rip

v. 0.0.3

Project Documentation

Technearts

2018-04-11

Table Of Content

1 Read Me!

RIP - Rest in peace

Gerador de serviços mockados estupidamente simples

1.1 Objetivos

1. Gerar um serviço rest mockado sem preocupações de infraestrutura
2. Serviços simples, sem qualquer inteligência
3. Possibilidade de respostas distintas de acordo com a presença de conteúdo na requisição
4. Respondendo os verbos mais comuns (GET, POST, PUT, DELETE), outros podem ser implementados se necessário

O objetivo principal é criar um endpoint rest local que responda sempre o mesmo conteúdo dado que o corpo da mensagem contenha determinada string.

1.2 Utilização

O servidor e os serviços são criados programaticamente, para serem incluídos em um teste unitário, uma aplicação web ou um servidor remoto. O servidor responde em localhost, em uma porta definida.

1.2.1.1 Exemplos

Cria o servidor na porta padrão (7777) respondendo à `GET /test` e, se *xpto* estiver no body da requisição, responde *Hello World*:

```
localhost().get("/test").contains("xpto").respond("Hello World");
```

Cria o servidor na porta 8888 respondendo à `POST /test` e, se *xpto* e *1234* estiverem no body, responde com o conteúdo do arquivo `/test/hello.json`

```
localhost(8888).post("/test").contains("xpto").and().contains("1234").respond(withF
```

Responde à `POST /test` de acordo com o conteúdo do body, caindo no caso default (último) se não encontrar nenhuma correspondência

```
localhost().post("/test").contains("test1").respond(withFile("/test/1.json"));
localhost().post("/test").contains("test2").respond(withFile("/test/2.json"));
localhost().post("/test").respond(withFile("File not found"));
```

1.3 Como criar os serviços mockados em uma aplicação dentro de um servlet container qualquer (Ex. tomcat, liberty, jboss)

Para testes unitários basta a chamada aos métodos de criação de rotas, como acima, preferencialmente em um método `@BeforeClass`. Se for necessário executar testes em uma aplicação web publicada, os serviços podem ser iniciados na mesma aplicação com a utilização de um filtro.

A sintaxe de criação de rotas não é alterada, mas naturalmente a porta não deve ser utilizada.

1.3.1 Acrescentar a dependencia no pom

```
        <!-- Mocks -->
        <dependency>
            <groupId>net.technearts</groupId>
            <artifactId>rip</artifactId>
            <version>0.0.3</version>
        </dependency>
```

1.3.2 Criar uma classe extendendo RipWebApp

```
package rest.mock;

import static net.technearts.rip.RipServer.localhost;
import static net.technearts.rip.RipServer.withFile;

import net.technearts.rip.RipWebApp;

public class RestMock extends RipWebApp {

    @Override
    public void setup() {
        localhost().post("/test").contains("0123456789")
                .respond(withFile("/files/test1.json"));
        localhost().post("/test")
                .respond(withFile("/files/test2.json"));
    }
}
```

1.3.3 Acrescentar o filtro no web.xml

```
    <filter>
        <filter-name>RipFilter</filter-name>
        <filter-class>net.technearts.rip.RipWebFilter</filter-class>
        <init-param>
            <param-name>applicationClass</param-name>
            <param-value>rest.mock.RestMock</param-value>
        </init-param>
    </filter>
    <filter-mapping>
        <filter-name>RipFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
```

Obs.: o serviço irá responder na mesma porta e contexto da aplicação!

1.3.4 Acrescentar os arquivos com as respostas ao path. No maven, sob /src/main/resources, por exemplo:

- /files/test1.json
- /files/test2.json