Jax-RS(JSR-311)

Pre-Requisitos

- Instalacao do Wildfly - Servidor de Aplicacao
  Download - Application Server Distribution
- Instalacao do Postman - Ferramenta para comunicacao com servicos Rest
  Download

Instalacao do Wildfly

- Na aba servers
  - Clicar no link (No servers are available. Click this link to create a new server...)
  - Em Select the server type, digite: wildfly
  -

O que sao RESTFul Web Services?

- Representational State Transfer(REST),
- E basicamente uma arquitetura baseada em servicos web
- Em uma arquitetura rest tudo eh um recurso
- Sao leves, altamente escalaveis e de facil manutencao
- Sao comumente usados para criacao de APIs para aplicacoes web

Vamos criar nosso primeiro exemplo

import javax.ws.rs.GET;

import javax.ws.rs.Produces;

import javax.ws.rs.Path;

```
@ApplicationPath("/")
public class Resource extends Application {

}
```

```java
// The Java class will be hosted at the URI path "/helloworld"
@Path("/helloworld")
public class HelloWorldResource extends Resource {

    // The Java method will process HTTP GET requests
    @GET
    // The Java method will produce content identified by the MIME Media
    // type "text/plain"
    @Produces("text/plain")
    public String getClichedMessage() {
        // Return some cliched textual content
        return "Hello World";
    }
}
```

Verbos mais utilizados

GET - sao apenas para leitura.

PUT and DELETE, operacoes sao "idempotent" (o resultado sempre sera o mesmo nao importa quantas vezes serao chamados)

PUT and POST, somente com a difereca que o PUT e "idempotent" e a operacao POST causa um novo resultado.

Quando utilizar

GET - geralmente usado para ler dados

POST - geralmente usado para inserir novos dados

PUT - geralmente usado para atualizacao de dados

DELETE - geralmente usado para a remocao de dados

Exemplos

Criar a classe

```
@Path("/book")
@Produces({MediaType.APPLICATION_XML, MediaType.APPLICATI
@Consumes({MediaType.APPLICATION_XML, MediaType.APPLICATI
public class BookRestService {
   //Proximos metodos abaixo
}
```

```java
@POST
public Response createBook(Book book) {
  return Response.ok().build();
}

@PUT
public Response updateBook(Book book) {
  if (book == null)
    throw new BadRequestException();
  return Response.ok().build();
}

@GET
@Path("{id}")
public Response getBook(@PathParam("id") String id) {
  if (book == null)
    throw new NotFoundException();
  return Response.ok(book).build();
}
...
```

```
    ...

    @DELETE
    @Path("{id}")
    public Response deleteBook(@PathParam("id") String id) {

    return Response.noContent().build();
    }

    @GET
    public Response getAllBooks() {
          //Arrays.asList(...books)
          return Response.ok(books).build();
    }
}
```

Exercicio

- Criar uma classe chamada BookDAO
- Simular um banco estatico