Duo API Testing Report

Wen whsu05@nyit.edu

The Scope of API Testing

The test we are going to do is only focus on frontend testing instead of backend testing because we do not have any access to the Duo backend service, so it is impossible to parse the Duo Data flow.

The test cases are as below, which are essential to the HTTP Request and Response.

- HTTP Response Status Checking
- HTTP Response Header Checking
- Response Time Checking
- Cookie Testing

The Testing Methodology

Using Postman for testing API of HTTP requests, utilizing a GUI, the script uses Chai Assertion Library BDD syntax to perform the test cases.

The Objective

Status Code Testing:

Sending the HTTP Get Request to the customized Duo login page, and seeing the response status is included with the status set.

The script is:

```
pm.test("Status code Testing", () =>{
 pm.expect(pm.response.code).to.be.oneOf([201,202,200,304,400,401,403,404,409,410,500]);
});
```

The complete picture of the status code can be found in the below cheatsheet.

1xx: HTTP Informational Codes	
100	Continue
101	Switching Protocols
102	Processing WebDAV
103	Checkpoint draft POST PUT
122	Request-URI too long IE7

2xx: HTTP Successful Codes	
200	OK
201	Created
202	Accepted
203	Non-Authoritative Information 1.1
204	No Content
205	Reset Content
206	Partial Content
207	Multi-Status WebDAV 4918
208	Already Reported WebDAV 5842
226	IM Used 3229 GET

3xx: HTTP Redirection Codes	
300	Multiple Choices
301	Moved Permanently
302	Found
303	See Other 1.1
304	Not Modified
305	Use Proxy 1.1
306	Switch Proxy unused
307	Temporary Redirect 1.1
308	Permanent Redirect 7538

307 and **308** are similar to **302** and **301**, but the new request method after redirect must be the same, as on initial request.

4xx: HTTP Client Error Code	
400	Bad Request
401	Unauthorized
402	Payment Required res
403	Forbidden
404	Not Found
405	Method Not Allowed
406	Not Acceptable
407	Proxy Authentication Required
408	Request Timeout
409	Conflict
410	Gone
411	Length Required
412	Precondition Failed
413	Request Entity Too Large
414	Request-URI Too Long
415	Unsupported Media Type
416	Requested Range Not Satisfiable
417	Expectation Failed
418	I'm a teapot 2324
422	Unprocessable Entity WebDAV 4918
423	Locked WebDAV 4918
424	Failed Dependency WebDAV 4918
425	Unordered Collection 3648
426	Upgrade Required 2817
428	Precondition Required draft
429	Too Many Requests draft
431	Request Header Fields Too Large draft
444	No Response nginx
449	Retry With MS
450	Blocked By Windows Parental Controls MS
451	Unavailable For Legal Reasons draft
499	Client Closed Request nginx

5xx: HTTP Server Error Codes	
500	Internal Server Error
501	Not Implemented
502	Bad Gateway
503	Service Unavailable
504	Gateway Timeout
505	HTTP Version Not Supported
506	Variant Also Negotiates 2295
507	Insufficient Storage WebDAV 4918
508	Loop Detected WebDAV 5842
509	Bandwidth Limit Exceeded nostd
510	Not Extended 2774
511	Network Authentication Required draft
598	Network read timeout error nostd
599	Network connect timeout error nostd

HTTP Code Comments	
WebDAV	WebDAV extension
1.1	HTTP/1.1
GET, POST, PUT, POST	For these methods only
IE	IE extension
MS	MS extension
nginx	nginx extension
2518, 2817, 2295, 2774, 3229, 4918, 5842	RFC number
draft	Proposed draft
nostd	Non standard extension
res	Reserved for future use
unused	No more in use, deprecated

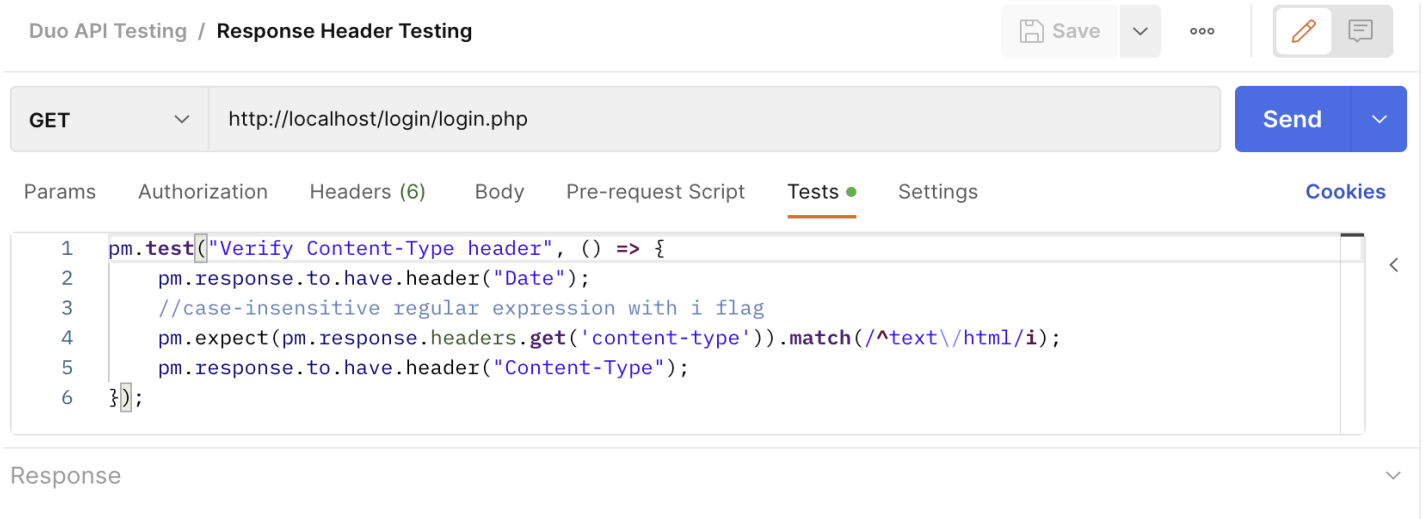
Wikipedia was used to produce all HTTP codes content:
http://en.wikipedia.org/wiki/HTTP_status

Header Testing:

Checking what types of response header the customized login page utilizes. And, add case-insensitive regular expression i flag at the end of the matched string.

The script is:

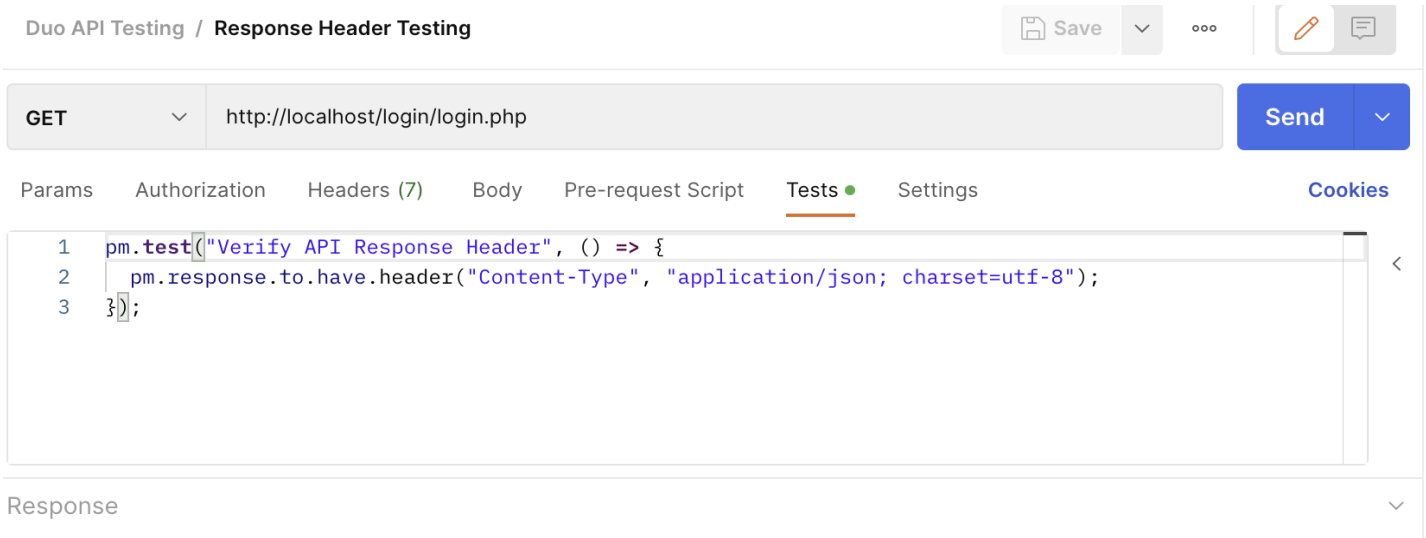
```
pm.test("Verify Content-Type header", () => {
    pm.response.to.have.header("Content-Type");
   pm.expect(pm.response.headers.get('content-type')).match(/^text\/html/i);
});
```



Checking what types of response header the customized login page utilizes.

The script is:

```
pm.test("Verify API Response Header", () => {
 pm.response.to.have.header("Content-Type", "/^application/json/i; charset=utf-8");
});
```

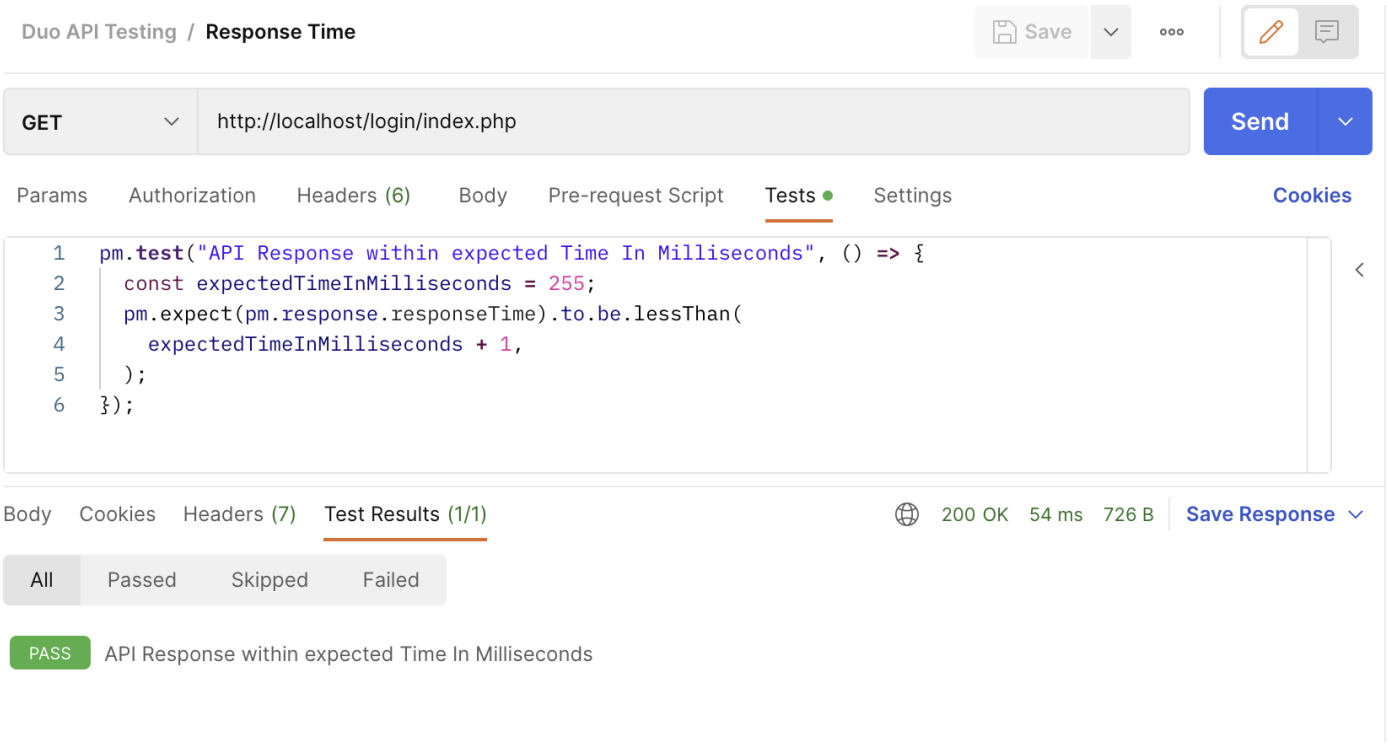


Response Time Testing:

The test is to check the HTTP response time that is in the expected time.

The script is:

```
pm.test("API Response within expected Time In Milliseconds", () => {
 const expectedTimeInMilliseconds = 255;
 pm.expect(pm.response.responseTime).to.be.lessThan(
  expectedTimeInMilliseconds + 1,
  );
});
```

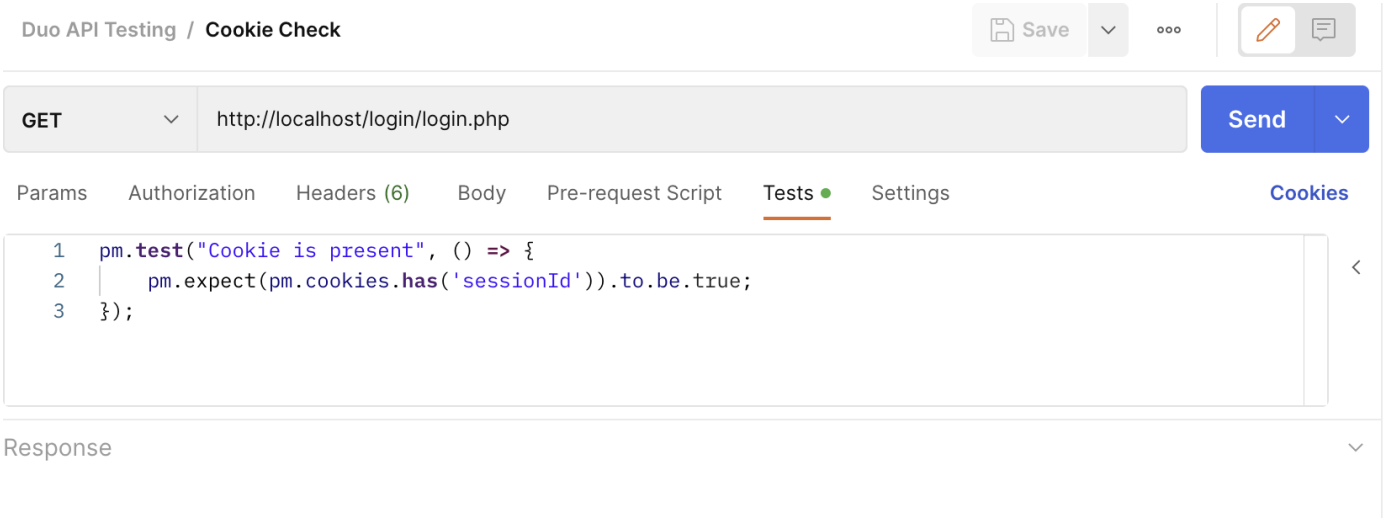


Cookie Testing:

Check the cookie exists if we send the HTTP request to the login page.

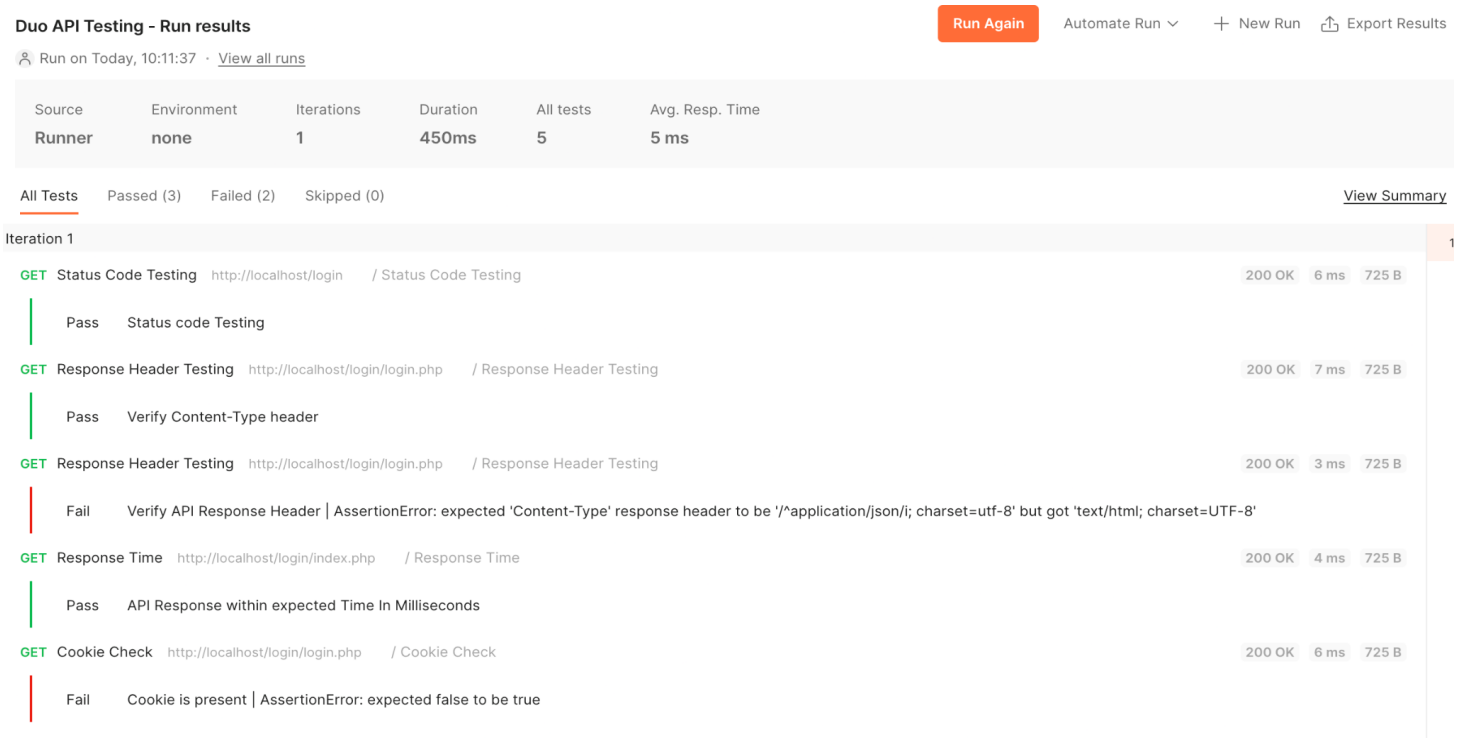
The script is:

```
pm.test("Cookie is present", () => {
  pm.expect(pm.cookies.has('sessionId')).to.be.true;
});
```



The complete picture of Testing results:

If each test case is put together, run the collection with the Postman plugin. Some test cases failed because of the Response header and cookie existence.



Resources

https://www.youtube.com/watch?v=VywxIQ2ZXw4

https://www.chaijs.com/api/bdd/

https://community.postman.com/t/test-content-type-ignoring-charset/953