UNIVERSIDADE FEDERAL RURAL DO SEMIÁRIDO – UFERSA
Centro de Ciências Exatas e Naturais - CCEN
Graduação em Ciência da Computação
Disciplina: Programação para Internet
Prof.: Paulo Henrique Lopes Silva
RAUL CORREIA PINHEIRO
WESLEY ADAMO COSTA DO SANTOS

DESENVOLVIMENTO DE UM SISTEMA DE GERENCIAMENTO DE ALUGUÉIS DE CARRO COM RESTful.

1. Introdução

1.1. Objetivos

O objetivo do projeto é desenvolver uma aplicação de gerenciamento de carros alugados usando Web services tendo como base o projeto desenvolvido na segunda unidade da disciplina de Programação para Internet(UFERSA-Mossoró). Ao decorrer deste documento serão apresentadas as principais caracteristicas do projeto(implementação e funcionamento).

2. Processo de desenvolvimento

O projeto está estruturado da seguinte forma: *ListCarrosXJ (“/list-carros”)*, responsável por receber as requisições de acordo com o path acessado, sendo cada uma url com funções distintas, todas relacionadas aos carros, *ListAlugueisXJ (“/list-alugueis”)*, de forma análoga responsável somente pelas funções envolvendo os alugueis e *ListClientesXJ (“/list-clientes”)*, responsável pelos métodos envolvendo os clientes. Todos as classes mencionadas acima acessam o Dao atráves de outro método para que possa se ter uma manutenção no código mais controlada.

2.1. ListCarrosXJ

O ListCarroXJ possui os métodos de obter os carros pelo renavan o *getXml()*, que recebe como parâmetro o renavan do carro, acessado através do *"/{renavan}/renavanxml"* que tem o retorno do carro em Xml e o método *getJson()* acessado através do *“/{renavan}/renavanjson”* que retorna o carro em formato Json, ambos os métodos chamam o método *obterPorRenavan()* passando o renavan como parâmetro e é retornado o carro desejado, caso não exista o carro é feito um tratamento para retornar uma mensagem de aviso.

```java
// Obtem o carro pelo renavan em xml
@GET
@Path("/{renavan}/renavanxml")
@Produces(MediaType.APPLICATION_XML)
public Response getXml(@PathParam("renavan") String renavan) {

    Carro c = obterPorRenavan(Long.parseLong(renavan));

    if (c.getAnoFabricacao() != null)
        return Response.ok(c).header(HttpHeaders.CONTENT_TYPE, MediaType.APPLICATION_XML).build();

    return Response.ok("Carro não encontrado").header(HttpHeaders.CONTENT_TYPE, MediaType.TEXT_PLAIN).build();
}

// Obtem o carro pelo renavan em Json
@GET
@Path("/{renavan}/renavanjson")
@Produces(MediaType.APPLICATION_JSON)
public Response getJson(@PathParam("renavan") String renavan) {

    Carro c = obterPorRenavan(Long.parseLong(renavan));

    if (c.getAnoFabricacao() != null)
        return Response.ok(c).header(HttpHeaders.CONTENT_TYPE, MediaType.APPLICATION_JSON).build();

    return Response.ok("Carro não encontrado").header(HttpHeaders.CONTENT_TYPE, MediaType.TEXT_PLAIN).build();

}
```

Imagem 01: métodos getXml e getJson, respectivamente

Há também métodos de obter a lista com os carros disponíveis, *getListDispXml()*, acessado através do *“/list/disponivelxml”*, que retorna uma lista em Xml e o método *getListDispJson()*, acessado através do *“/list/disponiveljson” com* retorno de uma lista em Json, ambos os métodos chamam o método *obterListaCarrosDisponivel()* que acessa o dao e retorna uma lista com os carros

disponíveis, também foi feito um tratamento pois caso a lista tenha tamanho 0 significa que não têm carros disponíveis então é exibida uma mensagem para o usuário.

```java
// Obtem a lista de carro disponivel em xml
@GET
@Path("/list/disponivelxml")
@Produces(MediaType.APPLICATION_XML)
public Response getListDispXml() {

    // return obterListaCarrosDisponivel();
    CarrosList listCarros = new CarrosList();
    listCarros.setCarros(obterListaCarrosDisponivel());

    if (listCarros.getCarros().size() > 0)
        return Response.ok(listCarros).header(HttpHeaders.CONTENT_TYPE, MediaType.APPLICATION_XML).build();

    return Response.ok("Não existe carro disponível").header(HttpHeaders.CONTENT_TYPE, MediaType.TEXT_PLAIN)
            .build();

}

// Obtem a lista de carro disponivel em json
@GET
@Path("/list/disponiveljson")
@Produces(MediaType.APPLICATION_JSON)
public Response getListDispJson() {

    CarrosList listCarros = new CarrosList();
    listCarros.setCarros(obterListaCarrosDisponivel());

    if (listCarros.getCarros().size() > 0)
        return Response.ok(listCarros).header(HttpHeaders.CONTENT_TYPE, MediaType.APPLICATION_JSON).build();

    return Response.ok("Não existe carro disponível").header(HttpHeaders.CONTENT_TYPE, MediaType.TEXT_PLAIN)
            .build();

}
```

Imagem 02: métodos getListDispXml e getListDispJsom, respectivamente.

Já os métodos de obter os carros pelo tipo, *getListTipoXml()*, acessado através do *"/{tipo}/tipoxml"* que retorna a lista com os carros em Xml e o método *getListTipoJson()*, acessado através do *"/{tipo}/tipojson"* que retorna a lista em Json, ambos os métodos acessam o método *obterListaCarrosTipo()* passando por parâmetro o tipo, o mesmo acessa o Dao e retorna a lista completa.

```java
@GET
@Path("/{tipo}/tipoxml")
@Produces(MediaType.APPLICATION_XML)
public Response getListTipoXml(@PathParam("tipo") String tipo) {

    // return obterListaCarrosTipo(tipo);
    CarrosList listCarros = new CarrosList();
    listCarros.setCarros(obterListaCarrosTipo(tipo));

    if (listCarros.getCarros().size() > 0)
        return Response.ok(listCarros).header(HttpHeaders.CONTENT_TYPE, MediaType.APPLICATION_XML).build();

    return Response.ok("Categoria não encontrada!").header(HttpHeaders.CONTENT_TYPE, MediaType.TEXT_PLAIN).build()

}

// Obtem a lista de carro por tipo em json
@GET
@Path("/{tipo}/tipojson")
@Produces(MediaType.APPLICATION_JSON)
public /* List<Carro> */ Response getListTipoJson(@PathParam("tipo") String tipo) {

    // return obterListaCarrosTipo(tipo);
    CarrosList listCarros = new CarrosList();
    listCarros.setCarros(obterListaCarrosTipo(tipo));

    if (listCarros.getCarros().size() > 0)
        return Response.ok(listCarros).header(HttpHeaders.CONTENT_TYPE, MediaType.APPLICATION_JSON).build();

    return Response.ok("Categoria não encontrada!").header(HttpHeaders.CONTENT_TYPE, MediaType.TEXT_PLAIN).build()
```

Imagem 03: métodos getListTipoXlm e getListTipoJson, respectivamente.

Os métodos para editar um carro passando um carro em Xml ou Json, *editCarroXml()*, acessando através do *"/editcarro/xml"* retorna uma mensagem informando se foi editado ou não e o *editCarroJson()*, acessado através do *"/editcarro/json"*, ambos acessam o método *editaCarro()* passando por parâmetro o objeto carro construído após receber um carro em xml ou json.

```java
@PUT
@Path("/editcarro/xml")
@Consumes(MediaType.APPLICATION_XML)
public String editCarroXml(Carro carro) {

    boolean ok = editaCarro(carro);

    if (ok)
        return "Carro alterado.";
    else
        return "Algum erro ocorreu";
}
```

```java
@PUT
@Path("/editcarro/json")
@Consumes(MediaType.APPLICATION_JSON)
public String editCarroJson(Carro carro) {

    boolean ok = editaCarro(carro);

    if (ok)
        return "Carro alterado.";
    else
        return "Algum erro ocorreu";
}
```

Imagem 04: métodos editCarroXml e editCarroJson.

Possui também os métodos de deletar um carro passando o renavan, *deleteCarroXml()*, acessando através do *"/{renavan}/deletcarroxml"* que retorna em Xml o carro deletado e *deleteCarroJson()*, acessado através do *"/{renavan}/deletcarrojson"* que retorna em Json o carro deletado, em ambos se não houver o carro com o renavan do parâmetro é informado que não existe o carro, isto é feito através do método *deletaCarro() passando como parâmetro o renavan.*

```java
@DELETE
@Path("/{renavan}/deletcarroxml")
@Produces(MediaType.APPLICATION_XML)
public Response deleteCarroXml(@PathParam("renavan") long renavan) {

    Carro carro = deletaCarro(renavan);

    if (carro.getAnoFabricacao() != null)
        return Response.ok(carro).header(HttpHeaders.CONTENT_TYPE, MediaType.APPLICATION_XML).build();

    return Response.ok("Carro não encontrado").header(HttpHeaders.CONTENT_TYPE, MediaType.TEXT_PLAIN).build();

}

// Deleta o carro por id em em json
@DELETE
@Path("/{renavan}/deletcarrojson")
@Produces(MediaType.TEXT_PLAIN)
public Response deleteCarroJson(@PathParam("renavan") long renavan) {

    Carro carro = deletaCarro(renavan);

    if (carro.getAnoFabricacao() != null)
        return Response.ok(carro).header(HttpHeaders.CONTENT_TYPE, MediaType.APPLICATION_JSON).build();

    return Response.ok("Carro não encontrado").header(HttpHeaders.CONTENT_TYPE, MediaType.TEXT_PLAIN).build();
```

Imagem 05: métodos deletCarroXml e deletCarroJson.

2.2. ListAlugueisXJ

Possui os métodos de obter a lista com os alugueis, *getListAluXml()*, acessado através do *"/list/allalugueisxml"* que retorna uma lista em Xml e o método *getListAluJson()*, acessado através do *"/list/allalugueisjson"* que retorna uma lista em Json, ambos os métodos chamam o método *obterTodosAlugueis()* que acessa o dao e retorna uma lista com os alugueis feitos, também e feito um tratamento pois caso a lista tenha tamanho 0 significa que não têm alugueis ,então é exibida uma mensagem para o usuário informando que não há reserva cadastrada.

```java
@GET
@Path("/list/allalugueisxml")
@Produces(MediaType.APPLICATION_XML)
public Response getListAluXml() {

    AlugueisList alugueis = new AlugueisList();
    alugueis.setAlugueis(obterTodosAlugueis());
    if (alugueis.getAlugueis().size() > 0) {
        return Response.ok(alugueis).header(HttpHeaders.CONTENT_TYPE, MediaType.APPLICATION_XML).build();
    }

    return Response.ok("Não há reserva cadastrada").header(HttpHeaders.CONTENT_TYPE, MediaType.TEXT_PLAIN).build();
    }

// Obtem a lista de alugueis em json
@GET
@Path("/list/allalugueisjson")
@Produces(MediaType.APPLICATION_JSON)
public Response getListAluJson() {

    AlugueisList alugueis = new AlugueisList();
    alugueis.setAlugueis(obterTodosAlugueis());
    if (alugueis.getAlugueis().size() > 0) {
        return Response.ok(alugueis).header(HttpHeaders.CONTENT_TYPE, MediaType.APPLICATION_JSON).build();
    }

    return Response.ok("Não há reserva cadastrada").header(HttpHeaders.CONTENT_TYPE, MediaType.TEXT_PLAIN).build();
```

Imagem 06: métodos getListAluXml e getListAluJson.

Possui também os métodos de obter aluguel pelo id, *getAluguelIdXml()*, acessado através do *"/{id}/allalugueisxml"* que retorna o aluguel em Xml e o método *getAluguelIdJson()*, acessado através do *"/{id}/allalugueisjson"* que retorna o aluguel em Json, ambos os métodos acessam o método *obterAluguelId()* passando por parâmetro o id, o mesmo acessa o Dao e retorna o aluguel selecionado.

```java
@GET
@Path("/{id}/allalugueisxml")
@Produces(MediaType.APPLICATION_XML)
public Response getAluguelIdXml(@PathParam("id") int id) { // Verificar se pode receber int ou tem que ser String e
                                                           // transformar em int

    Aluguel aluguel = obterAluguelId(id);

    if (aluguel.getCpfCliente() != null)
        return Response.ok(aluguel).header(HttpHeaders.CONTENT_TYPE, MediaType.APPLICATION_XML).build();
    return Response.ok("Nao ha aluguel cadastrado nesse ID").header(HttpHeaders.CONTENT_TYPE, MediaType.TEXT_PLAIN)
            .build();

}

// Obtem o aluguel por id em json
@GET
@Path("/{id}/allalugueisjson")
@Produces({ MediaType.APPLICATION_JSON, MediaType.TEXT_PLAIN })
public Response getAluguelIdJson(@PathParam("id") int id) {

    Aluguel aluguel = obterAluguelId(id);

    if (aluguel.getCpfCliente() != null)
        return Response.ok(aluguel).header(HttpHeaders.CONTENT_TYPE, MediaType.APPLICATION_JSON).build();
    return Response.ok("Nao ha aluguel cadastrado nesse ID").header(HttpHeaders.CONTENT_TYPE, MediaType.TEXT_PLAIN)
            .build();

}
```

Imagem 07: métodos getAluguelIdXml e getAluguelIdJson.

2.3. ListClientesXJ

Métodos de obter a lista com os clientes, *getAllClientesXml()*, acessado através do *"/allclientesxml"* retorna uma lista em Xml e o método *getAllClientesJson()*, acessado através do *"/allclientesjson"* retorna uma lista em Json, ambos os métodos chamam o método *obterTodosClientes()* que acessa o dao e retorna uma lista com os clientes, também é feito um tratamento

pois caso a lista tenha tamanho 0 significa que não têm clientes ,então é exibida uma mensagem para o usuário.

```java
@GET
@Path("/allclientesxml")
@Produces(MediaType.APPLICATION_XML)
public Response getAllClientesXml() {

    ClientesList clientes = new ClientesList();

    clientes.setClientesLista(obterTodosClientes());

    if (clientes.getClientesLista().size() > 0)
        return Response.ok(clientes).header(HttpHeaders.CONTENT_TYPE, MediaType.APPLICATION_XML).build();
    return Response.ok("Nao ha clientes cadastrados").header(HttpHeaders.CONTENT_TYPE, MediaType.TEXT_PLAIN)
            .build();

}

// Obtem todos os clientes em Json
@GET
@Path("/allclientesjson")
@Produces(MediaType.APPLICATION_JSON)
public Response getAllClientesJson() {

    ClientesList clientes = new ClientesList();

    clientes.setClientesLista(obterTodosClientes());

    if (clientes.getClientesLista().size() > 0)
        return Response.ok(clientes).header(HttpHeaders.CONTENT_TYPE, MediaType.APPLICATION_JSON).build();
    return Response.ok("Nao ha clientes cadastrados").header(HttpHeaders.CONTENT_TYPE, MediaType.TEXT_PLAIN)
```

Imagem 08: métodos getAllClientesXml e getAllClientesJson.

Métodos de obter o cliente pelo id, *getClienteIdXml()*, acessado através do *"/{id}/clienteidxml"* retorna o cliente em Xml e o método *getClienteIdJson()*, acessado através do *"/{id}/clienteidjson"* retorna o cliente em Json, ambos os métodos acessam o método *obterClienteId()* passando por parâmetro o id, o mesmo acessa o Dao e retorna o cliente selecionado.

```java
// Obtem cliente por id em xml
@GET
@Path("/{id}/clienteidxml")
@Produces(MediaType.APPLICATION_XML)
public Response getClienteIdXml(@PathParam("id") int id) {

    Cliente clientes = obterClienteId(id);

    if (clientes.getCpf() != null)
        return Response.ok(clientes).header(HttpHeaders.CONTENT_TYPE, MediaType.APPLICATION_XML).build();
    return Response.ok("Nao ha cliente cadastrado com esse ID")
            .header(HttpHeaders.CONTENT_TYPE, MediaType.TEXT_PLAIN).build();

}

// Obtem cliente por id em Json
@GET
@Path("/{id}/clienteidjson")
@Produces(MediaType.APPLICATION_JSON)
public Response getClienteIdJson(@PathParam("id") int id) {

    Cliente clientes = obterClienteId(id);

    if (clientes.getCpf() != null)
        return Response.ok(clientes).header(HttpHeaders.CONTENT_TYPE, MediaType.APPLICATION_JSON).build();
    return Response.ok("Nao ha cliente cadastrado com esse ID")
            .header(HttpHeaders.CONTENT_TYPE, MediaType.TEXT_PLAIN).build();
```

Imagem 10: métodos getClientesIdXml e getClientesIdJson.

3. Conclusão

3.1. Vantagens e desvantagens

As vantagens da utilização de Web Services no estilo Rest é a simplicidade no desenvolvimento do sistema. Foi possível perceber que o fluxo do funcionamento do sistema é mais fácil de entender e também houve uma diminuição no tamanho do sistema como um todo.

Como a parte da lógica do funcionamento da locadora de carros já estava feito, a transformação das informações da locadora para o modelo proposto foi relativamente fácil, o que não trouxe dificuldade nas alterações. Dessa forma, não foi possível identificar possíveis desvantagens na utilização de Web Services.

3.2 Lições aprendidas

Como mencionado, a simplicidade da utilização do padrão produces/consumes foi uma das lições aprendidas em relação as demais tecnologias já utilizadas nas unidades passadas. A utilização dos métodos POST, GET, DELETE e PUT nos deu um entendimento melhor dos métodos do HTTP.