

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA PARAÍBA

*CAMPUS* CAMPINA GRANDE

EMANUEL JOSÉ GUIMARÃES BRITO

ERI JONHSON OLIVEIRA DA SILVA

FELIPE NASCIMENTO SOUSA

IVANILDO FRANCISCO DOS SANTOS TERCEIRO

**TUTORIAL SOBRE COMUNICAÇÃO ENTRE COMPONENTES EM AMBIENTES DISTRIBUÍDOS NA SOLUÇÃO BASEADA EM SOFTWARE QUALITY MANAGER**

**CAMPINA GRANDE – PB**

**SETEMBRO DE 2014**

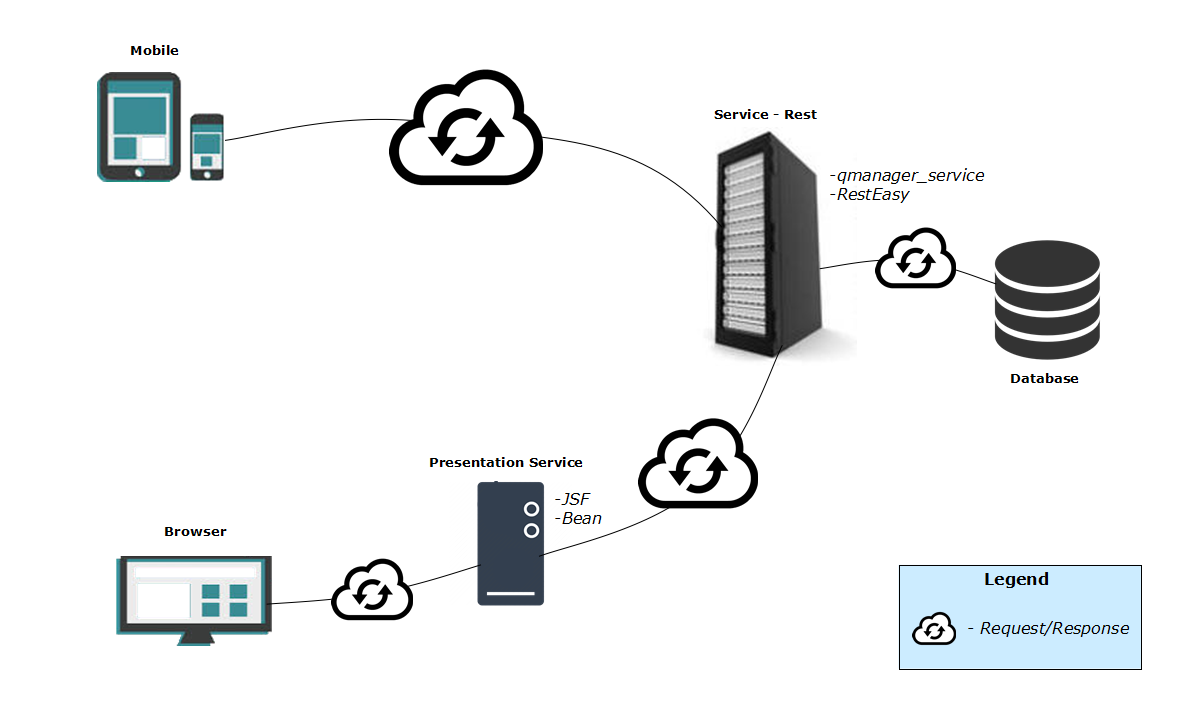
**Índice**

1. Caracterização do problema e infraestrutura da solução
2. Implementação
   1. Web Service
   2. Banco de Dados
   3. Servidor de Apresentação
   4. Cliente Móvel Android
3. Apêndices
4. Referências

**1. Caracterização do problema e infraestrutura da solução**

A Coordenação de Pesquisa e Extensão do Campus Campina Grande, do Instituto Federal de Educação, Ciência e Tecnologia da Paraíba (IFPB), tem como função gerenciar as atividades que dizem respeito aos projetos de pesquisa e extensão do campus. Atualmente, o processo de gerenciamento utiliza arquivos em papel, exigindo muito esforço, tempo e, o mais preocupante, é sujeito a muitos erros, tendo em vista o grande número de projetos.

Nesse sentido, se faz necessário uma solução baseada em software para automatizar esse processo de gerenciamento. Em virtude da popularização do uso de dispositivos móveis, essa solução deve contar com um cliente móvel. Devido ao poder de processamento limitado dos dispositivos móveis, é interessante fazer um sistema que forneça serviços com tráfego de pacotes no formato JSON, dada sua à sua leveza. A Análise do Projeto evidencia que o sistema deve ser distribuído da seguinte maneira:



1. Web service: camada responsável por atender a requisições de serviços, validar informações e enviar para o banco de dados.
2. Banco de Dados: camada responsável por persistir os dados do sistema.
3. Servidor de Apresentação: camada responsável por atender a requisições de páginas web e requisitar serviços correspondentes a tais páginas.
4. Dispositivo Móvel: cliente que requisita serviços.
5. Navegador: cliente que requisita páginas web.

**2. Implementação**

A implementação dos artefatos de software da solução foi feita em Java 7, utilizando a IDE Eclipse Enterprise Edition Luna. No caso do aplicativo móvel, utilizamos o Android Development Tools (ADT) Bundle.

O Web Service utiliza a RESTEasy - implementação da especificação JAX-RS que fornece uma API Java para serviços Web RESTful através do protocolo HTTP.

O Banco de Dados utilizado é o MySQL e a aplicação comunica-se com este através de tecnologias JDBC.

O Servidor de Apresentação utiliza tecnologias JavaServer Faces (JSF), Java Beans e implementação RESTEasy Client.

O sistema operacional para dispositivos móveis escolhido foi o Android, por conta da sua popularidade, documentação e ambiente de desenvolvimento adequados. Também usa implementação RESTEasy Client.

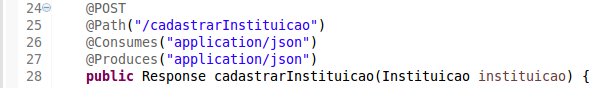
**2.1. Web Service**

Nessa primeira parte, é importante esmiuçar como construímos o Web Service seguindo a implementação RESTEasy.

1. Crie um Maven Project;
2. Caracterize o Maven Project como Dinamic Web Project e adicione Maven Dependencies na Propriedade Deployment Assembly;
3. Configure o pom.xml para baixar as APIs que a implementação RESTEasy utiliza (Apêndice 01).
4. Defina ouvintes, classes de inicialização e mapeie o servlet no web.xml (Apêndice 02).
5. Crie a classe singleton que usará os recursos JAX-RS através da classe Application (Apêndice 03).
6. Faça quantas classes RESTful quiser, adicione os serviços necessários nessas classes (Apêndice 04). e não esqueça de declará-las no construtor da sua Application (Apêndice 03).
7. Entenda como fazer um serviço que consome e produz um pacote JSON:
   * 1. Defina um caminho (@Path) para a classe onde se encontra o serviço:



* + 1. Defina método (@POST), caminho do serviço (@Path), que tipo de objeto será consumido do corpo da requisição HTTP (@Consumes) e que tipo de resposta que o recurso dará (@Produces):



* + 1. Repare no parâmetro desse método (Instituicao). Quando definimos a classe Instituicao fizemos anotações na classe (@XmlRootElement) e em seus métodos de busca (@XmlElement) para que as APIs do JAX-RS consigam converter o pacote JSON, que é recebido, em uma instância do tipo esperado (Apêndice 05).
    2. A classe ResponseBuilder foi usada para empacotar JSON da entidade, após ela ser persistida no banco de dados, e retorná-lo para o cliente (Apêndice 04).



8. A estrutura de diretórios para o projeto do quality-manager\_SERVICE consta no Apêndice 06.

**2.2. Banco de Dados**

Nessa seção, é importante explanar apenas sobre a conexão entre o Web Service e o banco de dados, logo abstrairemos detalhes íntimos sobre a construção deste.

1. Baixe o driver Java Database Connection (JDBC) no site do fabricante: <http://dev.mysql.com/downloads/connector/j/>, descompacte e importe para seu projeto.
2. Faça uma classe de conexão com seu Banco de Dados fazendo referência ao driver de conexão (Apêndice 07).
3. Defina uma interface GenericDAO para padronizar a implementação das classes que manipulam o banco (Apêndice 08).
4. Defina suas classes DAO implementando a interface GenericDAO (Apêndice 09).
5. Use Objetos do tipo Connection, PreparedStatement, Statement, ResultSet e SQLException para interagir com o Banco (Apêndice 10-14).
6. A estrutura de diretórios para o projeto do QManager consta no Apêndice 15.

É importante deixar claro que esse projeto que se comunica com o banco de dados, nomeado como QManager, é exportado para os projetos dos servidores, quality-manager\_SERVICE e quality-manager\_WEB, por ser código comum na implementação deles.

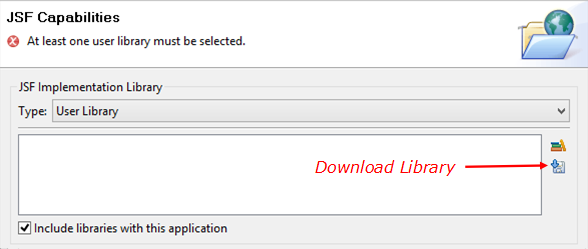
**2.3. Servidor de Apresentação**

Nesse segmento é importante destacar como construímos o Servidor de Apresentação e como este faz intermédio das requisições de um navegador com os serviços disponíveis no Web Service.

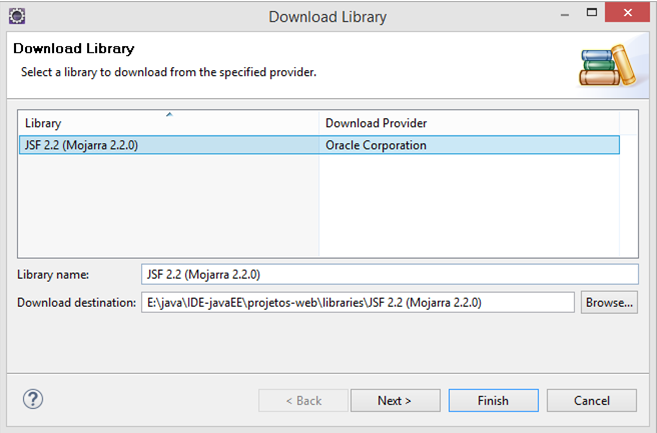
Como já foi dito, utilizaremos para o servidor de apresentação, o JSF e o Managed Bean. O JSF cuidará da parte de interface do site, e o Managed Bean cuidará de receber os dados JSF e enviar requisições ao Servidor de Serviços. Além disso, para facilitar e deixar uma interface mais agradável, usaremos a Framework JFS chamada *Primefaces*.

Seguem os passos para a configuração do servidor de apresentação.

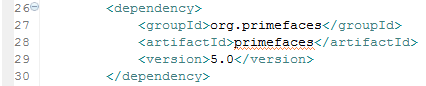
1. Selecione *New Dynamic Web Project*. Na aba em que define o nome do projeto, vá em *Configuration* e selecione a opção *JavaServer Faces* correspondente à sua versão (utilizamos a versão 2.2).
2. Siga os procedimentos padrões para criar um Dinamic Web Project.
3. Em *JSF Compabilities,* abra *Download Library.*



1. Agora, selecione*JSF 2.2 (Mojarra 2.2.0)*. Opte por *Finish*



1. Configure o *pom.xml* para baixar a dependência do *Primefaces (note que utilizamos a versão 5.0).*



Pronto, seu projeto já está configurado para utilizar essas ferramentas. Obs: Vimos apenas como fazer a configuração do JSF e a Framework Primefaces, já que o Managed Bean utiliza apenas a tecnologia Java.

Antes de explicar como acontece a comunicação entre JSF, Managed Bean e Service, é necessário esclarecer algumas informações, pois quem utiliza apenas *HTML,* pode se deparar com alguns detalhes novos.

1. O JSF utiliza arquivos com extensões *.xhtml*.
2. Cada *Input/Output* deve informar o caminho para onde deve ser redirecionado a informação.
3. As *Tag’ s* são compostas por um prefixo, o que deve ser declarado na Tag *html* da página.
   1. Declaração do prefixo *h (o JSF contém vários prefixos):*



* 1. Utilizando uma tag que necessita do prefixo *h*:



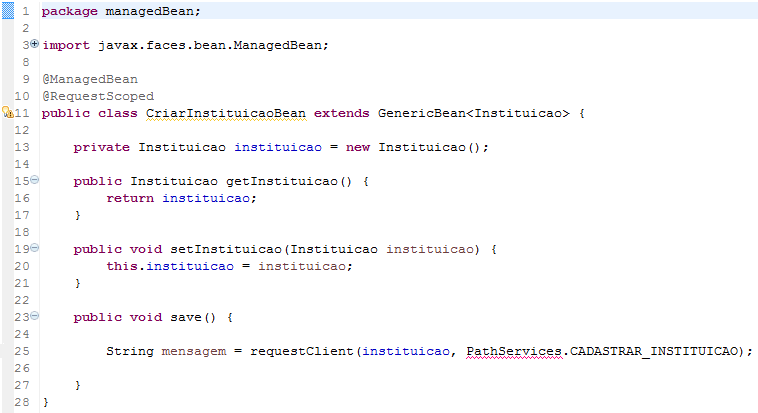
Veja um exemplo de um formulário em JSF, com a Framework PrimeFaces:



Agora, vamos explicar o que temos nesse código, separando por *Tag’s*. Vamos nos concentrar apenas nas Tag’s que possuem mais importância funcional, tag’s para deixar a página mais apresentável não é o foco.

1. No Doctype, existe algumas informações da página. Que quer dizer que está sendo usado HTML 4 e que o validador da página é o Transitional 1.0. Mas não precisa se preocupar com isso, pois essas informações são padrões, só basta copiar o colar.
2. Como já foi dito, na Tag *Html* contém a declaração dos nossos prefixos. O valores para cada prefixo também é padrão, basta saber a qual prefixo cada valor pertence (note que da Tag *body* em diante, utiliza-se prefixos em todas as tags).
3. Chegamos na parte que interessa, onde nosso JSF irá se comunicar com o Managed Bean. Perceba que na tag *<p:inputText>* contém o atributo *value.* Este valor corresponde pra onde este valor deve ser setado, e como o Bean deve se comportar.
   1. A serquilha indica que aquele valor deve obedecer a uma requisição, e dentro das chaves o seu caminho. Esse valor contém três informações separadas por ponto.
   2. O primeiro valor corresponde ao nome do Managed Bean análogo aquela página. O segundo valor corresponde a entidade que o Managed Bean contém. E o terceiro valor, é o atributo da entidade, para onde esse valor do Input está sendo setado (para isso funcionar, é necessário o get e set de todos os atributos, tanto no Managed Bean quanto na própria entidade, sejam criados).
   3. Com isso temos que: Cnpj é um atributo de Instituição. E instituição deve ser instanciada no Managed Bean.
4. A tag <p:commadButtton> utiliza a tag *actionListener* para chamar o método do Bean que fará a requisição do serviço, nesse caso, o metódo save.

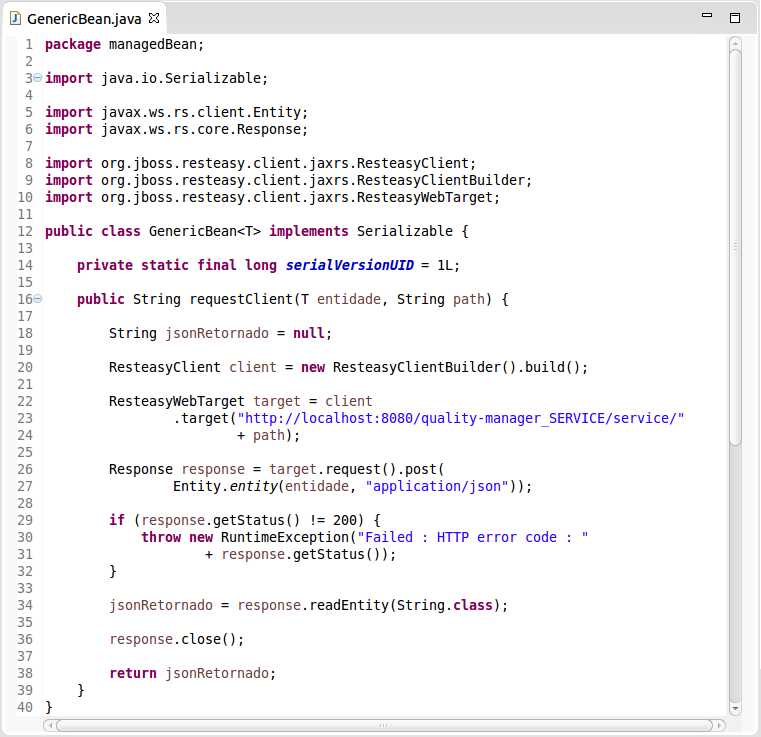
Apresentando como fica o nosso Managed Bean, e consequentemente nossa entidade instituição:



Para evitar o trabalho de ter que ficar mapeando os Managed Beans no .xml, utilizamos a anotação *@ManagedBean e @RequestScoped* antes de declaramos a classe.

O método Save chama o método requestClient, que está na classe *GenericBean*, passando por parâmetro a sua respectiva entidade e o nome do serviço.

Aqui está a classe GenericBean:



O método requestClient é capaz de receber qualquer entidade e o nome do serviço a ser solicitado. Fazendo uso da classe ResteasyClient indicamos qual recurso solicitar. Em seguida usamos esse recurso para empacotar a entidade e fazer a requisição via método POST por meio da classe ResteasyWebTarget.

Após isso, o Web Service devolve um response que tem em seu corpo um pacote JSON que é transformado em String para ser processada no save do manageBean e mostrar o status da operação para o usuário.

**2.4. Cliente Móvel Android**

Nessa parte, é importante destacar como construímos a comunicação do aplicativo QManager com o Web Service, utilizando tráfego de pacotes no formato JSON. Dessa forma, abstrairemos detalhes íntimos sobre construção de Activities, seus layouts e características não relacionadas com esse propósito.

1. Construa uma classe para resolver requisição e resposta do protocolo Http (Apêndice16):

Nesse cliente, definimos o endereço do Web Server até o caminho para os serviços e três métodos principais que retornam respostas às requisições do protocolo Http:

* + 1. Um para fazer solicitação via método GET, cujo parâmetro é o nome do serviço a ser solicitado;
    2. Outro para enviar JSON via método POST, cujo parâmetros são o nome do serviço a ser solicitado e o JSON a ser enviado;
    3. E um terceiro para enviar uma lista de parâmetros, cujo parâmetros são o nome do serviço a ser solicitado e a lista a ser enviada.

1. Construa classes que implementem AsyncTask (Apêndice 17):

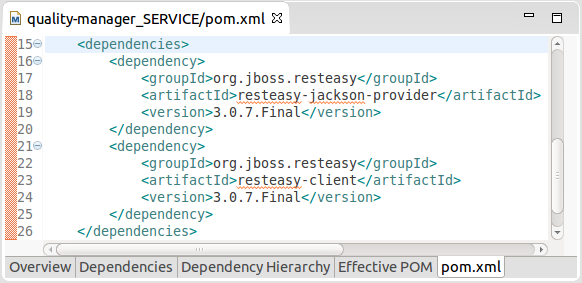
Enquanto o usuário interage com a interface gráfica da aplicação via thread de interface (UI), é interessante fazer uma nova thread para se comunicar com o servidor a fim de deixar a aplicação leve e sem travamentos indesejáveis. A API já se encarrega de tratar a concorrência de recursos através da classe AsyncTask, que tem quatro métodos principais:

* + 1. onPreExecute: Executada na UI thread, antes de executar a tarefa em background. Em resumo, é basicamente um evento visual que mostra para o usuário que algo está acontecendo;
    2. doInBackGround: Invocado em uma thread em background e recebe uma lista de parâmetros do tipo indicado na sua criação. Ao finalizar, deve retornar o objeto do tipo Resultado(JSON ou XML);
    3. onProgressUpdate: Executado na UI thread, utilizado para atualizar algum indicativo de progresso da operação. Recebe o código que está sendo executado no método doInBackground. Um exemplo desta utilização é o progresso do Download;
    4. onPostExecute: Executado na UI thread e é chamado após a execução do doInBackground, recebendo o resultado obtido no mesmo.

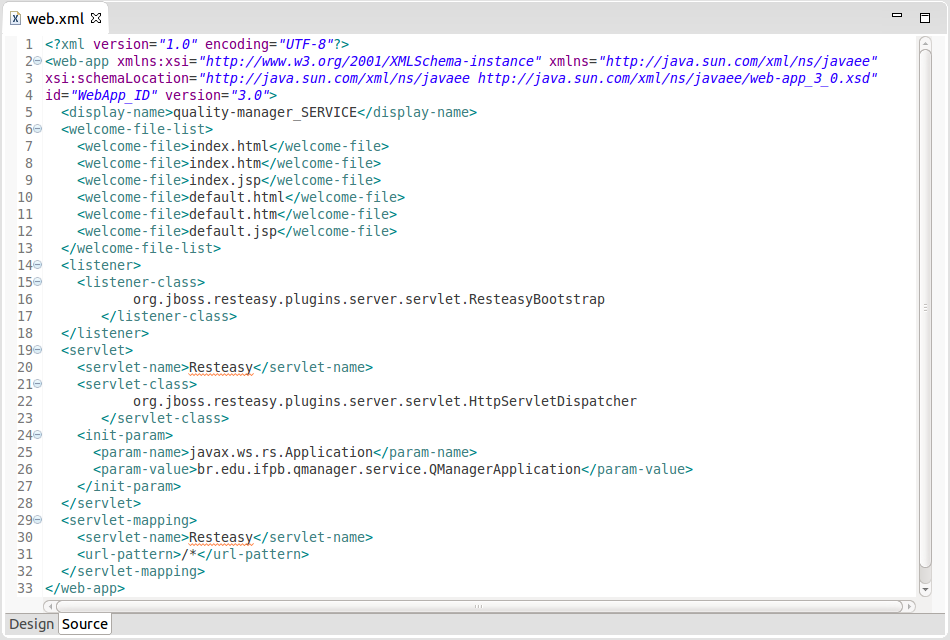
3. No doInBackGround de sua classe AsyncTask, faça o cliente chamar o servidor através dos métodos explicados em 1 (Apêndice 17).

APÊNDICES

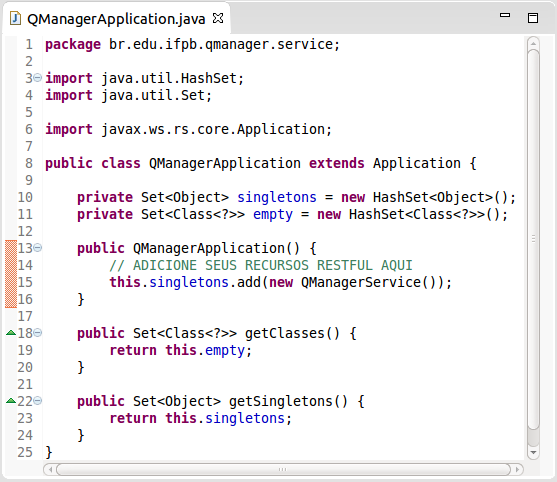
Apêndice 01: Configuração do pom.xml



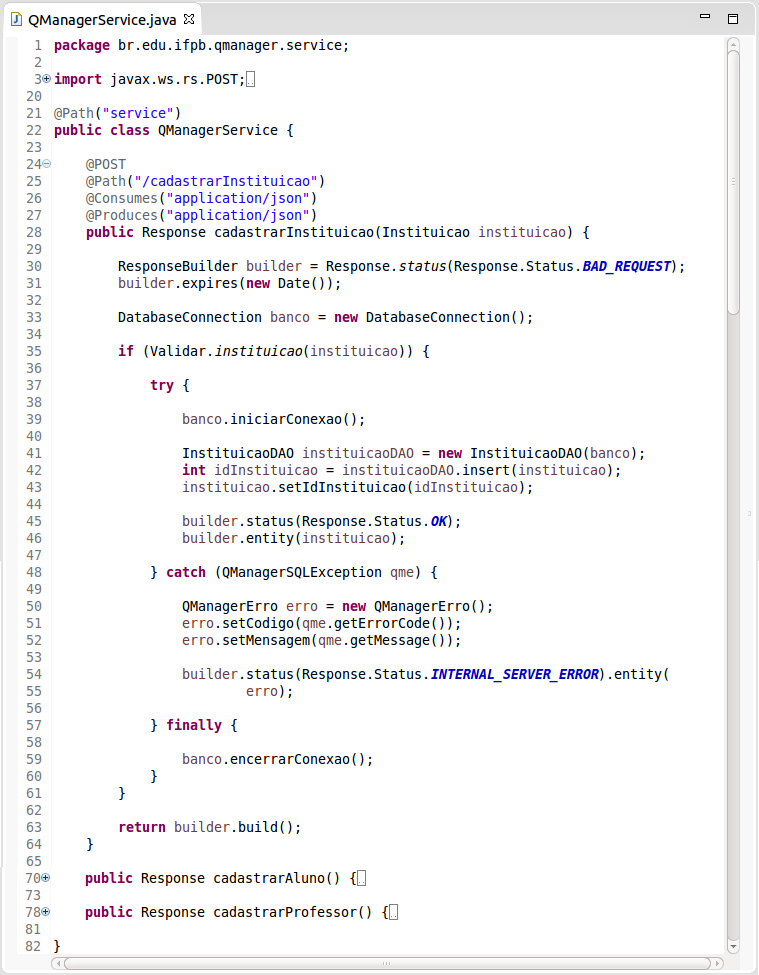
Apêndice 02: Configuração do web.xml



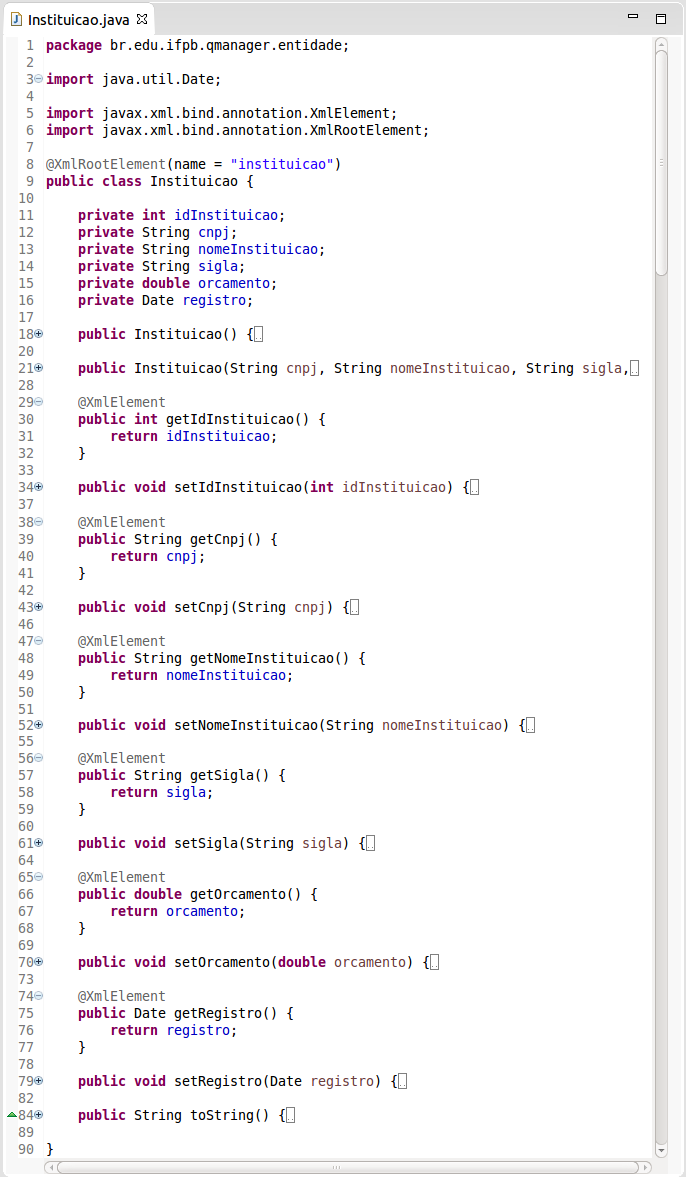
Apêndice 03: Configuração do web.xml



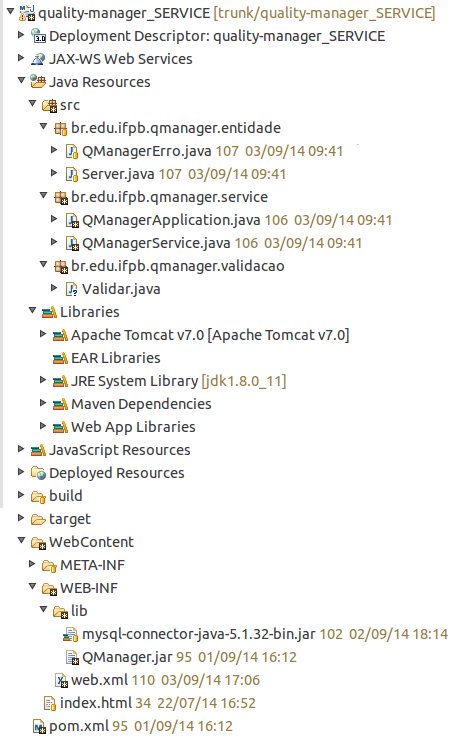
Apêndice 04: Exemplo de RESTful



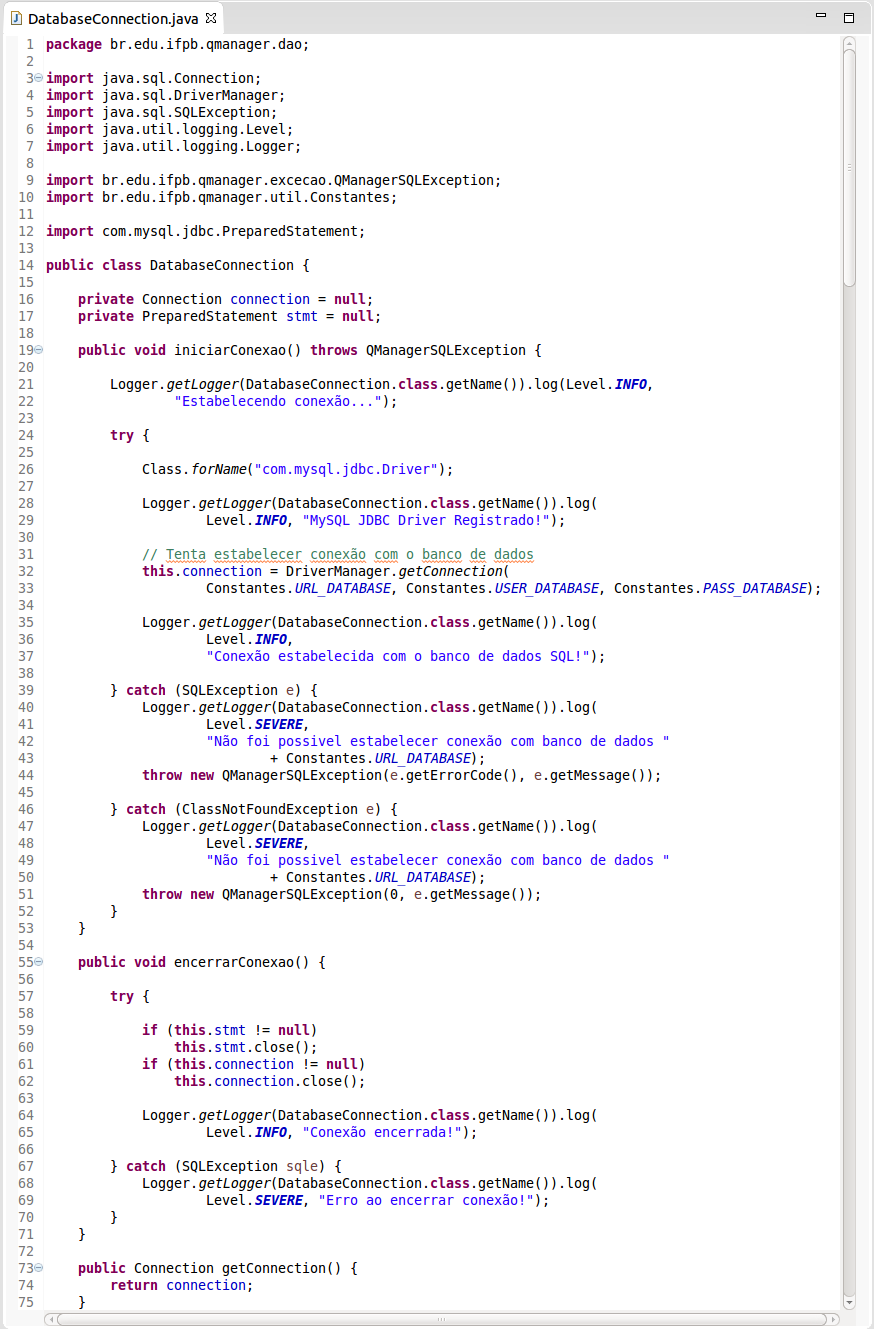
Apêndice 05: Exemplo de RESTful



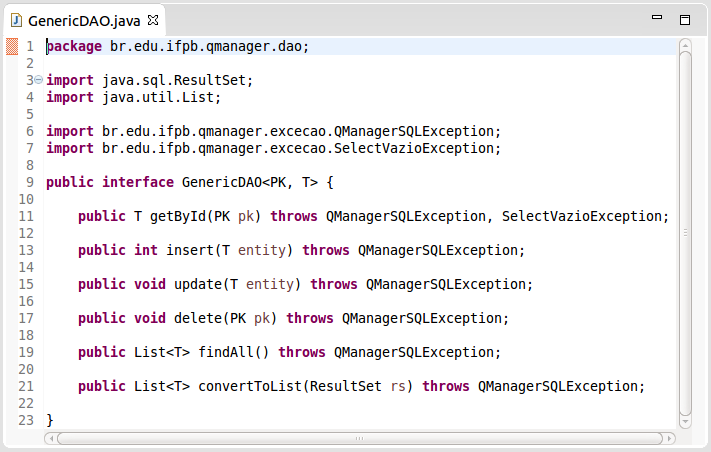
Apêndice 06: Estrutura de diretórios do quality-manager\_SERVICE



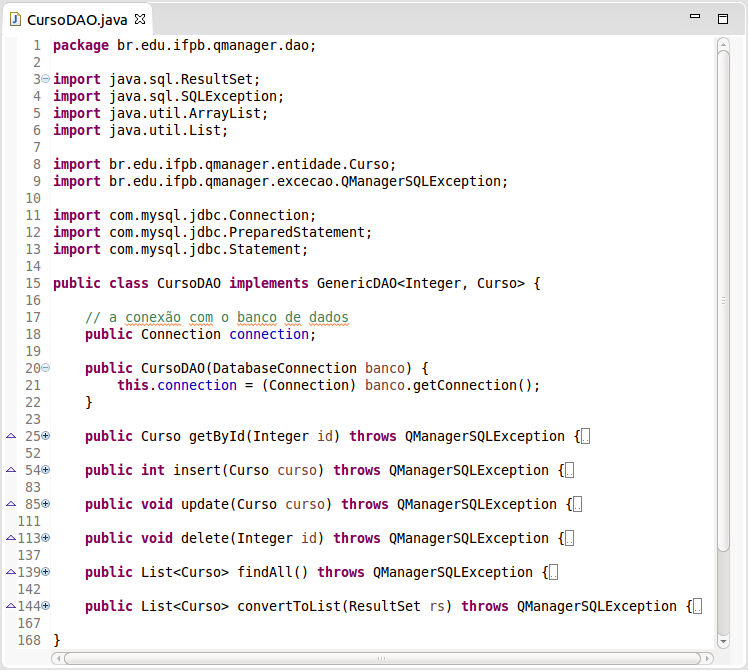
Apêndice 07: Classe de conexão com Banco de Dados.



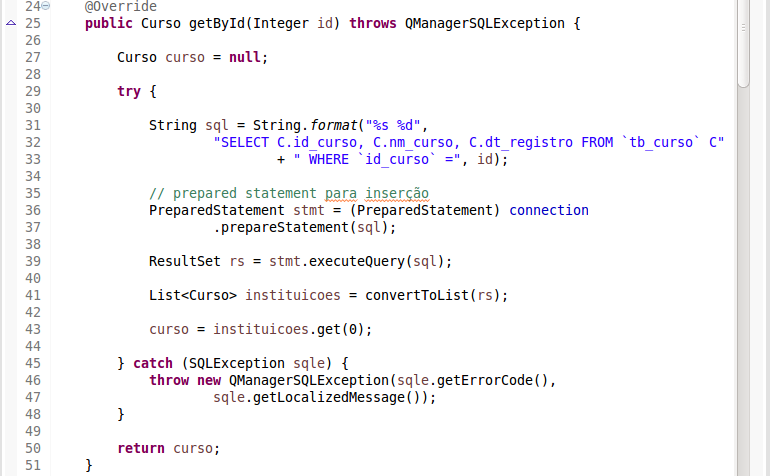
Apêndice 08: Interface GenericDAO



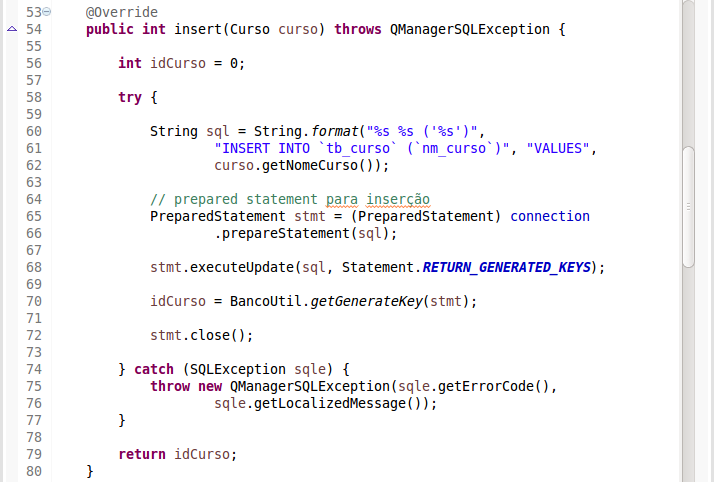
Apêndice 09: Exemplo de classe DAO



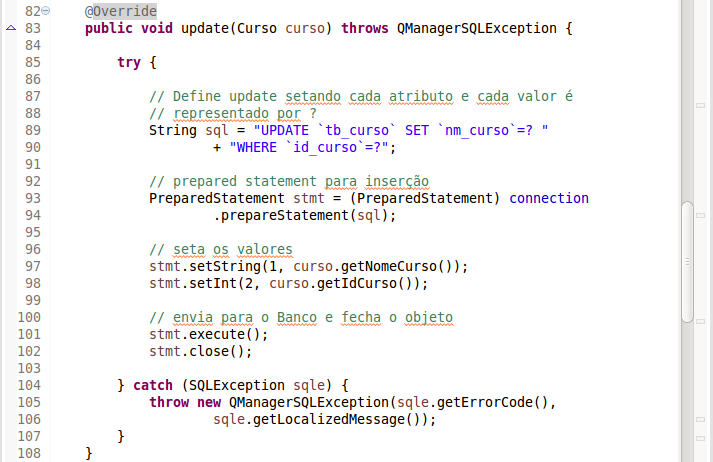
Apêndice 10: Exemplo de método T getById(PK pk)



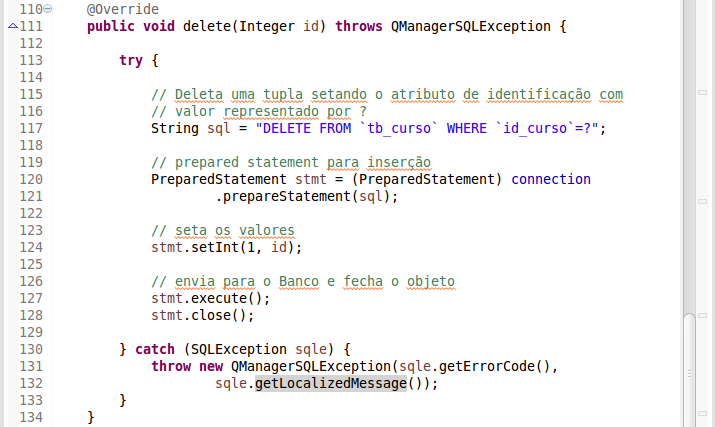
Apêndice 11: Exemplo de método int insert(T entity)



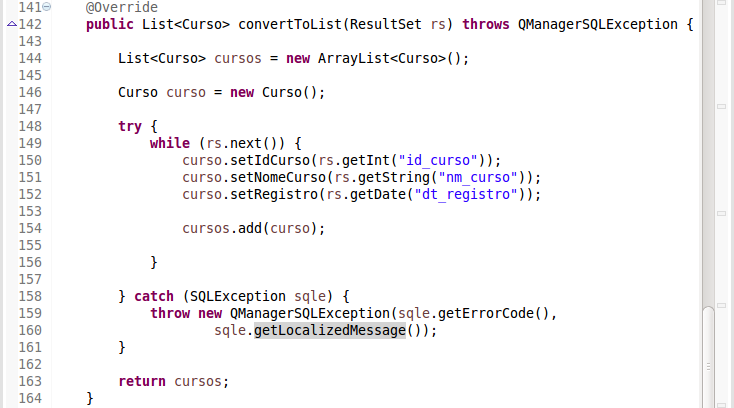
Apêndice 12: Exemplo de método void update(T entity)



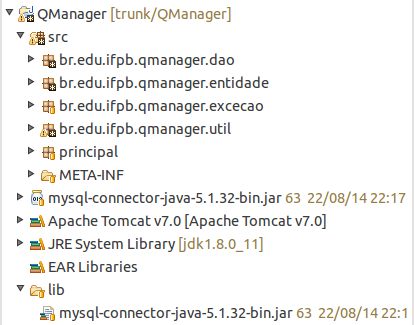
Apêndice 13: Exemplo de método void delete(T entity)



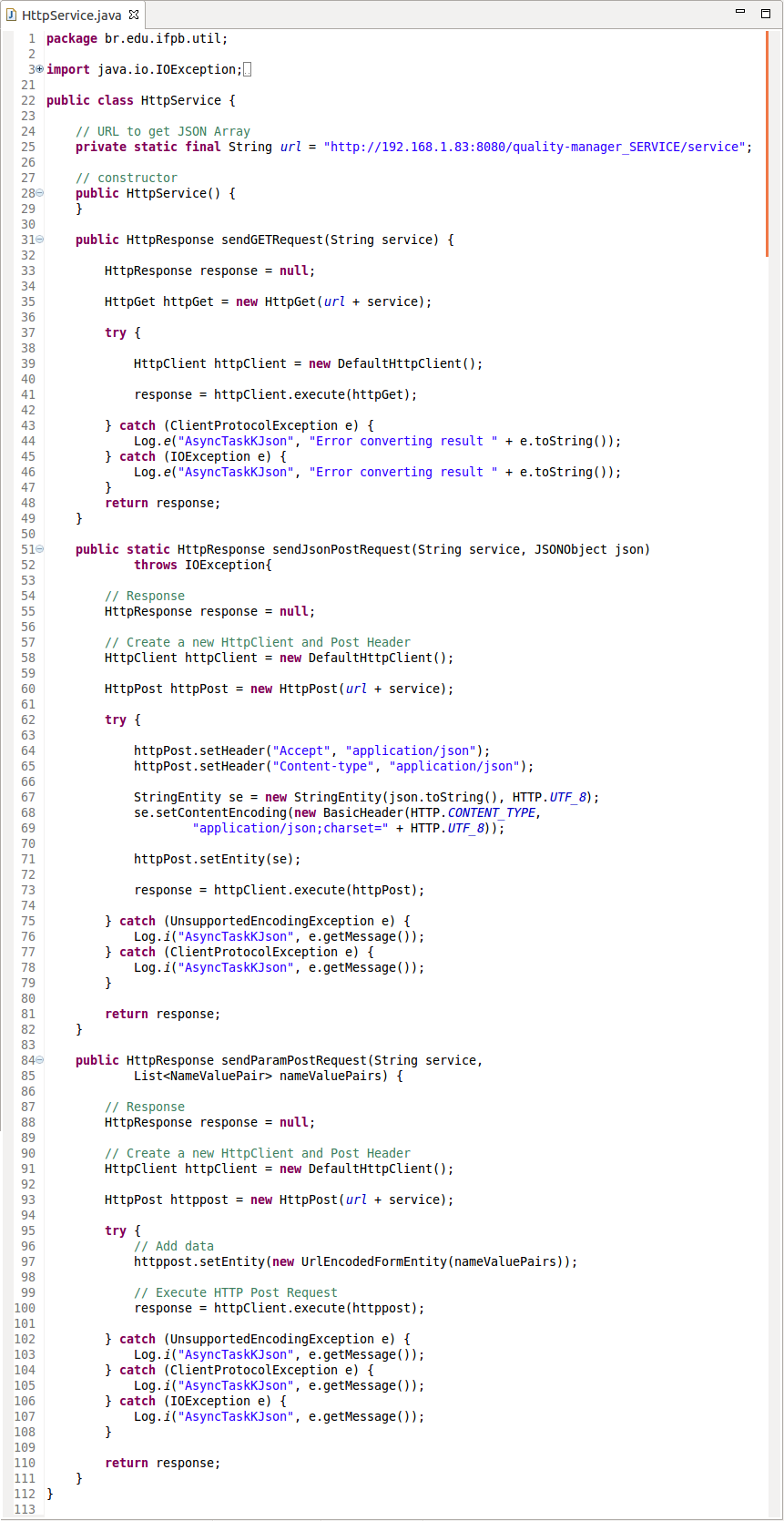
Apêndice 14: Exemplo de método List<T> convertToList(ResultSet rs)



Apêndice 15: Estrutura de diretórios do QManager

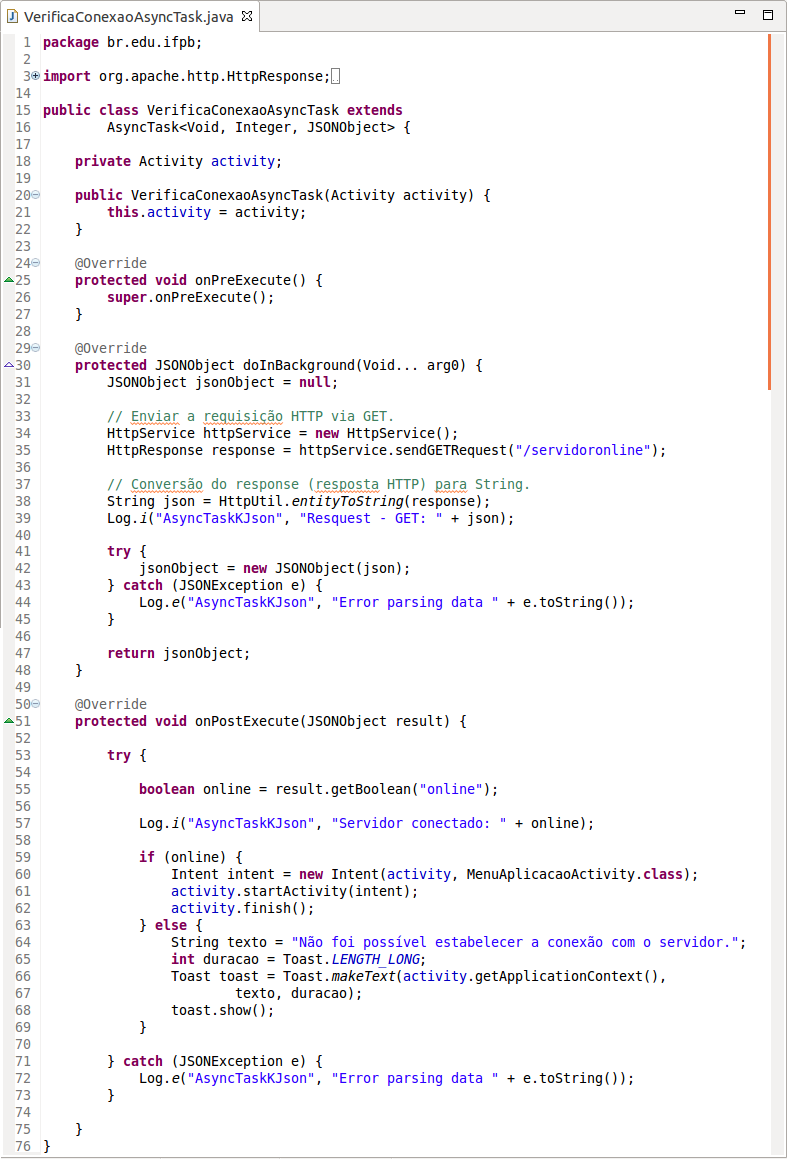


Apêndice 16: Classe que resolve requisições e respostas do cliente Android



Apêndice 17: Exemplo de classe que estende AsyncTask

REFERÊNCIAS



AMAL, Raj. **Android AsyncTask with JSON Parsing–Example**. Disponível em: <http://www.learn2crack.com/2013/10/android-asynctask-json-parsing-example.html>. Acesso em: 29 de jul. de 2014.

Google. **AsyncTask**. Disponível em: <http://developer.android.com/reference/ android/os/AsyncTask.html>. Acesso em: 29 de jul. de 2014.

GUEDES, Rhavy Maia. **Como funciona o RESTEasy**. Campina Grande: IFPB, 26 ago. 2014. Aula ministrada aos alunos do curso Técnico em Informática do IFPB - Campus Campina Grande.

MARAVITSAS, Nikos. **JSON example with RESTEasy + Jackson**. Disponível em: <http://examples.javacodegeeks.com/enterprise-java/rest/resteasy/json-example-with-resteasy-jackson/>. Acesso em: 22/08/2014.

MARCHIONI, Francesco. RESTEasy Tutorial. Disponível em: <http://www.mastertheboss.com/resteasy/resteasy-tutorial>. Acesso em: 22 de ago. de 2014.

SOUSA, [Filipe. Criando e configurando um projeto Web - JSF 2, Primefaces 3 e CDI](http://www.devmedia.com.br/autor/filipe-souza/303228) Disponível em: <http://www.devmedia.com.br/criando-e-configurando-um-projeto -web-jsf-2-primefaces-3-e-cdi/25251>. Acesso em: 02 de Set. de 2014.

LANHELLAS, [Ronaldo. Introdução ao JSF Managed Bean](http://www.devmedia.com.br/autor/ronaldo-lanhellas/244531). Disponível em: <http://www.devmedia.com.br/introducao-ao-jsf-managed-bean/29390>. Acesso em: 03 de Set. de 2014