System and Unit Test Report

Portfolio Tracker - Portfolio Trackers
Timothy, Kenneth, Herbert, Michael, Jesus
12/8/2020

System Test Scenarios

A. Sprint 1 story 1: As a user, I want to be able to find out how many people have seen my portfolio.
    a. Install the product using the install guide.
       Run the server and go to the /addrow endpoint on the installed domain.
       The Pageloads table should now have a new row
B. Sprint 2 story 1: As a user, I want to be able to visualize how much traffic my portfolio has received.
    a. Run the server and go to /addrow.
       Load / on the analytics subdomain
       There should now be a pageload on the graph
C. Sprint 2 story 2: As a user, I want to be able to select a range and see my website traffic within that date.
    a. Run the server and go to /addrow.
       Go to / on the analytics subdomain and select a date range containing the current date.
       There should be a pageload on the graph
D. Sprint 2 story 3: As a customer, I want to be able to embed the tracker into my website without causing HTML validation errors.
   Sprint 2 story 4: As a user, I want to be able to find out what companies viewed pages.
    a. Run the server and add the tag <script src="{{analytics_domain}}/addrow"></script> (where {{analytics_domain}} is replaced with the configured domain) to the main domain.
       Log into the UCSC vpn and load the page.
       The Pageloads table should now contain a page load from “ucsc.edu”
E. Sprint 3 story 1: As a user, I want to be able to visualize which companies viewed which pages.
    a. Run the server and add the tag <script src="{{analytics_domain}}/addrow"></script> (where {{analytics_domain}} is replaced with the configured domain) to the main domain.
       Log into the UCSC vpn and load the page.
       The dashboard located on the analytics domain should now show a page load from ucsc.edu in the companies table.
F. Sprint 3 story 2: As a user, I want to be able to know page viewers got to the end of the page.

    a. Run the server and add the tag <script src="{{analytics_domain}}/addrow"></script> (where {{analytics_domain}} is replaced with the configured domain) to the main domain.
       Check the dashboard to see the percentage of times the “unknown” company has read a page to completion.
       Load the main page on the tracked domain and scroll to bottom.
       Reload the dashboard.
       The dashboard located on the analytics domain should now show an increased percentage of viewers from “unknown” got to the end of the page.

G. Sprint 4 story 1: As a user, I want to have my data protected by an account system.
    a. Run add_user.py with the username and password as arguments in the app directory
       Run the server
       Go to /. The website redirects to the login page
       Login with the username and password
       The website redirects to /, the dashboard
       Data can now be viewed in the dashboard
       Click the logout button.
       The website redirects to the login page
    b. Run add_user.py with the username and password as arguments in the app directory
       Run the server
       Go to /. The website redirects to the login page
       Login with the username and a bad password
       The login should fail

Unit Tests

Note:

1. Tests are located in the app/tests folder
2. To run the tests, cd to the project directory and run pytest

HTTP Test

Equivalence Classes and Test Cases

Module: /addrow
Equivalence Classes:

1. Valid
    a. Valid response status code: 200
    b. Valid response “Content-Type” header: “text/javascript”
    c. One entry with correct timestamp added into the Pageloads table
    d. One entry with correct page name added into the Pageloads table
    e. One entry with correct company name added into the Pageloads table
2. Invalid
    a. Invalid response status code: anything other than 200
    b. Invalid response “Content-Type” header: anything other than “text/javascript”
    c. One entry with incorrect timestamp added into the Pageloads table
    d. More than one entries added into the Pageloads table
    e. One entry with incorrect page name added into the Pageloads table
    f. One entry with incorrect company name added into the Pageloads table

Test Cases:

1. Test 1: test_addrow_js(); Equivalence class 1a and 1b
    a. Input: GET request
    b. Expected output: 200 response code; “Content-Type” header is “text/javascript”
2. Test 2: test_addrow_db(); Equivalence class 1a and 1c
    a. Input: GET request
    b. Expected Output: One entry added into the Pageloads table with correct timestamp
3. Test 3: test_addrow_page_db(); Equivalence class 1a, 1c, and 1d
    a. Input: GET request with the “Referer” header set to “https://www.example.com/resume.html”
    b. Expected Output: One entry added into the Pageloads table with the correct timestamp and “https://www.example.com/resume.html” as the page name
4. Test 4: test_addrow_org_db(); Equivalence class 1a, 1c, and 1e
    a. Input: GET request with the “X-Real-IP” header set to “128.114.119.88”

    b. Expected Output: One entry added into the Pageloads table with the correct timestamp and “ucsc.edu” as the company name

Module: /pageloads
Equivalence Classes:

1. Valid
    a. Valid response status code: 200
    b. Valid response header: “application/json” and “text/json”
    c. Valid JSON object: company and page name field are correct
    d. Valid company data in the JSON object: JSON object with company name “ucsc.edu”, its view count 1, and its view to page bottom count 0
2. Invalid
    a. Invalid response status code: anything other than 200
    b. Invalid response header: anything other than “application/json” and “text/json”
    c. Invalid JSON object: company or page name field are not correct
    d. Invalid company data in the JSON object: error

Test Cases:

1. Test 5: test_pageloads_json(); Equivalence class 1a, 1b, and 1c
    a. Input: Pageloads table entry and GET request
    b. Expected Output: 200 OK response that contains a JSON object with correct company and page name field
2. Test 6: test_pageloads_bounded_json(); Equivalence class 1a, 1b, and 1c
    a. Input: Pageloads table entry and GET request with time bounds encoded in the URL
    b. Expected Output: 200 OK response that contains a JSON object with correct company and page name field
3. Test 7: test_pageloads_per_company_bounded_json(); Equivalence class 1a, 1b, and 1d
    a. Input: Pageloads table entry and GET request with time bounds encoded in the URL
    b. Expected Output: 200 OK response that contains a JSON object with correct company name, view count, and view to page bottom count.

Test 1: Return type of the “/addrow” endpoint

Description: The client receives a response with javascript as the content type from the “/addrow” endpoint.

Precondition: The server is set up and running properly.

Assumption: None.

Test Steps:
1. The Flask test client sends a GET request to the “/addrow” endpoint
2. The client receives a response from the endpoint
3. Check if the response status code is 200
4. Check if the “Content-Type” header of the response is “text/javascript”

Expected Result: The addrow endpoint returns 200; The “Content-Type” header of the response should be “text/javascript”.

Correct Test Result: Yes

Test 2: The “/addrow” endpoint adds an entry with timestamp into the database

Description: After the client accessed the “/addrow” endpoint, an entry with the time of the request is created and added into the database.

Precondition: The server is set up and running properly.

Assumption: None.

Test Steps:
1. Create an SQLAlchemy session
2. Record the start time
3. The Flask test client sends a GET request to the “/addrow” endpoint
4. The client receives a response from the endpoint
5. Record the stop time
6. Check if the response status code is 200
7. Check if the database has successfully added 1 entry with the correct timestamp into the database

Expected Result: The addrow endpoint returns 200; The database should contain 1 entry with the correct timestamp.

Correct Test Result: Yes

Test 3: The “/addrow” endpoint adds an entry with timestamp and page name into the database

Description: After the client accessed the “/addrow” endpoint, an entry with the time of the request and the request Referer header is created and added into the database.

Precondition: The server is set up and running properly.

Assumption: None.

Test Steps:

1. Create an SQLAlchemy session
2. Record the start time
3. The Flask test client sends a GET request with a Referer header to the “/addrow” endpoint
4. The client receives a response from the endpoint
5. Record the stop time
6. Check if the response status code is 200
7. Check if the database has successfully added 1 entry with the correct timestamp and page name into the database

Expected Result: The addrow endpoint returns 200; The database should contain 1 entry with the correct timestamp and the page name.

Correct Test Result: Yes

Test 4: The “/addrow” endpoint adds an entry with timestamp and company name into the database

Description: After the client accessed the “/addrow” endpoint, an entry with the time of the request and the domain of the request X-Real-IP header is created and added into the database.

Precondition: The server is set up and running properly.

Assumption: None.

Test Steps:

1. Create an SQLAlchemy session
2. Record the start time

3. The Flask test client sends a GET request with an X-Real-IP header to the “/addrow” endpoint
4. The client receives a response from the endpoint
5. Record the stop time
6. Check if the response status code is 200
7. Check if the database has successfully added 1 entry with the correct timestamp and company name into the database

Expected Result: The addrow endpoint returns 200; The database should contain 1 entry with the correct timestamp and the company name.

Correct Test Result: Yes

Test 5: The “/pageloads” endpoint returns correct JSON

Description: After the client accessed the “/addrow” endpoint, the “pageloads” endpoint returns a JSON file with the correct pageload object.

Precondition: The server is set up and running properly.

Assumption: None.

Test Steps:

1. Create an SQLAlchemy session
2. Clear the database
3. Record the start time
4. The Flask test client sends a GET request with a Referer and an X-Real-IP header to the “/addrow” endpoint
5. The client receives a response from the endpoint
6. Record the stop time
7. Check if the response status code is 200
8. The Flask test client sends a GET request to the “/pageloads” endpoint
9. The client receives a response from the endpoint
10. Check if the response status code is 200
11. Check if the correct content type is provided in the response
12. Check if the JSON file data is compliant

Expected Result: The pageloads endpoint returns 200; The JSON file should contain 1 object with the correct timestamp format, the company name, and the page name.

Correct Test Result: Yes

Test 6: The “/pageloads” endpoint with time bounds returns correct JSON

Description: After the client accessed the “/addrow” endpoint, the “pageloads” endpoint with time bounds returns a JSON file with the correct pageload object.

Precondition: The server is set up and running properly.

Assumption: None.

Test Steps:

1. Create an SQLAlchemy session
2. Clear the database
3. Record the start time
4. The Flask test client sends a GET request with a Referer and an X-Real-IP header to the “/addrow” endpoint
5. The client receives a response from the endpoint
6. Record the stop time
7. Check if the response status code is 200
8. Convert start and stop time to time bounds
9. The Flask test client sends a GET request to the “/pageloads” endpoint with time bounds
10. The client receives a response from the endpoint
11. Check if the response status code is 200
12. Check if the correct content type is provided in the response
13. Check if the JSON file data is compliant

Expected Result: The pageloads endpoint returns 200; The JSON file should contain 1 object with the correct timestamp format, the company name, and the page name.

Correct Test Result: Yes

Test 7: The “/pageloads” endpoint with time bounds returns correct JSON

Description: After the client accessed the “/addrow” endpoint, the “pageloads” endpoint with time bounds returns a JSON file with the correct company data.

Precondition: The server is set up and running properly.

Assumption: None.

Test Steps:

1. Create an SQLAlchemy session
2. Clear the database
3. Record the start time
4. The Flask test client sends a GET request with a Referer and an X-Real-IP header to the "/addrow" endpoint
5. The client receives a response from the endpoint
6. Record the stop time
7. Check if the response status code is 200
8. Convert start and stop time to time bounds
9. The Flask test client sends a GET request to the "/pageloads" endpoint with time bounds
10. The client receives a response from the endpoint
11. Check if the response status code is 200
12. Check if the correct content type is provided in the response
13. Check if the JSON file data has the correct view count and reach to the bottom of page count

Expected Result: The pageloads endpoint returns 200; The JSON file should contain 1 object with the correct company name, view count and reach to the bottom of page count.

Correct Test Result: Yes

IP Addr Trans Test

Equivalence Classes and Test Cases

Module: get_company_from_request()
Equivalence Classes:

1. Valid
    a. Valid domain: building5.corp.google.com
    b. IP under available domain: 128.114.119.88 for ucsc.edu
    c. Simplified domain: www.bbc.co.uk into bbc.co.uk
2. Invalid
    a. Invalid domain: 255-255-255-255.snttca.lightspeed.sbcglobal.net, None
    b. IP under unavailable domain
    c. Invalid IP: 256.256.256.256

Test Cases:

5. Test 1: test_domain_filter(); Equivalence class 1a and 2a
    a. Input: 255-255-255-255.snttca.lightspeed.sbcglobal.net
    b. Expected output: None
    c. Input: building5.corp.google.com
    d. Expected output: building5.corp.google.com
6. Test 2: test_domain_simplify(); Equivalence class 1a and 1c
    a. Input: None
    b. Expected output: None
    c. Input: building5.corp.google.com
    d. Expected output: google.com
    e. Input: home.komatsu
    f. Expected output: home.komatsu
    g. Input: xn--mgbaa2be1idb4afr.xn--lgbbat1ad8j
    h. Expected output: xn--mgbaa2be1idb4afr.xn--lgbbat1ad8j
    i. Input: www.bbc.co.uk
    j. Expected output: bbc.co.uk
7. Test 3: test_get_domain(); Equivalence class 1a and 2a
    a. Input 128.114.119.88
    b. Expected Output: string ends with “ucsc.edu”
    c. Input 256.256.256.256
    d. Expected Output: None
8. Test 4: test_full_ip_to_domain(); Equivalence class 1b and 2c
    a. Input: 128.114.119.88
    b. Expected Output: string equals “ucsc.edu”
    c. Input: 256.256.256.256
    d. Expected Output: None

Test 1: Domain Filter

Description: The filter domains function can accept and reject domains based on the ignore list.

Precondition: The mock request object and environment is set up.

Assumption: None.

Test Steps:

1. Check if a domain on the ignore list is rejected by the filter domains function
2. Check if a domain on the ignore list is accepted by the filter domains function

Expected Result: The filter domains function should reject domains on the ignore list and accept otherwise.

Correct Test Result: Yes

Test 2: Domain Simplify

Description: The simplify domain function simplifies valid domains.

Precondition: The mock request object and environment is set up.

Assumption: None.

Test Steps:
1. Check for None object
2. Check for generic Top-Level Domain with more than 3 characters
3. Check for encoded non-ASCII country-code Top-Level Domain
4. Check for encoded ASCII country-code Top-Level Domain

Expected Result: The simplify domain function should correctly simplify valid domains and return None for any invalid domains.

Correct Test Result: Yes

Test 3: Get Domain

Description: The get domain function converts the IP address in a request into a domain if a domain is available.

Precondition: The mock request object and environment is set up.

Assumption: None.

Test Steps:
1. Create a mock request with UCSC’s IP address
2. Pass the request into the get domain function
3. Check if the get domain function returns a domain that ends with “ucsc.edu”
4. Create an invalid mock request
5. Pass the request into the get domain function
6. Check if the get domain function returns None

Expected Result: The get domain function should correctly return a domain given a valid IP address and return None given an invalid IP address.

Correct Test Result: Yes

Test 4: Get Company From Request

Description: The get company from request function converts the IP address in a request into a simplified domain if a domain is available and is not on the ignore list.

Precondition: The mock request object and environment is set up.

Assumption: None.

Test Steps:

1. Create a mock request with UCSC's IP address
2. Pass the request into the get company from request function
3. Check if the get company from request function returns a domain that ends with "ucsc.edu"
4. Create an invalid mock request
5. Pass the request into the get company from request function
6. Check if the get company from request function returns None

Expected Result: The get company from request function should correctly return a simplified domain if it's available and not on the ignore list given a valid IP address and return None given an invalid IP address.

Correct Test Result: Yes

Read Test

Equivalence Classes and Test Cases

Module: /read
Equivalence Classes:

1. Valid
    a. Valid response status code: 200
    b. Valid read timestamp: timestamp indicating when the user reached the bottom of the page must be between the start and end time of the GET request
2. Invalid
    a. Invalid response status code: anything other than 200
    b. Invalid view to page bottom timestamp: error

Test Cases:

1. Test 1: test_read_add_timestamp(); Equivalence class 1a and 1b
    a. Input: Pageloads table entry and GET request with the session id encoded in the URL, the “X-Real-IP” header set to “128.114.119.88” and the “Referer” header set to “https://www.example.com/about.html”
    b. Expected Output: the read timestamp is between the start and end time of the GET request

Test 1: Correct read timestamp in Pageloads table

Description: The read endpoint saves the timestamp into the Pageloads entry that has the correct session id.

Precondition: The server is set up and running properly.

Assumption: None.

Test Steps:

1. Create a session
2. Add a Pageloads entry
3. Record the start time
4. Client sends a GET request with session id encoded in the URL
5. Record the end time
6. Check if the response status code is 200
7. Check if the timestamp in the added Pageloads entry is between the start and end time
8. Close the session

Expected Result: The read endpoint returns 200, and the timestamp in the added Pageloads entry should be between the start and end time

Correct Test Result: Yes

Sessions Test

Equivalence Classes and Test Cases

Module: /check_session
Equivalence Classes:

1. Valid
    a. Valid response code: 401
    b. Empty response body
2. Invalid
    a. Response code other than 401
    b. Anything in response body

Test Cases:

1. Test 1: test_session_expired(), equivalence class 1a, 1b
    a. Input: Sessions table entry and GET request with session header set
    b. Expected Output: A 401 error response

Test 1: Expired Sessions

Description: The get request for an expired session should return an error

Precondition: The server is set up and running properly

Assumption: None

Test Steps:

1. Create an SQLAlchemy session
2. Clear the database
3. Insert a new session into the sessions table with an expired time equal to 1 second ago
4. The flask test client sets its cookie to that inserted session
5. The flask test client sends a GET request to “/check_sessions”
6. Check that the status code is 401

Expected Result: The check_sessions endpoint should return a 401 error.

Correct Test Result: Yes

Login Test

Equivalence Classes and Test Cases

Module: /login
Equivalence Classes:

1. Valid
    a. Valid response status code: 200
    b. Only 1 session cookie
    c. Less than 1 hour between session expire time and response time
2. Invalid
    a. Invalid response status code: 200
    b. More than 1 session cookies
    c. More than 1 hour between session expire time and response time

Test Cases:

1. Test 1: test_login_duration(); Equivalence class 1a, 1b, and 1c
    a. Input: POST request with username and password encoded in a JSON object
    b. Expected Output: the session expire time for the client's session entry in the sessions table is less than 1 hour from when the client received the response

Test 1: Login endpoint creates sessions of no more than 1 hour

Description: The login endpoint creates an session entry in the sessions table with a timestamp of no more than 1 hour from its creation

Precondition: The server is set up and running properly.

Assumption: None.

Test Steps:

1. Create a session
2. Clear Users and Sessions table
3. Add a user
4. Client sends a POST request to the login endpoint with correct username and password
5. Record the time

6. Check if the response status code is 200
7. Check if there’s only 1 cookie
8. Get the session id from the cookie
9. Get the timestamp for the current session using the session id
10. Check if the timestamp is less than 1 hour from when the client received the response

Expected Result: The login endpoint returns 200, and the timestamp in the added session entry should be less than 1 hour from when the client received the response

Correct Test Result: Yes

Logout Test

Equivalence Classes and Test Cases

Module: /logout
Equivalence Classes:

1. Valid
    a. Valid response code: 200
    b. Session row with the same ID as the user is deleted from the Session table
    c. Cookie is deleted
2. Invalid
    a. Response code other than 401
    b. Session row the same ID as the user remains in the Sessions table
    c. Another session with a different ID as the user is deleted from the Sessions table
    d. Cookie is not deleted

Test Cases:

1. Test 1: test_logout(), equivalence class 1a, 1b, and 1c
    a. Input: Sessions table entry and GET request with session header set
    b. Expected Output: A 200 response with cookie deleted

Test 1: Session entry and cookie deleted after logout

Description: The client’s session entry in the sessions table and cookie are deleted after sending a request to the logout endpoint

Precondition: The server is set up and running properly

Assumption: None

Test Steps:

1. Create an SQLAlchemy session
2. Clear the database
3. Insert a new session into the sessions table with 1 hour expiration time
4. The flask test client sets its cookie to that inserted session
5. The flask test client sends a GET request to the logout endpoint
6. Check that the response status code is 200
7. Check that the session entry is deleted
8. Check that the cookie is deleted

Expected Result: the logout endpoint should return 200, and the session entry in the sessions table and the cookie are deleted after that.

Correct Test Result: Yes