MM - Camada de Servicos

Camada de Serviços

Bem como o objetivo da camada de visualização, a camada de serviços também tem o propósito de trazer facilidades para o desenvolvimento de aplicações, tentando amenizar os problemas ocasionados por um desenvolvimento sem padrões definidos e no modo "go horse". Foi pensando nestes objetivos que surgiu o projeto COMMONS-BUSINESS, que contempla a camada de serviços e a camada de persistência.

A camada de serviços nada mais é do que uma adaptação do padrão de Restful Objects. Ela é responsável por oferecer uma forma de comunicação entre as diferentes partes do plugin através de URI que são expostos como serviços.

O modelo de URI é baseado nas operações HTTP (POST, PUT, DELETE e GET). E tratando-se do padrão Restful a URI do serviço exposto tem uma estrutura diferente, mantendo, entretanto, sua estrutura base, conforme podemos ver a seguir:.

http://{host}:{port}/{context}/rest/shield/latest/domain/{object}/

Onde:

Variável	**Descrição**
{host}	Define onde a aplicação esta rodando
{port}	É a porta do host onde a aplicação esta rodando
{context}	É o Servlet Context da aplicacao. Por exemplo, no crowd normalmente é /crowd
{object}	É o objeto da camada de persistência. Ou seja, é a classe Java que representa para a aplicação a entidade de banco de dados

Operações suportadas

Conforme dito acima, a camada de serviços expõe serviços que podem ser acionados através de operações HTTP, são elas:

Operação	Descrição
DELETE	É a forma utilizada para remover registros identificados através da URI.
GET	É usado para buscar ou ler recursos do banco de dados. A representação desse recurso pode ser expostos em alguns formatos, bem como XML ou JSON.
POST	É utilizado geralmente para a criação de novos recursos.
PUT	É geralmente utilizado para a atualização de recursos

Leia mais: HTTP Methods

A seguir, as mesmas serão exploradas, dando exemplos utilização através de arquitetura.

GET

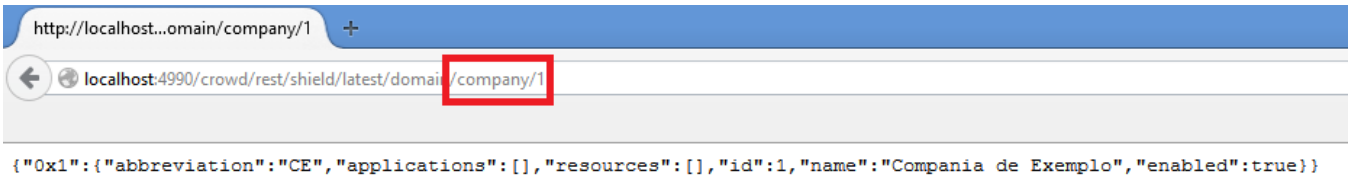
Como já dito, o método GET é utilizado para resgatar recursos de uma origem de dados. No caso da arquitetura criada, essa operação pode ser feita de duas maneiras. A primeira é passando o ID do objeto a ser buscado e a segunda é passando uma condicional para buscar uma lista de um determinado objeto.

Estas duas maneiras serão abordadas a seguir:

Passando ID:

URI: *~/{id}*

É a maneira simplificada de resgatar objetos através do serviço, onde será retornado um registro único caso o mesmo exista. Assim sendo, a forma de utilizar este serviço é através da URI base concatenando ela com o identificador único (ID) do objeto a ser resgatado. A camada de serviço interpretará a operação requisitada e realizará uma pesquisa utilizando o identificador passado retornando o referido objeto, caso ele exista. Abaixo podemos ver um exemplo:



Onde:

Parâmetro	Descrição
company	Este é o objeto representativo do banco de dados. Ou seja, este parâmetro substitui o atributo {object} da URI base da camada de serviço. Neste caso, existe uma classe "Company" que representa uma tabela do banco de dados
1	É o identificador único (ID) da entidade company no banco de dados.

Assim sendo, no exemplo acima foi requisitada uma operação do tipo GET para o serviço, tentando buscar uma "company" com ID "1". O retorno desta consulta, como podemos ver, é uma String com os atributos do classe representativa da entidade de banco de dados Company.

Usando condicional:

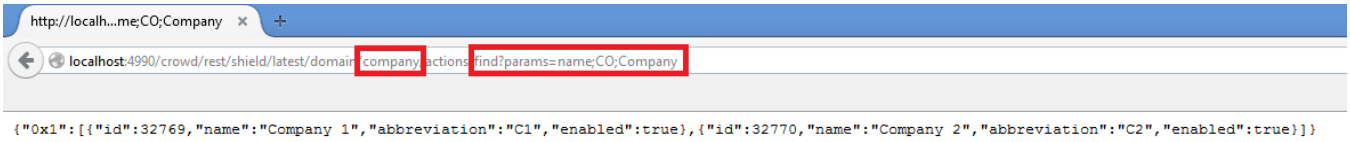
URI: *~/actions/find*

É a maneira que a camada de serviço realiza operações de busca um pouco mais complexa. Nela são utilizados operadores pré-determinados pela arquitetura conforme podemos ver na listagem abaixo:

Constante	Representação em SQL	Descrição	Exemplos	URI
EQ	=	É o operador de igualdade. Ou seja, através deste, são feitas comparações entre dois objetos dizendo se são iguais.	1 = 1: Verdadeiro 1 = 2: Falso "teste" = "teste": Verdadeiro "teste" = "teste1": Falso	URI~actions/find?params=enabled;eq;true
LTE	<=	É o operador que diz se um registro é menor ou igual a outro. Pode ser usado somente com números.	1 <= 1: Verdadeiro 1 <= 2: Verdadeiro 2 <= 1: Falso	
GTE	>=	É o operador que diz se um registro é maior ou igual a outro. Pode ser usado somente com números.	1 >= 1: Verdadeiro 1 >= 2: Falso 2 >= 1: Verdadeiro	
LT	<	É o operador que diz se um registro é menor a outro. Pode ser usado somente com números.	1 < 1: Falso 1 < 2: Verdadeiro 2 < 1: Verdadeiro	
GT	>	É o operador que diz se um registro é maior a outro. Pode ser usado somente com números.	1 > 1: Falso 1 > 2: Falso 2 > 1: Verdadeiro	
ST		Não especificado ainda		
EN		Não especificado ainda		
CO	CONTAINS	É o operador utilizado para verificar se um determinado valor esta contido num campo	CONTAINS (nome, "Joao"	URI~actions/find?params=abbreviation;co;C1
NST		Não especificado ainda		
NEN		Não especificado ainda		
NCO		Não especificado ainda		

IN	in	É a condicional que verifica se um determinado valor está em um conjunto de dados.	1 in (1,2,4,6): Verdadeiro 1 in (2,4,6,8): Falso	
IS	is			
NEQ	!=	É o inverso do operador "=". Ou seja, ele verifica se um valor é diferente de outro.	1 != 1: Falso 1 != 2: Verdadeiro "teste" != "teste": Falso "teste" != "teste1": Verdadeiro	

Esses parâmetros são enviados juntamente com a requisição HTTP com a finalidade de restringir os resultados na hora de realizar uma busca. Abaixo temos um exemplo de sua utilização:



Onde:

Parâmetro	Descrição
company	Este é o objeto representativo do banco de dados. Ou seja, este parâmetro substitui o atributo {object} da URI base da camada de serviço. Neste caso, existe uma classe "Company" que representa uma tabela do banco de dados

find?
params=name;
CO;Company
	São as condicionais utilizadas para restringir o retorno da busca. Neste caso, o operador utilizado é o "CO" (contains, em SQL) buscando no campo "name" algum registro que contenha "Company". Se convertermos isso em SQL, a instrução seria:
	
	`Select *`
	`  from company c`
	`where c.name like '%Company%'`

POST

PUT

DELETE