**Not worked approach 1**

package com.reqres.filters;

import io.restassured.filter.Filter;

import io.restassured.filter.FilterContext;

import io.restassured.filter.log.RequestLoggingFilter;

import io.restassured.filter.log.ResponseLoggingFilter;

import io.restassured.specification.FilterableRequestSpecification;

import io.restassured.specification.FilterableResponseSpecification;

import io.restassured.filter.log.LogDetail;

import io.restassured.response.Response;

public class CustomLoggingFilter implements Filter {

private final LogDetail requestLogDetail;

private final LogDetail responseLogDetail;

private final boolean logOnFailureOnly;

private final String methodToLog;

private final Integer statusCodeToLog;

// Constructor to configure logging

public CustomLoggingFilter(LogDetail requestLogDetail, LogDetail responseLogDetail,

boolean logOnFailureOnly, String methodToLog, Integer statusCodeToLog) {

this.requestLogDetail = requestLogDetail;

this.responseLogDetail = responseLogDetail;

this.logOnFailureOnly = logOnFailureOnly;

this.methodToLog = methodToLog;

this.statusCodeToLog = statusCodeToLog;

}

@Override

public Response filter(FilterableRequestSpecification requestSpec,

FilterableResponseSpecification responseSpec,

FilterContext ctx) {

// Log Request if specified

if (requestLogDetail != null) {

new RequestLoggingFilter(requestLogDetail).filter(requestSpec, responseSpec, ctx);

}

// Proceed with the request and get the response

Response response = ctx.next(requestSpec, responseSpec);

// Log Response if specified

if (responseLogDetail != null) {

new ResponseLoggingFilter(responseLogDetail).filter(requestSpec, responseSpec, ctx);

}

// Log only on failure

if (logOnFailureOnly && (response.statusCode() < 200 || response.statusCode() >= 300)) {

new RequestLoggingFilter(LogDetail.ALL).filter(requestSpec, responseSpec, ctx);

new ResponseLoggingFilter(LogDetail.ALL).filter(requestSpec, responseSpec, ctx);

}

// Log specific HTTP Method or Status Code

if ((methodToLog != null && methodToLog.equalsIgnoreCase(requestSpec.getMethod())) ||

(statusCodeToLog != null && response.statusCode() == statusCodeToLog)) {

new RequestLoggingFilter(LogDetail.ALL).filter(requestSpec, responseSpec, ctx);

new ResponseLoggingFilter(LogDetail.ALL).filter(requestSpec, responseSpec, ctx);

}

return response;

}

// Static methods for common logging strategies (logAll, logHeadersOnly, logBodyOnly, etc.)

public static CustomLoggingFilter logAll() {

return new CustomLoggingFilter(LogDetail.ALL, LogDetail.ALL, false, null, null);

}

public static CustomLoggingFilter logHeadersOnly() {

return new CustomLoggingFilter(LogDetail.HEADERS, LogDetail.HEADERS, false, null, null);

}

public static CustomLoggingFilter logBodyOnly() {

return new CustomLoggingFilter(LogDetail.BODY, LogDetail.BODY, false, null, null);

}

public static CustomLoggingFilter logCookiesOnly() {

return new CustomLoggingFilter(LogDetail.COOKIES, LogDetail.COOKIES, false, null, null);

}

public static CustomLoggingFilter logUri() {

return new CustomLoggingFilter(LogDetail.URI, LogDetail.ALL, false, null, null);

}

public static CustomLoggingFilter logUriAndStatusOnly() {

return new CustomLoggingFilter(LogDetail.URI, LogDetail.STATUS, false, null, null);

}

public static CustomLoggingFilter logOnFailureOnly() {

return new CustomLoggingFilter(LogDetail.ALL, LogDetail.ALL, true, null, null);

}

public static CustomLoggingFilter logPostOnly() {

return new CustomLoggingFilter(LogDetail.ALL, LogDetail.ALL, false, "POST", null);

}

public static CustomLoggingFilter logStatusCode(int statusCode) {

return new CustomLoggingFilter(LogDetail.ALL, LogDetail.ALL, false, null, statusCode);

}

}

expalin from basics

// In your test

.filters(CustomLoggingFilter.logHeadersOnly())

// calls this

→ logHeadersOnly() {

return new CustomLoggingFilter(...);

}

// which calls this constructor

→ CustomLoggingFilter(requestLogDetail, responseLogDetail, ...)

**Not worked approach 2**

package com.reqres.filters;

import io.restassured.filter.Filter;

import io.restassured.filter.FilterContext;

import io.restassured.filter.log.LogDetail;

import io.restassured.response.Response;

import io.restassured.specification.FilterableRequestSpecification;

import io.restassured.specification.FilterableResponseSpecification;

public class RequestLoggingFilter implements Filter {

private final LogDetail logDetail;

private final boolean logOnFailureOnly;

private final String methodToLog;

public RequestLoggingFilter(LogDetail logDetail, boolean logOnFailureOnly, String methodToLog) {

this.logDetail = logDetail;

this.logOnFailureOnly = logOnFailureOnly;

this.methodToLog = methodToLog;

}

@Override

public Response filter(FilterableRequestSpecification requestSpec,

FilterableResponseSpecification responseSpec,

FilterContext ctx) {

// Log the request based on config BEFORE calling ctx.next

if (logDetail != null) {

logRequest(requestSpec, logDetail);

}

// Proceed with the request

Response response = ctx.next(requestSpec, responseSpec);

// Log on failure (after getting response)

if (logOnFailureOnly && (response.statusCode() < 200 || response.statusCode() >= 300)) {

logRequest(requestSpec, LogDetail.ALL);

}

// Log only specific method (e.g., GET)

if (methodToLog != null && methodToLog.equalsIgnoreCase(requestSpec.getMethod())) {

logRequest(requestSpec, LogDetail.ALL);

}

return response;

}

private void logRequest(FilterableRequestSpecification requestSpec, LogDetail detail) {

try {

switch (detail) {

case ALL:

requestSpec.log().all();

break;

case BODY:

requestSpec.log().body();

break;

case HEADERS:

requestSpec.log().headers();

break;

case COOKIES:

requestSpec.log().cookies();

break;

case PARAMS:

requestSpec.log().params();

break;

default:

// No logging

break;

}

} catch (Exception e) {

System.err.println("Error while logging request: " + e.getMessage());

e.printStackTrace();

}

}

// Factory methods

public static RequestLoggingFilter logAll() {

return new RequestLoggingFilter(LogDetail.ALL, false, null);

}

public static RequestLoggingFilter logHeadersOnly() {

return new RequestLoggingFilter(LogDetail.HEADERS, false, null);

}

public static RequestLoggingFilter logBodyOnly() {

return new RequestLoggingFilter(LogDetail.BODY, false, null);

}

public static RequestLoggingFilter logOnFailure() {

return new RequestLoggingFilter(null, true, null);

}

public static RequestLoggingFilter logForMethod(String method) {

return new RequestLoggingFilter(null, false, method);

}

}

package com.reqres.filters;

import io.restassured.filter.Filter;

import io.restassured.filter.FilterContext;

import io.restassured.filter.log.LogDetail;

import io.restassured.response.Response;

import io.restassured.specification.FilterableRequestSpecification;

import io.restassured.specification.FilterableResponseSpecification;

public class ResponseLoggingFilter implements Filter {

private final LogDetail logDetail;

private final boolean logOnFailureOnly;

private final Integer statusCodeToLog;

public ResponseLoggingFilter(LogDetail logDetail, boolean logOnFailureOnly, Integer statusCodeToLog) {

this.logDetail = logDetail;

this.logOnFailureOnly = logOnFailureOnly;

this.statusCodeToLog = statusCodeToLog;

}

@Override

public Response filter(FilterableRequestSpecification requestSpec,

FilterableResponseSpecification responseSpec,

FilterContext ctx) {

// Proceed with the request and get the response

Response response = ctx.next(requestSpec, responseSpec);

// Log response if specified

if (logDetail != null) {

logResponse(response, logDetail);

}

// Log only on failure if specified

if (logOnFailureOnly && (response.statusCode() < 200 || response.statusCode() >= 300)) {

logResponse(response, LogDetail.ALL);

}

// Log specific status code if specified

if (statusCodeToLog != null && response.statusCode() == statusCodeToLog) {

logResponse(response, LogDetail.ALL);

}

return response;

}

private void logResponse(Response response, LogDetail detail) {

try {

switch (detail) {

case ALL:

response.then().log().all();

break;

case BODY:

response.then().log().body();

break;

case HEADERS:

response.then().log().headers();

break;

case COOKIES:

response.then().log().cookies();

break;

case PARAMS:

response.then().log().params();

break;

default:

// No logging

break;

}

} catch (Exception e) {

System.err.println("Error while logging response: " + e.getMessage());

e.printStackTrace();

}

}

// Factory methods for common use cases

public static ResponseLoggingFilter logAll() {

return new ResponseLoggingFilter(LogDetail.ALL, false, null);

}

public static ResponseLoggingFilter logHeadersOnly() {

return new ResponseLoggingFilter(LogDetail.HEADERS, false, null);

}

public static ResponseLoggingFilter logBodyOnly() {

return new ResponseLoggingFilter(LogDetail.BODY, false, null);

}

public static ResponseLoggingFilter logOnFailure() {

return new ResponseLoggingFilter(null, true, null);

}

public static ResponseLoggingFilter logForStatusCode(int statusCode) {

return new ResponseLoggingFilter(null, false, statusCode);

}

}

**Worked from calude**

package com.reqres.filters;

import io.restassured.filter.Filter;

import io.restassured.filter.FilterContext;

import io.restassured.filter.log.LogDetail;

import io.restassured.response.Response;

import io.restassured.specification.FilterableRequestSpecification;

import io.restassured.specification.FilterableResponseSpecification;

public class RequestLoggingFilter implements Filter {

private final LogDetail logDetail;

private final boolean logOnFailureOnly;

private final String methodToLog;

public RequestLoggingFilter(LogDetail logDetail, boolean logOnFailureOnly, String methodToLog) {

this.logDetail = logDetail;

this.logOnFailureOnly = logOnFailureOnly;

this.methodToLog = methodToLog;

}

@Override

public Response filter(FilterableRequestSpecification requestSpec,

FilterableResponseSpecification responseSpec,

FilterContext ctx) {

// First check if we should log based on method

boolean shouldLog = methodToLog == null ||

methodToLog.equalsIgnoreCase(requestSpec.getMethod());

// Log request details before sending if configured to do so and method matches

if (shouldLog && logDetail != null && !logOnFailureOnly) {

logRequest(requestSpec, logDetail);

}

// Proceed with the request - avoid manipulating the filter chain during iteration

Response response = ctx.next(requestSpec, responseSpec);

// After getting response, check if we should log on failure

if (shouldLog && logOnFailureOnly && (response.statusCode() < 200 || response.statusCode() >= 300)) {

logRequest(requestSpec, LogDetail.ALL);

}

return response;

}

private void logRequest(FilterableRequestSpecification requestSpec, LogDetail detail) {

try {

switch (detail) {

case ALL:

System.out.println("REQUEST: " + requestSpec.getMethod() + " " + requestSpec.getURI());

System.out.println("Headers: " + requestSpec.getHeaders());

if (requestSpec.getBody() != null) {

System.out.println("Body: " + requestSpec.getBody().toString());

}

break;

case HEADERS:

System.out.println("REQUEST HEADERS: " + requestSpec.getHeaders());

break;

case BODY:

if (requestSpec.getBody() != null) {

System.out.println("REQUEST BODY: " + requestSpec.getBody().toString());

}

break;

case COOKIES:

System.out.println("REQUEST COOKIES: " + requestSpec.getCookies());

break;

case PARAMS:

System.out.println("REQUEST PARAMS: " + requestSpec.getQueryParams());

break;

default:

// No logging

break;

}

} catch (Exception e) {

System.err.println("Error while logging request: " + e.getMessage());

}

}

// Factory methods

public static RequestLoggingFilter logAll() {

return new RequestLoggingFilter(LogDetail.ALL, false, null);

}

public static RequestLoggingFilter logHeadersOnly() {

return new RequestLoggingFilter(LogDetail.HEADERS, false, null);

}

public static RequestLoggingFilter logBodyOnly() {

return new RequestLoggingFilter(LogDetail.BODY, false, null);

}

public static RequestLoggingFilter logOnFailure() {

return new RequestLoggingFilter(null, true, null);

}

public static RequestLoggingFilter logForMethod(String method) {

return new RequestLoggingFilter(null, false, method);

}

}

package com.reqres.filters;

import io.restassured.filter.Filter;

import io.restassured.filter.FilterContext;

import io.restassured.filter.log.LogDetail;

import io.restassured.response.Response;

import io.restassured.specification.FilterableRequestSpecification;

import io.restassured.specification.FilterableResponseSpecification;

public class ResponseLoggingFilter implements Filter {

private final LogDetail logDetail;

private final int maxStatusCode;

public ResponseLoggingFilter(LogDetail logDetail, int maxStatusCode) {

this.logDetail = logDetail;

this.maxStatusCode = maxStatusCode;

}

@Override

public Response filter(FilterableRequestSpecification requestSpec,

FilterableResponseSpecification responseSpec,

FilterContext ctx) {

// Execute the request first

Response response = ctx.next(requestSpec, responseSpec);

// Then log response based on configuration

boolean shouldLog = maxStatusCode <= 0 || response.statusCode() <= maxStatusCode;

if (shouldLog && logDetail != null) {

logResponse(response, logDetail);

}

return response;

}

private void logResponse(Response response, LogDetail detail) {

try {

switch (detail) {

case ALL:

System.out.println("RESPONSE Status: " + response.getStatusCode() + " " + response.getStatusLine());

System.out.println("Headers: " + response.getHeaders());

System.out.println("Body: " + response.asPrettyString());

break;

case HEADERS:

System.out.println("RESPONSE HEADERS: " + response.getHeaders());

break;

case BODY:

System.out.println("RESPONSE BODY: " + response.asPrettyString());

break;

case STATUS:

System.out.println("RESPONSE STATUS: " + response.getStatusCode() + " " + response.getStatusLine());

break;

default:

// No logging

break;

}

} catch (Exception e) {

System.err.println("Error while logging response: " + e.getMessage());

}

}

// Factory methods

public static ResponseLoggingFilter logAll() {

return new ResponseLoggingFilter(LogDetail.ALL, 0);

}

public static ResponseLoggingFilter logHeadersOnly() {

return new ResponseLoggingFilter(LogDetail.HEADERS, 0);

}

public static ResponseLoggingFilter logBodyOnly() {

return new ResponseLoggingFilter(LogDetail.BODY, 0);

}

public static ResponseLoggingFilter logStatusOnly() {

return new ResponseLoggingFilter(LogDetail.STATUS, 0);

}

public static ResponseLoggingFilter logOnlySuccessResponses() {

return new ResponseLoggingFilter(LogDetail.ALL, 299);

}

}