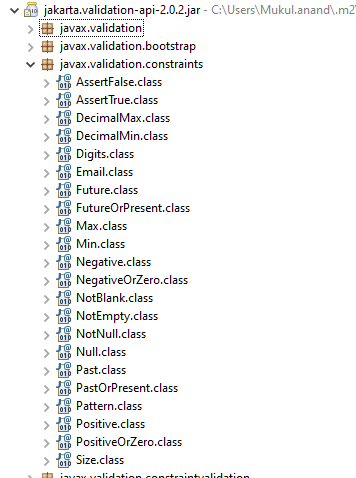
#To set log level

logging.level.org.springframework = debug

* It will show AutoConfig Report which tells about all Auto configuration done by spring boot like DispatcherServelet etc.
* Spring Boot Auto Config will check all the class present in class path and Configure.
* DispatcherServlet handles all incoming requests.
* DispatcherServlet follows front controller pattern. It is front controller for Spring MVC/Boot projects.
* All the validation is present in Jakarta-validation-api.jar



HATEOAS

Hypermedia—better known as Hypermedia as the Engine of Application State (**HATEOAS**)—is one of the main constraints of Representational State Transfer (**REST**). The idea is that hypermedia artifacts, such as links or forms, can be used to describe how clients can interact with a set of HTTP services.

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-hateoas</artifactId>

</dependency>

@GetMapping("/users/{id}")

**public** EntityModel<User> retrieveUser(@PathVariable **int** id) {

User user = service.findOne(id);

**if**(user==**null**)

**throw** **new** UserNotFoundException("id-"+ id);

//HATEOAS

//"all-users", SERVER\_PATH + "/users"

//retrieveAllUsers

EntityModel<User> resource = EntityModel.*of*(user);

WebMvcLinkBuilder linkTo =

*linkTo*(*methodOn*(**this**.getClass()).retrieveAllUsers());

resource.add(linkTo.withRel("all-users"));

//We can add other links as well.

**return** resource;

}

<http://localhost:8080/users/1>

{

"id": 1,

"name": "Adam",

"birthDate": "2021-07-02T10:25:28.316+00:00",

"\_links": {

"all-users": {

"href": "http://localhost:8080/users"

}

}

}

Internationalization for RESTful Services

@GetMapping(path = "/hello-world-internationalized")

**public** String helloWorldInternationalized(

//@RequestHeader(name="Accept-Language", required = false) Locale locale

) {

**return** messageSource.getMessage("good.morning.message", **null**, "Default Message", LocaleContextHolder.*getLocale*());

}

messages.properties

good.morning.message=Good Morning

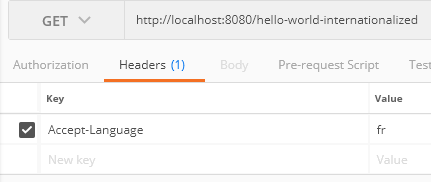
messages\_fr.properties

good.morning.message=Bonjour

messages\_nl.properties

good.morning.message=Goede Morgen

Header



<http://localhost:8080/hello-world-internationalized>

Bonjour

We can use parameter.

@RequestHeader(name="Accept-Language", required = false) Locale locale

Or

LocaleContextHolder.*getLocale*()

Content Negotiation - Implementing Support for XML

<dependency>

<groupId>com.fasterxml.jackson.dataformat</groupId>

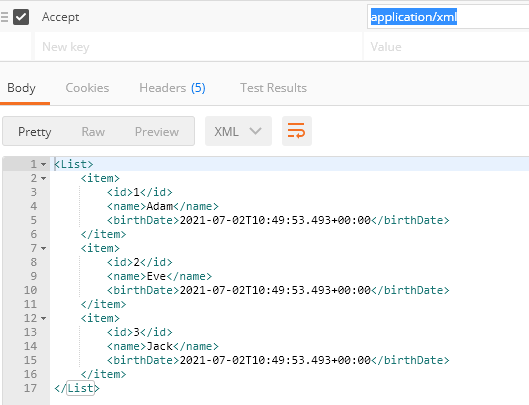
<artifactId>jackson-dataformat-xml</artifactId>

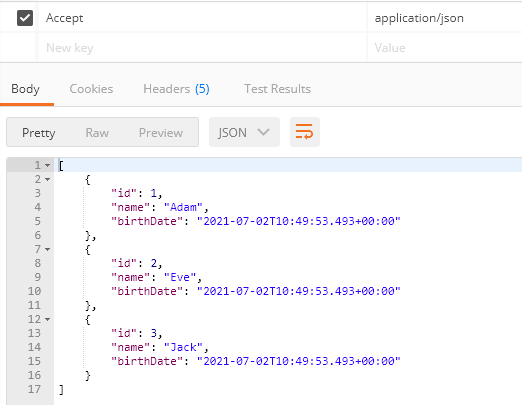
</dependency>

<http://localhost:8080/users>

Set Header

Accept application/xml





Configuring Auto Generation of Swagger Documentation

**OpenAPI definition Swagger implementation version 3.0.1**

Just add jar and hit swagger URL –

<dependency>

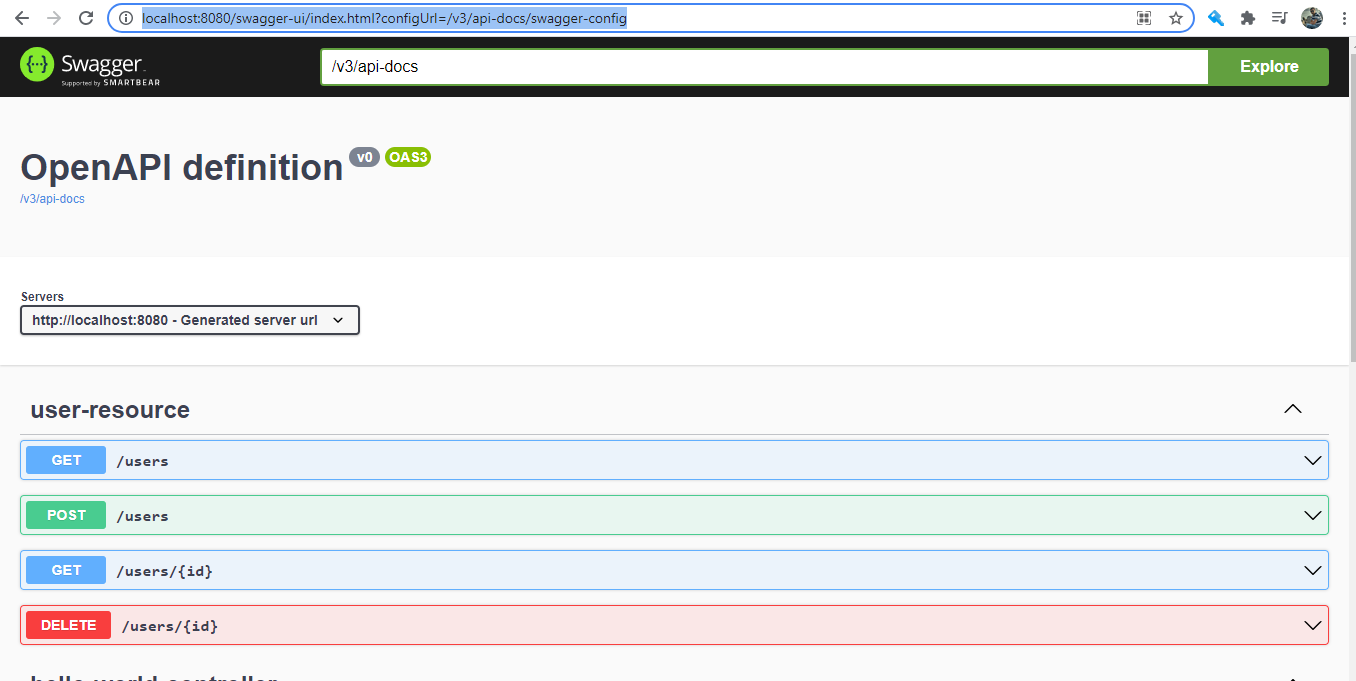
<groupId>org.springdoc</groupId>

<artifactId>springdoc-openapi-ui</artifactId>

<version>1.5.9</version>

</dependency>

<http://localhost:8080/swagger-ui/index.html?configUrl=/v3/api-docs/swagger-config>



<http://localhost:8080/v3/api-docs>



Monitoring APIs with Spring Boot Actuator

To check health, to monitor application, to get metrics etc we use actuator.

Spring Boot includes a number of additional features to help you monitor and manage your application when you push it to production. You can choose to manage and monitor your application by using HTTP endpoints or with JMX. Auditing, health, and metrics gathering can also be automatically applied to your application.

Actuator endpoints let you monitor and interact with your application. Spring Boot includes a number of built-in endpoints and lets you add your own. For example, the health endpoint provides basic application health information.

Each individual endpoint can be enabled or disabled and exposed (made remotely accessible) over HTTP or JMX. An endpoint is considered to be available when it is both enabled and exposed. The built-in endpoints will only be auto-configured when they are available. Most applications choose exposure via HTTP, where the ID of the endpoint along with a prefix of /actuator is mapped to a URL. For example, by default, the health endpoint is mapped to /actuator/health.

<http://localhost:8080/actuator>

By Default only 3 URLs



Add below to access all actuator URL in application.properties

#To access all actuator URL

management.endpoints.web.exposure.include=\*

#To exclude any URL

management.endpoints.web.exposure.exclude=health, metrics

{"\_links": {"self": {"href": "<http://localhost:8080/actuator>","templated": false},"beans": {"href": "<http://localhost:8080/actuator/beans>","templated": false},"caches-cache": {"href": "[http://localhost:8080/actuator/caches/{cache}](http://localhost:8080/actuator/caches/%7Bcache%7D)","templated": true},"caches": {"href": "<http://localhost:8080/actuator/caches>","templated": false},"health-path": {"href": "[http://localhost:8080/actuator/health/{\*path}](http://localhost:8080/actuator/health/%7B*path%7D)","templated": true},"health": {"href": "<http://localhost:8080/actuator/health>","templated": false},"info": {"href": "<http://localhost:8080/actuator/info>","templated": false},"conditions": {"href": "<http://localhost:8080/actuator/conditions>","templated": false},"configprops": {"href": "<http://localhost:8080/actuator/configprops>","templated": false},"configprops-prefix": {"href": "[http://localhost:8080/actuator/configprops/{prefix}](http://localhost:8080/actuator/configprops/%7Bprefix%7D)","templated": true},"env": {"href": "<http://localhost:8080/actuator/env>","templated": false},"env-toMatch": {"href": "[http://localhost:8080/actuator/env/{toMatch}](http://localhost:8080/actuator/env/%7BtoMatch%7D)","templated": true},"loggers": {"href": "<http://localhost:8080/actuator/loggers>","templated": false},"loggers-name": {"href": "[http://localhost:8080/actuator/loggers/{name}](http://localhost:8080/actuator/loggers/%7Bname%7D)","templated": true},"heapdump": {"href": "<http://localhost:8080/actuator/heapdump>","templated": false},"threaddump": {"href": "<http://localhost:8080/actuator/threaddump>","templated": false},"metrics-requiredMetricName": {"href": "[http://localhost:8080/actuator/metrics/{requiredMetricName}](http://localhost:8080/actuator/metrics/%7BrequiredMetricName%7D)","templated": true},"metrics": {"href": "<http://localhost:8080/actuator/metrics>","templated": false},"scheduledtasks": {"href": "<http://localhost:8080/actuator/scheduledtasks>","templated": false},"mappings": {"href": "<http://localhost:8080/actuator/mappings>","templated": false}}}

<http://localhost:8080/actuator/beans>



<http://localhost:8080/actuator/metrics>

{

"names": [

"http.server.requests",

"jvm.buffer.count",

"jvm.buffer.memory.used",

"jvm.buffer.total.capacity",

"jvm.classes.loaded",

"jvm.classes.unloaded",

"jvm.gc.live.data.size",

"jvm.gc.max.data.size",

"jvm.gc.memory.allocated",

"jvm.gc.memory.promoted",

"jvm.gc.pause",

"jvm.memory.committed",

"jvm.memory.max",

"jvm.memory.used",

"jvm.threads.daemon",

"jvm.threads.live",

"jvm.threads.peak",

"jvm.threads.states",

"logback.events",

"process.cpu.usage",

"process.start.time",

"process.uptime",

"system.cpu.count",

"system.cpu.usage",

"tomcat.sessions.active.current",

"tomcat.sessions.active.max",

"tomcat.sessions.alive.max",

"tomcat.sessions.created",

"tomcat.sessions.expired",

"tomcat.sessions.rejected"

]

}

<http://localhost:8080/actuator/metrics/process.cpu.usage>

{

"name": "process.cpu.usage",

"description": "The \"recent cpu usage\" for the Java Virtual Machine process",

"baseUnit": null,

"measurements": [{

"statistic": "VALUE",

"value": 6.762712079346401E-4

}],

"availableTags": []

}

Visualizing APIs with HAL Explorer

<dependency>

<groupId>org.springframework.data</groupId>

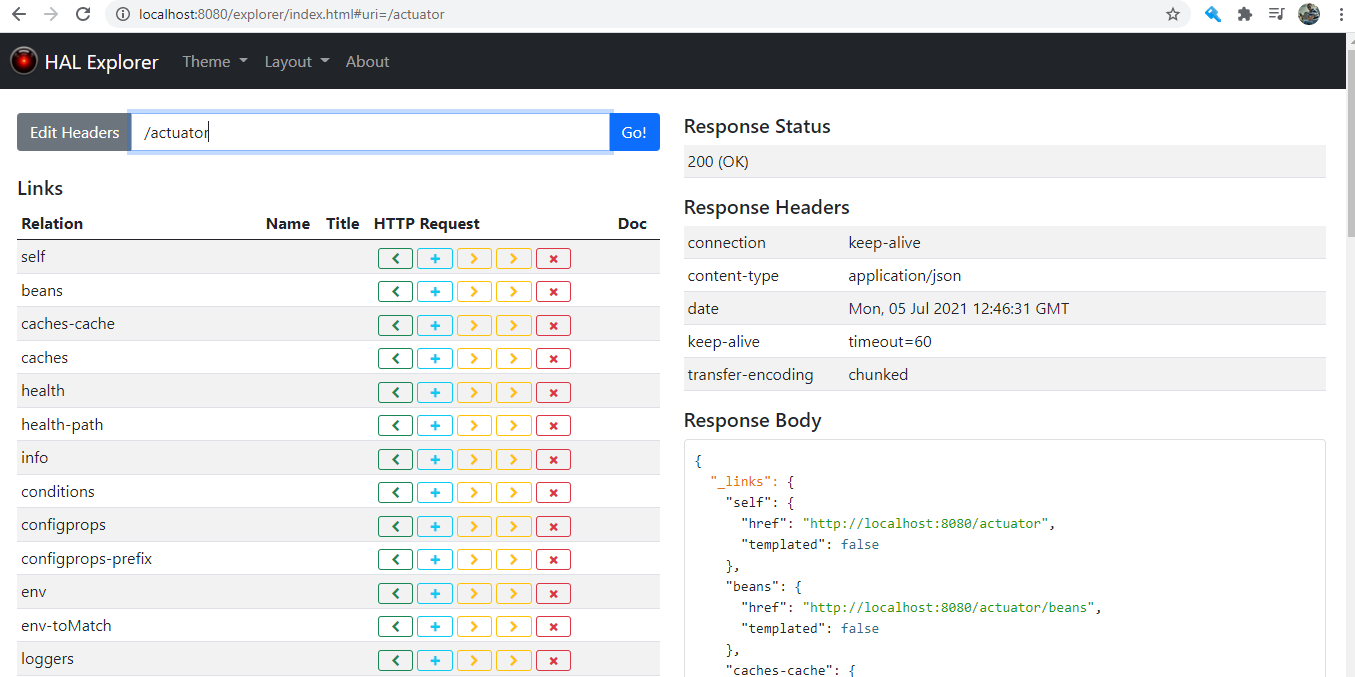
<artifactId>spring-data-rest-hal-explorer</artifactId>

</dependency>

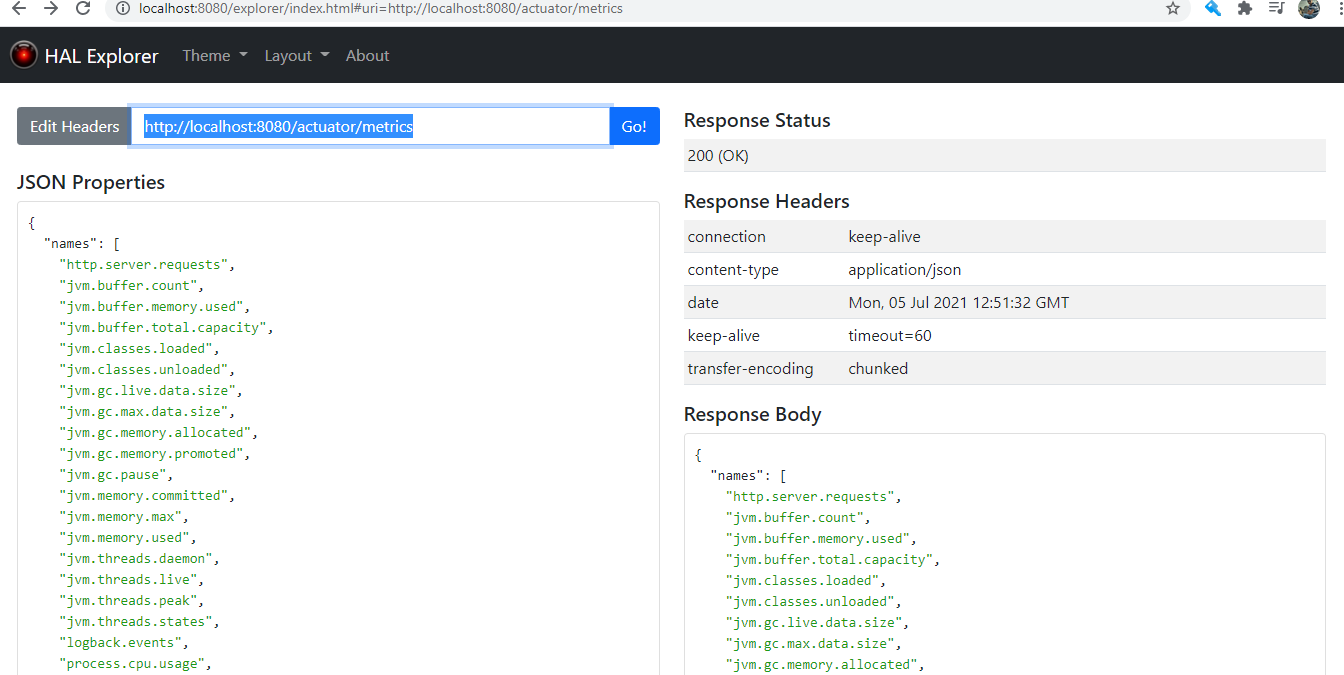
It provides visualization of actuator links –

<http://localhost:8080/>

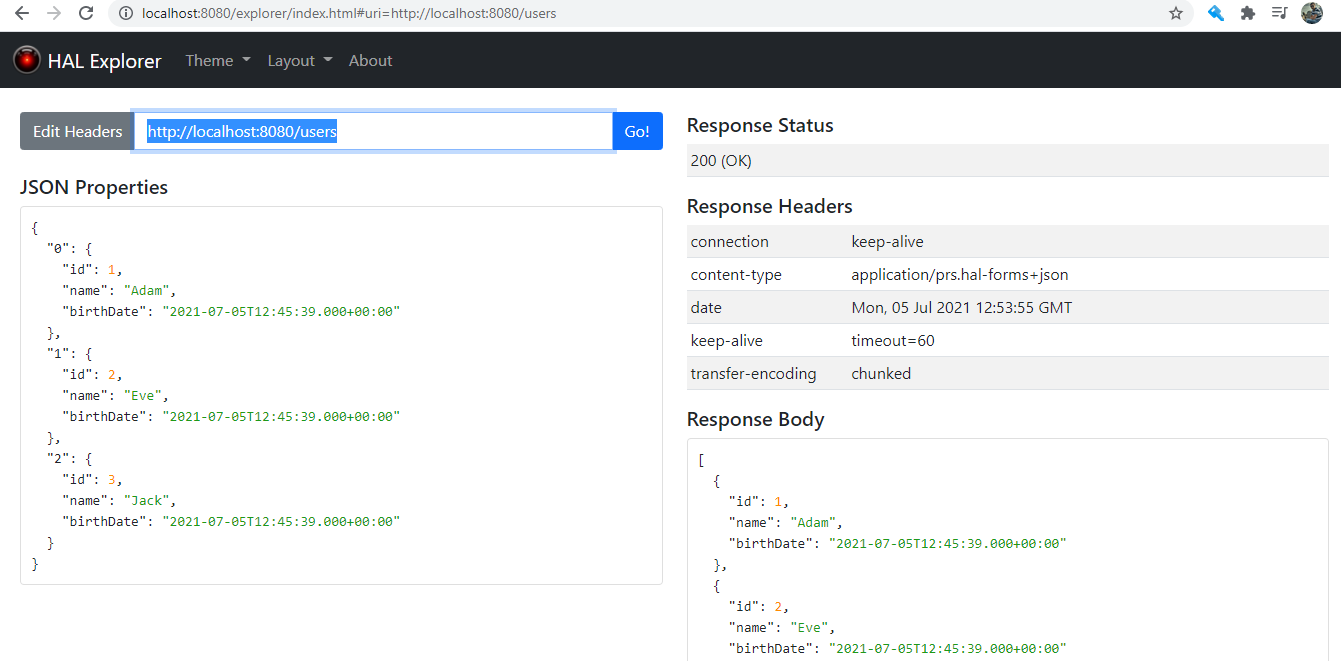
/actuator



<http://localhost:8080/actuator/metrics>



<http://localhost:8080/users>



Implementing Static Filtering for RESTful Service

We can do static filtering using below annotations –

We can use **@JsonIgnore** On the field which we want to exclude

Or at class level @JsonIgnoreProperties(value = {"field1","field2"})

 Implementing Dynamic Filtering for RESTful Service

**package** com.in28minutes.rest.webservices.restfulwebservices.filtering;

**import** com.fasterxml.jackson.annotation.JsonFilter;

**import** com.fasterxml.jackson.annotation.JsonIgnore;

**import** com.fasterxml.jackson.annotation.JsonIgnoreProperties;

@JsonFilter("SomeBeanFilter")

//@JsonIgnoreProperties(value = {"field1","field2"})

**public** **class** SomeBean {

//@JsonIgnore

**private** String field1;

**private** String field2;

**private** String field3;

**public** SomeBean(String field1, String field2, String field3) {

**super**();

**this**.field1 = field1;

**this**.field2 = field2;

**this**.field3 = field3;

}

**public** String getField1() {

**return** field1;

}

**public** **void** setField1(String field1) {

**this**.field1 = field1;

}

**public** String getField2() {

**return** field2;

}

**public** **void** setField2(String field2) {

**this**.field2 = field2;

}

**public** String getField3() {

**return** field3;

}

**public** **void** setField3(String field3) {

**this**.field3 = field3;

}

}

----------------------------------------------

package com.in28minutes.rest.webservices.restfulwebservices.filtering;

import java.util.Arrays;

import java.util.List;

import org.springframework.http.converter.json.MappingJacksonValue;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RestController;

import com.fasterxml.jackson.databind.ser.FilterProvider;

import com.fasterxml.jackson.databind.ser.impl.SimpleBeanPropertyFilter;

import com.fasterxml.jackson.databind.ser.impl.SimpleFilterProvider;

@RestController

public class FilteringController {

// field1,field2

@GetMapping("/filtering")

public MappingJacksonValue retrieveSomeBean() {

SomeBean someBean = new SomeBean("value1", "value2", "value3");

SimpleBeanPropertyFilter filter = SimpleBeanPropertyFilter.filterOutAllExcept("field1", "field2");

FilterProvider filters = new SimpleFilterProvider().addFilter("SomeBeanFilter", filter);

MappingJacksonValue mapping = new MappingJacksonValue(someBean);

mapping.setFilters(filters);

return mapping;

}

// field2, field3

@GetMapping("/filtering-list")

public MappingJacksonValue retrieveListOfSomeBeans() {

List<SomeBean> list = Arrays.asList(new SomeBean("value1", "value2", "value3"),

new SomeBean("value12", "value22", "value32"));

SimpleBeanPropertyFilter filter = SimpleBeanPropertyFilter.filterOutAllExcept("field2", "field3");

FilterProvider filters = new SimpleFilterProvider().addFilter("SomeBeanFilter", filter);

MappingJacksonValue mapping = new MappingJacksonValue(list);

mapping.setFilters(filters);

return mapping;

}

}

