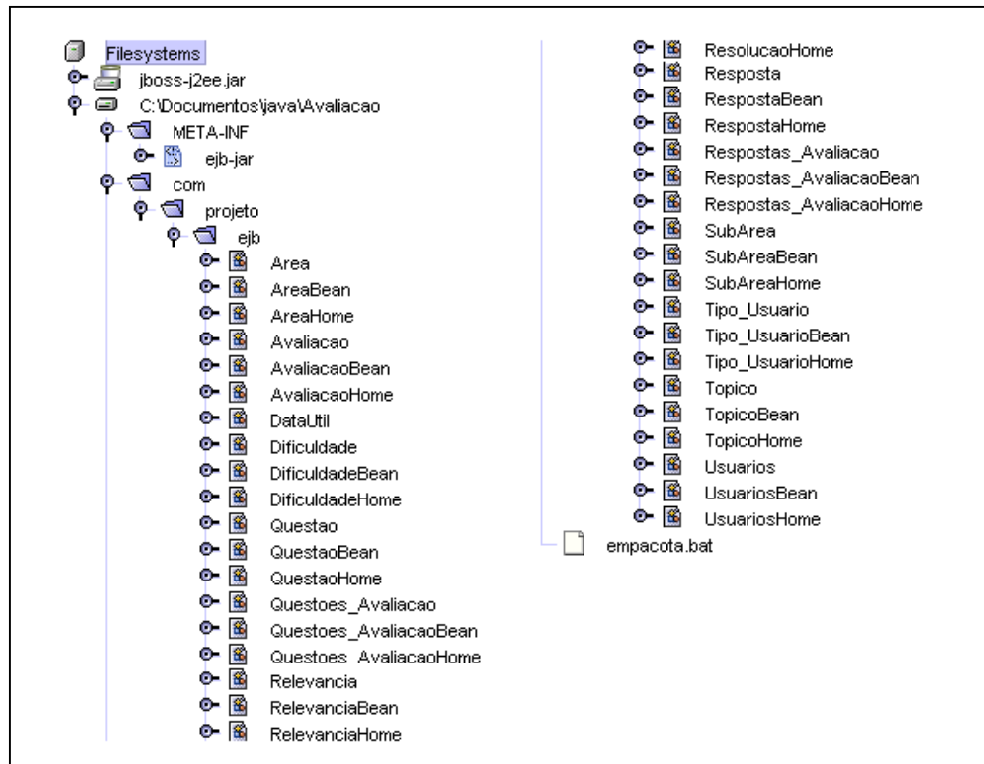


Figura 8 – Visão Geral do Filesystem no NetBeans

3.6 PUBLICANDO O PROTÓTIPO NO JBOSS

Uma aplicação EJB precisa ter um descritor de distribuição que é um arquivo XML e descreve cada EJB, qual é a sua interface *home*, interface *remote* e a classe que implementa toda lógica de negócio definida na interface *remote*. Esse arquivo deve se chamar *ejb-jar.xml* e ser armazenado dentro da pasta *META-INF*. O descritor do nosso protótipo está em anexo na seção 7.2.

Estando todos as classes dos *session beans* compilados e o descritor de distribuição criado, o próximo passo é a geração do arquivo de distribuição, que nada mais é que o empacotamento de todos os arquivos de classe em um arquivo *.jar*. Para facilitar a geração desse arquivo e a sua publicação no JBoss, foi criado um arquivo chamado *empacota.bat* que está no mesmo nível da pasta *com* contendo os comandos necessários para a geração. O conteúdo desse arquivo é o seguinte:

```
jar cvf projeto.jar com/projeto/ejb/*.* META-INF/ejb-jar.xml
copy projeto.jar C:\JBoss-2.4.10\deploy
```

`jar cfv projeto.jar com/projeto/ejb/*.* META-INF/ejb-jar.xml`: Esse comando é o principal, que gera o arquivo `projeto.jar`. Os dois parâmetros seguintes são os arquivos que farão parte desse pacote, todos os arquivos da pasta `com/projeto/ejb` e o arquivo de distribuição que está na pasta `META-INF`.

`copy projeto.jar C:\JBoss-2.4.10\deploy`: Após a geração do arquivo `projeto.jar`, esse comando se encarrega de fazer a publicação do mesmo no JBoss. Como o JBoss tem a característica de “hot deploy”, que é a publicação de arquivos sem a necessidade de ser reiniciado, basta copiar o arquivo para dentro da pasta `c:\jboss-2.4.10\deploy`, que o servidor detecta um novo pacote e se encarrega de carregá-lo.

3.7 O APLICATIVO CLIENTE EM JSP

Uma aplicação EJB é um componente do lado servidor que fica esperando por uma requisição de cliente. Assim, para testar se uma aplicação está funcionando há a necessidade de criar uma aplicação cliente.

Uma aplicação EJB nunca chama os métodos diretamente da classe que os implementa, ela só pode ver as interfaces `home` e `remote` como demonstrado na Figura 3.

O acesso ao *session bean* é feito através de JNDI que vem como parte de JDK 1.3 e posterior, e oferece dois serviços: um serviço de nomeação e um serviço de diretório.

Um serviço de nomeação é um recurso muito importante, pois ele é responsável por encontrar um objeto associado a um determinado nome. No contexto de EJB, um serviço de nomeação encontra um EJB, se você souber o nome dado a ele.

Se um aplicativo cliente quiser chamar um EJB, ele só precisa fornecer o seu nome que JNDI o encontrará.

Os serviços de diretório não são utilizados para acessar EJB.

A JNDI é dividida em cinco pacotes que são: `javax.naming`, `javax.naming.directory`, `javax.naming.event`, `javax.naming.ldap`, `javax.naming.spi`.

Para acessar um EJB é necessário conhecer apenas o pacote `javax.naming` e dois membros: a interface `Context` e a classe `InitialContext`.

A interface `Context` representa um contexto de nomeação, em um conjunto de associações nome-a-objeto. Essa associação em termos de JNDI é chamada de *binding*. A interface `Context` é importante, pois ela contém métodos para examinar e

modificar as ligações nome-a-objeto. O método usado com mais freqüência na interface `Context`, provavelmente é o método `lookup`. Esse método retorna uma referência a um objeto, dado o seu nome. Há duas sobrecargas desse método, e a assinatura de ambas são:

```
public Object lookup(javax.naming.Name name) throws
javax.naming.NamingException;
public Object lookup(String name) throws javax.naming.NamingException;
```

O método `lookup` é usado para obter uma referência a um objeto *home* do EJB. A segunda sobrecarga do método `lookup` é bem direta, é necessário simplesmente passar a representação de *String* ao nome do *bean* para obter uma referência ao objeto *home* do *bean*, ela devolve um objeto `javax.naming.NamingException` se a resolução de nome falhar.

As operações de nomeação são relativas a um contexto. A classe `InitialContext` implementa a interface `Context` e oferece o contexto de início para resoluções de nomes.

Para um contexto, você precisa definir uma série de propriedades para o ambiente do contexto. Normalmente, você cria um objeto `java.util.Properties` e usa o seu método `put` para acrescentar pares de chave/valor, representando quaisquer propriedades de nomes e valores necessárias.

Ao acessar um EJB você precisa fornecer duas propriedades. A primeira é a propriedade de ambiente `java.naming.factory.Initial`. O valor dessa propriedade é o nome de classe totalmente qualificada que será usado para criar o contexto inicial. A segunda propriedade é `java.naming.provider.url`, que é a propriedade de ambiente, cujo valor especifica informações de configuração para uso de serviço de provedor. Ambas as propriedades de nomes estão convenientemente contidas em dois campos estáticos na interface `Context`. São eles `Context.INITIAL_CONTEXT_FACTORY` e `Context.PROVIDER_URL`, respectivamente.

Então, para se obter uma referência a um *session bean* de dentro da aplicação cliente, é necessário passar inicialmente pelas seguintes etapas:

1. Criar um objeto `java.util.Properties`;
2. Acrescentar as propriedades necessárias ao objeto `java.util.Properties` para construir o contexto inicial;
3. Criar um objeto `javax.naming.InitialContext`;

4. Usar o método `lookup` da interface `javax.naming.Context` para obter uma referência ao objeto *home* do *bean*, passando o seu nome;
5. Criar uma cópia do *bean* no servidor;
6. E finalmente, chamar os métodos expostos do *bean*.

O código para esses passos seria o seguinte:

```
<%@ page import="javax.naming.*"%>
<%@ page import="javax.rmi.PortableRemoteObject"%>
<%@ page import="java.util.Properties"%>
<%@ page import="com.projeto.ejb.Area"%>
<%@ page import="com.projeto.ejb.AreaHome"%>

Properties properties = new Properties();
properties.put(Context.INITIAL_CONTEXT_FACTORY,"
    org.jnp.interfaces.NamingContextFactory");
properties.put(Context.PROVIDER_URL, "localhost:1099");
try
{
    InitialContext jndiContext = new InitialContext(properties);
    Object ref = jndiContext.lookup("SessionArea");
    AreaHome home = (AreaHome) PortableRemoteObject.narrow (ref, AreaHome.class);
    Area area = home.create();
    If (area.inclui(Integer.parseInt(id), vnome);){
        .
        .
        .
    }
}
catch (NamingException e){
}
```

Até aqui, foi conseguida uma referência ao objeto *home* do *session bean*. A próxima etapa é criar uma cópia do *bean* no servidor. Isso é feito usando o método *create* da interface *home*.

A partir desse ponto, é só chamar quaisquer métodos expostos do *bean*, que nesse exemplo, foi chamado o método *inclui* do *session bean* *Area*.

Com base nesse exemplo, foram criadas páginas JSP para o protótipo, acessando os *session beans* que fazem a manipulação do banco de dados, chamando métodos para incluir, excluir, alterar e consultar as tabelas do banco de dados.

Para que a nossa aplicação JSP cliente funcione, ela deve ser publicada no Tomcat. Dentro da pasta de instalação do Tomcat, existe uma subpasta chamada *webapps*, que ficam os arquivos que podem ser alcançados através do browser, sendo assim, para que o nosso site seja publicado, foi criada uma pasta dentro de *webapps* chamada *Projeto*, onde dentro dela foi colocada todos os arquivo JSP do projeto.

Dentro dessa pasta *Projeto*, deve ser respeitada uma estrutura como mostrado na figura abaixo:

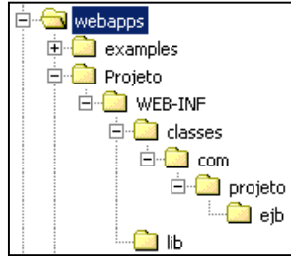


Figura 9 – Estrutura de pastas para publicação da aplicação cliente JSP no Tomcat

Dentro da pasta *lib* deverão ficar os arquivos .jar requeridos para acessar os *beans*, sendo eles o *jboss.jar*, *jboss-j2ee.jar* e *jnpserver.jar*, conseguidos dentro da pasta de instalação do JBoss.

Dentro da pasta *classes* deverão ficar todas as interfaces *home* e *remote* que utilizaremos. Como elas formam o pacote *com.projeto.ejb*, essa estrutura de pastas deve ser criada, e dentro do último nível, *ejb*, devem ser colocadas todas essas interfaces.

Resumindo:

- 1) Dentro da pasta *Projeto* ficam todos os arquivos .JSP;
- 2) Dentro da pasta *lib* ficam os arquivos *jboss.jar*, *jboss-j2ee.jar* e *jnpserver.jar*;
- 3) Dentro da pasta *classes/com/projeto/ejb* ficam as interfaces utilizadas nas páginas .JSP, no caso, as interfaces *home* e *remote* dos *session beans*, participantes do pacote *com.projeto.ejb*.

3.8 RESUMO

Neste capítulo, foram apresentados todos os passos executados na implementação do protótipo, desde a instalação de todo o ambiente de desenvolvimento e execução, até a compilação e publicação dos pacotes.

No próximo capítulo será apresentado o resultado obtido com o desenvolvimento da aplicação, bem como algumas de suas telas com os *session*

beans que foram utilizados, indicando quais métodos foram utilizados em cada situação.

4 RESULTADOS OBTIDOS

Para testar a aplicação EJB publicada no JBoss, foi desenvolvido uma aplicação cliente que realiza o acesso aos *session beans*. Essa aplicação cliente é composta de um site JSP, através do qual conseguimos nos conectar com o servidor de aplicação através da rede e acessar todos os métodos publicados na aplicação EJB.

Essa possibilidade de conexão com o servidor de aplicação através da rede e acesso à aplicação EJB através de chamada a métodos específicos, nos forneceu um alto nível de segurança, pois a aplicação cliente pode estar separada do servidor de aplicação por um firewall somente com a porta de comunicação específica liberada (1099). Assim, a aplicação cliente não faz nenhum tipo de acesso ao banco de dados, somente conhece a aplicação EJB e quais são os métodos expostos. Somente através desses métodos que é efetuada a comunicação entre cliente e servidor, ficando o banco de dados em outra camada que somente é acessada através do servidor de aplicação dentro da rede interna ao firewall.

Através da implementação da aplicação cliente denominada de “Sistema de Apoio a Aplicação de Testes através da Web”, conseguimos comprovar a eficiência e rapidez no desenvolvimento de aplicações multi-camadas utilizando-se de EJB.

A seguir, demonstraremos algumas telas do site.

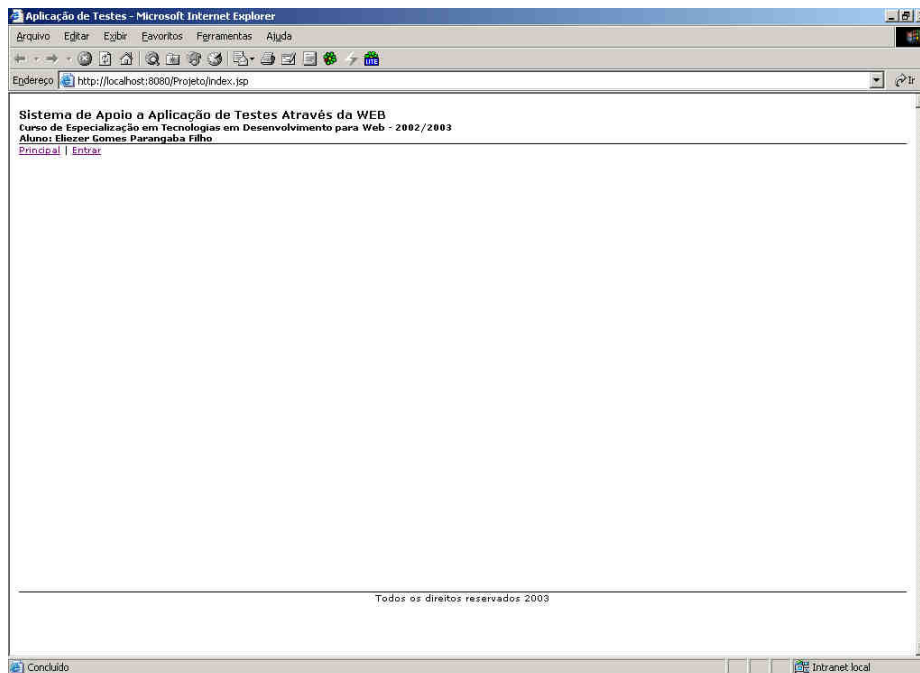


Figura 10 – Tela principal do site

A tela da figura 10 fica disponível, quando o usuário ainda não está logado, apenas um link para a tela de login. Após efetuado o login, o usuário verá, nessa tela, outros links para telas de cadastros específicos.

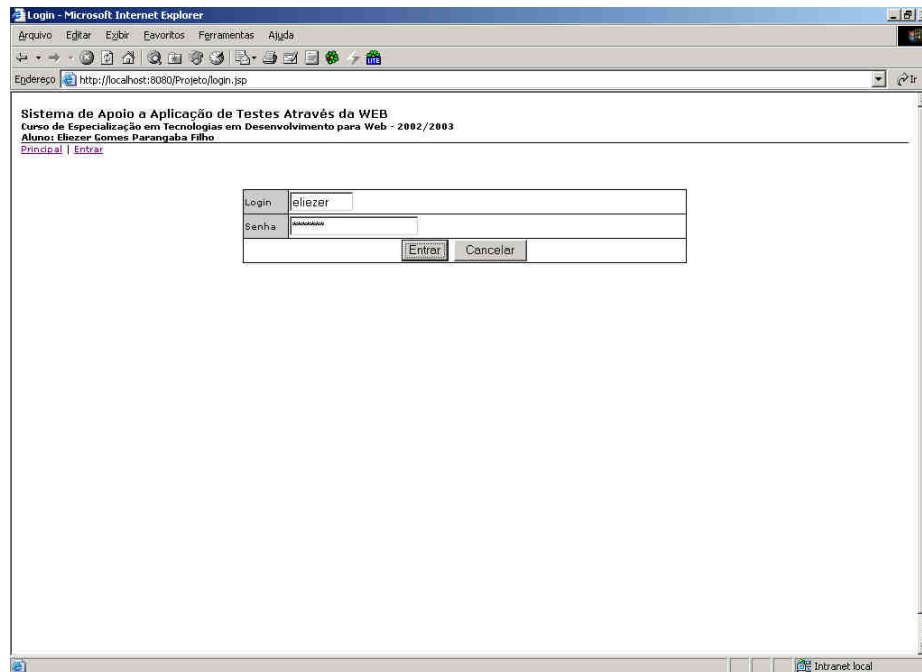


Figura 11 – Tela de login

A tela da figura 11 é responsável pelo login do usuário no site e está relacionado com o *session bean* *Usuarios*. O método utilizado foi:

UsuarioValido: Quando o usuário fornece o seu login e a sua senha e clica no botão *Entrar*, esse método é chamado para verificar se existe na tabela de usuários um registro com o login e senha informados.

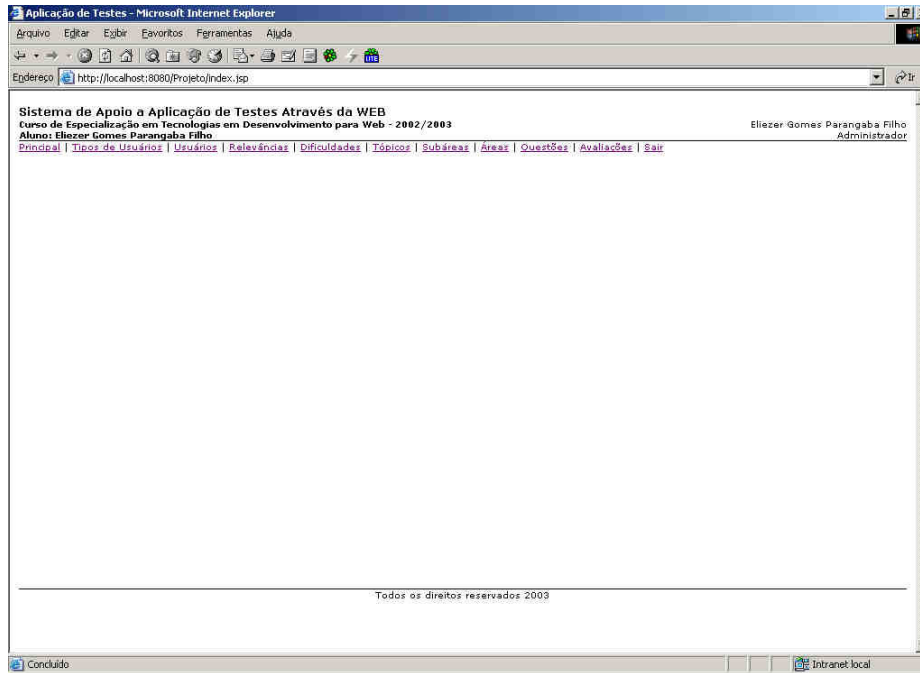


Figura 12 – Tela principal após login

Após o login do usuário no sistema, na tela principal do site aparece outros links, conforme a tela da figura 12, para cadastros como: Tipos de usuários, Usuários, Relevâncias, Dificuldades, Tópicos, Subáreas, Áreas, Questões e Avaliações.

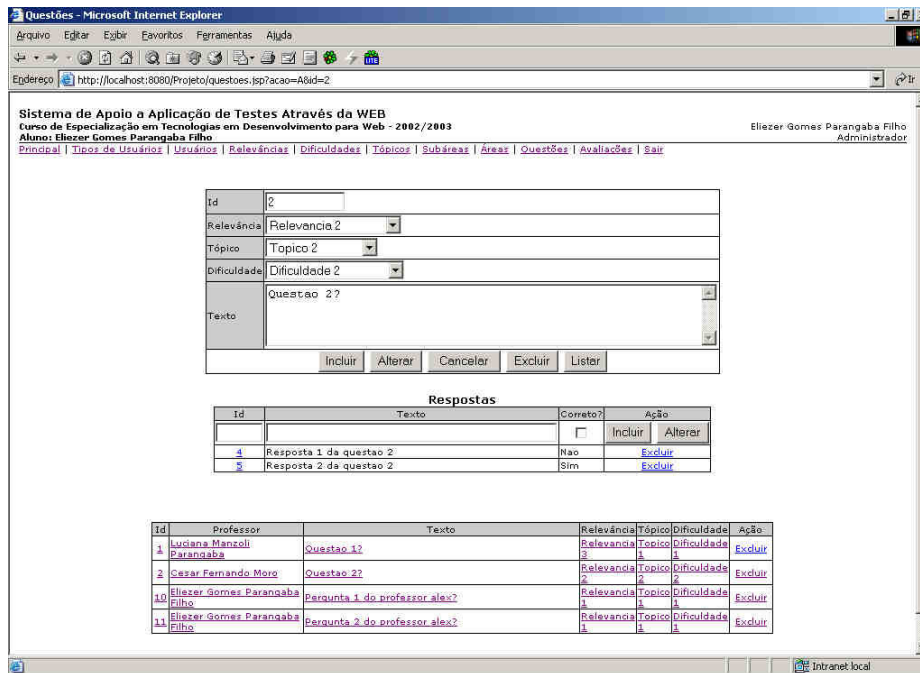


Figura 13 – Tela de cadastro de questões

A tela da figura 13 é utilizada para o cadastramento das questões, e está relacionada com os *session beans* Questao, Relevancia, Topico, Dificuldade, Resposta e Usuarios. Os métodos utilizados de cada *bean* são:

Questao:

incluir: Quando o usuário preenche as informações da questão e clica no botão Incluir, esse método é chamado para que uma nova questão seja gravada.

altera: Quando o usuário altera as informações da questão e clica no botão Alterar, esse método é chamado para que os dados da questão sejam alterados.

exclui: Quando o usuário clica no botão Exclui, esse método é chamado para que essa questão seja excluída.

localizaId: Esse método é utilizado para localizar uma questão através de seu código, e caso seja encontrada, retorna seus dados para exibição.

localizaTudo: Quando o administrador do site clica no botão Listar, esse método é chamado para que todas as questões atualmente cadastradas sejam exibidas em uma lista, para que o mesmo possa vê-las e decidir se gostaria de alterá-las ou excluí-las através de links sobre essas questões.

localizaProf: Quando um professor clica no botão Listar, esse método é chamado para que todas as suas questões atualmente cadastradas sejam exibidas em uma lista, para que o mesmo possa vê-las e decidir se gostaria de alterá-las ou excluí-las através de links sobre essas questões.

Relevancia:

localizaTudo: É utilizado para preencher o campo do formulário select com a lista de todas as relevâncias cadastradas.

localizaId: É utilizado na lista de questões, para a exibição do nome da relevância e não seu código.

Topico:

localizaTudo: É utilizado para preencher o campo do formulário select com a lista de todos os tópicos cadastrados.

localizaId: É utilizado na lista de questões, para a exibição do nome do tópico e não seu código.

Dificuldade:

localizaTudo: É utilizado para preencher o campo do formulário select com a lista de todas as dificuldades cadastradas.

localizaId: É utilizado na lista de questões, para a exibição do nome da dificuldade e não seu código.

Resposta:

inclui: Quando o usuário preenche as informações de uma resposta e clica no botão Incluir, esse método é chamado para que uma nova resposta para uma questão específica seja gravada.

altera: Quando o usuário altera as informações de uma resposta e clica no botão Alterar, esse método é chamado para que os dados da resposta sejam alterados.

exclui: Quando o usuário clica no link Excluir da lista de respostas, esse método é chamado para que a respectiva resposta seja excluída.

localizaId: Quando o usuário clica em uma resposta da lista, esse método é chamado para que todos dados referentes aquela resposta seja recuperado e exibido.

localizaQuestao: Esse método é chamado para que todas as respostas referências a uma questão sejam exibidas em uma lista.

Usuarios:

localizaId: É utilizado na lista de questões, para a exibição do nome do professor que a criou.

4.1 RESUMO

Neste capítulo foram apresentados os resultados obtidos com o desenvolvimento da aplicação, bem como algumas de suas telas com os *session beans* que foram utilizados, indicando quais métodos foram utilizados em cada situação.

Finalmente o próximo capítulo apresenta as conclusões do presente trabalho e as propostas de trabalhos futuros.

5 CONCLUSÃO E TRABALHOS FUTUROS

Através de todo o estudo realizado sobre EJB e do projeto desenvolvido, foi possível verificar a importância cada vez maior dessa tecnologia de componentes no servidor, de centralização e reutilização das regras de negócios. Esse é o desejo de desenvolvedores de aplicações multi-camadas, ter um ambiente prático e fácil para desenvolvimento dessas aplicações, podendo distribuí-las através da rede.

O aprendizado dessa tecnologia é custoso, demanda uma quantidade razoável de tempo para entender todo o processo e tecnologias envolvidas, a quantidade de APIs é bastante grande, além de ter que conhecer o funcionamento do servidor de aplicação que será utilizado. Significa uma mudança na maneira de pensar, de idealizar um processo de desenvolvimento de aplicações para internet, mas, uma vez entendido todo o esquema, o desenvolvimento de aplicações que usam componentes no servidor é fácil e rápido.

Durante os testes finais houve a comprovação da funcionalidade desse projeto e da tecnologia estudada. Cada camada da aplicação ficou em execução em um micro distinto. O cliente através de um browser que estava sendo executado em um micro (camada 1), estava acessando a aplicação cliente do contêiner que estava sendo executado em um servidor (camada 2), este contêiner, por sua vez, servia páginas JSP que são clientes dos EJB. O servidor de aplicação JBoss estava sendo executado em um outro servidor (camada 3), onde estavam publicados todos os EJB da aplicação, e por último, esses EJBs estavam acessando dados contidos em um banco de dados MySQL, que estava sendo executado em um terceiro servidor (camada 4).

Toda a lógica de negócio não fica na aplicação cliente, e sim no servidor de aplicação, encapsulada dentro dos EJBs, o que adiciona a aplicação segurança, podendo existir, inclusive, um firewall entre o cliente e o servidor de aplicação. As aplicações clientes contém somente as interfaces com os métodos publicados pelos EJBs.

Durante a implementação foi utilizado apenas *session beans*, que encapsulam toda a lógica de negócio e também manipulam o banco de dados através de instruções SQLs. Para um projeto futuro, é possível sugerir o desenvolvimento de *entity beans*, que são um tipo de EJB que mapeiam registros de tabelas no banco de dados, substituindo as instruções SQLs de dentro dos *session beans* por chamadas a esses *entity beans*, o que poderia tornar mais prático as tarefas relacionadas a persistência das informações da aplicação.

6 REFERÊNCIAS BIBLIOGRÁFICAS

Documentação do JBoss 2.4+, Disponível em:

<<http://jboss.sourceforge.net/doc-24/>>, Acesso em: 20 nov. 2002.

Especificação JavaServer Pages 2.0, Disponível em:

<<http://java.sun.com/products/jsp/download.html#specs>>, Acesso em: 05 jan. 2003.

Guia de Referência Rápida JSP, Disponível em:

<<http://java.sun.com/products/jsp/syntax/1.2/card12.pdf>>, Acesso em: 06 jan. 2003.

Java Server Pages Technology, Disponível em:

<<http://java.sun.com/j2ee/1.4/docs/tutorial/doc/JSPIntro.html>>, Acesso em: 17 jan. 2003.

Kurniawan, Budi, **Java para Web com Servlets, JSP e EJB**, Editora Ciência Moderna, 2002.

Manhães, Marcelo Mota, **Enterprise Java Beans, Fundamentos**, 2002, Disponível em: <http://www.mundooo.com.br/php/printmooart.php?pa=showpage&pid=6>>, Acesso em: 13 fev. 2003.

Roman, Ed, Ambler, Scott W., Jewell, Tyler, **Mastering Enterprise JavaBeans Second Edition**, John Wiley & Sons, 2001.

Roman, Ed., **Mastering Enterprise JavaBeans and Java 2 Platform, Enterprise Edition**, John Wiley & Sons, 1999.

Romão, Mário J. B., Caldeira, Mário M., **Componentes de Software: a caminho de uma indústria?**, 2000, Disponível em: <<http://www.indeg.org/rpbgrpg5/estudos.html>>, Acesso em: 21 mai. 2003.

Sun Microsystems, **JavaBeans Concepts**, 2003, Disponível em:

<<http://java.sun.com/docs/books/tutorial/javabeans/whatis/beanDefinition.html>>,

Acesso em: 05 nov. 2003.

Sun Microsystems, **J2EE 1.4 Documentation**, 2003, Disponível em:

<<http://java.sun.com/j2ee/1.4/docs/>>, Acesso em 05 nov. 2003.

Sun Microsystems, **Trail: Learning the Java Language**, 2003, Disponível em:

<<http://java.sun.com/docs/books/tutorial/java/index.html>>, Acesso em 05 nov. 2003.

Almeida, Rodrigo Rebouças de, **Model-View-Controller (MVC)**, 2003, Disponível em:

<<http://www.dsc.ufpb.br/~jacques/cursos/map/html/arqu/mvc/mvc.htm>>, Acesso em:
29 mar. 2003.

7 ANEXOS

7.1 Classe para Manipulação e Conversão de Data

DataUtil.java

```
package com.projeto.ejb;

import java.util.Date;
import java.util.GregorianCalendar;
import java.text.SimpleDateFormat;

public class DataUtil {
    String vdatastr="";
    Date vdata = new Date(0);
    int vdia, vmes, vano=0;
    boolean vdatavalida=false;

    public DataUtil() {

    }

    public DataUtil(String data) {
        setData(data, false);
    }

    public DataUtil(String data, boolean invertida) {
        setData(data, invertida);
    }

    public void setData(String data, boolean invertida) {
        try{
            this.vdatastr=data;
            vdatavalida=false;
            if (vdatastr.length()==10){
                if (!invertida){
                    GregorianCalendar gc = new
GregorianCalendar(Integer.parseInt(vdatastr.substring(6, 10)),
Integer.parseInt(vdatastr.substring(3, 5))-1, Integer.parseInt(vdatastr.substring(0, 2)));
                    vdia=gc.get(GregorianCalendar.DATE);
                    vmes=gc.get(GregorianCalendar.MONTH)+1;
                    vano=gc.get(GregorianCalendar.YEAR);
                    vdata = gc.getTime();
                }
                else{
                    GregorianCalendar gc = new
GregorianCalendar(Integer.parseInt(vdatastr.substring(0, 4)),
Integer.parseInt(vdatastr.substring(5, 7))-1, Integer.parseInt(vdatastr.substring(8, 10)));
                    vdia=gc.get(GregorianCalendar.DATE);
                    vmes=gc.get(GregorianCalendar.MONTH)+1;
                    vano=gc.get(GregorianCalendar.YEAR);
                    vdata = gc.getTime();
                }
            }
        }
    }
}
```

```

        }
        vdatavalida=true;
    }
}
catch(Exception e){

}
}

public Date getDate(){
    return vdata;
}

public int getDia(){
    return vdia;
}

public int getMes(){
    return vmes;
}

public int getAno(){
    return vano;
}

public String getData(String separador, boolean invertida){
    SimpleDateFormat df;
    if (invertida){
        df = new SimpleDateFormat("yyyy"+separador+"MM"+separador+"dd");
    }
    else{
        df = new SimpleDateFormat("dd"+separador+"MM"+separador+"yyyy");
    }
    return df.format(vdata);
}

public String getData(boolean invertida){
    SimpleDateFormat df;
    if (invertida){
        df = new SimpleDateFormat("yyyy-MM-dd");
    }
    else{
        df = new SimpleDateFormat("dd/MM/yyyy");
    }
    return df.format(vdata);
}

public boolean dataValida(){
    return vdatavalida;
}
}

```

7.2 Descritores de Distribuição

ejb-jar.java

```
<?xml version="1.0" encoding="UTF-8"?>

<ejb-jar>
  <description>Projeto</description>
  <display-name>Projeto</display-name>
  <enterprise-beans>
    <session>
      <ejb-name>SessionArea</ejb-name>
      <home>com.projeto.ejb.AreaHome</home>
      <remote>com.projeto.ejb.Area</remote>
      <ejb-class>com.projeto.ejb.AreaBean</ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Bean</transaction-type>
    </session>
    <session>
      <ejb-name>SessionSubArea</ejb-name>
      <home>com.projeto.ejb.SubAreaHome</home>
      <remote>com.projeto.ejb.SubArea</remote>
      <ejb-class>com.projeto.ejb.SubAreaBean</ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Bean</transaction-type>
    </session>
    <session>
      <ejb-name>SessionTipo_Usuario</ejb-name>
      <home>com.projeto.ejb.Tipo_UsuarioHome</home>
      <remote>com.projeto.ejb.Tipo_Usuario</remote>
      <ejb-class>com.projeto.ejb.Tipo_UsuarioBean</ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Bean</transaction-type>
    </session>
    <session>
      <ejb-name>SessionTopico</ejb-name>
      <home>com.projeto.ejb.TopicoHome</home>
      <remote>com.projeto.ejb.Topico</remote>
      <ejb-class>com.projeto.ejb.TopicoBean</ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Bean</transaction-type>
    </session>
    <session>
      <ejb-name>SessionUsuarios</ejb-name>
      <home>com.projeto.ejb.UsuariosHome</home>
      <remote>com.projeto.ejb.Usuarios</remote>
      <ejb-class>com.projeto.ejb.UsuariosBean</ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Bean</transaction-type>
    </session>
  </enterprise-beans>
</ejb-jar>
```

```
</session>
<session>
  <ejb-name>SessionRelevancia</ejb-name>
  <home>com.projeto.ejb.RelevanciaHome</home>
  <remote>com.projeto.ejb.Relevancia</remote>
  <ejb-class>com.projeto.ejb.RelevanciaBean</ejb-class>
  <session-type>Stateless</session-type>
  <transaction-type>Bean</transaction-type>
</session>
<session>
  <ejb-name>SessionDificuldade</ejb-name>
  <home>com.projeto.ejb.DificuldadeHome</home>
  <remote>com.projeto.ejb.Dificuldade</remote>
  <ejb-class>com.projeto.ejb.DificuldadeBean</ejb-class>
  <session-type>Stateless</session-type>
  <transaction-type>Bean</transaction-type>
</session>
<session>
  <ejb-name>SessionQuestao</ejb-name>
  <home>com.projeto.ejb.QuestaoHome</home>
  <remote>com.projeto.ejb.Questao</remote>
  <ejb-class>com.projeto.ejb.QuestaoBean</ejb-class>
  <session-type>Stateless</session-type>
  <transaction-type>Bean</transaction-type>
</session>
<session>
  <ejb-name>SessionResposta</ejb-name>
  <home>com.projeto.ejb.RespostaHome</home>
  <remote>com.projeto.ejb.Resposta</remote>
  <ejb-class>com.projeto.ejb.RespostaBean</ejb-class>
  <session-type>Stateless</session-type>
  <transaction-type>Bean</transaction-type>
</session>
<session>
  <ejb-name>SessionAvaliacao</ejb-name>
  <home>com.projeto.ejb.AvaliacaoHome</home>
  <remote>com.projeto.ejb.Avaliacao</remote>
  <ejb-class>com.projeto.ejb.AvaliacaoBean</ejb-class>
  <session-type>Stateless</session-type>
  <transaction-type>Bean</transaction-type>
</session>
<session>
  <ejb-name>SessionQuestoes_Avaliacao</ejb-name>
  <home>com.projeto.ejb.Questoes_AvaliacaoHome</home>
  <remote>com.projeto.ejb.Questoes_Avaliacao</remote>
  <ejb-class>com.projeto.ejb.Questoes_AvaliacaoBean</ejb-class>
  <session-type>Stateless</session-type>
  <transaction-type>Bean</transaction-type>
</session>
<session>
  <ejb-name>SessionResolucao</ejb-name>
  <home>com.projeto.ejb.ResolucaoHome</home>
  <remote>com.projeto.ejb.Resolucao</remote>
```

```
<ejb-class>com.projeto.ejb.ResolucaoBean</ejb-class>
<session-type>Stateless</session-type>
<transaction-type>Bean</transaction-type>
</session>
<session>
  <ejb-name>SessionRespostas_Avaliacao</ejb-name>
  <home>com.projeto.ejb.Respostas_AvaliacaoHome</home>
  <remote>com.projeto.ejb.Respostas_Avaliacao</remote>
  <ejb-class>com.projeto.ejb.Respostas_AvaliacaoBean</ejb-class>
  <session-type>Stateless</session-type>
  <transaction-type>Bean</transaction-type>
</session>
</enterprise-beans>
</ejb-jar>
```

7.3 Exemplo de dois dos treze *session beans* do projeto

Interface Home: Questoes AvaliacaoHome.java

```
package com.projeto.ejb;

import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.EJBHome;

public interface Questoes_AvaliacaoHome extends EJBHome {
    Questoes_Avaliacao create() throws RemoteException, CreateException;
}
```

Session bean – Implementação dos Métodos: Questoes AvaliacaoBean.java

```
package com.projeto.ejb;

import java.rmi.RemoteException;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;
import java.sql.*;
import java.util.ArrayList;

public class Questoes_AvaliacaoBean implements SessionBean {

    private Connection getConnection() {
        Connection connection = null;
        try {
            Class.forName("org.gjt.mm.mysql.Driver");
            connection = DriverManager.getConnection("jdbc:mysql://localhost/avaliacao", "avaliacao",
"avaliacao");
        }
        catch (Exception e) {
        }
        return connection;
    }

    public boolean inclui(int Avaliacao, int Topico, int Relevancia, int Dificuldade, int
Numero_Questoes, int Peso){
        Connection con = null;
        PreparedStatement ps = null;
        boolean incluiu = false;
        try {
            String sql = "Insert Into Questoes_Avaliacao (Avaliacao, Topico, Relevancia, Dificuldade,
Numero_Questoes, Peso) Values(?, ?, ?, ?, ?, ?)";
            con=getConnection();
            ps = con.prepareStatement(sql);
            ps.setInt(1, Avaliacao);
            ps.setInt(2, Topico);
            ps.setInt(3, Relevancia);
        }
    }
}
```

```

        ps.setInt(4, Dificuldade);
        ps.setInt(5, Numero_Questoes);
        ps.setInt(6, Peso);
        ps.executeUpdate();
        incluiu = true;
    }
    catch (Exception e) {
        System.out.println(e.toString());
    }
    finally{
        try {
            if (ps!=null)
                ps.close();
            if (con!=null)
                con.close();
        }
        catch (SQLException e) {
        }
    }
    return incluiu;
}

```

```

public boolean altera(int Avaliacao, int Topico, int Relevancia, int Dificuldade, int
Numero_Questoes, int Peso){
    Connection con = null;
    PreparedStatement ps = null;
    boolean alterou = false;

    try {
        String sql = "Update Questoes_Avaliacao Set Numero_Questoes = ?, Peso = ? Where
Avaliacao = ? and Topico = ? and Relevancia = ? and Dificuldade = ?";
        con=getConnection();
        ps = con.prepareStatement(sql);
        ps.setInt(1, Numero_Questoes);
        ps.setInt(2, Peso);
        ps.setInt(3, Avaliacao);
        ps.setInt(4, Topico);
        ps.setInt(5, Relevancia);
        ps.setInt(6, Dificuldade);
        ps.executeUpdate();
        alterou = true;
    }
    catch (Exception e) {
        System.out.println(e.toString());
    }
    finally{
        try {
            if (ps!=null)
                ps.close();
            if (con!=null)
                con.close();
        }
        catch (SQLException e) {
        }
    }
    return alterou;
}

```

```

public boolean exclui(int Avaliacao, int Topico, int Relevancia, int Dificuldade){
    Connection con = null;
    PreparedStatement ps = null;
    boolean excluiu = false;

    try {
        String sql = "Delete From Questoes_Avaliacao Where Avaliacao = ? and Topico = ? and
Relevancia = ? and Dificuldade = ?";
        con=getConnection();
        ps = con.prepareStatement(sql);
        ps.setInt(1, Avaliacao);
        ps.setInt(2, Topico);
        ps.setInt(3, Relevancia);
        ps.setInt(4, Dificuldade);
        ps.executeUpdate();
        excluiu = true;
    }
    catch (Exception e) {
        System.out.println(e.toString());
    }
    finally{
        try {
            if (ps!=null)
                ps.close();
            if (con!=null)
                con.close();
        }
        catch (SQLException e) {
        }
    }
    return excluiu;
}

```

```

public String[] localizaId(int Avaliacao, int Topico, int Relevancia, int Dificuldade){
    Connection con = null;
    PreparedStatement ps = null;
    String[] row = null;

    try {
        String sql = "Select Avaliacao, Topico, Relevancia, Dificuldade, Numero_Questoes, Peso
"+
        "From Questoes_Avaliacao Where Avaliacao = ? and Topico = ? and Relevancia
= ? and Dificuldade = ?";
        con=getConnection();
        ps = con.prepareStatement(sql);
        ps.setInt(1, Avaliacao);
        ps.setInt(2, Topico);
        ps.setInt(3, Relevancia);
        ps.setInt(4, Dificuldade);
        ResultSet rs = ps.executeQuery();
        if (rs.next()) {
            row = new String [6];
            row[0] = rs.getString("Avaliacao");
            row[1] = rs.getString("Topico");
            row[2] = rs.getString("Relevancia");
            row[3] = rs.getString("Dificuldade");
            row[4] = rs.getString("Numero_Questoes");
            row[5] = rs.getString("Peso");
        }
    }
}

```

```

    }
}
catch (Exception e) {
    System.out.println(e.toString());
}
finally{
    try {
        if (ps!=null)
            ps.close();
        if (con!=null)
            con.close();
    }
    catch (SQLException e) {
    }
}
return row;
}

public ArrayList localizaAvaliacao(int Avaliacao){
    Connection con = null;
    PreparedStatement ps = null;
    ArrayList retval = new ArrayList(50);
    try {
        String sql = "Select Avaliacao, Topico, Relevancia, Dificuldade, Numero_Questoes, Peso
"+
            "From Questoes_Avaliacao Where Avaliacao = ?";
        con=getConnection();
        ps = con.prepareStatement(sql);
        ps.setInt(1, Avaliacao);
        ResultSet rs = ps.executeQuery();
        while (rs.next()) {
            String[] row = new String [6];
            row[0] = rs.getString("Avaliacao");
            row[1] = rs.getString("Topico");
            row[2] = rs.getString("Relevancia");
            row[3] = rs.getString("Dificuldade");
            row[4] = rs.getString("Numero_Questoes");
            row[5] = rs.getString("Peso");
            retval.add(row);
        }
    }
    catch (Exception e) {
        System.out.println(e.toString());
    }
    finally{
        try {
            if (ps!=null)
                ps.close();
            if (con!=null)
                con.close();
        }
        catch (SQLException e) {
        }
    }
    return retval;
}

public ArrayList localizaTudo(){

```

```

Connection con = null;
PreparedStatement ps = null;
ArrayList retval = new ArrayList(50);
try {
    String sql = "Select Avaliacao, Topico, Relevancia, Dificuldade, Numero_Questoes, Peso
"+
        "From Questoes_Avaliacao";
    con=getConnection();
    ps = con.prepareStatement(sql);
    ResultSet rs = ps.executeQuery();
    while (rs.next()) {
        String[] row = new String [6];
        row[0] = rs.getString("Avaliacao");
        row[1] = rs.getString("Topico");
        row[2] = rs.getString("Relevancia");
        row[3] = rs.getString("Dificuldade");
        row[4] = rs.getString("Numero_Questoes");
        row[5] = rs.getString("Peso");
        retval.add(row);
    }
}
catch (Exception e) {
    System.out.println(e.toString());
}
finally{
    try {
        if (ps!=null)
            ps.close();
        if (con!=null)
            con.close();
    }
    catch (SQLException e) {
    }
}
return retval;
}

public void ejbCreate() {
}

public void ejbRemove() {
}

public void ejbActivate() {
}

public void ejbPassivate() {
}

public void setSessionContext(SessionContext sc) {
}
}

```

Interface Home: ResolucaoHome.java

```
package com.projeto.ejb;

import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.EJBHome;

public interface ResolucaoHome extends EJBHome {
    Resolucao create() throws RemoteException, CreateException;
}
```

Session bean – Implementação dos Métodos: ResolucaoBean.java

```
package com.projeto.ejb;

import java.rmi.RemoteException;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;
import java.sql.*;
import java.util.ArrayList;

public class ResolucaoBean implements SessionBean {

    private Connection getConnection() {
        Connection connection = null;
        try {
            Class.forName("org.gjt.mm.mysql.Driver");
            connection = DriverManager.getConnection("jdbc:mysql://localhost/avaliacao", "avaliacao",
"avaliacao");
        }
        catch (Exception e) {
        }
        return connection;
    }

    public boolean inclui(int Id, int Avaliacao, int Usuario, String Data_Inicio, String Data_Final, int
Acertos, int Erros, int Branco, int Conceito){
        Connection con = null;
        PreparedStatement ps = null;
        boolean incluiu = false;
        try {
            String sql = "Insert Into Resolucao (Id, Avaliacao, Usuario, Data_Inicio, Data_Final,
Acertos, Erros, Branco, Conceito) Values (?, ?, ?, ?, ?, ?, ?, ?, ?)";
            con=getConnection();
            ps = con.prepareStatement(sql);
            ps.setInt(1, Id);
            ps.setInt(2, Avaliacao);
            ps.setInt(3, Usuario);

```

```

        java.sql.Date data = new java.sql.Date(0);
        data.valueOf(Data_Inicio);
        ps.setDate(4, data);
        data.valueOf(Data_Final);
        ps.setDate(5, data);
        ps.setInt(6, Acertos);
        ps.setInt(7, Erros);
        ps.setInt(8, Branco);
        ps.setInt(9, Conceito);
        ps.executeUpdate();
        incluiu = true;
    }
    catch (Exception e) {
        System.out.println(e.toString());
    }
    finally{
        try {
            if (ps!=null)
                ps.close();
            if (con!=null)
                con.close();
        }
        catch (SQLException e) {
        }
    }
    return incluiu;
}

public boolean altera(int Id, int Avaliacao, int Usuario, String Data_Inicio, String Data_Final, int
Acertos, int Erros, int Branco, int Conceito){
    Connection con = null;
    PreparedStatement ps = null;
    boolean alterou = false;

    try {
        String sql = "Update Resolucao Set Avaliacao = ?, Usuario = ?, Data_Inicio = ?,
Data_Final = ?, Acertos = ?, Erros = ?, Branco = ?, Conceito = ? Where Id = ?";
        con=getConnection();
        ps = con.prepareStatement(sql);
        ps.setInt(1, Avaliacao);
        ps.setInt(2, Usuario);
        java.sql.Date data = new java.sql.Date(0);
        data.valueOf(Data_Inicio);
        ps.setDate(3, data);
        data.valueOf(Data_Final);
        ps.setDate(4, data);
        ps.setInt(5, Acertos);
        ps.setInt(6, Erros);
        ps.setInt(7, Branco);
        ps.setInt(8, Conceito);
        ps.setInt(9, Id);
        ps.executeUpdate();
        ps.executeUpdate();
        alterou = true;
    }
    catch (Exception e) {
        System.out.println(e.toString());
    }
}

```

```

finally{
    try {
        if (ps!=null)
            ps.close();
        if (con!=null)
            con.close();
        }
    catch (SQLException e) {
    }
}
return alterou;
}

```

```

public boolean exclui(int Id){
    Connection con = null;
    PreparedStatement ps = null;
    boolean excluiu = false;

    try {
        String sql = "Delete From Resolucao Where Id=?";
        con=getConnection();
        ps = con.prepareStatement(sql);
        ps.setInt(1, Id);
        ps.executeUpdate();
        excluiu = true;
    }
    catch (Exception e) {
        System.out.println(e.toString());
    }
    finally{
        try {
            if (ps!=null)
                ps.close();
            if (con!=null)
                con.close();
        }
        catch (SQLException e) {
        }
    }
    return excluiu;
}

```

```

public String[] localizaId(int Id){
    Connection con = null;
    PreparedStatement ps = null;
    String[] row = null;

    try {
        String sql = "Select Id, Avaliacao, Usuario, Data_Inicio, Data_Final, Acertos, Erros, Branco, Conceito From Resolucao Where Id = ?";
        con=getConnection();
        ps = con.prepareStatement(sql);
        ps.setInt(1, Id);
        ResultSet rs = ps.executeQuery();
        if (rs.next()) {
            row = new String [9];
            row[0] = rs.getString("Id");
            row[1] = rs.getString("Avaliacao");

```

```

        row[2] = rs.getString("Usuario");
        row[3] = rs.getString("Data_Inicio");
        row[4] = rs.getString("Data_Final");
        row[5] = rs.getString("Acertos");
        row[6] = rs.getString("Erros");
        row[7] = rs.getString("Branco");
        row[8] = rs.getString("Conceito");
    }
}
catch (Exception e) {
    System.out.println(e.toString());
}
finally{
    try {
        if (ps!=null)
            ps.close();
        if (con!=null)
            con.close();
    }
    catch (SQLException e) {
    }
}
return row;
}

public ArrayList localizaAvaliacao(int Avaliacao){
    Connection con = null;
    PreparedStatement ps = null;
    ArrayList retval = new ArrayList(50);
    try {
        String sql = "Select Id, Avaliacao, Usuario, Data_Inicio, Data_Final, Acertos, Erros,
Branco, Conceito From Resolucao Where Avaliacao = ?";
        con=getConnection();
        ps = con.prepareStatement(sql);
        ps.setInt(1, Avaliacao);
        ResultSet rs = ps.executeQuery();
        while (rs.next()) {
            String[] row = new String [9];
            row[0] = rs.getString("Id");
            row[1] = rs.getString("Avaliacao");
            row[2] = rs.getString("Usuario");
            row[3] = rs.getString("Data_Inicio");
            row[4] = rs.getString("Data_Final");
            row[5] = rs.getString("Acertos");
            row[6] = rs.getString("Erros");
            row[7] = rs.getString("Branco");
            row[8] = rs.getString("Conceito");
            retval.add(row);
        }
    }
    catch (Exception e) {
        System.out.println(e.toString());
    }
    finally{
        try {
            if (ps!=null)
                ps.close();
            if (con!=null)

```

```

        con.close();
    }
    catch (SQLException e) {
    }
}
return retval;
}

```

```

public ArrayList localizaUsuario(int Usuario){
    Connection con = null;
    PreparedStatement ps = null;
    ArrayList retval = new ArrayList(50);
    try {
        String sql = "Select Id, Avaliacao, Usuario, Data_Inicio, Data_Final, Acertos, Erros,
Branco, Conceito From Resolucao Where Usuario = ?";
        con=getConnection();
        ps = con.prepareStatement(sql);
        ps.setInt(1, Usuario);
        ResultSet rs = ps.executeQuery();
        while (rs.next()) {
            String[] row = new String [9];
            row[0] = rs.getString("Id");
            row[1] = rs.getString("Avaliacao");
            row[2] = rs.getString("Usuario");
            row[3] = rs.getString("Data_Inicio");
            row[4] = rs.getString("Data_Final");
            row[5] = rs.getString("Acertos");
            row[6] = rs.getString("Erros");
            row[7] = rs.getString("Branco");
            row[8] = rs.getString("Conceito");
            retval.add(row);
        }
    }
    catch (Exception e) {
        System.out.println(e.toString());
    }
    finally{
        try {
            if (ps!=null)
                ps.close();
            if (con!=null)
                con.close();
        }
        catch (SQLException e) {
        }
    }
    return retval;
}

```

```

public ArrayList localizaTudo(){
    Connection con = null;
    PreparedStatement ps = null;
    ArrayList retval = new ArrayList(50);
    try {
        String sql = "Select Id, Avaliacao, Usuario, Data_Inicio, Data_Final, Acertos, Erros,
Branco, Conceito From Resolucao";
        con=getConnection();
        ps = con.prepareStatement(sql);
    }
}

```

```

        ResultSet rs = ps.executeQuery();
        while (rs.next()) {
            String[] row = new String [9];
            row[0] = rs.getString("Id");
            row[1] = rs.getString("Avaliacao");
            row[2] = rs.getString("Usuario");
            row[3] = rs.getString("Data_Inicio");
            row[4] = rs.getString("Data_Final");
            row[5] = rs.getString("Acertos");
            row[6] = rs.getString("Erros");
            row[7] = rs.getString("Branco");
            row[8] = rs.getString("Conceito");
            retval.add(row);
        }
    }
    catch (Exception e) {
        System.out.println(e.toString());
    }
    finally{
        try {
            if (ps!=null)
                ps.close();
            if (con!=null)
                con.close();
        }
        catch (SQLException e) {
        }
    }
    return retval;
}

public void ejbCreate() {
}

public void ejbRemove() {
}

public void ejbActivate() {
}

public void ejbPassivate() {
}

public void setSessionContext(SessionContext sc) {
}
}

```

7.4 Exemplo de duas das quatorze páginas JSP da aplicação cliente

avaliacoes.jsp

```
<%@ page contentType="text/html; charset=iso-8859-1" language="java" import="java.sql.*"
errorPage="" %>
<%@ page import="javax.naming.*"%>
<%@ page import="javax.rmi.PortableRemoteObject"%>
<%@ page import="java.util.Properties"%>
<%@ page import="com.projeto.ejb.Avaliacao"%>
<%@ page import="com.projeto.ejb.AvaliacaoHome"%>
<%@ page import="com.projeto.ejb.Relevancia"%>
<%@ page import="com.projeto.ejb.RelevanciaHome"%>
<%@ page import="com.projeto.ejb.Topico"%>
<%@ page import="com.projeto.ejb.TopicoHome"%>
<%@ page import="com.projeto.ejb.Dificuldade"%>
<%@ page import="com.projeto.ejb.DificuldadeHome"%>
<%@ page import="com.projeto.ejb.Questoes_Avaliacao"%>
<%@ page import="com.projeto.ejb.Questoes_AvaliacaoHome"%>
<%@ page import="com.projeto.ejb.Usuarios"%>
<%@ page import="com.projeto.ejb.UsuariosHome"%>

<%@ page import="com.projeto.ejb.DataUtil"%>

<%@ page import="java.util.ArrayList"%>
<%
String id=request.getParameter("id");
String datainicio=request.getParameter("datainicio");
String datafinal=request.getParameter("datafinal");
String totalquestoes=request.getParameter("totalquestoes");
String peso=request.getParameter("peso");
String duracao=request.getParameter("duracao");
String corrige=request.getParameter("corrige");
String idtopico=request.getParameter("idtopico");
String idrelevancia=request.getParameter("idrelevancia");
String iddificuldade=request.getParameter("iddificuldade");
String numeroquestoes=request.getParameter("numeroquestoes");
String pesoquestoes=request.getParameter("pesoquestoes");

String vid="";
String vdatainicio="";
String vdatafinal="";
String vttotalquestoes="";
String vpeso="";
String vduracao="";
String vcorrige="";
String vidtopico="";
String vidrelevancia="";
String viddificuldade="";
String vnumeroquestoes="";
String vpesoquestoes="";
String idusuario = session.getAttribute("id").toString();

if (id!=null) vid=id;
if (datainicio!=null) vdatainicio=datainicio;
```

```
if (datafinal!=null) vdatafinal=datafinal;
if (totalquestoes!=null) vtotalquestoes=totalquestoes;
if (peso!=null) vpeso=peso;
if (duracao!=null) vduracao=duracao;
if (corrige!=null) vcorrige=corrige;
if (idtopico!=null) vidtopico=idtopico;
if (idrelevancia!=null) vidrelevancia=idrelevancia;
if (iddificuldade!=null) viddificuldade=iddificuldade;
if (numeroquestoes!=null) vnumeroquestoes=numeroquestoes;
if (pesoquestoes!=null) vpesoquestoes=pesoquestoes;
String acao=request.getParameter("acao");
```

```
Properties properties = new Properties();
properties.put(Context.INITIAL_CONTEXT_FACTORY,
"org.jnp.interfaces.NamingContextFactory");
properties.put(Context.PROVIDER_URL, "localhost:1099");
InitialContext jndiContext = new InitialContext(properties);
```

```
Object refre = jndiContext.lookup("SessionRelevancia");
RelevanciaHome homere = (RelevanciaHome) PortableRemoteObject.narrow (refre,
RelevanciaHome.class);
Relevancia relevancia = homere.create();
```

```
Object refto = jndiContext.lookup("SessionTopico");
TopicoHome hometo = (TopicoHome) PortableRemoteObject.narrow (refto, TopicoHome.class);
Topico topico = hometo.create();
```

```
Object refdi = jndiContext.lookup("SessionDificuldade");
DificuldadeHome homedi = (DificuldadeHome) PortableRemoteObject.narrow (refdi,
DificuldadeHome.class);
Dificuldade dificuldade = homedi.create();
```

```
Object refav = jndiContext.lookup("SessionAvaliacao");
AvaliacaoHome homeav = (AvaliacaoHome) PortableRemoteObject.narrow (refav,
AvaliacaoHome.class);
Avaliacao avaliacao = homeav.create();
```

```
Object refqav = jndiContext.lookup("SessionQuestoes_Avaliacao");
Questoes_AvaliacaoHome homeqav = (Questoes_AvaliacaoHome)
PortableRemoteObject.narrow (refqav, Questoes_AvaliacaoHome.class);
Questoes_Avaliacao questoes_avaliacao = homeqav.create();
```

```
Object refusu = jndiContext.lookup("SessionUsuarios");
UsuariosHome homeusu = (UsuariosHome) PortableRemoteObject.narrow (refusu,
UsuariosHome.class);
Usuarios usuarios = homeusu.create();
```

```
if (request.getParameter("b_cancelar")!=null){
    id="";
    datainicio="";
    datafinal="";
    totalquestoes="";
    peso="";
    duracao="";
    corrige="";
    idtopico="";
    idrelevancia="";
    iddificuldade="";
```

```

        numeroquestoes="";
        pesoquestoes="";
    }

    if (request.getParameter("b_incluir")!=null){
        try
        {
            DataUtil data = new DataUtil();
            data.setData(vdatainicio, false);
            if (data.dataValida()){
                String vdi=data.getData(true);
                data.setData(vdatafinal, false);
                if (data.dataValida()){
                    String vdf=data.getData(true);
                    avaliacao.inclui(Integer.parseInt(vid), Integer.parseInt(idusuario), vdi, vdf,
                    Integer.parseInt(vtotalquestoes), Integer.parseInt(vpeso), Integer.parseInt(vduracao),
                    vcorrige.equals("1"));
                }
            }
        }
        catch(Exception e){
            System.out.println(e.toString());
        }
    }

    if (request.getParameter("b_incluirq")!=null){
        try
        {
            questoes_avalicao.inclui(Integer.parseInt(vid), Integer.parseInt(vidtopico),
            Integer.parseInt(vidrelevancia), Integer.parseInt(viddificuldade),
            Integer.parseInt(vnumeroquestoes), Integer.parseInt(vpesoquestoes));
        }
        catch(Exception e){
            System.out.println(e.toString());
        }
    }

    if (request.getParameter("b_alterar")!=null){
        try
        {
            DataUtil data = new DataUtil();
            data.setData(vdatainicio, false);
            if (data.dataValida()){
                String vdi=data.getData(true);
                data.setData(vdatafinal, false);
                if (data.dataValida()){
                    String vdf=data.getData(true);
                    avaliacao.altera(Integer.parseInt(vid), Integer.parseInt(idusuario), vdi, vdf,
                    Integer.parseInt(vtotalquestoes), Integer.parseInt(vpeso), Integer.parseInt(vduracao),
                    vcorrige.equals("1"));
                }
            }
        }
        catch(Exception e){
            System.out.println(e.toString());
        }
    }
}

```

```

if ((request.getParameter("b_excluir")!=null) || ((acao!=null) && (acao.equals("E")))){
    try
    {
        avaliacao.exclui(Integer.parseInt(id));
        id="";
        datainicio="";
        datafinal="";
        totalquestoes="";
        peso="";
        duracao="";
        corrige="";
    }
    catch(Exception e){
        System.out.println(e.toString());
    }
}

if ((acao!=null) && (acao.equals("EQ"))){
    try
    {
        questoes_avaliacao.exclui(Integer.parseInt(vid), Integer.parseInt(vidtopico),
Integer.parseInt(vidrelevancia), Integer.parseInt(viddificuldade));
    }
    catch(Exception e){
        System.out.println(e.toString());
    }
}

if ((acao!=null) && ((acao.equals("A") || acao.equals("EQ"))) && (id!=null)){
    try
    {
        String[] rec = avaliacao.localiza(Integer.parseInt(id));
        DataUtil data = new DataUtil();
        id=rec[0];
        data.setData(rec[2], true);
        datainicio=data.getData(false);
        data.setData(rec[3], true);
        datafinal=data.getData(false);
        totalquestoes=rec[4];
        peso=rec[5];
        duracao=rec[6];
        corrige=rec[7];
    }
    catch(Exception e){
        System.out.println(e.toString());
    }
}

if (id!=null) vid=id;
if (datainicio!=null) vdatainicio=datainicio;
if (datafinal!=null) vdatafinal=datafinal;
if (totalquestoes!=null) vtotalquestoes=totalquestoes;
if (peso!=null) vpeso=peso;
if (duracao!=null) vduracao=duracao;
if (corrige!=null) vcorrige=corrige;
if (idtopico!=null) vidtopico=idtopico;
if (idrelevancia!=null) vidrelevancia=idrelevancia;
if (iddificuldade!=null) viddificuldade=iddificuldade;
if (numeroquestoes!=null) vnumeroquestoes=numeroquestoes;

```

```

if (pesoquestoes!=null) vpesoquestoes=pesoquestoes;
%>
<html>
<head>
<title>Avalia&ccedil;&otilde;es</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<link href="estilos.css" rel="stylesheet" type="text/css">
</head>

<body>
<table width="100%" height="550" border="0" cellpadding="0" cellspacing="0">
<tr>
<td height="73" align="left" valign="top">
<jsp:include page="cabecalho.jsp" flush="true" />
</td>
</tr>
<tr>
<td height="520" align="center" valign="top"><br>
<form action="avaliacoes.jsp" method="post" name="f_avaliacoes" id="f_avaliacoes">
<table width="50%" border="0" cellpadding="1" cellspacing="1" bgcolor="#000000">
<tr align="center" valign="middle" bgcolor="#FFFFFF">
<td width="14%" align="left" bgcolor="#CCCCCC"><font size="1">Id</font></td>
<td width="86%" align="left"> <input name="id" type="text" id="id"
value="<%out.println(vid);%>" size="11" maxlength="11"></td>
</tr>
<tr align="center" valign="middle" bgcolor="#FFFFFF">
<td align="left" bgcolor="#CCCCCC"><font size="1">Data In&iacute;cio</font></td>
<td align="left"><input name="datainicio" type="text" id="datainicio"
value="<%out.println(vdatainicio);%>" size="10" maxlength="10"></td>
</tr>
<tr align="center" valign="middle" bgcolor="#FFFFFF">
<td align="left" bgcolor="#CCCCCC"><font size="1">Data Final</font></td>
<td align="left"><input name="datafinal" type="text" id="datafinal"
value="<%out.println(vdatafinal);%>" size="10" maxlength="10"></td>
</tr>
<tr align="center" valign="middle" bgcolor="#FFFFFF">
<td align="left" bgcolor="#CCCCCC"><font size="1">Total Quest&otilde;es</font></td>
<td align="left"><input name="totalquestoes" type="text" id="totalquestoes"
value="<%out.println(vtotalquestoes);%>" size="11" maxlength="11"></td>
</tr>
<tr align="center" valign="middle" bgcolor="#FFFFFF">
<td align="left" bgcolor="#CCCCCC"><font size="1">Peso</font></td>
<td align="left"><input name="peso" type="text" id="peso"
value="<%out.println(vpeso);%>" size="11" maxlength="11"></td>
</tr>
<tr align="center" valign="middle" bgcolor="#FFFFFF">
<td align="left" bgcolor="#CCCCCC"><font size="1">Dura&ccedil;&atilde;o</font></td>
<td align="left"><input name="duracao" type="text" id="duracao"
value="<%out.println(vduracao);%>" size="11" maxlength="11"></td>
</tr>
<tr align="center" valign="middle" bgcolor="#FFFFFF">
<td align="left" bgcolor="#CCCCCC"><font size="1">Corrige</font></td>
<td align="left"><input name="corrigir" type="checkbox" id="corrigir" value="1"<%if
(vcorrigir.equals("1")) out.println(" checked ");%>></td>
</tr>
<tr align="center" valign="middle" bgcolor="#FFFFFF">
<td colspan="2"> <input name="b_incluir" type="submit" id="b_incluir" value="Incluir">
<input name="b_alterar" type="submit" id="b_alterar" value="Alterar">

```

```



```

```

        out.println("<option
value=\""+s[0]+"\""+selected+">"+s[1]+"</option>");
    }
}
catch(Exception e){
    System.out.println(e.toString());
}
}
%>
</select></td>
<td align="center"><select name="iddificuldade" id="iddificuldade">
    <option value="0" selected>Escolha a dificuldade</option>
    <%
        try
        {
            int i;
            ArrayList di = dificuldade.localizaTudo();
            int rowCount = di.size();
            String selected;
            for(i=0; i<rowCount; i++){
                String[] s = (String[]) di.get(i);
                selected="";
                if (viddificuldade.equals(s[0])) selected=" selected ";
                out.println("<option
value=\""+s[0]+"\""+selected+">"+s[1]+"</option>");
            }
        }
        catch(Exception e){
            System.out.println(e.toString());
        }
    }
    %>
</select> </td>
<td align="center"><input name="numeroquestoes" type="text" id="numeroquestoes"
value="<%out.println(vnumeroquestoes);%>" size="11" maxlength="11"></td>
<td align="center"><input name="pesoquestoes" type="text" id="pesoquestoes"
value="<%out.println(vpesoquestoes);%>" size="11" maxlength="11"></td>
<td align="center"> <input name="b_incluirq" type="submit" id="b_incluirq"
value="Incluir"></td>
</tr>
<%
int iquest;
ArrayList aquest = questoes_avalicao.localizaAvalicao(Integer.parseInt(vid));
int rowCountquest = aquest.size();
for(iquest=0; iquest<rowCountquest; iquest++){
    String[] squest = (String[]) aquest.get(iquest);
%>
<tr align="center" valign="middle" bgcolor="#FFFFFF">
    <td align="left" valign="middle"><font size="1">
        <%
            String[] aux = topico.localizaId(Integer.parseInt(squest[1]));
            out.println(aux[1]);
        %>
    </font></td>
    <td align="left" valign="middle"><font size="1">
        <%
            String[] auxr = relevancia.localizaId(Integer.parseInt(squest[2]));
            out.println(auxr[1]);
        %>
    </font></td>

```

```

<td align="left" valign="middle"><font size="1">
    <%
        String[] auxd = dificuldade.localizaId(Integer.parseInt(squest[3]));
        out.println(auxd[1]);
    %>
</font></td>
<td align="left" valign="middle"><font size="1">
    <%out.println(squest[4]);%>
</font></td>
<td align="left" valign="middle"><font size="1">
    <%out.println(squest[5]);%>
</font></td>
<td><font size="1"><a
href="avaliacoes.jsp?acao=EQ&id=<%out.println(id);%>&idtopico=<%out.println(squest[1]);%>&
idrelevancia=<%out.println(squest[2]);%>&id dificuldade=<%out.println(squest[3]);%>">Excluir</
a></font></td>
</tr>
<%
}
%>
</table>
<%
}
catch(Exception e) {
    System.out.println(e.toString());
}
}
%>
</form>
<br>
<br>
<%
if ((request.getParameter("b_listar")!=null) || (request.getParameter("acao")!=null)){
    try
    {
        %>
<table width="70%" border="0" cellpadding="1" cellspacing="1" bgcolor="#000000">
    <tr align="center" valign="middle" bgcolor="#CCCCCC">
        <td width="3%" height="15"><font size="1">Id</font></td>
        <td width="48%"><font size="1">Professor</font></td>
        <td width="7%"><font size="1">Data In<i>icio</i></font></td>
        <td width="7%"><font size="1">Data Fim</font></td>
        <td width="8%"><font size="1">Total de Quest<i>ões</i></font></td>
        <td width="5%"><font size="1">Peso</font></td>
        <td width="8%"><font size="1">Dura<i>ccedil;&atilde;o</i></font></td>
        <td width="7%"><font size="1">Corrige</font></td>
        <td width="7%"><font size="1">A<i>ccedil;&atilde;o</i></font></td>
    </tr>
    <%
    int i;
        ArrayList a = new ArrayList();
        if (session.getAttribute("idtipousuario").equals("1")){
            a = avaliacao.localizaTudo();
        }
        if (session.getAttribute("idtipousuario").equals("2")){
            a = avaliacao.localizaProf(Integer.parseInt(idusuario));
        }
        int rowCount = a.size();

```

```

        for(i=0; i<rowCount; i++){
            String[] s = (String[]) a.get(i);
            DataUtil data = new DataUtil();
%>
            <tr align="center" valign="middle" bgcolor="#FFFFFF">
                <td align="center"><font size="1"> <a
href="avaliacoes.jsp?acao=A&id=<%out.println(s[0]);%>">
                    <%out.println(s[0]);%>
                </a> </font></td>
                <td align="left"><font size="1"><a
href="avaliacoes.jsp?acao=A&id=<%out.println(s[0]);%>">
                    <%
                        String[] aux = usuarios.localizald(Integer.parseInt(s[1]));
                        out.println(aux[4]);
                    %>
                </a></font></td>
                <td align="center"><font size="1"> <a
href="avaliacoes.jsp?acao=A&id=<%out.println(s[0]);%>">
                    <%
                        data.setData(s[2], true);
                        out.println(data.getData(false));%>
                </a></font></td>
                <td align="center"><font size="1"><a
href="avaliacoes.jsp?acao=A&id=<%out.println(s[0]);%>">
                    <%
                        data.setData(s[3], true);
                        out.println(data.getData(false));%>
                </a></font></td>
                <td align="center"><font size="1"><a
href="avaliacoes.jsp?acao=A&id=<%out.println(s[0]);%>">
                    <%out.println(s[4]);%>
                </a></font></td>
                <td align="center"><font size="1"><a
href="avaliacoes.jsp?acao=A&id=<%out.println(s[0]);%>">
                    <%out.println(s[5]);%>
                </a></font></td>
                <td align="center"><font size="1"><a
href="avaliacoes.jsp?acao=A&id=<%out.println(s[0]);%>">
                    <%out.println(s[6]);%>
                </a></font></td>
                <td align="center"><font size="1"><a
href="avaliacoes.jsp?acao=A&id=<%out.println(s[0]);%>">
                    <%if (s[7].equals("1")) out.println("Sim"); else out.println("Nao");%>
                </a> </font></td>
                <td align="center"><font size="1"><a
href="avaliacoes.jsp?acao=E&id=<%out.println(s[0]);%>">Excluir</a></font></td>
            </tr>
            <%
        }
%>
    </table>
<%
}
}
catch(Exception e) {
    System.out.println(e.toString());
}
}
%>

```

```

        </td>
    </tr>
    <tr>
        <td height="31" align="center" valign="bottom">
            <jsp:include page="rodape.jsp" flush="true" />
        </td>
    </tr>
</table>
</body>
</html>

```

questoes.jsp

```

<%@ page contentType="text/html; charset=iso-8859-1" language="java" import="java.sql.*"
errorPage="" %>
<%@ page import="javax.naming.*"%>
<%@ page import="javax.rmi.PortableRemoteObject"%>
<%@ page import="java.util.Properties"%>
<%@ page import="com.projeto.ejb.Questao"%>
<%@ page import="com.projeto.ejb.QuestaoHome"%>
<%@ page import="com.projeto.ejb.Relevancia"%>
<%@ page import="com.projeto.ejb.RelevanciaHome"%>
<%@ page import="com.projeto.ejb.Topico"%>
<%@ page import="com.projeto.ejb.TopicoHome"%>
<%@ page import="com.projeto.ejb.Dificuldade"%>
<%@ page import="com.projeto.ejb.DificuldadeHome"%>
<%@ page import="com.projeto.ejb.Resposta"%>
<%@ page import="com.projeto.ejb.RespostaHome"%>
<%@ page import="com.projeto.ejb.Usuarios"%>
<%@ page import="com.projeto.ejb.UsuariosHome"%>

<%@ page import="java.util.ArrayList"%>
<%
String id=request.getParameter("id");
String idresposta=request.getParameter("idresposta");
String textoresposta=request.getParameter("textoresposta");
String idrelevancia=request.getParameter("idrelevancia");
String correto=request.getParameter("correto");
String idtopico=request.getParameter("idtopico");
String iddificuldade=request.getParameter("iddificuldade");
String texto=request.getParameter("texto");
String vid="";
String vidresposta="";
String vtextoresposta="";
String vcorreto="0";
String vidrelevancia="";
String vidtopico="";
String viddificuldade="";
String vtexto="";
if (id!=null) vid=id;
if (idresposta!=null) vidresposta=idresposta;
if (textoresposta!=null) vtextoresposta=textoresposta;
if (correto!=null) vcorreto=correto;
if (idrelevancia!=null) vidrelevancia=idrelevancia;
if (idtopico!=null) vidtopico=idtopico;
if (iddificuldade!=null) viddificuldade=iddificuldade;

```

```

if (texto!=null) vtexto=texto.trim();
String acao=request.getParameter("acao");

Properties properties = new Properties();
properties.put(Context.INITIAL_CONTEXT_FACTORY,
"org.jnp.interfaces.NamingContextFactory");
properties.put(Context.PROVIDER_URL, "localhost:1099");
InitialContext jndiContext = new InitialContext(properties);

Object refre = jndiContext.lookup("SessionRelevancia");
RelevanciaHome homere = (RelevanciaHome) PortableRemoteObject.narrow (refre,
RelevanciaHome.class);
Relevancia relevancia = homere.create();

Object refto = jndiContext.lookup("SessionTopico");
TopicoHome hometo = (TopicoHome) PortableRemoteObject.narrow (refto, TopicoHome.class);
Topico topico = hometo.create();

Object refdi = jndiContext.lookup("SessionDificuldade");
DificuldadeHome homedi = (DificuldadeHome) PortableRemoteObject.narrow (refdi,
DificuldadeHome.class);
Dificuldade dificuldade = homedi.create();

Object refusu = jndiContext.lookup("SessionUsuarios");
UsuariosHome homeusu = (UsuariosHome) PortableRemoteObject.narrow (refusu,
UsuariosHome.class);
Usuarios usuarios = homeusu.create();

if (request.getParameter("b_cancelar")!=null){
    id="";
    idrelevancia="";
    idtopico="";
    iddificuldade="";
    texto="";
    acao="";
}

if (request.getParameter("b_incluir")!=null){
    try
    {
        Object ref = jndiContext.lookup("SessionQuestao");
        QuestaoHome home = (QuestaoHome) PortableRemoteObject.narrow (ref,
QuestaoHome.class);
        Questao questao = home.create();
        String idusuario = session.getAttribute("id").toString();
        questao.inclui(Integer.parseInt(vid), Integer.parseInt(vidrelevancia),
Integer.parseInt(idusuario), Integer.parseInt(vidtopico), Integer.parseInt(viddificuldade), vtexto);
    }
    catch(Exception e){
        System.out.println(e.toString());
    }
}

if (request.getParameter("b_inclurr")!=null){
    try
    {
        Object ref = jndiContext.lookup("SessionResposta");

```

```

        RespostaHome home = (RespostaHome) PortableRemoteObject.narrow (ref,
RespostaHome.class);
        Resposta resposta = home.create();
        resposta.inclui(Integer.parseInt(vidresposta), Integer.parseInt(vid), vtextoresposta,
vcorreto.equals("1"));
    }
    catch(Exception e){
        System.out.println(e.toString());
    }
}

if (request.getParameter("b_alterarr")!=null){
    try
    {
        Object ref = jndiContext.lookup("SessionResposta");
        RespostaHome home = (RespostaHome) PortableRemoteObject.narrow (ref,
RespostaHome.class);
        Resposta resposta = home.create();
        resposta.altera(Integer.parseInt(vidresposta), Integer.parseInt(vid), vtextoresposta,
vcorreto.equals("1"));
    }
    catch(Exception e){
        System.out.println(e.toString());
    }
}

if (request.getParameter("b_alterar")!=null){
    try
    {
        Object ref = jndiContext.lookup("SessionQuestao");
        QuestaoHome home = (QuestaoHome) PortableRemoteObject.narrow (ref,
QuestaoHome.class);
        Questao questao = home.create();
        String idusuario = session.getAttribute("id").toString();
        questao.altera(Integer.parseInt(vid), Integer.parseInt(vidrelevancia),
Integer.parseInt(idusuario), Integer.parseInt(vidtopico), Integer.parseInt(viddificuldade), vtexto);
    }
    catch(Exception e){
        System.out.println(e.toString());
    }
}

if ((request.getParameter("b_excluir")!=null) || ((acao!=null) && (acao.equals("E")))){
    try
    {
        Object ref = jndiContext.lookup("SessionQuestao");
        QuestaoHome home = (QuestaoHome) PortableRemoteObject.narrow (ref,
QuestaoHome.class);
        Questao questao = home.create();
        questao.exclui(Integer.parseInt(id));
        id="";
        idrelevancia="";
        idtopico="";
        iddificuldade="";
        texto="";
    }
    catch(Exception e){
        System.out.println(e.toString());
    }
}

```

```

    }
}

if ((acao!=null) && (acao.equals("ER"))){
    try
    {
        Object ref = jndiContext.lookup("SessionResposta");
        RespostaHome home = (RespostaHome) PortableRemoteObject.narrow (ref,
RespostaHome.class);
        Resposta resposta = home.create();
        resposta.exclui(Integer.parseInt(idresposta));
        vidresposta="";
    }
    catch(Exception e){
        System.out.println(e.toString());
    }
}

if ((acao!=null) && (acao.equals("AR"))){
    try
    {
        Object ref = jndiContext.lookup("SessionResposta");
        RespostaHome home = (RespostaHome) PortableRemoteObject.narrow (ref,
RespostaHome.class);
        Resposta resposta = home.create();
        String[] rec = resposta.localizaId(Integer.parseInt(vidresposta));
        vidresposta=rec[0];
        vtextoresposta=rec[2];
        vcorreto=rec[3];
    }
    catch(Exception e){
        System.out.println(e.toString());
    }
}

if ((acao!=null) && ((acao.equals("A") || acao.equals("ER") || acao.equals("AR"))) && (id!=null)){
    try
    {
        Object ref = jndiContext.lookup("SessionQuestao");
        QuestaoHome home = (QuestaoHome) PortableRemoteObject.narrow (ref,
QuestaoHome.class);
        Questao questao = home.create();

        String[] rec = questao.localizaId(Integer.parseInt(id));
        id=rec[0];
        idrelevancia=rec[1];
        idtopico=rec[3];
        iddificuldade=rec[4];
        texto=rec[5];
    }
    catch(Exception e){
        System.out.println(e.toString());
    }
}

if (id!=null) vid=id;
if (idrelevancia!=null) vidrelevancia=idrelevancia;
if (idtopico!=null) vidtopico=idtopico;
if (iddificuldade!=null) viddificuldade=iddificuldade;

```

```

if (texto!=null) vtexto=texto.trim();
%>
<html>
<head>
<title>Quest&otilde;es</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<link href="estilos.css" rel="stylesheet" type="text/css">
</head>

<body>
<table width="100%" height="550" border="0" cellpadding="0" cellspacing="0">
<tr>
<td height="73" align="left" valign="top">
<jsp:include page="cabecalho.jsp" flush="true" />
</td>
</tr>
<tr>
<td height="520" align="center" valign="top"><br>
<form action="questoes.jsp" method="post" name="f_questoes" id="f_questoes">
<table width="50%" border="0" cellpadding="1" cellspacing="1" bgcolor="#000000">
<tr align="center" valign="middle" bgcolor="#FFFFFF">
<td width="10%" align="left" bgcolor="#CCCCCC"><font size="1">Id</font></td>
<td width="90%" align="left"> <input name="id" type="text" id="id"
value="<%out.println(vid);%>" size="11" maxlength="11"></td>
</tr>
<tr align="center" valign="middle" bgcolor="#FFFFFF">
<td align="left" bgcolor="#CCCCCC"><font size="1">Relev&acirc;ncia</font></td>
<td align="left"><select name="idrelevancia" id="idrelevancia">
<option value="0" selected>Escolha a relev&acirc;ncia</option>
<%
try
{
int i;
ArrayList re = relevancia.localizaTudo();
int rowCount = re.size();
String selected;
for(i=0; i<rowCount; i++){
String[] s = (String[]) re.get(i);
selected="";
if (vidrelevancia.equals(s[0])) selected=" selected ";
out.println("<option
value=\""+s[0]+"\""+selected+"\""+s[1]+"</option>");
}
}
catch(Exception e){
System.out.println(e.toString());
}
%>
</select></td>
</tr>
<tr align="center" valign="middle" bgcolor="#FFFFFF">
<td align="left" bgcolor="#CCCCCC"><font size="1">T&ocute;pico</font></td>
<td align="left"><select name="idtopico" id="idtopico">
<option value="0" selected>Escolha o t&ocute;pico</option>
<%
try
{
int i;

```

```

        ArrayList to = topico.localizaTudo();
        int rowCount = to.size();
        String selected;
        for(i=0; i<rowCount; i++){
            String[] s = (String[]) to.get(i);
            selected="";
            if (vidtopico.equals(s[0])) selected=" selected ";
            out.println("<option
value=\""+s[0]+"\""+selected+">"+s[1]+"</option>");
        }
    }
    catch(Exception e){
        System.out.println(e.toString());
    }
}
%>

</select></td>
</tr>
<tr align="center" valign="middle" bgcolor="#FFFFFF">
    <td align="left" bgcolor="#CCCCCC"><font size="1">Dificuldade</font></td>
    <td align="left"><select name="iddificuldade" id="iddificuldade">
        <option value="0" selected>Escolha a dificuldade</option>
        <%
            try
            {
                int i;
                ArrayList di = dificuldade.localizaTudo();
                int rowCount = di.size();
                String selected;
                for(i=0; i<rowCount; i++){
                    String[] s = (String[]) di.get(i);
                    selected="";
                    if (viddificuldade.equals(s[0])) selected=" selected ";
                    out.println("<option
value=\""+s[0]+"\""+selected+">"+s[1]+"</option>");
                }
            }
            catch(Exception e){
                System.out.println(e.toString());
            }
        }
        %>

        </select></td>
    </tr>
    <tr align="center" valign="middle" bgcolor="#FFFFFF">
        <td align="left" bgcolor="#CCCCCC"><font size="1">Texto</font></td>
        <td align="left"> <textarea name="texto" cols="60" rows="4"
id="texto"><%out.println(vtexto);%></textarea></td>
    </tr>
    <tr align="center" valign="middle" bgcolor="#FFFFFF">
        <td colspan="2"> <input name="b_incluir" type="submit" id="b_incluir" value="Incluir">
        <input name="b_alterar" type="submit" id="b_alterar" value="Alterar">
        <input name="b_cancelar" type="submit" id="b_cancelar" value="Cancelar">
        <input name="b_excluir" type="submit" id="b_excluir" value="Excluir">
        <input name="b_listar" type="submit" id="b_listar" value="Listar"></td>
    </tr>
</table><br>
<%
    if (!vid.equals("")){
        %>

```

```

        <font size="2"><strong>Respostas</strong></font>
        <%
        try{
            Object refresp = jndiContext.lookup("SessionResposta");
            RespostaHome homeresp = (RespostaHome) PortableRemoteObject.narrow
(refresp, RespostaHome.class);
            Resposta resposta = homeresp.create();
            %>
            <table width="50%" border="0" cellpadding="1" cellspacing="1" bgcolor="#000000">
            <tr align="center" valign="middle" bgcolor="#CCCCCC">
                <td width="5%"><font size="1">Id</font></td>
                <td width="60%"><font size="1">Texto</font></td>
                <td width="8%"><font size="1">Correto?</font></td>
                <td width="27%"><font size="1">A&ccedil;&atilde;o</font></td>
            </tr>
            <tr align="left" valign="middle" bgcolor="#FFFFFF">
                <td <input name="idresposta" type="text" id="idresposta" value="<%=vidresposta%>"
size="5" maxlength="11"></td>
                <td><input name="textoresposta" type="text" id="textoresposta"
value="<%=vtextoresposta%>" size="50" maxlength="200"></td>
                <td align="center">
<input name="correto" type="checkbox" id="correto" value="1"<%if (vcorreto.equals("1"))
out.println(" checked ");%>></td>
                <td align="center" nowrap>
                <input name="b_inclui" type="submit" id="b_inclui" value="Incluir">
                <input name="b_alterar" type="submit" id="b_alterar" value="Alterar">
                </td>
            </tr>
            <%
            int iresp;
            ArrayList aresp = resposta.localizaQuestao(Integer.parseInt(vid));
            int rowCountResp = aresp.size();
            for(iresp=0; iresp<rowCountResp; iresp++){
                String[] sresp = (String[]) aresp.get(iresp);
                %>
                <tr align="center" valign="middle" bgcolor="#FFFFFF">
                    <td align="center"><a
href="questoes.jsp?acao=AR&id=<%out.println(id);%>&idresposta=<%out.println(sresp[0]);%>"
><font size="1">
                        <%out.println(sresp[0]);%>
                        </font></a></td>
                    <td align="left"><font size="1">
                        <%out.println(sresp[2].replaceAll("\n", "<br>"));%>
                        </font></td>
                    <td align="left"><font size="1">
                        <%if (sresp[3].equals("1")) out.println("Sim"); else out.println("Nao");%>
                        </font></td>
                    <td><font size="1"><a
href="questoes.jsp?acao=ER&id=<%out.println(id);%>&idresposta=<%out.println(sresp[0]);%>"
>Excluir</a></font></td>
                </tr>
            <%
            }
            %>
            </table>
            <%
            }
            catch(Exception e) {

```

```

        System.out.println(e.toString());
    }
}
%>
    </form>
    <br>
<br>
<%
if ((request.getParameter("b_listar")!=null) || (request.getParameter("acao")!=null)){
    try
    {
        Object ref = jndiContext.lookup("SessionQuestao");
        QuestaoHome home = (QuestaoHome) PortableRemoteObject.narrow (ref,
QuestaoHome.class);
        Questao questao = home.create();
    %>
<table width="70%" border="0" cellpadding="1" cellspacing="1" bgcolor="#000000">
    <tr align="center" valign="middle" bgcolor="#CCCCCC">
        <td width="3%"><font size="1">Id</font></td>
        <td width="22%"><font size="1">Professor</font></td>
        <td width="45%"><font size="1">Texto</font></td>
        <td width="9%"><font size="1">Relev&acirc;ncia</font></td>
        <td width="5%"><font size="1">T&oacute;pico</font></td>
        <td width="9%"><font size="1">Dificuldade</font></td>
        <td width="7%"><font size="1">A&ccedil;&atilde;o</font></td>
    </tr>
    <%
    int i;
        ArrayList a = new ArrayList();
        if (session.getAttribute("idtipousuario").equals("1")){
            a = questao.localizaTudo();
        }
        if (session.getAttribute("idtipousuario").equals("2")){
            String idusuario = session.getAttribute("id").toString();
            a = questao.localizaProf(Integer.parseInt(idusuario));
        }
        int rowCount = a.size();
        for(i=0; i<rowCount; i++){
            String[] s = (String[]) a.get(i);
    %>
    <tr align="center" valign="middle" bgcolor="#FFFFFF">
        <td align="center"><font size="1"> <a
href="questoes.jsp?acao=A&id=<%out.println(s[0]);%>">
        <%out.println(s[0]);%>
        </a> </font></td>
        <td align="left"><font size="1"><a
href="questoes.jsp?acao=A&id=<%out.println(s[0]);%>">
        </a><a href="avaliacoes.jsp?acao=A&id=<%out.println(s[0]);%>">
        <%
            String[] aux = usuarios.localizaId(Integer.parseInt(s[1]));
            out.println(aux[4]);
        %>
        </a><a href="questoes.jsp?acao=A&id=<%out.println(s[0]);%>"> </a></font></td>
        <td align="left"><font size="1"> <a
href="questoes.jsp?acao=A&id=<%out.println(s[0]);%>">
        </a><a href="questoes.jsp?acao=A&id=<%out.println(s[0]);%>">
        <%out.println(s[5].replaceAll("\n", "<br>"));%>
        </a></font></td>

```

```

        <td align="left"><font size="1"><a
href="questoes.jsp?acao=A&id=<%out.println(s[0]);%>">
        <%
                String[] rec_relevancia = relevancia.localizaId(Integer.parseInt(s[1]));
                if (rec_relevancia!=null) out.println(rec_relevancia[1].trim());
                %>
        </a></font></td>
        <td align="left"><font size="1"><a
href="questoes.jsp?acao=A&id=<%out.println(s[0]);%>">
        <%
                String[] rec_topico = topico.localizaId(Integer.parseInt(s[3]));
                if (rec_topico!=null) out.println(rec_topico[1].trim());
                %>
        </a></font></td>
        <td align="left"><font size="1"><a
href="questoes.jsp?acao=A&id=<%out.println(s[0]);%>">
        <%
                String[] rec_dificuldade = dificuldade.localizaId(Integer.parseInt(s[4]));
                if (rec_dificuldade!=null) out.println(rec_dificuldade[1].trim());
                %>
        </a></font></td>
        <td><font size="1"><a
href="questoes.jsp?acao=E&id=<%out.println(s[0]);%>">Excluir</a></font></td>
    </tr>
    <%
    }
    %>
    </table>
    <%
    }
    catch(Exception e) {
        System.out.println(e.toString());
    }
    }
    %>
        </td>
    </tr>
    <tr>
        <td height="31" align="center" valign="bottom">
            <jsp:include page="rodape.jsp" flush="true" />
        </td>
    </tr>
</table>
</body>
</html>

```